Choose a design pattern for your agentic AI system

🧱 What is an Agent Design Pattern?

An **Agent Design Pattern** is a common architectural approach used to build agentic applications. It provides a distinct framework for:

1. **Organizing Components:** Structuring the agent's Model, Tools, and Memory.
2. **Integrating the Model:** Defining the role of the LLM (e.g., is it the sole planner or just a tool?).
3. **Orchestration:** Governing the flow and collaboration among single or multiple agents to accomplish a workflow.

Why Agents Need Patterns

AI agents are best suited for:

- **Open-ended problems** requiring autonomous decision-making.
- **Complex multi-step workflow** management.
- **Knowledge-intensive tasks** that require real-time external data (grounding).

The patterns provide the structure needed to manage this complexity and autonomy safely and efficiently.

🗺️ The Design Process: Choosing a Pattern

Choosing the right design pattern is not a one-time decision but a process guided by your specific needs. It involves three high-level steps:

1. Define Your Requirements (Assess the Workload)

This initial step involves closely examining the characteristics of the task you want the agent to solve:

- **Task Complexity:** Is it a simple lookup (Level 1) or a multi-step planning and collaboration task (Level 3)?
- **Latency and Performance:** How quickly must the agent respond? (e.g., real-time customer service vs. nightly reporting).
- **Cost Budget:** More complex patterns involving multiple agent calls and large models cost more.
- **Need for Human Involvement (Safety):** Does the task involve high-stakes actions that require approval (e.g., financial transactions)?

2. Review the Common Agent Design Patterns

Once you know your requirements, you learn the capabilities of the available patterns (which include both **single-agent systems** and **multi-agent systems**).

3. Select a Pattern

You match the pattern's architecture and strengths to the workload characteristics defined in Step 1.

Crucial Note: This guide assumes you understand that agent architecture is fundamentally different from simpler AI applications like:

- **Direct Model Reasoning:** Calling the LLM once without tools or memory.
- **RAG (Retrieval-Augmented Generation):** Calling the LLM once with external documents (but usually without multi-step planning).

This foundational understanding ensures you choose an agent pattern only when the task truly requires the **autonomous decision-making** and **complex workflow management** that agents provide.

Here are your complete, structured notes on **Agentic AI Design Patterns**. These are formatted specifically for you to copy and paste directly into a Google Doc for your portfolio or study guide.

📘Agentic AI Design Patterns

Definition: Agent design patterns are architectural frameworks used to organize an AI system's components (Model, Tools, Memory) and orchestrate how they solve problems. Choosing the right pattern depends on the trade-off between

Autonomy (flexibility) and **Control** (reliability).

---

1. Foundational Architectures

👤 Single-Agent System

The fundamental building block. A single AI model uses a defined set of tools and a comprehensive system prompt to handle a request from start to finish.

- **How it works:** The user sends a prompt → The Agent reasons → Calls Tools → Returns Answer.
- **Best For:** Prototypes, MVPs, and tasks requiring simple tool use (e.g., "Search for the weather and convert to Celsius").
- **Pros:** Simple to build, low cost, easy to debug.
- **Cons:** Fails as complexity grows; context window gets overcrowded; single point of failure.

🤝 Multi-Agent System (General Concept)

A system that orchestrates multiple **specialized agents** to solve a complex problem.

- **Core Principle: Decomposition.** Breaking a massive goal into smaller sub-tasks (e.g., Researcher, Writer, Coder).
- **Why Use It:** When a single prompt is too complex for one context window, or when different sub-tasks require different tools/personas.
- **Key Component: Context Engineering** (managing what information is passed between agents).

---

2. Deterministic Workflows (Predictable)

Use when the workflow path is known in advance and does not change.

➡️ Sequential Pattern (The Assembly Line)

Executes agents in a fixed, linear order.

- **Structure:** Input → [Agent A] → Output A/Input B → [Agent B] → Output.
- **Best For:** Highly structured pipelines (e.g., Data Extraction → Data Cleaning → Database Insertion).
- **Trade-off:** Low latency and cost, but very rigid. If one step fails or is slow, the whole chain stops.

🔀 Parallel Pattern (Concurrent Processing)

Executes multiple sub-agents at the same time, then synthesizes the results.

- **Structure:** Input is sent simultaneously to Agent A, Agent B, and Agent C. Their outputs are aggregated by a final step.

- **Best For:** Tasks needing diverse perspectives or speed (e.g., Analyzing a stock symbol with Sentiment Agent, Technical Agent, and Fundamental Agent simultaneously).
- **Trade-off:** Reduces latency (speed), but increases cost (multiple calls at once) and complexity in merging conflicting results.

🔄 Loop Pattern (The Monitor)

Repeatedly executes a task until a specific condition is met.

- **Structure:** Run Agent → Check Exit Condition. If "No", run again. If "Yes", stop.
- **Best For:** Monitoring tasks or retries (e.g., "Check server status every 5 minutes until it returns 200 OK").
- **Trade-off:** Risk of infinite loops (high cost) if the exit condition is never met.

---

3. Iterative & Quality Workflows (The "Quality" Loop)

Use when the first draft is likely imperfect and needs refinement.

🧠 ReAct Pattern (Reason + Act)

The standard operating system for modern agents. It forces the model to "think" before it "acts."

- **Structure:**
    1. **Thought:** Model analyzes the request.
    2. **Action:** Model chooses a tool.
    3. **Observation:** Model reads the tool output.
    4. **Repeat:** Updates thought based on observation until done.
- **Best For:** Dynamic tasks requiring continuous planning (e.g., "Find the phone number of the CEO of the company that makes the iPhone").
- **Visual:**

🧐 Review and Critique Pattern (Generator & Critic)

A specific quality assurance workflow using two distinct agents.

- **Structure:**
    1. **Generator Agent:** Creates the content (e.g., Code).

    2. **Critic Agent:** Evaluates it against hard criteria (e.g., "Does it have security bugs?").
    3. **Feedback:** If rejected, sends feedback back to Generator.
- **Best For:** High-stakes content generation (Code generation, Legal drafting) where accuracy is paramount.
- **Trade-off:** Higher cost (paying for two agents per turn).

💎 Iterative Refinement Pattern (The Polisher)

The general strategy of progressively improving an output through cycles.

- **Structure:** Generate Draft → Evaluate → Enhance Prompt/Context → Regenerate Draft.
- **Advanced Feature:** Often uses a **Prompt Enhancer** subagent to rewrite the instructions for the next loop based on previous failures.
- **Best For:** Complex creative or reasoning tasks where the "perfect" answer requires polishing (e.g., writing a novel chapter).
- **Visual:**

---

4. Dynamic Orchestration (The "Manager" Models)

Use when the path is unknown and requires AI intelligence to navigate.

🧭 Coordinator Pattern (The Router)

Uses a central "Manager" agent to dynamically route tasks.

- **Structure:** User → **Coordinator Agent** (Decides "Who handles this?") → Routes to **Specialist Subagent**.
- **Best For:** Adaptive workflows like Customer Support (Classifying "Refund" vs. "Tech Support" vs. "Sales" and routing accordingly).
- **Trade-off:** Flexible and smart, but adds latency (the Manager must "think" before anyone works).
- **Visual:**

🌳 Hierarchical Task Decomposition (The Organization Chart)

A multi-level hierarchy of agents handling massive planning.

- **Structure: Root Agent** (CEO) breaks goal into chunks → **Middle Agents** (Managers) break chunks into tasks → **Leaf Agents** (Workers) execute tasks.
- **Best For:** Extremely ambiguous, large-scale goals (e.g., "Plan a 3-day conference," "Write a software application from scratch").
- **Trade-off:** Most powerful, but highest latency and hardest to debug.
- **Visual:**

🐝 Swarm Pattern (The Roundtable)

Collaborative, all-to-all communication without a central boss.

- **Structure:** Agents act as peers. Agent A can talk to B, who talks to C, who talks back to A. They debate and hand off tasks dynamically.
- **Best For:** Brainstorming, creative problem solving, and tasks requiring diverse expert viewpoints without a rigid process.
- **Trade-off:** High creativity, but high risk of "looping" (arguing forever) and very expensive.

---

5. Special Patterns (Safety)

✋ Human-in-the-Loop (HITL)

Integrates a mandatory checkpoint for human approval.

- **Structure:** Agent works → Reaches Checkpoint → **Pauses State** → Sends Notification (Email/UI) → **Waits for Human Click** → Resumes.
- **Best For:** Sensitive actions (Refunds > $100, deploying code to production, deleting database records).
- **Visual:**

📊 Quick Decision Matrix (Cheatsheet)

If your workflow is...	**Use this Pattern**

Predictable & Linear	**Sequential**
Needs Speed / Independent Tasks	**Parallel**
Requires Monitoring/Retries	**Loop**
Requires Quality/Safety	**Review & Critique** or **HITL**
Needs Dynamic Routing	**Coordinator**
Huge, Ambiguous Project	**Hierarchical Decomposition**
Creative Brainstorming	**Swarm**