**📘 Deep Notes: Understanding AI Agent Patterns (LangGraph + General AI Agentic Architectures)**

This document provides a **thorough, example-rich, interview-ready guide** to understanding major types of AI agents used in LangGraph and modern agentic architectures. The goal is to help you explain, implement, and design AI agents confidently.

**🚀 1. What Are AI Agents?**

**AI agents** are intelligent systems that use LLMs (Large Language Models) to reason, plan, take actions (via tools), and provide dynamic responses.

They typically:

1. Receive user input (query/instruction)
2. Use **reasoning** to decide what to do
3. Optionally **invoke tools** (search, code, DB)
4. Process tool output
5. Return final response
6. In advanced cases, they collaborate with other agents

**🎯 Example:**

User: "What’s the current weather in Delhi?"

* Agent thinks: "I need a weather API"
* Agent calls: get\_weather(city='Delhi')
* Gets response: 32°C, Clear
* Returns: "It’s 32°C and clear in Delhi."

**🧠 2. ReAct Agents (Reasoning + Acting)**

**📝 Overview**

* Named after the paper: *"ReAct: Synergizing Reasoning and Acting in Language Models"*
* Combines **thought**, **action**, and **observation**.
* LLM does both reasoning and tool selection.

**🧩 Key Parts:**

* **Thought**: Reason through the problem
* **Action**: Call a relevant tool
* **Observation**: Analyze tool output
* **Final Answer**: Compose final response

**📊 Example Flow:**

User: "What is 37 multiplied by 4?"

→ Thought: I should use a calculator.

→ Action: CalculatorTool(input="37 \* 4")

→ Observation: 148

→ Thought: Now I can reply.

→ Final Answer: "The result is 148."

**✅ Best For:**

* Multi-step reasoning
* Conditional tool use
* QA, RAG, search, calculations

**💡 Analogy:**

A smart assistant who reasons, checks Google if needed, and then answers.

**⚡ 3. Reflex Agents (Rule-Based Logic)**

**📝 Overview**

* Simple, reactive agents
* Uses **if-else** rules or classification
* No reasoning or memory

**🧩 How It Works:**

* Detects intent based on keywords/rules
* Routes to a specific tool directly
* Optional LLM rephrasing

**📊 Example:**

User: "Stock price of Apple"

→ Rule: keyword="stock" → Use StockTool

→ StockTool("Apple") → 188.25

→ Final Answer: "Apple stock is $188.25."

**✅ Best For:**

* Static flows
* Fast decisions
* FAQ-style bots

**💡 Analogy:**

A phone IVR: "Press 1 for balance, 2 for recharge."

**🔁 4. Reflexion Agents (Self-Improving Agents)**

**📝 Overview**

* Introduced in the *Reflexion* paper
* Adds a **Critic LLM** that reviews the response
* Forms a feedback loop

**🧩 How It Works:**

1. Query → ReAct Agent → Initial Response
2. Critic Agent checks quality
3. If poor → Ask to retry
4. Loop continues until acceptable answer

**📊 Example:**

User: "Find key risks in this document"

→ ReAct → Misses 1 section

→ Critic: "You ignored Section B. Try again."

→ Retry → Better Answer

→ Critic: "Looks good."

**✅ Best For:**

* Tasks needing high accuracy
* Summarization, code generation, RAG QA

**💡 Analogy:**

A junior drafts → senior reviews → junior fixes → repeat

**🧭 5. Router Agents (Intent Routers)**

**📝 Overview**

* Single LLM decides **which sub-agent** should handle the query
* Only routes, doesn’t monitor execution

**🧩 How It Works:**

1. User query → Classify intent
2. Call the correct agent/tool
3. Return response

**📊 Example:**

User: "Summarize this file"

→ LLM intent: 'summarize'

→ Route to SummarizerAgent

→ Output returned

**✅ Best For:**

* Multi-agent systems with clear roles
* Entry-point dispatch

**💡 Analogy:**

Receptionist who redirects you to the right department.

**👩‍💼 6. Supervisor Agents (Task Coordinators)**

**📝 Overview**

* Higher-level controller agent
* **Assigns tasks to other agents**, monitors their output

**🧩 How It Works:**

1. User gives complex task
2. Supervisor splits it into subtasks
3. Assigns to child agents
4. Waits for all outputs
5. Combines and replies

**📊 Example:**

User: "Summarize and extract risks from this audit report"

→ Supervisor →

- Task 1: SummarizerAgent

- Task 2: RiskExtractorAgent

→ Gathers both → Combines → Returns

**✅ Best For:**

* Multi-task execution
* Coordination between agents

**💡 Analogy:**

A manager assigning tasks to team members and combining their work.

**🧬 7. Multi-Agent Systems (MAS)**

**📝 Overview**

* Full systems involving **multiple agents**, tools, memory, and routing logic

**🧠 May include:**

* Router
* Supervisor
* ReAct agents
* Critics
* Tool interfaces

**📊 Use Case:**

AI assistant with:

* Summarizer agent
* Search agent
* Code agent
* Planning agent

All coordinated together to complete a task.

**💡 Analogy:**

A company with departments (agents), managers (supervisors), and entry receptionist (router).

**🧭 Summary Comparison Table**

| **Agent Type** | **Tool Use** | **Reasoning** | **Self-Check** | **Multi-Agent** | **Use Case** |
| --- | --- | --- | --- | --- | --- |
| **ReAct** | ✅ Yes | ✅ Yes | ❌ No | ✅ Possible | QA, Search, RAG |
| **Reflex** | ✅ Yes | ❌ No | ❌ No | ✅ Possible | Rule-based bots, FAQs |
| **Reflexion** | ✅ Yes | ✅ Yes | ✅ Yes | ✅ Yes | High-accuracy QA or summaries |
| **Router** | ❌ No | ✅ Yes | ❌ No | ✅ Yes | Intent dispatching |
| **Supervisor** | ✅ Yes | ✅ Yes | ❌ (maybe) | ✅ Yes | Coordinated subtasks |
| **Multi-Agent** | ✅ Yes | ✅ Yes | ✅ (maybe) | ✅ Yes | End-to-end platforms |

**🎓 Learning Path (Recommended)**

**🔹 Step 1: Understand core ReAct agents**

* Build 1 tool + LLM
* Use LangGraph create\_react\_agent()

**🔹 Step 2: Create simple reflex agent**

* Use intent classification or keyword match

**🔹 Step 3: Add critic to form reflexion loop**

* Simulate review-retry logic

**🔹 Step 4: Build router agent**

* Use prompt to decide intent and redirect

**🔹 Step 5: Create a supervisor**

* Split task → run agents → combine results

**🔹 Step 6: Full Multi-agent project**

* Build marketplace / platform with all agent types

**🧠 Final Thought**

**Agentic design = modular, scalable, and intelligent workflows**.

Each pattern fits a different mindset:

* ReAct: Think + Act
* Reflex: Fast reaction
* Reflexion: Self-correction
* Router: Dispatcher
* Supervisor: Planner/Coordinator
* MAS: Full system

Start with one. Build intuition. Then combine.

These are the **building blocks of next-gen AI systems** — from copilots to autonomous assistants.

Let me know if you want:

* A diagram version
* PPT version
* LangGraph implementation for each
* Interview cheat sheet

**Difference between supervisor agent and router agents**

## Case 1: ****Router Agent****

### 🔹 Concept:

A **Router agent** acts like a **traffic cop** or a **receptionist**. It looks at the user query, figures out the **intent**, and **forwards the entire task** to **one best-fit agent**.

It does **not** break the task into parts, nor does it combine outputs. It simply chooses the **right tool or agent** for the job and gets out of the way.

**🔄 Flow:**

text

CopyEdit

User: "Please summarize this report and extract all key risks from it."

Step 1: Router analyzes the intent

→ Tries to classify what the user is asking

→ May use keyword rules or LLM classifier

Step 2: Router decides:

→ This is mostly a "risk extraction" task

→ Send to "RiskExtractorAgent" (only)

Step 3: RiskExtractorAgent processes input

→ Returns risks (maybe also includes summary, or not)

Step 4: Final answer returned to user

**🧠 Key Behavior:**

* **Single path**: Only **one** child agent is invoked.
* **No splitting** of the task.
* **No post-processing** or combining logic.

**🧑 Real Analogy:**

A receptionist who hears your problem and sends you to **one** specialist — even if your problem had two parts.

**🧑‍💼 Case 2: Supervisor Agent**

**🔹 Concept:**

A **Supervisor agent** is like a **project manager**. It sees the whole problem, knows there are **multiple subgoals**, and intelligently **delegates subtasks** to different agents or tools — then **waits**, **reviews**, and **combines** outputs.

**🔄 Flow:**

text

CopyEdit

User: "Please summarize this report and extract all key risks from it."

Step 1: Supervisor receives the full query

Step 2: Supervisor understands there are two tasks:

→ Task A: Summarize the report

→ Task B: Extract key risks

Step 3: Supervisor routes:

→ Task A → SummarizerAgent

→ Task B → RiskExtractorAgent

Step 4: Waits for both results:

→ Summary from A

→ Risks from B

Step 5: Supervisor combines final output:

→ "Here’s the summary: … And here are the key risks: …"

Step 6: Sends answer to user

**🧠 Key Behavior:**

* **Multi-task orchestration**
* **Delegates subtasks to multiple agents**
* **Combines or refines outputs**
* **Can retry or reroute failed tasks**

**🧑 Real Analogy:**

A project manager who divides a large task between two team members (summarizer and risk analyst), waits for both, and then sends the combined result to the client.

**🧨 Key Differences (Side-by-Side)**

| **Feature** | **Router Agent** | **Supervisor Agent** |
| --- | --- | --- |
| **Primary Role** | Dispatcher / Classifier | Orchestrator / Coordinator |
| **# of agents used** | One per task | Many per task |
| **Can split subtasks?** | ❌ No | ✅ Yes |
| **Combines multiple outputs?** | ❌ No | ✅ Yes |
| **Reacts to failures?** | ❌ Minimal logic | ✅ Can retry or replace agent |
| **Best For** | Simple routing based on intent | Complex workflows needing multiple tools/agents |
| **Analogy** | Receptionist or switchboard | Project manager |

**📘 Visual Summary (Mental Model)**

**Router Agent:**

text

CopyEdit

┌────────────┐

User Query → │ Router │

└────┬───────┘

↓

┌────────────┐

│ One Agent │ → Final Answer

└────────────┘

**Supervisor Agent:**

text

CopyEdit

┌───────────────┐

User Query → │ Supervisor │

└─────┬─────────┘

┌──────────┴──────────────┐

Summarizer Agent Risk Extractor Agent

↓ ↓

"Summary..." "Key Risks..."

└──────────┬──────────────┘

↓

Final Combined Answer

**🎓 Interview Tip**

If someone asks:

“What’s the difference between a Router and a Supervisor agent?”

You can confidently say:

“A Router agent chooses **one best-fit agent** based on intent and sends the full task there. A Supervisor agent understands the query as **multi-part**, breaks it into subtasks, delegates them to different agents, and **aggregates** the results — much like a project manager coordinating a team.”