**Agentic AI:**

**Agentic AI** refers to **AI systems that can independently make decisions, act in dynamic environments, and pursue goals** over time. Unlike standard AI systems that respond only to specific inputs, agentic AI combines **reasoning**, **planning**, and **acting** to solve complex tasks **autonomously** while interacting with its environment.

Agentic AI is a type of AI that’s all about **autonomy**. This means it can make decisions, take actions, and even learn on its own to achieve specific goals.

Agentic AI is more than just a model—it behaves like an autonomous entity, capable of adapting to changing situations and iterating through tasks without constant human intervention.

* **Why Agentic AI Required:**
* **Handling Complexity:** Real-world problems often require multi-step reasoning, dynamic decision-making, and interaction with external systems.
* **Autonomy:** Agentic AI reduces the need for human supervision, enabling tasks to be completed with minimal input.
* **Adaptability:** It adjusts its behaviour in response to new information or changing circumstances.
* **Efficiency:** By acting independently, agentic AI can manage workflows, optimize tasks, and achieve goals faster.
* **Importance of Agentic AI:**
* **Scalability:** Supports large-scale applications like personalized healthcare, autonomous vehicles, and intelligent customer service.
* **Problem-Solving:** Excels in scenarios where goals are ambiguous or require iterative decision-making.
* **Innovation:** Enables entirely new capabilities, such as intelligent agents performing complex web navigation or managing multi-system integrations.
* **Human-AI Collaboration:** Augments human decision-making by handling repetitive or logic-intensive tasks.
* **Examples of Agentic AI:**
  1. **Self-Driving Cars:** One of the most exciting uses of Agentic AI is in autonomous vehicles. These AI systems perceive their surroundings, make driving decisions, and learn from every trip. Over time, they get better at navigating and handling new challenges on the road. For example, **Tesla’s Full Self-Driving system**is an example of Agentic AI that continuously learns from the driving environment and adjusts its behaviour to improve safety and efficiency.
  2. **Supply Chain Management:** Agentic AI is also helping companies optimize their supply chains. By autonomously managing inventory, predicting demand, and adjusting delivery routes in real-time, AI can ensure smoother, more efficient operations. **Amazon’s Warehouse Robots,**powered by AI, are an example — these robots navigate complex environments, adapt to different conditions, and autonomously move goods around warehouses.
  3. **Cybersecurity:** In the world of cybersecurity, Agentic AI can detect threats and vulnerabilities by analysing network activity and automatically responding to potential breaches.[**Darktrace, an AI cybersecurity company,**](https://darktrace.com/) uses Agentic AI to autonomously detect, respond to, and learn from potential cyber threats in real-time.
  4. **Healthcare:** AI is playing a big role in healthcare, too. Agentic AI can assist with diagnostics, treatment recommendations, and patient care management. It analyzes medical data, identifies patterns, and helps doctors make more informed decisions. For instance, **IBM’s Watson Health** uses AI to analyze massive amounts of healthcare data, learning from new information to offer insights that help doctors and healthcare professionals.
* **Example of Agentic AI in Action:**

**Scenario: Travel Planning Assistant**

A user asks an AI system: "Plan a 5-day trip to Paris, including flights, hotels, and sightseeing."

**How Agentic AI Works:**

1. **Goal Setting:** The AI identifies the goal (create a complete travel plan).
2. **Planning:**
   * Searches for flights and compares prices.
   * Books a suitable hotel based on preferences.
   * Schedules sightseeing tours.
3. **Dynamic Interaction:** Adjusts plans if flights or hotels are unavailable or if the user provides new input like, "Include a visit to the Eiffel Tower."
4. **Execution:** Finalizes bookings, creates an itinerary, and sends it to the user.

**Output:** A complete itinerary with booked flights, hotel accommodations, and sightseeing activities, adaptable to last-minute changes.

* **Comparison: Agentic AI vs. AI Agent vs. Generative AI**

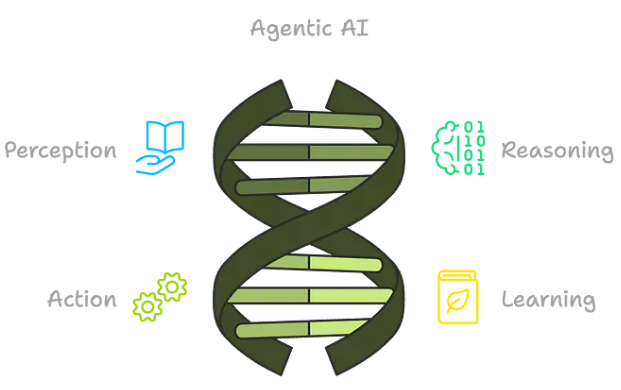
|  |  |  |  |
| --- | --- | --- | --- |
| **Aspect** | **Agentic AI** | **AI Agent** | **Generative AI** |
| **Definition** | Autonomous systems capable of reasoning, planning, and acting. | Task-specific systems using tools for discrete steps. | AI that generates outputs (text, images, code). |
| **Goal** | Long-term, multi-step problem-solving. | Perform single tasks by calling tools or APIs. | Generate creative or factual content. |
| **Autonomy** | High autonomy iterates and adapts over time. | Partial autonomy relies on predefined workflows. | No autonomy, responds to specific prompts. |
| **Example Task** | Plan and execute a 5-day trip. | Use a weather API to answer "What's the weather?" | Generate an essay on "Benefits of AI." |
| **Key Strength** | Handles dynamic, goal-driven tasks. | Efficient for single-step tasks. | Creative, content-focused output generation. |
| **Interactivity** | Constantly interacts with the environment. | Uses predefined tools as needed. | Static responses without external interactions. |
| **Complexity Handling** | Handles ambiguous or multi-step goals. | Best for clear, well-defined queries. | No reasoning or external interaction. |

* **How Agentic AI Works:**

Agentic AI operates by combining **reasoning**, **planning**, **action-taking**, and **feedback loops** to achieve goals autonomously. It dynamically interacts with its environment, adapting to new information and iterating its actions to refine results.

**Agentic AI Operates in Four Key Stages:**

1. **Perception:** It gathers data from the world around it.
2. **Reasoning:** It processes this data to understand what’s going on.
3. **Action:** It decides what to do based on its understanding.
4. **Learning:** It improves and adapts over time, learning from feedback and experience.



This makes Agentic AI highly autonomous and able to handle complex tasks that require reasoning, problem-solving, and adapting to new situations.

**Here is detailed explanation of its working principle:**

1. **Goal Understanding:**

Agentic AI begins by interpreting the user’s query or task and defining a clear goal.

**Example:** For a query like "Plan a 5-day trip to Paris," the AI identifies the objective: create a comprehensive travel itinerary.

1. **Multi-Step Planning:**

Agentic AI creates a **structured plan** that breaks the goal into smaller, manageable subtasks.

**Example:**

* **Task 1:** Search for flights.
* **Task 2:** Book a hotel.
* **Task 3:** Suggest sightseeing locations.

1. **Tool and Resource Usage:**

Agentic AI interacts with **tools**, **APIs**, **databases**, or external systems to gather information and execute tasks.

**Example:**

* Call a **flight API** to find the best prices.
* Use a **weather API** to ensure sightseeing plans align with favourable weather.

1. **Iterative Reasoning and Decision-Making:**

At every step, Agentic AI evaluates the intermediate results and updates its plan if necessary.

**Example:**

* If no flights are available for the specified dates, the AI searches for nearby airports or adjusts the travel dates.

1. **Dynamic Adaptation:**

Agentic AI dynamically adapts its actions based on user feedback or new data.

**Example:**

* **User input:** "Include a visit to the Eiffel Tower."
* The AI modifies the sightseeing plan to prioritize this request.

1. **Feedback Loops:**

Agentic AI integrates **feedback loops** to refine its decisions and outputs continuously.

**Example:**

* After creating an itinerary, the AI asks: "Does this plan meet your requirements?" If the user requests changes, it revises the itinerary.

1. **Final Output and Execution:**

After completing all steps and ensuring alignment with the goal, Agentic AI presents the final result or executes the task.

**Example:**

* Provide the full itinerary, including booked flights, hotel reservations, and sightseeing plans.

**Working Model:**

Agentic AI operates in a **cyclic process**:

**Input Understanding** 🡪 **Planning** 🡪 **Tool Use** 🡪 **Observation** 🡪 **Reasoning** 🡪 **Adaptation**.

**Multi-Modal Agent:**

A **Multi-Modal Agent** is an AI system that can process and integrate multiple types of data or inputs, such as **text, images, audio, and video**, to perform tasks or make decisions. Unlike single-modal agents that work with only one data type, multi-modal agents combine different modalities to enhance understanding and provide more comprehensive responses.

* **Example: Multi-Modal Agent in a Virtual Assistant**

**Scenario:** A user interacts with a virtual assistant by sending both a text query and an image.

1. **User Query:** "What kind of flower is this?" (along with an image of a flower)
2. **Multi-Modal Processing:**

* The agent analyzes the **text** query to understand that the user is asking for flower identification.
* It processes the **image** to recognize the flower using computer vision techniques.

1. **Response:** "This is a sunflower."

In this case, the **text query** guides the agent's task, while the **image** provides the necessary visual data to perform the identification. The agent integrates these inputs to deliver a precise answer.

**LangGraph:**

**LangGraph** is **visualization and debugging tool** designed for LangChain workflows. It provides an intuitive way to **visualize chains, agents, and components** by representing them as graphs. This graphical representation helps developers understand, debug, and optimize their LangChain-based pipelines.

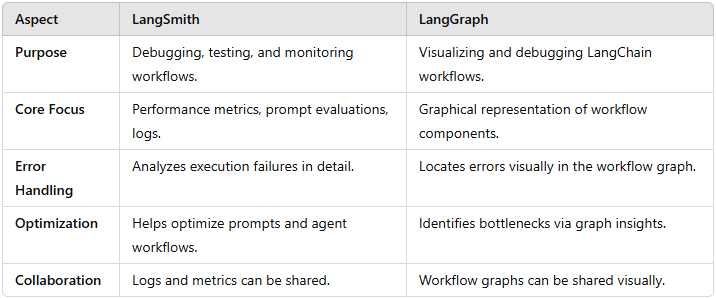
* **Why LangGraph is required:**
  1. **Complex Workflow Debugging:** LangChain pipelines can involve multiple chains, agents, and tools. LangGraph visually simplifies these complex workflows.
  2. **Optimization:** It helps developers identify bottlenecks and inefficiencies in their pipelines.
  3. **Error Tracing:** LangGraph makes it easier to pinpoint errors and debug specific parts of the workflow.
  4. **Better Collaboration:** Teams can use visual graphs to explain and share pipeline designs.
* **Concepts of LangGraph:**
  1. **Workflow:**
     + **Definition:** A **workflow** in LangGraph represents the complete sequence of tasks or operations involved in solving a problem or completing a process.
     + **Structure:** Composed of **nodes** (components like chains, agents, or tools) and **edges** (connections between nodes showing data flow). **Node**  itself is a function or collection of function.
     + **Purpose:** Helps visualize, debug, and optimize LangChain-based pipelines.
  2. **Direct Cyclic Graph (DCG):**
     + **Definition:** A graph with directed edges **where cycles (loops) exist**.
     + **Meaning:** The workflow may involve repetitive processes or feedback loops.
     + **Usage in LangGraph:** Represents iterative workflows where outputs of one node might be fed back as inputs to a previous node.
     + **Example:**

Generate an initial response → Analyze response quality → Modify inputs → Repeat.

* 1. **Direct Acyclic Graph (DAG):**
     + **Definition:** A graph with directed edges **where no cycles (loops) exist**.
     + **Meaning:** A purely forward-moving workflow, where data flows from start to end without revisiting any nodes.
     + **Usage in LangGraph:** Represents linear workflows without iterations.
     + **Example:**

Parse query → Retrieve documents → Generate answer → Return result.

* 1. **Normal Edges:**
     + **Definition:** These edges represent **regular data flow** between two nodes.
     + **Purpose:** Show connections where outputs of one component are inputs to the next.
     + **Example:** If an **LLM** generates a query and passes it to a **retriever**, the edge connecting them is a **normal edge**.
  2. **Conditional Edges:**
     + **Definition:** These edges represent **conditional data flow**, where the path taken depends on a decision or condition evaluated at a node.
     + **Purpose:** Enable dynamic workflows where execution changes based on intermediate results.
     + **Example:** A decision node might check if the user’s query involves a numerical calculation:
       - **Condition**: If true, the edge connects to a calculator tool.
       - If false, the edge connects to a document retriever.
* **LangGraph vs. LangSmith:**



**Different Types of Agentic Patterns:**

**Agentic Patterns** refer to **architectural and behavioral frameworks** used to design AI agents that can autonomously reason, plan, and act in dynamic environments. These patterns define how **LLM-powered agents** interact with users, external tools, memory, and decision-making mechanisms to solve complex tasks.

Agentic patterns are crucial in **Agentic AI**, where agents operate **independently**, making informed decisions by retrieving relevant knowledge, using tools, and refining responses through iterative feedback loops.

* **Key Characteristics of Agentic Patterns:**
* **Autonomy:** Agents operate with minimal human intervention.
* **Multi-Step Reasoning:** Breaks down tasks into logical steps before execution.
* **Tool Usages:** Calls APIs, databases, or external tools dynamically.
* **Memory Handling:** Maintains long-term and short-term context across interactions.
* **Iterative Improvement:** Adapts and refines its output based on new information.
* **Types of Agentic Patterns:**

1. **RaAct (Reasoning + Acting):**

* **What?** 
  + Combines **thought generation (reasoning)** with **tool execution (acting)** in an iterative loop.
* **How it Works?**
  + The agent first thinks about a step, then executes an action, observes the result, and repeats the process.
* **Example: Customer Support Chatbot**
  + **Query:** “My internet is slow. What should I do?”
  + **Agent’s Steps:**
    1. **Reason:** “I need to check the user's internet speed.”
    2. **Act:** Calls a speed test API.
    3. **Observe:** Gets speed results.
    4. **Reason Again: “**Now, I will provide troubleshooting steps.”
    5. **Act:** Suggests solutions like restarting the router.

1. **Self-Ask with Search:**

* **What?**
  + Breaks down a complex question into smaller sub-questions and searches for answers before responding.
* **How it Works?**
  + The agent decomposes a query into **sub-queries**, retrieves information, and integrates the answers.
* **Example: Question Answering System**
  + **Query:** “Who was the U.S. President when the Eiffel Tower was completed, and how tall is it?”
  + **Agent’s Steps:**
    1. **Sub-question-1:** “When was the Eiffel Tower completed?” 🡪 1889
    2. **Sub-question-2:** “Who was the U.S. President in 1889?” 🡪 Benjamin Harrison
    3. **Sub-question-3:** “What is the height of the Eiffel Tower?” 🡪 330 meters
    4. **Final Answer:** “The Eiffel Tower was completed in 1889 when Benjamin Harrison was President, and it is 330 meters tall.”

1. **Tool-Using Agent Pattern:**

* **What?**
  + Uses external **APIs, databases, and plugins** to fetch real-world information.
* **How it Works?**
  + The agent detects **when a tool is needed**, executes API calls, and integrates responses.
* **Example: Weather Assistant**
  + **Query:** “What’s the weather in New York right now?”
  + **Agent’s Steps:**
    1. **Decides a tool is needed:** Calls a **weather API**
    2. **Fetches API results:** “New York is 75°F with clear skies.”
    3. **Returns the Answers:** “The current temperature in New York is 75°F with clear skies.”

1. **Multi-Agent Collaboration:**

* **What?**
  + Multiple specialized agents work together to complete a task.
* **How it Works?**
  + Each agent is responsible for a **specific role** (e.g., Researcher, Writer, and Editor).
* **Example: AI Research and Writing Team**
  + **Task:** “Generate a report on AI advancements in healthcare.”
  + **Agent Workflow:**
    1. **Research Agent:** Gathers latest AI healthcare studies.
    2. **Writing Agent:** Summarizes findings into a structured report.
    3. **Editing Agent:** Reviews grammar, coherence, and factual correctness.
    4. **Final Output:** “Here’s a well-researched report on AI in healthcare.”

1. **Human-in-the-Loop (HITL):**

* **What?**
  + The agent **asks a human for input** when uncertainty is high.
* **How it Works?**
  + If the AI is **unsure**, it requests human validation before proceeding.
* **Example: Legal AI Assistant**
  + **Task:** “Analyze this contract for risky clauses.”
  + **Agent’s Steps:**
    1. Scans contract and detects **potential risks**.
    2. Highlights **ambiguous clauses** for **human review**.
    3. The **human lawyer approves or edits** flagged sections.
    4. AI generates a final **risk assessment report**.

1. **Memory-Augmented Agents:**

* **What?**
  + Agents that remember past interactions for better long-term responses.
* **How it Works?**
  + Uses **short-term and long-term memory** to maintain continuity.
* **Example: Personalized Virtual Assistant**
  + **User:** “Remind me about my meeting with John next Monday.”
  + **Agent Stores:** Meeting with John on Monday at 3 PM.
  + **Next Week:** The assistant **remembers** and sends a timely reminder.
* **Why are Agentic Patterns Important:**
  + **Enhance AI Autonomy:** Reduce manual intervention.
  + **Improved Context Understanding:** Agents can retain **memory and reasoning**.
  + **Enable Real-World Applications:** Used in **finance, healthcare, customer service, and automation**.
  + **Boost AI Efficiency:** Faster decision-making with **tools and retrieval mechanisms**.

**Agentic Patterns** empower AI models to function like **intelligent assistants** by reasoning, planning, and acting in structured workflows. Each pattern serves **different use cases**, from **retrieving real-time data (Tool-Using Agents)** to **collaborating with humans (HITL)**.

**Different Types of RAG Agent:**

There is different type of RAGs, like **a)** **Agentic RAG**, **b)** **Adaptive RAG**, **c)** **Corrective RAG**, **d)** **Self-RAG**, **e)** **SQL Agent**.

* **Agentic RAG:**

**Agentic RAG (Retrieval-Augmented Generation)** is an advanced AI framework that enhances traditional **RAG** by incorporating **agentic behaviors** like **reasoning, tool usage, planning, and decision-making**. Unlike basic RAG, where the model passively retrieves information, **Agentic RAG** actively selects **which sources to query, how to process retrieved data, and when to refine the query** for better results.

* **Difference between Traditional RAG & Agentic RAG:**

|  |  |  |
| --- | --- | --- |
| **Feature** | **Traditional RAG** | **Agentic RAG** |
| **Retrieval Process** | Passive retrieval of documents based on query | Active retrieval with dynamic query refinement |
| **Query Understanding** | Direct retrieval based on user input | Breaks down complex queries into sub-queries |
| **Reasoning Ability** | No reasoning, just retrieves relevant chunks | Uses reasoning to decide the best retrieval strategy |
| **Verification & Filtering** | No verification, retrieved docs used as-is | Cross-checks and filters retrieved results for accuracy |
| **Adaptability** | Static process | Adapts based on query complexity and available data |
| **Handling Multi-Step Tasks** | Limited, single-step retrieval | Supports multi-step reasoning and iterative refinement |
| **Response Accuracy** | Can contain irrelevant or outdated info | Improves accuracy with iterative verification |
| **Tool Usage** | Limited or none | Can use APIs, databases, and external tools to enhance retrieval |
| **Example of Use Case** | Simple FAQ-style chatbot | AI research assistant, legal AI, advanced knowledge workers |

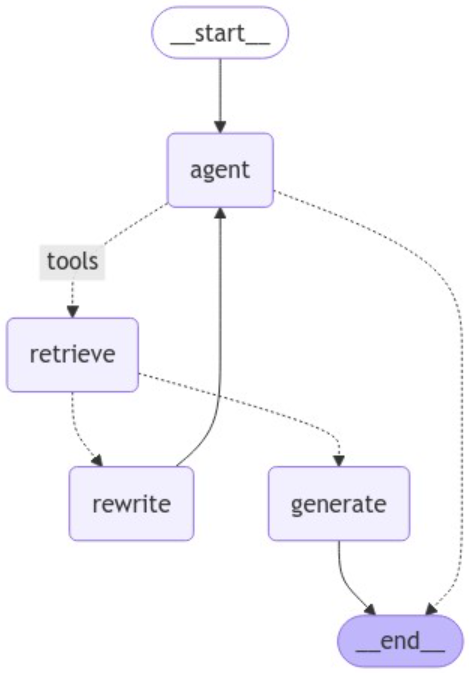
* **Why Agentic RAG:**
* **Improves retrieval accuracy** by dynamically adjusting queries.
* **Reduces hallucinations** by verifying and filtering retrieved content.
* **Handles multi-step reasoning** with an iterative approach.
* **Uses multiple retrieval strategies** (e.g., keyword-based, vector search, hybrid search).
* **How does Agentic RAG Works?**

1. **Traditional RAG Flow:**

**User Query 🡪 Retrieve Relevant Documents 🡪 Generate Response**

1. **Agentic RAG Flow:**

**User Query 🡪 Agent Reasoning (Decide Retrieval Strategy) 🡪 Retrieve Data Dynamically 🡪 Verify & Filter Information 🡪 Generate Accurate Response**



Agentic RAG allows the AI to **think, refine, and improve responses** based on retrieved documents before generating a final answer.

* **Benefits of Agentic RAG:**

1. **Enhanced Query Understanding:**

* **Breaks down** complex questions into smaller parts.
* **Chooses the best retrieval method** dynamically.

1. **More Accurate and Reliable Response:**

* Uses **iterative refinement** to improve retrieval.
* **Cross-checks** sources to minimize hallucinations.

1. **Adaptive Hallucination and Improves Trust:**

* Adjusts **retrieval strategy** based on query intent.
* Supports **multi-modal inputs** (text, images, etc.).

1. **Reduces Hallucination and Improved Trust:**

* Applies **fact-checking mechanisms** before generating responses.
* Uses **hybrid search** (keyword + vector) for better results.

1. **Ideal for Complex Use Cases:**

* Works well for **legal AI, research assistants, and enterprise knowledge management**.
* **Limitation of Agentic RAG:**

1. **Higher Computational Cost:**

* Requires **more processing power** due to multi-step reasoning.
* Can be **slower than traditional RAG**.

1. **More Complex to Implement:**

* Needs **agents, retrievers, and tool integrations**.
* Requires **custom logic** for query breakdown and refinement.

1. **Potential Over-Reliance on External Tools:**

* If an external API or retriever fails, performance drops.
* Requires **constant monitoring** to maintain accuracy.

1. **Risk of Overfitting to Retrieval Patterns:**

* May **bias towards certain sources** if not properly balanced.
* Needs **regular updates** to avoid stale information.

* **Retrieval Strategy Used by Agentic RAG:**

Agentic RAG improves upon Traditional RAG by employing **adaptive retrieval strategies** to enhance response accuracy and reduce hallucinations. **The key retrieval strategies used by Agentic RAG include:**

1. **Iterative Retrieval:**

**What it is:**

* Instead of retrieving all information in one step, the agent **refines the query** dynamically.
* If the initial retrieval isn’t satisfactory, the agent **modifies the query** and retrieves again.

**Example:**

* A user asks: “What was the impact of the Industrial Revolution on modern AI?”
* The agent **breaks it down** into:
  + What were the key technological advancements of the Industrial Revolution?
  + How did these technologies influence the development of computing?
  + How does computing relate to AI?
* It retrieves information **step by step** before generating a final response.

1. **Multi-Hop Retrieval:**

**What it is:**

* Used when answering requires multiple knowledge sources.
* The agent **connects information from multiple documents** before answering.

**Example:**

* **User:** “How did Einstein’s work influence modern AI ethics?”
* **Step 1:** Retrieve information on Einstein’s theories.
* **Step 2:** Retrieve information on how his theories influenced computing.
* **Step 3:** Retrieve information on AI ethics and check for connections.

The agent **links the retrieved documents** to form a **comprehensive answer**.

1. **Self-Reflective Retrieval:**

**What it is:**

* The agent **evaluates its own retrieved data** before responding.
* If the retrieved information is **insufficient or irrelevant**, it **self-corrects** and retrieves again.

**Example:**

* User: “What were the key economic policies of the Great Depression?”
* Initial retrieval contains **historical events but lacks economic policies**.
* The agent detects this gap and **retrieves again with a refined query**.

This ensures **only relevant information is used** in the final response.

1. **Hybrid Search (Sparse + Dense Retrieval):**

**What it is:**

* Combines traditional keyword search (BM25) and vector search (semantic search).
* **Keyword search** ensures factual precision, and **vector search** captures semantic meaning.

**Example:**

* User: “Tell me about neural network optimizers like Adam and SGD.”
* **Sparse Retrieval (BM25)** finds documents with keywords “Adam Optimizer”, “SGD algorithm”.
* **Dense Retrieval (Vector Search)** finds semantically related papers on “optimization techniques in deep learning”.
* The agent **merges both results** for a **comprehensive answer**.

1. **Active Retrieval with Tool Usage:**

**What it is:**

* If retrieval from the document store is insufficient, the agent **queries APIs, databases, or external tools**.

**Example:**

* User: “What is the latest research on quantum computing?”
* The agent first **searches internal knowledge sources**.
* If outdated, it **calls an external API like ArXiv or Google Scholar** to fetch recent papers.

1. **Re-Ranking and Filtering Mechanisms:**

**What it is:**

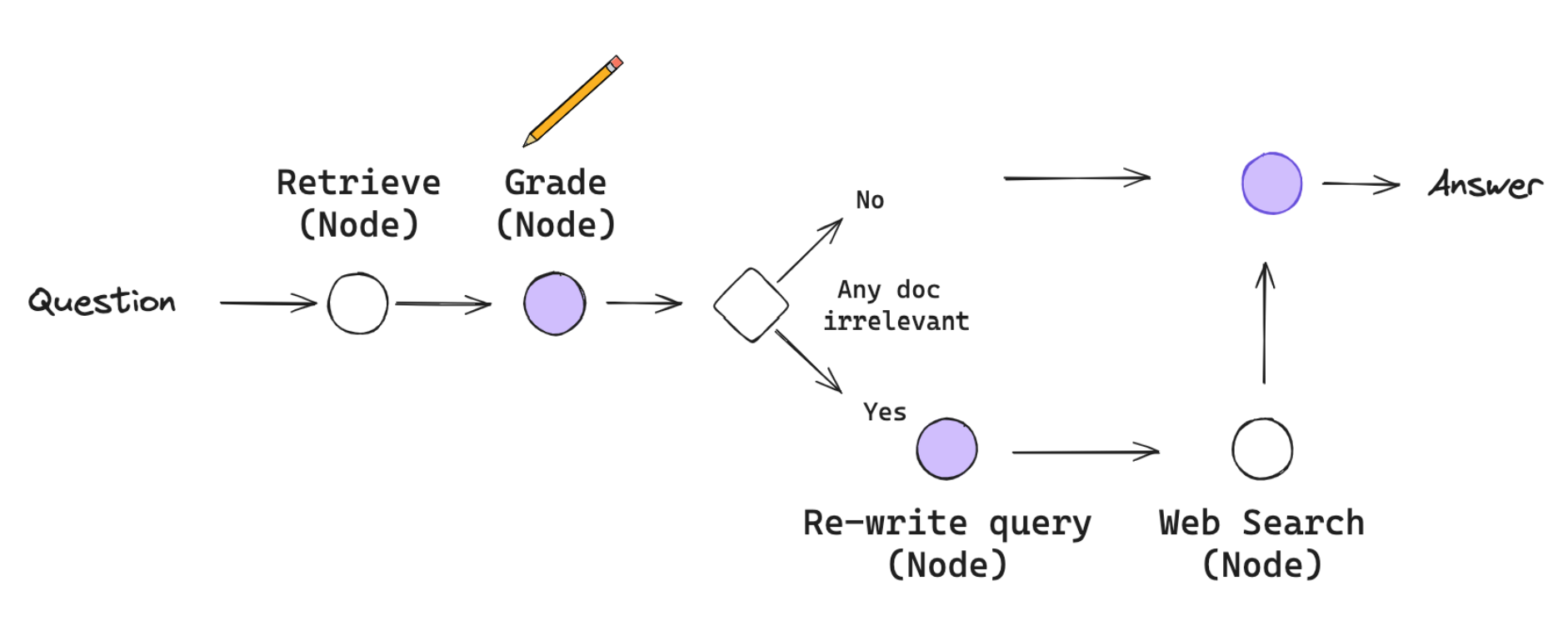
* The agent **sorts retrieved documents** based on **relevance, recency, and credibility**.
* Can **discard irrelevant results** before responding.

**Example:**

* User: “What are the effects of climate change on agriculture?”
* The agent retrieves multiple sources but **filters out outdated or low-quality ones**.
* It then **ranks the results** based on scientific credibility.
* **Final Thoughts: Why Agentic RAG’s Retrieval Strategies Matters:**
* **More Accurate:** Eliminates irrelevant or outdated info.
* **Context-Aware:** Adapts to complex, multi-step queries.
* **Reduces Hallucinations:** Uses reasoning + re-ranking for high-quality responses.
* **Scalable:** Works for research, legal AI, and enterprise use cases.
* **Corrective RAG:**

**Corrective RAG (CRAG)** is an enhancement of the traditional **Retrieval-Augmented Generation (RAG)** approach that introduces a **self-correcting mechanism**. It is designed to detect and **correct hallucinations** or inaccuracies in responses generated by LLMs.

It stands for **Corrective Retrieval-Augmented Generation** and combines **generation**, **validation**, and **retrieval correction** in a **feedback loop.**



* **Why is CRAG Needed:**

**Traditional RAG systems are helpful but not perfect:**

* They **retrieve relevant documents** and pass them to an LLM for response generation.
* However, LLMs can still **hallucinate**, **misinterpret documents**, or **miss important context**.
* CRAG addresses this by introducing **checks and balances** that **validate** and **improve** the generated answers.

**Key reasons why CRAG is needed:**

* To reduce **hallucinations and misinformation**.
* To make LLMs **more reliable for critical domains** (e.g., legal, finance, healthcare).
* To enable **self-correction and iterative improvement** of answers.
* **Working Flow of Corrective RAG:**

**Here’s a high-level step-by-step flow of how CRAG works:**

1. **Initial Retrieval:**

* A user query is processed.
* Relevant documents are retrieved from a **Vector DB** (e.g., Pinecone, Chroma).

1. **Answer Generation:**

* The LLM uses the retrieved documents to **generate an answer**.

1. **Answer Validation:**

* A **validator node** (another LLM call or a custom function) checks:
  + ✅ is the answer grounded in the retrieved documents?
  + ❌ is it hallucinated or missing information?

1. **Conditional Check:**

* If **valid**, return the answer.
* If **invalid**, trigger a **corrective retrieval**.

1. **Corrective Retrieval and Update:**

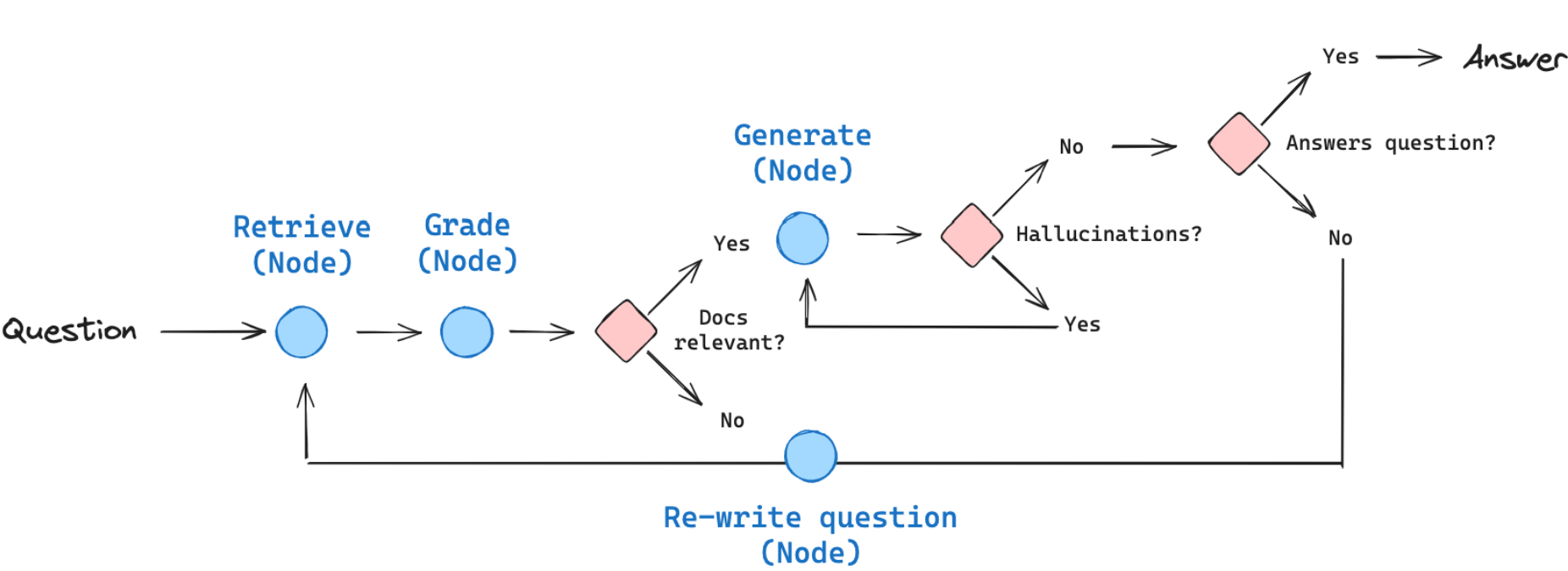
* Retrieve **additional or different documents**.
* Generate a **new, corrected answer** based on updated context.
* Optionally re-validate.

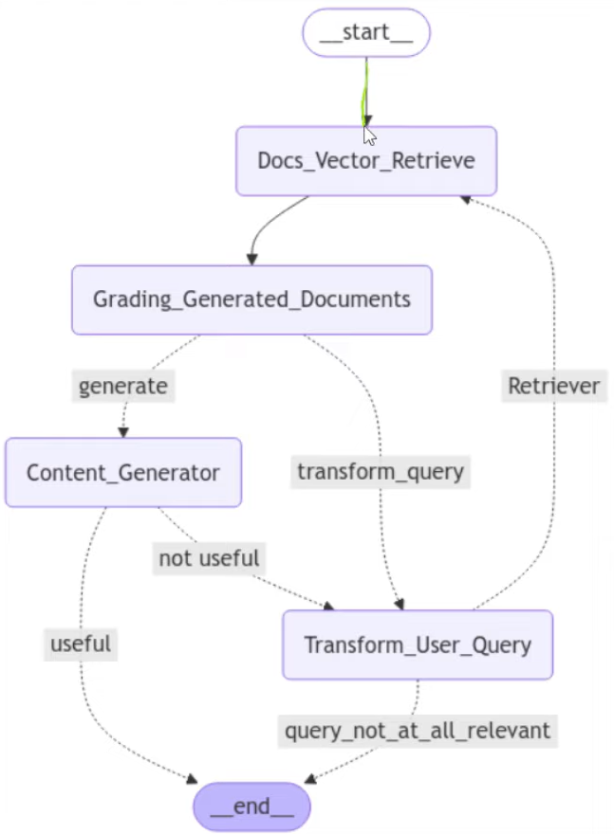
1. **Final Output:**

* The best grounded, validated answer is returned to the user.
* **Benefits of CRAG:**
* **Self-correcting loop:** Improves answer accuracy with validation and re-retrieval.
* **Modular design:** Easy to integrate with LangGraph / LangChain workflows.
* **Grounded Output:** Reduces hallucinations by requiring evidence.
* **Fine Control:** Customizable validation and correction logic.
* **Self-RAG:**

**Self-RAG** stands for **Self-Reflective Retrieval-Augmented Generation**.  
It is an **enhanced version of RAG (Retrieval-Augmented Generation)** that introduces a **reflection mechanism** to improve the quality of generated answers.

**The key innovation:** The LLM **reflects on its own response** to detect whether the **retrieved context was insufficient or irrelevant**, and then **triggers improved retrieval** before finalizing the output.





* **Working Flow of Self-RAG:**

1. **Initial Retrieval:**

* Use the query to retrieve documents from a vector database.

1. **Initial Answer Generation:**

* LLM generates a response using the retrieved documents.

1. **Self-Reflection:**

* LLM reflects on its own generated answer using a prompt like:
  + **“**Was this answer fully supported by the documents? Are there missing pieces?**”**
* The reflection generates a **critique** or **self-diagnosis**.

1. **Decision: Was Retrieval Good Enough?**

* If **yes**, finalize the answer.
* If **not**, use the **reflected critique** to:
  + Generate a **better query**.
  + Perform **improved or targeted retrieval**.

1. **Re-Generation with Better Context:**

* Generate a new answer using improved documents.

1. **Final Output:**

* Return the most supported and refined answer.
* **Benefits of Self-RAG:**
* **Self-awareness:** LLM reflects on its own output.
* **Adaptive Retrieval:** Triggers better queries and retrieval when needed.
* **Better grounded answers:** More accurate and supported responses.
* **Customizable Prompts:** Reflection logic can be tuned to domain-specific needs.

**Self-RAG** (Self-Reflective Retrieval-Augmented Generation) is an advanced RAG approach where the LLM **reflects on its initial response** to determine whether the retrieved context was sufficient. If not, it **revises the query**, performs improved retrieval, and regenerates a better answer. This **self-improvement loop** enhances answer quality and reduces hallucinations by allowing the model to **self-correct retrieval failures**.

* **Adaptive RAG:**
* **SQL Agent:**

**Learn Some Key Concepts:**

Learn about CoT vs. ToT vs. Agentic Flow:

* **Chain-of-Thought (ToT) Prompting:**
* **What it is:** A step-by-step reasoning approach where the model solves a problem linearly, one thought at a time.
* **Key Concepts:**
* **Linear Progression:** Thoughts follow a single path.
* **Human-like Reasoning:** Mimics how humans break down problems.
* **Best for:** Simple to moderately complex problems.
* **Examples:**
* **Question:** “If Alice has 3 apples and gives 1 to Bob, how many apples does she have left?”
* **CoT Response:**
  1. Alice starts with 3 apples.
  2. She gives 1 apple to Bob.
  3. She now has 3 - 1 = **2 apples.**
* **Tree-of-Thought (ToT) Prompting:**
* **What it is:** An extension of CoT where the model explores **multiple reasoning paths** (like branches of a tree), evaluates them, and selects the best one.
* **Key Concepts:**
* **Parallel Exploration:** Considers multiple solutions simultaneously.
* **Backtracking:** Can discard poor paths and revisit alternatives.
* **Best For:** Complex problems requiring creativity or strategy.
* **Examples:**
* **Question:** “Plan a 3-day trip to Paris with a budget of $1,000.”
* **ToT Response:**
  + **Path 1:** Budget-friendly (hostels, free attractions).
  + **Path 2:** Luxury-focused (hotels, fine dining).
  + **Path 3:** Balanced (mid-range hotels, mix of paid/free activities).

The model evaluates costs for each path and selects **Path 3** as the optimal balance.

* **Agentic Flow:**
  + **What it is:** A dynamic, goal-driven approach where **autonomous agents** (AI systems) collaborate, reason, and act iteratively using tools (e.g., search calculators).
  + **Key Concepts:**
* **Multi-agent Collaboration:** Agents specialize in tasks (e.g., research, math).
* **Tool Use:** Can fetch data or perform actions (e.g., search the web).
* **Best For:** Open-ended, real-world tasks requiring adaptability.
  + **Examples:**
* **Task:** “Find the latest AI research papers about climate change and summarize key trends.”
* **Agentic Flow Steps:**

1. **Research Agent:** Searches arXiv for recent papers.
2. **Analysis Agent:** Identifies common themes (e.g., "AI for carbon capture").
3. **Summary Agent:** Generates a concise report.

* **Key Takeaways:**
* **CoT:** Follows a straight line ("A → B → C").
* **ToT:** Explores a tree of options ("A → [B1, B2] → C").
* **Agentic Flow:** A team of AI agents working together ("Researcher → Analyst → Writer").

**Multi-Agentic System:**

Explore the Multi-Agentic System:

* **Definition:** A **Multi-Agent System (MAS)** is a computational framework where multiple autonomous **agents** (AI models, robots, or software programs) collaborate, compete, or negotiate to solve complex problems. Unlike single-agent systems, MAS distributes tasks among specialized agents, enabling **scalability, adaptability, and efficiency**.
* **Key Idea:**
* Agents work **independently** but communicate to achieve a common goal.
* Each agent has its own **knowledge, goals, and decision-making ability.**
* **Benefits of Multi-Agent System:**
* **Scalability:** Adding more agents improves performance without redesigning the whole system.
* **Fault Tolerance:** If one agent fails, others can compensate, ensuring robustness.
* **Parallel Processing:** Agents work simultaneously, speeding up complex tasks.
* **Flexibility:** Agents can adapt to dynamic environments (e.g., traffic routing, stock markets).
* **Specialization:** Each agent focuses on a specific task (e.g., one for research, another for analysis).
* **Core Components of Multi-Agent System:**
* **Agent:** Have a distinct **role**, **persona**, **and context**, powered by LLM.
* **Connection:** Define how agents interact and shared information.
* **Orchestration:** Determining the coordination strategy (e.g., sequential, hierarchical, bi-directional).
* **Human Oversight:** In the most cases, human intervention is required for **decision-making and evaluation**.
* **Tools:** Agent use tools for specific tasks (e.g., web search, document generation, code update).
* **LLM:** Act as a backbone of the system.
* **Environment:** The shared space where agents interact (e.g., a marketplace, a city).
* **Communication:** Agents exchange data via messages (e.g., APIs, natural language).
* **Coordination:** Rules for collaboration (e.g., auctions, voting, task delegation).
* **Decision-Making:** Agents use logic (e.g., game theory, reinforcement learning) to act.
* **Real World Application of Multi-Agent System:**

1. **Autonomous Vehicles & Traffic Management:**
   * + **Agents:** Self-driving cars, traffic lights, drones.
     + **How it Works:**
       - Cars negotiate lane changes to avoid collisions.
       - Traffic lights adjust timing based on real-time congestion.
     + **Example:** Tesla’s self-driving fleet shares road data to optimize routes.
2. **E-Commerce & Supply Chain:**
   * + **Agents:** Pricing bots, inventory managers, delivery drones.
     + **How it Works:**
       - Pricing bots compete to offer the best deals.
       - Delivery drones coordinate routes to minimize delays.
     + **Example:** Amazon’s warehouse robots optimize package sorting.
3. **Healthcare Diagnostics:**
   * + **Agents:** AI radiologists, patient chatbots, drug recommendation systems.
     + **How it Works:**
       - One agent analyzes X-rays, another checks patient history, and a third suggests treatments.
     + **Example:** IBM Watson Health uses MAS for cancer diagnosis.
4. **Financial Trading (Algorithmic Trading):**
   * + **Agents:** Trading bots, risk analyzers, fraud detectors.
     + **How it Works:**
       - Bots compete in stock markets, adjusting strategies in real-time.
       - Fraud detection agents flag suspicious transactions.
     + **Example:** Hedge funds use MAS for high-frequency trading.
5. **Smart Cities & Energy Grids:** 
   * + **Agents:** Smart meters, renewable energy controllers, emergency responders.
     + **How it Works:**
       - Energy agents balance solar/wind power distribution.
       - Emergency agents reroute ambulances based on traffic.
     + **Example:** Barcelona’s smart city IoT network.

* **Single Agent System vs. Multi-Agent System:**

|  |  |  |
| --- | --- | --- |
| **Feature** | **Single-Agent System** | **Multi-Agent System** |
| **Task Handling** | One agent does everything. | Multiple specialized agents. |
| **Scalability** | Limited. | Highly scalable. |
| **Fault Tolerance** | Fails if the agent crashes. | Resilient (other agents compensate). |
| **Use Case** | Simple chatbots. | Self-driving fleets, stock markets |

* **Communication between Multi-Agent System:**

Multi-Agent Systems (MAS) rely on different communication strategies to coordinate agents effectively. Below is an in-depth breakdown of four key communication mechanisms:

1. **Shared State Communication:**

* **How it Works:** 
  + Agents interact by reading and writing to a **centralized shared state** (e.g., a database, blackboard, or memory).
  + No direct messaging; agents observe and modify the shared state to collaborate.
* **Example:**
* **Stock Trading Bots:**
  + Agents (traders) read a shared order book (state) and place buy/sell orders.
  + No direct communication; decisions are based on the latest market data.
* **Pros:**
  + Simple to implement (no complex messaging).
  + Good for real-time, data-driven environments.
* **Cons:**
  + Can lead to race conditions (multiple agents modifying the same data).
  + No explicit coordination—agents may act redundantly.
* **Use Cases:**
  + Real-time bidding systems (ad auctions).
  + IoT sensor networks (shared environmental data).

1. **Tool-Based Communication:**

* **How it Works:**
  + Agents **explicitly call tools or APIs** to exchange information.
  + Similar to function calling in programming—agents request actions from other agents or services.
* **Example:**
* **AI Research Assistant:**
  + A "Search Agent" calls a web search tool to fetch papers.
  + A "Summarizer Agent" calls a text-processing tool to condense findings.
* **Pros:**
  + Structured and deterministic (clear input-output).
  + Easy to debug (each tool call is logged).
* **Cons:**
  + Requires predefined tool interfaces.
  + Less flexible for open-ended collaboration.
* **Use Cases:**
  + LangChain/LangGraph agents (ReAct-style workflows).
  + Automated customer support (calling APIs for order status).

1. **Mixed Communication State (Hybrid Approach):**

* **How it Works:**
  + Combines **shared state + direct messaging.**
  + Agents use a shared memory for common data but also send explicit messages when needed.
* **Example:**
  + **Autonomous Delivery Fleet:**
    - **Shared state:** Real-time traffic map (all drones see congestion).
    - **Direct messaging:** A drone asks another for battery status before swapping tasks.
* **Pros:**
  + Balances flexibility and efficiency.
  + Reduces redundant messaging (shared state handles common knowledge).
* **Cons:**
  + More complex to implement.
* **Use Cases:**
  + Self-driving car fleets (shared road data + V2V messaging).
  + Smart factories (shared production logs + machine-to-machine alerts).

1. **Message List Communication (Pub/Sub or Event-Driven):**

* **How it Works:** 
  + Agents **send and receive messages** via a **message queue** (e.g., Kafka, RabbitMQ).
  + Supports **publish-subscribe (Pub/Sub)** or **direct peer-to-peer** messaging.
* **Example:**
  + **Fraud Detection System:**
    - Transaction Agent → Publishes "Suspicious Payment" event.
    - Fraud Analyst Agent → Subscribes and investigates.
* **Pros:**
  + Highly scalable (decouples sender/receiver).
  + Supports asynchronous workflows (agents don’t wait for replies).
* **Cons:**
  + Requires robust message brokers.
  + Can lead to message overload if not managed.
* **Use Cases:**
  + Financial fraud detection.
  + Distributed microservices (e.g., Uber’s ride-matching system).
* **Challenges Solves by Multi-Agent System over Single-Agent System:**
* **Scalability & Parallelism:** Tasks can be distributed across agents, increasing system efficiency and scalability.
* **Specialization:** Each agent can focus on a specialized function (e.g., reasoning, planning, and memory).
* **Collaboration:** Agents can interact, share knowledge, and solve complex problems more effectively than a single agent.
* **Resilience:** Failure in one agent doesn't crash the entire system; others can adapt or take over.
* **Dynamic Adaption:** Multiple agents can better adapt to evolving tasks or environments by reassigning roles.
* **Different Types of Multi-Agentic Architectures:**

1. **Network Multi-Agentic System:** A decentralized structure where agents interact peer-to-peer without a central coordinator. This promotes high fault tolerance and flexibility. Agents communicate dynamically through message passing or shared memory, ideal for distributed problem solving and autonomous decision-making.
2. **Supervisor Multi-Agentic System:** Involves a central supervising agent managing and coordinating the behaviors of other agents. The supervisor assigns tasks, monitors progress, and resolves conflicts. This provides structured control and oversight but may become a bottleneck if overloaded.
3. **Hierarchical Multi-Agentic System:** Agents are organized in a tree-like structure with layered control. Higher-level agents oversee and coordinate lower-level agents, creating clear task delegation and modular design. This is effective for large-scale systems with well-defined subtasks.
4. **Custom Multi-Agentic System:** Tailored architecture combining elements of the above to meet specific use-case needs. For example, it might use a supervisor for task management and a network for knowledge sharing. Flexibility and adaptability make it suitable for evolving systems or hybrid workflows.

**Explore Different Types of Multi-Agentic System:**

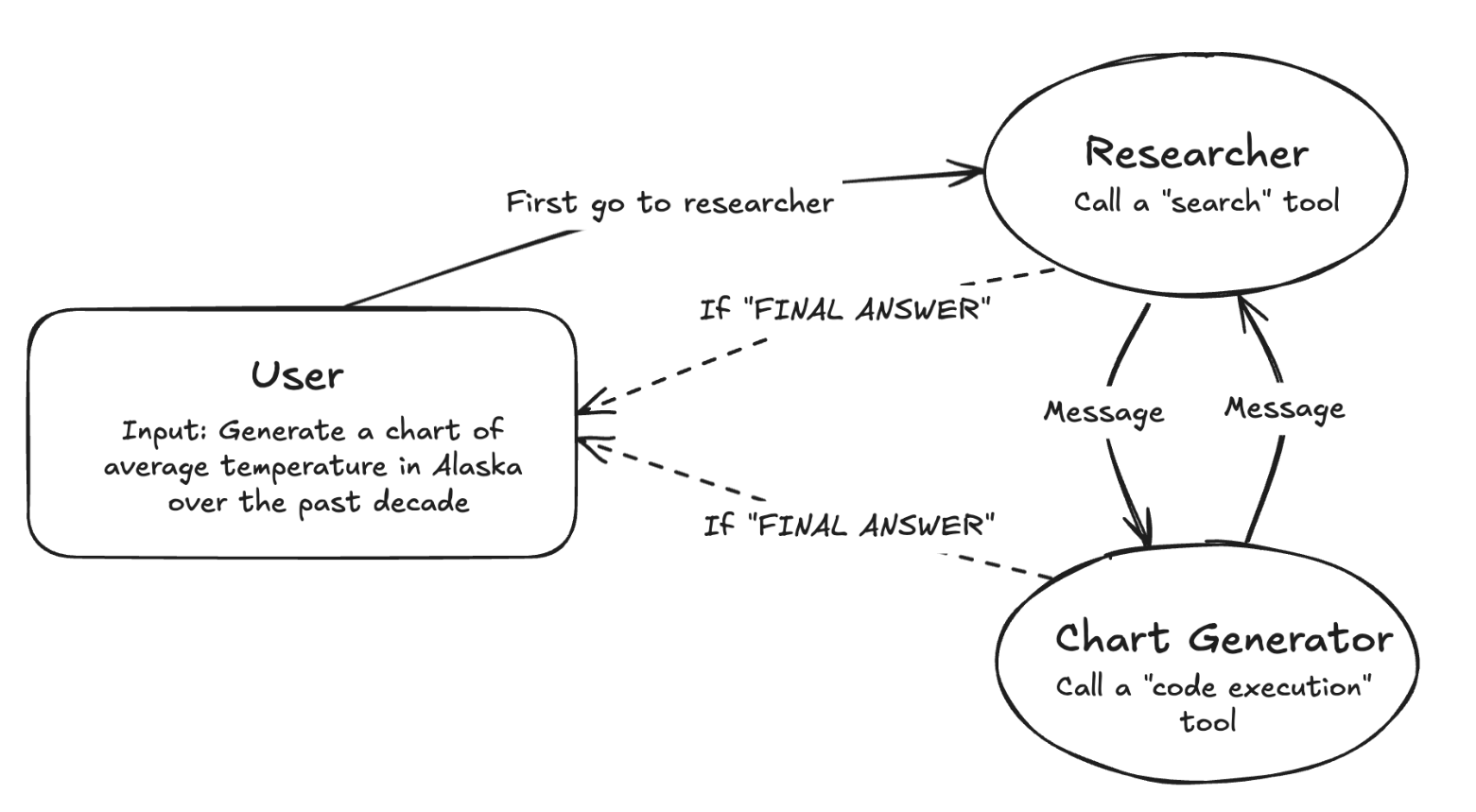
Let’s explore different types of Multi-Agentic Framework (by LangGraph):

* **Multi-Network Agent System:**

A **Multi-Network Agent System** is a **decentralized** architecture where **multiple agents interact directly with each other** (peer-to-peer) instead of depending on a single controller.

* Each agent has its **own responsibility** (like reasoning, memory, planning, coding, etc.).
* They **communicate, collaborate, and negotiate** through a **shared network**.
* **There’s no master agent** — the system is **dynamic**, **parallel**, and often **fault-tolerant.**
* Agents decide **when, how, and with who** to interact based on the task’s current needs.

This model fits **complex workflows** where the task needs a **combination of skills** and **real-time interaction** between different expert agents.



* **Flow:**
* User asks: "Give me a report on COVID-19 statistics analysis."
* **Researcher Agent** first searches for recent COVID-19 datasets.
* Then, **Researcher** sends the data to the **Coder Agent**.
* **Coder Agent** writes Python code to analyze the data.
* Once done, **Coder** sends results to the **Writer Agent**.
* **Writer Agent** composes the final report.

➡**Agents talk to each other** directly, without waiting for a boss agent to tell them.

➡**Each agent knows** when to take over based on the task context.

* **Collaborative & Network Agent:**
  + **Collaborative Agent:**
* A **Collaborative Agent** is an agent that **works together with other agents** to solve problems or achieve goals.
* It **shares information**, **helps other agents**, and **coordinates** its actions so that the **team as a whole** succeeds.
* The agent is **goal-driven** but **team-aware** — meaning, it doesn't only care about itself; it also adjusts its behaviour to support the collective success.

* + **Network Agent:**
* A **Network Agent** refers to an agent that is **connected** to other agents through a **communication network** (like a graph or peer-to-peer links).
* It **exchanges messages**, **requests help**, **offers services**, or **shares data** with other agents **through the network.**
* There is **no central boss**; all agents **independently** act and interact within the network.
* **Relationship Between Collaborative and Network Agent:**
* In a **multi-network agent system**, agents are often **collaborative** — they **communicate and cooperate** through the network.
* **Collaboration** is about **intent** (working together), while **networking** is about **communication structure** (how they talk).
* **Tiny Example:**

|  |  |
| --- | --- |
| **Collaborative Agent** | **Network Agent** |
| A shopping bot asks a payment bot for help in completing a purchase. | The shopping bot sends a message across the agent network to find available services. |

* **Detailed Workflow of Multi-Network Agent System:**

Here is the detailed workflow of Multi-Network Agent System Workflow:

1. **Agent Initialization:**

* Each agent is **created with a specific role** or expertise (e.g., researcher, planner, memory, and coder).
* Agents are given:
  + **Capabilities** (what tasks they can perform).
  + **Communication protocols** (how they send/receive information).
  + **Local goals or behaviors** (what they try to achieve individually).

1. **Task Reception:**

* A **user** or **external system** gives an initial **task, question, or problem** to one or more agents.
* Example: “Analyze recent sales data and generate insights.”
* The **first agent** (like a **task handler agent**) may **decide** whether it can:
  + Do it alone ✅
  + Or needs help ❌ (then it contacts other agents through the network)

1. **Dynamic Agent Collaboration:**

* Agents **communicate across the network:**
  + **Requesting help** (e.g., "I need a coder agent to process this data.")
  + **Sharing intermediate outputs** (e.g., "Here’s the cleaned data, pass it to the data analyst.")
  + **Negotiating responsibilities** (e.g., "Who’s available to summarize this report?")

**No central boss!** Agents **self-organize** and **choose** the right path forward.

1. **Knowledge/Context Sharing:**

* Agents **share context** or **important knowledge** with others, to make sure everyone understands the overall task better.
* They can:
  + Send outputs.
  + Send partial results.
  + Send metadata (e.g., "This is urgent", "Customer is waiting").

This prevents duplication and makes collaboration smoother.

1. **Execution of Subtasks:**

* Each agent works on **its own small piece** of the whole job, based on its skills:
  + Research agent finds data.
  + Coder agent writes Python scripts.
  + Analyzer agent interprets the results.
  + Writer agent writes a final summary.

**Parallel execution** speeds things up massively.

1. **Intermediate Feedback Loops:**

* After a task/subtask is done:
  + Agents **inform** other agents.
  + Other agents may **ask questions**, **suggest changes**, or **trigger new tasks.**

**Example:** After analyzing data, if the analyzer finds missing values, it can **ask** the researcher to **fetch better data** → a feedback loop.

1. **Final Aggregation & Response:**

* Once all pieces are completed:
  + **Agents assemble** the results together.
  + Either one agent or a group of agents **package** the final output for the user.

**Example:** Writer agent compiles the insights, graphs, and analysis into a clean final report for the customer.

1. **Termination or Reset:**

* Once the task is delivered:
  + Agents **reset** their state.
  + They **wait** for the next task or trigger.

Some agents may also **log results**, **update shared knowledge**, or **learn** for future tasks.

* **Key Features Throughout the WorkFlow:**

|  |  |
| --- | --- |
| **Features** | **Description** |
| Peer-to-peer Communication | Agents talk freely over the network without centralized control. |
| **Parallel Execution** | Multiple agents work at the same time on different subtasks. |
| Dynamic Task Distribution | Tasks shift between agents based on needs and availability. |
| Context Awareness | Agents keep track of task history and context when collaborating. |
| Feedback Loops | Agents refine work through mutual feedback before finalizing. |

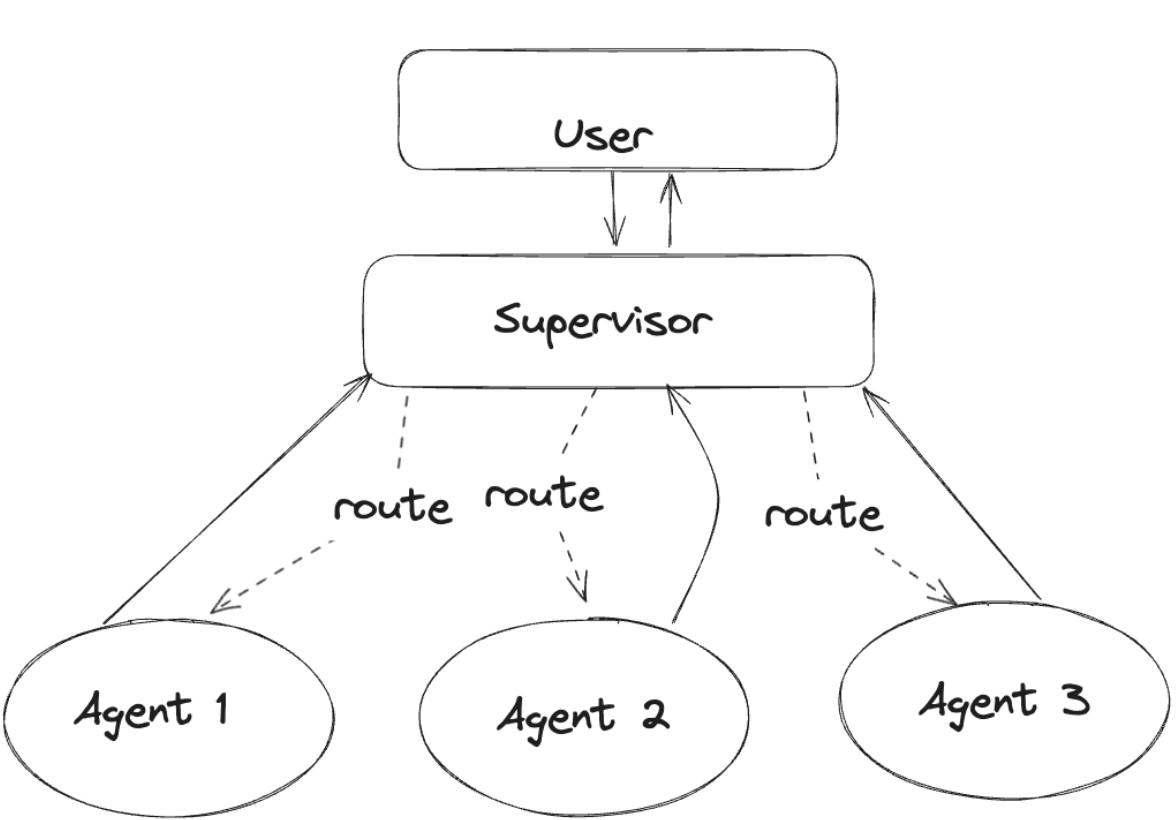
* **Multi-Agent Supervisor:**

A **Multi-Agent Supervisor** is a **special agent** (often called a “controller” or “coordinator”) that **oversees and manages the interaction** between multiple agents.

It doesn’t do the core task itself — instead, it:

* Decides which agent to activate
* Monitors agent outputs
* Loops back or redirects tasks
* Manages overall task progress

Think of it as a **project manager** that doesn’t write code or design slides, but assigns work to specialists and reviews the outcomes.



* **Flow:**

1. A **user request** is sent to the **supervisor agent**.
2. The **supervisor analyzes** the intent or task.
3. It chooses the appropriate **agent** (e.g., RAG agent, code interpreter, SQL agent).
4. Based on results or errors, it may:
   1. Call **another agent**
   2. Ask the **same agent to retry**
   3. Or **end** the task.

* **WorkFlow of Multi-Agent Supervisor:**

Here is a workflow of Multi-Agent Supervisor:

1. **User Sends a Task or Query:**

The user gives a natural language instruction or question like: **“**Summarize this article and also write a Python script to visualize its word frequency.**”**

1. **Supervisor Agent Receives the Input:**

The **Supervisor Agent** acts as the **central brain**.

**It parses the input to:**

* Understand intent
* Split the task (if needed)
* Decide which agent(s) to activate

**Example decision:**

* **Summarization** → Call summarizer\_agent
* **Python code** → Call code\_writer\_agent

1. **Supervisor Chooses an Agent:**

The supervisor **dynamically selects** the right agent based on task type.

**For Example:**

|  |  |
| --- | --- |
| **Task** | **Agent** |
| Retrieve external information | Rag\_agent |
| Write/Run Python code | Code\_agent |
| Generate SQL query | Sql\_agent |
| Summarize Content | Summarizer\_agent |

The decision is made using **rules**, **logic**, or even **language model reasoning.**

1. **Agent Executes Its Subtasks:**

The selected agent:

* Receives a portion of the task.
* Performs its role (e.g., writes code, summarizes, queries DB).
* Returns output to the **supervisor.**

1. **Supervisor Evaluate the Output:**

Once an agent returns a result:

* The **supervisor checks** if the result is valid or complete.
* It may:
  + **Accept** it and move on
  + **Refine** it using another agent
  + **Re-loop** the same agent with modified input
  + Or **request clarification** from the user

1. **Supervisor Orchestrates Multi-Steps Flows:**

If the task needs **multiple agents**, the supervisor:

* Chains them dynamically
* Maintains context and memory
* Passes outputs between agents

**Example Flow:**

User → Supervisor → RAG Agent → Code Agent → Supervisor → User

1. **Final Result Returned to User:**

After coordinating the agents:

* The **supervisor compiles** or formats the result.
* Sends it back to the user in a clean, understandable format

Task complete!