Agentic Workflow

🧠 Agentic Workflow — Detailed Explanation

An **Agentic Workflow** is a system design where a Large Language Model (LLM) acts as an **autonomous agent** that can **plan**, **reason**, **take actions**, **use tools**, and **iterate** until a goal is achieved.

Instead of only generating one-shot outputs, the LLM becomes an **active problem solver** with the ability to:

- break tasks into sub-tasks,
- call APIs/tools,
- check intermediate results,
- revise its own outputs,
- decide the next action autonomously,
- and stop only when the task is complete.

🔑 Key Characteristics of Agentic Workflows

1. **Planning**

   The LLM generates a plan or chain of actions (e.g., "search → filter → calculate → summarize").

2. **Tool use**

   The LLM calls external tools such as:

   - search engines,
   - databases,
   - calculators,
   - code execution environments,

- APIs.

3. **Reflection & Revision**

   The LLM evaluates its own intermediate outputs and decides whether they're correct or need revision.

4. **Memory / State Tracking**

   The system maintains an internal state to track steps, results, and context.

5. **Autonomous Looping**

   The LLM can run multiple iterations until the goal is met.

🔧 Non-Agentic Workflow (Traditional LLM Use)

A non-agentic workflow is the traditional "prompt → response" setup.

The LLM:

- receives a single input,
- generates a single output,
- **does NOT plan**,
- **does NOT use tools**,
- **does NOT check its own work**,
- **does NOT take actions or iterate**.

It's **passive** and produces outputs in one forward pass with no corrective mechanism.

⚖️ Agentic vs Non-Agentic Workflow — Comparison Table

Feature	**Agentic Workflow**	**Non-Agentic Workflow**
Role of LLM	Autonomous agent	Passive text generator
Actions	Can call tools, APIs, run code	No actions, only text
Planning	Yes — multi-step	No — single-step

Feature	**Agentic Workflow**	**Non-Agentic Workflow**
Reasoning	Iterative, reflective	One-shot reasoning
Error Correction	Self-correction loops	No revision
State / Memory	Maintains and updates state	Stateless for each prompt
Output	Plan + actions + final answer	Single answer
Use Cases	Automation, agents, workflows	Chat, Q/A, writing

🧩 Why Agentic Workflows are Powerful

Agentic LLMs:

- solve **complex multi-step tasks**,
- perform **dynamic decision-making**,
- reduce hallucinations by grounding information through tools,
- automate processes that normally require humans (research, coding, analysis),
- integrate with real-world systems.

Examples:

- AI assistants that browse the web,
- LLMs that analyze data with a Python tool,
- LLMs orchestrating business workflows,
- full autonomous research agents.

📘 Example of Agentic Workflow

1. User asks: *"Find the cheapest flight Bangalore → Delhi next week and summarize."*
2. Agent Plan:
    - search flight API →
    - filter lowest price →

- summarize result.

3. LLM calls flight search tool.
4. LLM processes the data.
5. LLM generates final answer.

This requires planning + tool use + iteration.

A non-agentic model cannot do this end-to-end.

🧾 5-line Exam-Ready Answer

1. An **Agentic Workflow** enables an LLM to autonomously plan, take actions, use tools, and iteratively refine outputs.
2. It treats the LLM as an **agent** that executes multi-step tasks using reasoning loops.
3. A **non-agentic workflow** is a simple "prompt-in, text-out" model with no tool use or self-correction.
4. Agentic systems maintain state, call external tools, and perform reflection to reduce errors.
5. Thus, Agentic workflows are essential for automation, decision-making, and complex real-world tasks.

🧠 Architecture of an AI Agent (LLM-based Agent)

An AI Agent is a system where an LLM becomes **action-taking**, not just **text-generating**.

To do this, the agent is built using several coordinated components:

🔷 1. Core Components of an Agent Architecture

Below is the standard architecture followed in modern AI agents (e.g., AutoGPT-style agents, Llama Agents, GPT-based tool-using agents):

1) The Policy / LLM (Brain of the Agent)

The LLM decides what to do next.

- Takes the current **observation + memory + goals** as input
- Decides an **action**:
  - Generate text
  - Call a tool
  - Query a database
  - Ask for more info
  - Plan next steps

Often represented as:

$$\text{action} = \pi_\theta(\text{state})$$

The LLM is the **decision-making engine**.

---

2) The Planner (Task Decomposer)

Many agent systems have a separate planning module (or planning prompts).

Role:

- Breaks big tasks into smaller steps
- Creates a plan, chain-of-thought, or workflow
- Helps reduce hallucinations and improve control

Some architectures merge planner + LLM. Some separate them.

---

3) Tools / External Functions (Agent's Abilities)

These are the **APIs, tools, or functions** the agent can call.

Examples:

- Web search
- Database lookup

- Code execution
- Calculator
- Image generator
- File system operations
- Custom enterprise APIs

The agent **selects and calls tools** when LLM output triggers them.

---

4) Tool Executor / Tool Router

This module receives the tool call from the LLM and:

- Validates the tool request
- Executes it using actual APIs
- Returns the structured result back to the agent

This ensures safety and correctness.

---

5) Memory System (Short-term + Long-term)

Agents operate over many steps → require memory.

Types:

1. **Short-term memory**
    - Stores context within the current task
    - E.g., conversation history, intermediate results
2. **Long-term memory**
    - Stores knowledge across tasks
    - Examples: vector DB, knowledge base, experience memory, episodic memory

Memory allows the agent to be **stateful** and context-aware.

---

6) Environment (Where the Agent Acts)

The environment provides:

- Tasks
- Feedback
- Observations
- Rewards (in RL settings)

Examples:

- A web browser
- Operating system
- API endpoints
- Simulation environment (RL)

This is where the agent performs "actions" and receives "state updates".

---

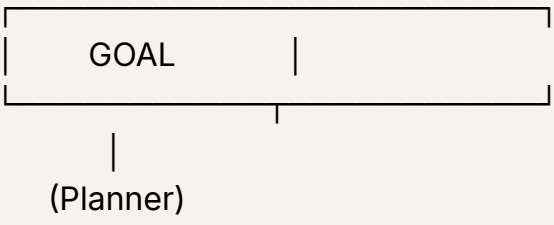
7) Controller / Orchestrator

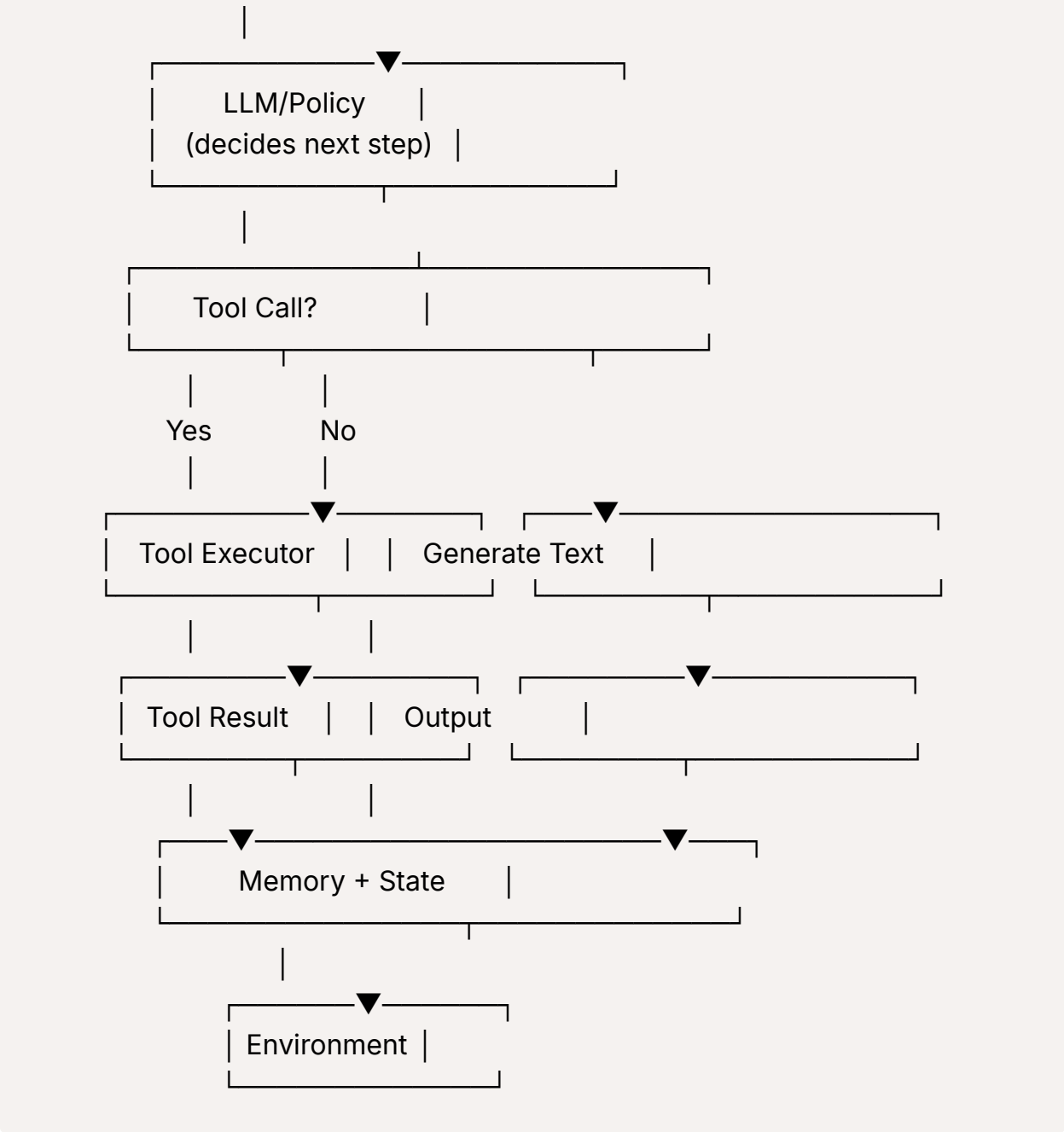
This module coordinates the entire agent loop:

1. Send prompt to LLM
2. Detect tool call
3. Execute tool
4. Return result to LLM
5. Repeat until task is complete

It prevents loops, enforces safety, and ensures smooth agent execution.

🔷 2. Agent Architecture Diagram (Textual)





🔷 3. Summary of the Agent Architecture

Component	Purpose
LLM / Policy	Brain → decides next action
Planner	Breaks tasks into steps
Tools	External abilities the agent can use

Component	Purpose
Tool Executor	Executes tools + returns results
Memory	Stores context and long-term knowledge
Environment	Where actions happen + feedback is received
Controller	Orchestrates the full agent loop

🔷 4. Key Difference From a Normal LLM

Non-Agentic LLM	Agent
Only generates text	Takes actions
No memory or state	Stateful (accumulates information)
No tools	Uses external tools/functions
No planning	Has planning + multi-step workflow
Passive	Autonomous / active

The **architecture** is what transforms a passive LLM into a **decision-making, tool-using, multi-step problem-solving agent**.
