Published by :
https://www.ijert.org/
An International Peer-Reviewed Journal

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 02 , February - 2026

Agentic AI: Concepts, Architectures, Frameworks, and Challenges

From Chatbots to Agents: A Guide to Architectures, Frameworks, and Design Patterns

Ashman Chetan Raut
Mumbai University: Department of Computer Engineering
Fr. Conceicao Rodrigues College of Engineering
Mumbai, India

Dr. Sujata Deshmukh
Mumbai University: Department of Computer Engineering
Fr. Conceicao Rodrigues College of Engineering
Mumbai, India

***Abstract*—Agentic AI refers to autonomous artificial intelligence systems which have Large Language Models (LLMs) as their base, and perform assigned tasks with goal-directed autonomy. These systems reason, plan, create their own workflows and use tools to accomplish the task, all with minimal human intervention. Unlike traditional LLMs, these systems offer adaptability to the changing requirements as they actively perceive the environment, take decisions, and execute with the help of feedback loops. This literature survey aims to provide a comprehensive beginner's guide to Agentic AI systems, exploring both theoretical foundations and practical implementations across single and multi agent architectures. The scope excludes pre-transformer reinforcement learning, and focuses on LLM-based systems. This survey is structured around the following three contributions: Theoretical Framework: The established foundations of agentic AI, such as the agent loop (Perception, Planning, Action, Observation/Learning), memory systems (short-term memory v/s long-term memory), and the core reasoning patterns (Chain-of-Thought, ReAct, Tree-of-Thought). Orchestration Frameworks: We analyze five frameworks (LangChain, LlamaIndex, CrewAI, AutoGen, MetaGPT) across seven characteristics: execution model, state management, loop support, use cases, learning curve, and maturity). Critical Challenges: We aim to address the hallucination, infinite looping, and token budget optimization.**

Keywords*—Agent Loop, ReAct Pattern, Orchestration Frameworks, Multi-Agent Systems, Tool Integration, Hallucination Mitigation, Planning Patterns, Memory Architecture, Agentic RAG, Self-Critique Mechanisms.

I. INTRODUCTION

A. Motivation

The paradigm shift from prompt engineering to flow engineering is representative of a fundamental evolution in AI agent and development. Prompt engineering involves creating precise text inputs to an LLM to get the best output, treating the LLM as a static question answering entity. Flow engineering on the other hand recognizes that LLM based agents and Agentic AI systems are capable of executing multi-step workflows, not just isolated prompts. This transition is necessary because static isolated prompts fail to adapt to real time environment changes. We cannot iteratively refine their outputs. Complex tasks require multi-step reasoning and dividing into smaller tasks, assigning them to specialized agents and coordinating the final output. Eg: Asking ChatGPT or Claude to write a python code to analyze stock market data v/s designing an agent that can continuously monitor financial APIs, and make changes according to the market conditions.

B. Scope

This survey covers:

- LLM based autonomous agents
- Single agent and multi agentic systems
- Tool integration and memory mechanisms
- Challenges and mitigation strategies

C. Audience

Any beginner in the field of AI Agents and Agentic AI systems who aims to:

- Learn about frameworks like AutoGen or LangChain.
- Understand architectural patterns, evaluation methodologies and cost implications.
- Assess use cases where these systems are not appropriate or needed.

II. CORE CONCEPTS AND TAXONOMY

A. The Agent Loop

Fig. 1, the agent loop is the operational cycle on which agentic systems are based on. Unlike traditional ML pipelines, agents engage in continuous feedback cycles.

Key differences from traditional ML systems:

- Iterative: Agentic systems loop until the goal is achieved or when limit is reached.
- Reactive: They respond to the changes in environment, and take feedback.
- Reflexive: They self-critique and manage errors.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 02 , February - 2026

1. The agent execution loop.

B. *Planning Patterns*

- Chain of Thought (CoT): We prompt the LLM to output its reasoning steps leading to the final answer. This is simple as no tool calls are required, but reasoning is limited as there is no feedback loop or error correction. Eg: “Think step by step and answer 12+39”. “First, add units: 2+9=11, write 1 and carry 1. Add tens: 1+1+3=5. Answer: 51.”
- ReAct (Reasoning+Acting): Agent performs both reasoning steps and tool calls in a single trajectory. Advantages include catching errors mid trajectory and correcting them. This however, means that more tokens are consumed, and if the error is not caught, it cascades.
- Tree of Thought (ToT): Agent builds a tree of possible reasoning paths, and backtracks upon failure of one path. This is used for complex reasoning and code generation. The drawback is that this is very computationally expensive.

In practice, most production systems use ReAct as the middle ground between having enough reasoning to be effective, and simple enough to control costs. Table 1 compares the different planning patterns.

I. PLANNING PATTERNS COMPARED

Pattern	Complexity	Mechanism	Best for	Limitation
Chain of thought	Baseline	Step-by-step reasoning in text	Simple multi-step tasks	No tool interaction, assumes static envirnoment
ReAct	Intermediate	Interleaved Reasoning+Acting	Dynamic environments with tool use	Limited lookahead, greedy action selection
Tree-of-Thought	Advanced	Explores multiple reasoning branches, and backtracks	Complex planning with high error cost	Expensive as it performs many LLM calls, rarely used in production

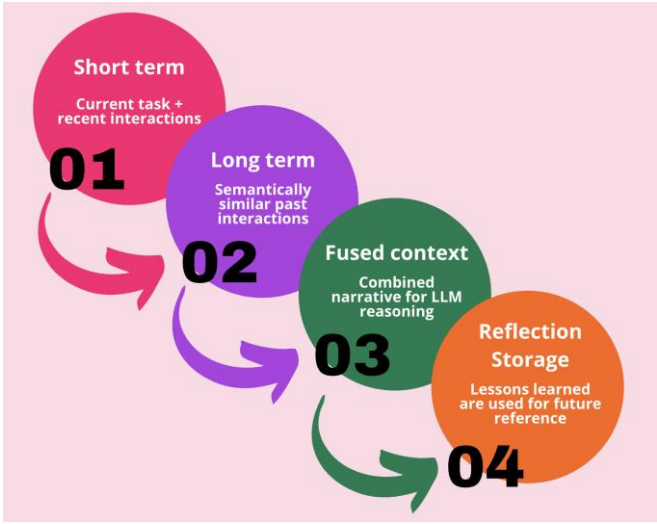
C. *Memory Systems*

AI agents have two memory systems, short-term memory and long-term memory. The short term memory or the context window includes the current conversation’s history and things like the user’s original query, previous reasoning steps and tool outputs, and the system instructions provided. This is all discarded after the task is completed. The larger the context window, the slower the response. Modern LLMs have about 128K-200K tokens. The long term memory, including vector stores and knowledge bases, maintain things like previous interaction history, user preferences, retrieved documents through RAG, and learned patterns. This enables long term learning across multiple sessions, but risks hallucinations and increases latency. Table 2 compares the two memory systems.

II. MEMORY SYSTEMS COMPARED

Parameter	Short-term memory (Context Window)	Long-term memory (Vector database, RAG)
Capacity	4K-200K tokens, depending on the model	Unlimited (dependent on external storage)
Persistence	Single session only	Not session dependent
Role	Immediate reasoning, and keeping conversation history	Accumulates knowledge and preferences from past interactions
Challenge	Limited capacity, as focus is on priority	Retrieval must be precise

AI agents can also have a hybrid memory configuration, as shown in Fig. 2.



2. Hybrid memory configuration.

D. *Tool Use and Function Calling*

AI Agents invoke external functions like API calls, to extend the systems capabilities beyond language prediction like traditional chatbots. Fig. 3 shows the step-by-step workflow of an AI agent using function calling. How it works:

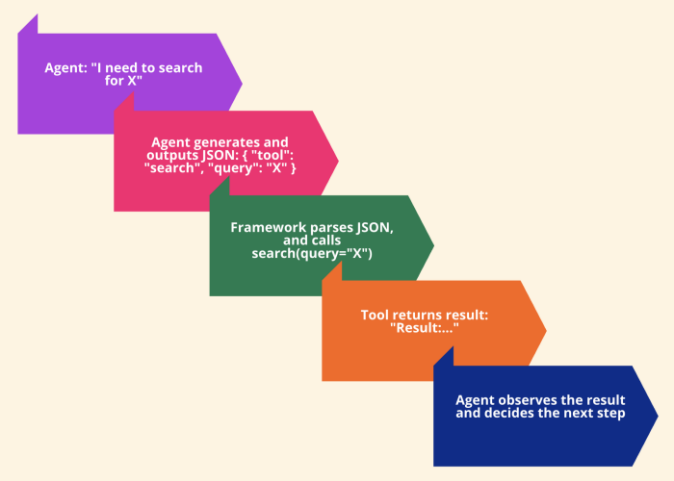
- Agent receives a goal (e.g., "Find average stock price for last 30 days")

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 02 , February - 2026

- Then the LLM generates a structured function call: {"function": "fetch_stock_data", "ticker": "AAPL", "days": 30}
- System invokes the function and captures the result returned by the function.
- This result is then fed back to the LLM.
- Finally, the LLM generates final step (e.g., "Average stock price for the last 30 days is: …")

Common Tools used by AI agents and agentic systems:

- Code Execution: Python interpreter, SQL engines
- Information Retrieval: Web search, vector database queries, APIs
- Perception: Computer vision APIs, speech-to-text
- Grounding: Knowledge graphs, ontologies, fact-checkers



3. Tool Use and Function Calling.

Key design partners include Schema-based prompting, where the agent is shown JSON schemas of available tools upfront, Dynamic tool selection, where Agent chooses which tools match its goal, and Argument validation, where the Framework checks tool arguments before execution and thus helps in preventing errors.

E. Reflection and Self-Critique Pattern

After executing every action, the agent evaluates if the task is completed and to what success metric. e.g., "Use tool X to solve problem Y".

- Execute: [Tool X produces result R for problem Y]
- Reflect: "Does R solve Y?"
- If Yes: Store success + proceed
- If No: Identify the failure causes, generate alternative approaches, retry with different tools/parameters, and update memory: "Tool X failed for scenario Y; try Z instead".

III. ORCHESTRATION FRAMEWORKS

A. Framework Comparison Table

Table 3 compares the different frameworks used to build AI Agents and Agentic AI systems powered by LLMs across a few key dimensions, like architecture, workflow design, and use cases.

III. FRAMEWORKS COMPARED

Dimension	LangChain	LlamaIndex	CrewAI	AutoGen	MetaGPT
Execution Model	Sequential chains, ReAct loops	ReAct agent + tool selection	Role-based team orchestration	Conversation-driven multi-agent	SOP-based workflows
State Management	Explicit memory objects (chain state)	ReAct agent state	Crew-level state + agent memory	Message pool + context vars	Shared message pool + role specs
Loop Support	Basic (chain → tool → chain)	ReAct loop (think-act-observe)	Fixed team conversation	Auto-reply mechanism + dynamic groups	Pub-subscribe + sequential roles
Typical Use Case	Document Q&A, code generation	RAG applications, data extraction	Specialized multi-step tasks	General-purpose multi-agent	Software engineering, complex workflows
Learning Curve	Moderate (many primitives)	Steep (requires deep tool integration)	Low (intuitive team metaphor)	Moderate (conversation patterns)	Steep (SOP design required)
Maturity	Production-ready	Early-stage	Emerging	Production-ready	Production-ready
Cost Efficiency	High (minimal LLM calls if designed well)	Moderate (tool calls increase costs)	Moderate (multi-agent overhead)	Low (extensive message passing)	Moderate (structured outputs reduce hallucinations)

B. Framework Analysis

1) LangChain and LangGraph:

a) LangChain (2022–2024): This was originally designed for chaining prompts and tool uses linearly. Strengths include simplicity in prototyping and RAG, and a large ecosystem of integrations with good documentation to refer to. Limitations include the fact that loops require a workaround (like: recursion + retries), and that State management is implicit (needs to be externally done) as it was not designed for stateful agents.

b) LangGraph (2024+): This Evolved from (was built on top of) LangChain and added support for graph-based state machines. Its strengths include native support for agentic loops, explicit state management (leading to clean debugging), and handling of flexible edge conditions (if-then logic). Its limitations are that as it is relatively new, it has a smaller community, and implementation requires a thinking approach in graph terms.

2) CrewAI: Agents have clearly defined roles and tasks, and the framework orchestrates their collaboration. Strengths include simplicity in defining multi-agents teams, and built in support for delegation (agents delegating tasks

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 02 , February - 2026

to other agents for help). Limitations include lack of flexibility compared to LangGraph for complex routing, basic memory (no advanced retrieval), and a limited tool ecosystem compared to LangChain.

3) MetaGPT: Each agent is given a role to play and SOPs (Standard Operating Procedures) to follow. Its strengths include structured workflows that are reproducible and debuggable, and agents independence with clear responsibilities, making it a strong choice for software engineering tasks. Its limitations are that the learning curve is steep due to complex API, fixed SOP definition making agents less adaptive mid task, and a smaller community compared to CrewAI.

4) AutoGen (Microsoft): Agents communicate via messages, which are dynamically routed by the framework based on the agent states. Strengths include high flexibility as the agents can be LLMs or humans, handling Human-in-the-loop naturally. It is good for research and exploration. Limitations include the requirement of careful conversation management as a mistake may lead to infinite looping, and debugging of multi-agent loops being tricky.

Table 4 depicts the best framework choice in certain scenarios.

IV. WHEN TO USE WHAT

Scenario	Best Choice	Rationale
Build Q&A system from documents	LangChain	Mature RAG integration, cost-efficiency
Multi-step research task	LangChain + Reflexion	Explicit tool use, self-critique
Specialized business workflow (content review, sales)	CrewAI	Clear role specialization; team metaphor
Complex software engineering	MetaGPT	SOP structure prevents hallucinations
Research/rapid prototyping	AutoGen	Flexibility for experimentation

IV. KEY CHALLENGES

A. *Hallucination Propagation*

Unlike a single error in output answer like with traditional chatbots, Agentic AI systems suffer from cascading hallucinations. E.g.:

- Agent hallucinates the output of tool A
- Downstream, tool b receives hallucinated and incorrect input.
- Therefore tool b produces another incorrect answer based on the previous wrong input.
- Another agent further uses this wrong output
- The one hallucinated error compounds across multiple iterations

1) Agent hallucinates that "ACME stock price is $500"
2) Agent uses $500 to compute trading volume threshold
3) Agent executes trade based on false threshold
4) Trade fails

Mitigation Strategies:

- Grounded answers via tool use: Replace the LLM that generates the facts with a verified tool. E.g.: An LLM might have the older price of a stock, but a finance API can fetch the current price of that stock.
- Cited and verified answers: Require the AI agent to cite its sources for the outputs it produces.
- Human-in-the-loop: Add a human approval step before important actions.
- Confidence scoring: Require the agent to check confidence in its every output, and reverify and generate a new output if the confidence score is lower than the set threshold.

B. *Infinite Loops*

Root causes for looping:

- Oscillation: The agent tries the same action again and again expecting a different or a better result. This may be due to bad tool output or due to the LLM not understanding the problem correctly.
- Goal drift: The agents interpretation of the given goal changes midway through execution.
- No termination condition: Agent is given a task with no clear “success” criterion, or it keeps finding new things to explore.

E.g.:

1) "I need to search for X"→[Search]→"Results unclear"
2) "I need to search for X"→[Search]→"Results unclear"
3) "I need to search for X"→[Search]→"Results unclear"
Leading to an infinite loop.

Prevention techniques:

- Set the max steps beforehand: max_iterations=10; if steps executed > max, then exit and use the best output so far.
- Action deduplication: track the last 5 actions executed, and if the current action is in execution again, force a new approach.
- Clear satisfaction metrics: E.g.: for finding cheapest flights, if current price > previous price, failed.
- Timeout and backtracking: if a particular step is taking too long, say more than 30 seconds, reduce the search scope and try again.
- Explicit termination condition: if confidence score > 0.9, success in execution of task.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 02 , February - 2026

C. Token Budget & Cost Optimization

Agentic loops consume significantly more tokens than traditional chatbots. A single task can cost more than 200 times more expensive, as shown in Table 5.

V. TOKEN USAGE COMPARISON

Operation	Tokens/Call	Calls/Task	Total Tokens	Cost (GPT-4, $0.03/K)
Simple Q&A	500	1	500	$0.015
ReAct (5 steps)	2,000	5	10,000	$0.30
Multi-agent (10 agents, 3 turns)	3,000	30	90,000	$2.70
Cost Multiplier	—	—	18–200×	—

Cost Drivers: Total Cost = (Context Size × Tool Calls) × Token Price + (Memory Retrieval × Vector DB Operations) × Retrieval Cost + (External API Calls) × API Cost

- Repeated context: multiple LLM calls per task, each
- Tool use: more tokens are needed, as large results that are fetched need to be included in the next LLM call.
- Agent dialogue and communication: each agent message further requires a new LLM call.

Optimization Strategies:

- Compress History: summarize past interactions, and store the full history in a vector database. This help drastically reduce the token consumption per LLM call.
- Tool use in batches: instead of calling the same tool again and again, plan multiple tool uses in a batch and call the tool once.
- Deciding Model: use cheaper models for the simpler steps, and keep the expensive models only for complex reasoning.
- Caching: store context window between the tool calls, and reuse the output when needed again instead of using the tool again.

V. CONCLUSION AND FUTURE DIRECTIONS

A. Key Takeaways

1) Agentic AI has brought a paradigm shift from static answer generation to dynamic and goal oriented systems capable of autonomously executing the given task.

2) All modern Agentic AI systems (based on LangChain, AutoGen, MetaGPT) use an agent loop, which is a variation of Perception → Planning → Action → Observation.

3) Different frameworks are based on different philosophies:

a) LangChain: It is a modular framework primarily for building LLM applications through prompt chaining. It is best suited for fast prototyping, using RAG, and for tool use workflows.

b) LangGraph: It is a graph based framework with a state machine layer based over LangChain, which enables it to develop stateful and loop driven Agentic AI systems for complex pipelines excelling in decision making.

c) CrewAI: it is a lightweight multi-agent orchestration framework which focuses on role based agent collaboration and delegation, and it is ideal for when coordinated task execution is necessary.

d) MetaGPT: It is a SOP (Standard Operating Procedures) based multi-agent framework which helps categorize responsibilities into structured workflows. This is very effective for automation in software engineering.

e) AutoGen: It is a message based framework for conversational agents. It supports dynamic routing and human-in-the-loop, which makes it suitable for exploratory research and adaptive multi-agent communication.

4) Hallucinations cascade while looping, and mitigation requires cited and verified answers, and/or human-in-the-loop, not just relying on the LLM for factual outputs.

5) Agentic systems are a lot more expensive than traditional chatbots, thus optimization becomes a key factor in deployment.

B. Future Directions

1) On device agents: To reduce latency as well as cost and increase privacy, we can aim to deploy small agents on edge devices like smartphones, laptops, etc.

2) Multi-Modal agents: We can integrate vision and speech combined with text reasoning for building AI that can embody humans.

3) Agent consensus: Find out how agents can reach consensus in disagreements or varying outputs.

4) Benchmarks: We need to develop standardized benchmarks for testing and comparing agent output, reliability, cost, latency, etc.

5) Certified reasoning: We need to add formal verification of the outputs generated by agent reasoning, so as to guarantee its correctness in high stakes fields.

REFERENCES

[1] J. Wei, X. Wang, D. Schuurmans, et al., “Chain-of-Thought prompting elicits reasoning in large language models,” in Advances in Neural Information Processing Systems (NeurIPS), vol. 35, pp. 24824–24837, 2022, arXiv:2201.11903.

[2] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao, “ReAct: Synergizing reasoning and acting in language models,” in Proc. Int. Conf. on Learning Representations (ICLR), 2023, arXiv:2210.03629.

[3] N. Shinn, F. Cassano, A. Gopinath, et al., “Reflexion: Language agents with verbal reinforcement learning,” arXiv:2303.11366, 2023.

[4] J. S. Park, J. C. O’Brien, C. J. Cai, et al., “Generative agents: Interactive simulacra of human behavior,” arXiv:2304.03442, 2023.

[5] S. Hong, M. Zhuge, J. Chen, et al., “MetaGPT: Meta programming for multi-agent collaborative framework,” in Proc. Int. Conf. on Learning Representations (ICLR), 2024, arXiv:2308.00352.

[6] Q. Wu, G. Bansal, J. Zhang, et al., “AutoGen: Enabling next-gen LLM applications via multi-agent conversation,” arXiv:2308.08155, 2023.

[7] S. S. Chowa, R. Alvi, S. S. Rahman, M. A. Rahman, M. A. K. Raiaan, M. R. Islam, M. Hussain, and S. Azam, “From language to action: A review of large language models as autonomous agents and tool users,” arXiv:2508.17281, 2025.

[8] V. de Lamo Castrillo, H. K. Gidey, A. Lenz, and A. Knoll, “Fundamentals of building autonomous LLM agents,” arXiv:2510.09244, 2025.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 02 , February - 2026

[9] U. Nisa, M. Shirazi, M. A. Saip, and M. S. M. Pozi, "Agentic AI: The age of reasoning—A review," Journal of Automation and Intelligence, 2025, in press, doi:10.1016/j.jai.2025.08.003.

[10] M. A. Ferrag, N. Tihanyi, and M. Debbah, "From LLM reasoning to autonomous AI agents: A comprehensive review," arXiv:2504.19678, 2025.

[11] Z. Xi, W. Chen, X. Guo, et al., "The rise and potential of large language model based agents: A survey," arXiv:2309.07864, 2023.

[12] L. Wang, C. Ma, X. Feng, et al., "A survey on large language model based autonomous agents," arXiv:2308.11432, 2023.