

Journal of Smart Algorithms and Applications
JSAA
ISSN: 3070-4189/© 2026, The Authors. (CC BY 4.0) License.
Journal Homepage
https://pub.scientificirg.com/index.php/JSAA

A Systematic Survey of LLM-Based Agentic AI Frameworks for Multi-Agent Coordination and Interoperability

Nada Mohamed[a,1], Prasun Chakrabarti[b], Shashi Kant Gupta[c]

[a] *Department of Computer Science, Higher Technological Institute, Beni Suef, Egypt*
Email: nada.mohamed4335@gmail.com
[b] *Sir Padampat Singhania University, Udaipur, Rajasthan, India. Email: drprasun.cse@gmail.com*
[c] *Post-Doctoral Fellow and Researcher, Computer Science and Engineering, Eudoxia Research University, USA. Email: raj2008enator@gmail.com*

ABSTRACT

Large Language Models (LLMs) have spurred the development of agentic artificial intelligence systems that can reason autonomously, plan, use tools, integrate memory, and carry out multi-step tasks. Unlike conventional prompt-response systems, LLM-based agents extend generative models to goal-oriented architectures that can decompose complex objectives, interact with external environments, and coordinate actions in iterative workflows. The survey systematically reviews LLM-based agent frameworks, multi-agent collaboration architectures, internal communication mechanisms, emerging cross-system interoperability protocols, and open research challenges. The review uses a PRISMA-based methodology, including literature from 2020 to March 2026, with particular focus on prominent frameworks such as LangChain, AutoGPT, AutoGen, MetaGPT, CAMEL, ChatDev, and CrewAI. The survey discusses the transition from modular chain-oriented to stateful graph-oriented and autonomous execution models and presents a systematic taxonomy of multi-agent collaboration architectures, including hierarchical, peer-to-peer, and role-based models. It also reviews the main internal communication mechanisms used to facilitate coordination and information sharing between LLM-based agents and distinguishes them from emerging interoperability protocols designed to support interaction across heterogeneous agents, tools, services, and frameworks. The analysis indicates that the promise of LLM-based agents for scalable automation, collaborative reasoning, and complex workflow execution comes with significant challenges in long-horizon reliability, evaluation standardization, communication security, cost-efficient orchestration, governance, and the interpretability of emergent multi-agent behavior. The survey covers architectural evolution, collaboration patterns, communication mechanisms, framework-level characteristics, and open research challenges. This provides a structured foundation for future research on reliable and trustworthy agentic artificial intelligence systems.

PAPER INFORMATION

HISTORY

Received: 15 May 2026
Revised: 20 June 2026
Accepted: 22 July 2026
Online: 28 August 2026

MSC:

68T07; 68R10; 94A60; 68M15

KEYWORDS

Agentic AI;
Multi-Agent Systems;
Agent Communication Protocols;
Agent Orchestration;
Autonomous Agents.

[1]*Corresponding author: Department of Computer Science, Higher Technological Institute, Beni Suef, Egypt.*
Email: nada.mohamed4335@gmail.com

1. INTRODUCTION

The emergence of Large Language Models (LLMs) has completely transformed the field of artificial intelligence, providing advanced capabilities in natural language understanding, reasoning, code generation, and knowledge-intensive tasks [1]. These capabilities notwithstanding, conventional LLM-based systems typically operate within prompt-response interaction patterns and are limited in their ability to autonomously manage complex, multi-step objectives. Agentic AI builds on the capabilities of LLMs by adding mechanisms for goal decomposition, planning, memory, tool use, iterative reasoning, interaction with the environment, and adaptation based on feedback [2]. Transitioning from prompts as instructions to goal-directed artificial intelligence systems capable of planning and executing sequences of actions with less human intervention is an important change [3].

This evolution has given rise to several frameworks to build and orchestrate LLM-based agents. Launched in 2022, LangChain was one of the first widely adopted frameworks for building modular LLM applications with composable chains, retrieval components, memory mechanisms, and external tools [4], [5]. Its architecture enables developers to combine various logic and functional components to create coherent workflows. AutoGPT, introduced in 2023, is a prominent early approach to deploying autonomous agents. It uses a goal-directed iterative process in which an agent decomposes a high-level goal into sub-goals, selects and executes actions, evaluates intermediate results, and incrementally improves its policy [6]. These developments illustrate the broader evolution of agentic systems from relatively static LLM pipelines toward autonomous and stateful execution architectures.

Another architectural shift has been from individual LLM agents to collaborative Multi-Agent Systems (MAS). Multi-agent architecture does not depend on a single agent to carry out all reasoning and execution tasks but distributes the responsibilities among multiple agents that specialize, communicate, coordinate, critique intermediate outputs, and collectively pursue complex objectives [7]. The use of LLMs in multi-agent systems has allowed for more flexible forms of role specialization, negotiation, collaborative reasoning, and distributed decision-making [8]. AutoGen, MetaGPT, CAMEL, CrewAI, and ChatDev illustrate diverse orchestration methods for LLM-agent teams with dialogs, roles, SOPs, shared memories, and structured workflows [9].

Despite rapid progress, the research landscape of LLM-based agents remains fragmented across natural language processing, software engineering, distributed systems, robotics, and human–AI interaction. Existing works often differ in terminology, architectural assumptions, coordination models, communication mechanisms, and evaluation methodologies, which makes systematic comparison and reproducibility more difficult [10]. Moreover, agentic systems are evolving from framework-level orchestration to heterogeneous ecosystems, where agents communicate with external tools, services, and agents developed with different platforms. Thus, understanding modern agentic AI requires not only individual frameworks but also orchestration architectures, multi-agent collaboration models, communication mechanisms, interoperability protocols, evaluation strategies, and deployment limitations [3].

The main contributions of this survey are**:** (1) a systematic synthesis of representative LLM-based agent frameworks and their evolution from modular chain-based execution to stateful, graph-based, and autonomous orchestration; (2) a structured taxonomy of multi-agent collaboration architectures, encompassing hierarchical, peer-to-peer, and role-based models; (3) an analysis of internal agent communication mechanisms and emerging cross-system interoperability protocols, explicitly distinguishing message-level coordination, shared-state interaction, graph-based control, and protocol-level communication across heterogeneous agent ecosystems; and (4) a comparative analysis of current limitations and research gaps related to benchmarking, long-horizon reliability, scalability, communication security, governance, cost efficiency, and the interpretability of emergent multi-agent behavior.

This survey is organized as follows. Section 2 presents the methodology of systematic literature review, including the search strategy, the study selection process, the inclusion and exclusion criteria, and the screening procedure based on PRISMA. Section 3 reviews representative LLM-based agent frameworks with a focus on the architecture evolution of LangChain and LangGraph and the autonomous goal-driven execution model of AutoGPT, including their empirical evaluations and limitations. Section 4 discusses multi-agent collaboration architectures, framework-level

comparisons, internal communication paradigms, and emerging interoperability protocols. Section 5 discusses open research challenges and provides directions for future work in benchmarking, reliability, security, interoperability, governance, cost-efficient orchestration, and interpretability. Finally, Section 6 concludes the survey with the main findings and conclusions.

2. LITERATURE REVIEW METHODOLOGY

This systematic review aimed to locate, appraise, and synthesize recent research on Agentic AI, LLM-based autonomous agents, and collaborative multi-agent systems. The review methodology was developed to ensure a transparent and reproducible process of identification, screening, eligibility assessment, and selection of the final studies. We paid special attention to studies that dealt with agent architectures, reasoning and planning mechanisms, memory systems, tool-augmented execution, orchestration frameworks, multi-agent collaboration, communication mechanisms, interoperability, evaluation, and reliability. The review process followed a PRISMA-based workflow comprising literature identification, duplicate removal, title and abstract screening, full-text eligibility assessment, and final study inclusion.

2.1 Search Strategy

A systematic search of literature was performed in four major scientific databases and digital libraries, namely Scopus, ScienceDirect, SpringerLink, and IEEE Xplore. Studies published between January 2020, and March 2026 were included in the formal systematic search. To capture rapidly evolving developments in agent frameworks, benchmarks, and interoperability standards, a limited number of authoritative technical and scholarly sources published after the formal search cutoff were consulted as supplementary background sources. Foundational pre-2020 sources were also consulted when necessary to provide historical context or established definitions. Neither the post-search updates nor the pre-2020 foundational sources were included in the PRISMA study counts or systematic evidence synthesis. This period was chosen to cover the advent and rapid evolution of LLM-powered autonomous agents and the subsequent shift toward collaborative and multi-agent orchestration architectures.

The search strategy encompassed terms concerning Agentic AI, autonomous LLM agents, multi-agent collaboration, reasoning, planning, and tool use. Boolean operators were used to combine the main concepts with each other so as to improve the relevance and coverage of the retrieved literature. The primary search term was structured as follows:

("Agentic AI" OR "LLM agents" OR "Autonomous AI") AND ("Multi-agent systems" OR "agent collaboration") AND ("reasoning" OR "planning" OR "tool use")

Then, retrieved literature was screened for studies discussing architectural and operational aspects of LLM-based agents such as autonomous reasoning, planning, memory, orchestration, tool integration, agent collaboration, and related evaluation mechanisms.

2.2 Study selection process

The study selection process was an adapted PRISMA flow for identification, duplicate removal, screening, eligibility, and final inclusion. The first search in Scopus, ScienceDirect, SpringerLink, and IEEE Xplore resulted in 1,034 records (304 from Scopus, 526 from ScienceDirect, 177 from SpringerLink, and 27 from IEEE Xplore).

The database search yielded 1034 records, of which 422 were duplicates, leaving 612 records to be screened by title and abstract. At this stage, the studies were screened for their relevance to the main themes of the review, including LLM-based autonomous agents, reasoning and planning mechanisms, agent orchestration, memory architecture, tool-augmented execution, and multi-agent collaboration. In total, 346 records were excluded at the title and abstract level, leaving 266 articles for full-text eligibility assessment.

Studies were assessed for full text according to predefined inclusion and exclusion criteria. 145 full-text articles were excluded due to the following reasons: studies not focused on Agentic AI or LLM-based agents; studies not addressing multi-agent collaboration or orchestration; studies lacking sufficient coverage of reasoning, planning, memory, or tool-augmented execution; studies that were not peer reviewed or lacked sufficient technical or methodological depth; and studies published in languages other than English. After eligibility assessment, 121 studies were included in the final review and thematic synthesis, as shown in **Figure 1**.

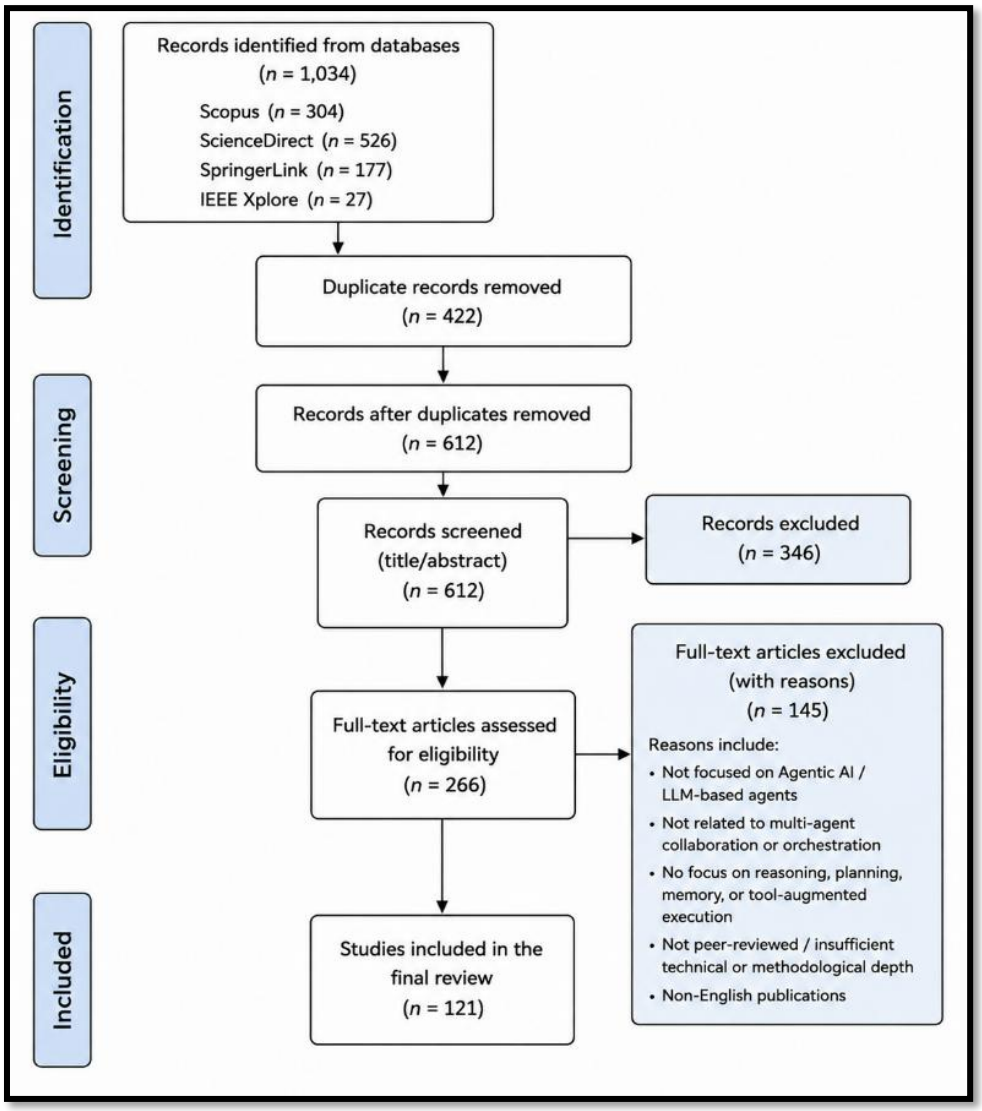


Figure 1: PRISMA flow diagram.

2.3 Inclusion and Exclusion Criteria

Studies were included if they met the review's thematic, methodological, and publication criteria. The inclusion criteria were (1) peer-reviewed journal articles or conference papers published between January 2020 and March 2026; (2) publications in English; (3) studies involving Agentic AI, LLM-based autonomous agents, or LLM-based multi-agent systems; and (4) studies containing relevant technical discussion of one or more core agentic components, including reasoning, planning, memory, orchestration, tool use, autonomous or semi-autonomous decision-making, multi-agent coordination, communication, or evaluation. For the final synthesis, priority was given

to contributions of an architectural nature, methodological developments, agentic systems that have been experimentally evaluated, or technically substantive analyses.

We excluded studies that were limited to traditional generative AI or prompt-based chatbot systems without autonomous agent capabilities. This survey also goes beyond studies on classical symbolic AI or traditional multi-agent systems that do not integrate large language models. Other exclusion criteria were non-English publications, editorials, opinion pieces, tutorials, blog posts, and other informal, non-peer-reviewed sources. We also excluded duplicate records and studies that did not provide adequate architectural description, technical depth, methodological rigor, or direct relevance to autonomous or collaborative LLM-based agent systems.

Methodological quality was considered qualitatively during full-text assessment based on the clarity of the architectural description, technical depth, methodological rigor, evaluation transparency where applicable, and direct relevance to the review scope.

3. LLM-Based Agent Frameworks

The emergence of large language models (LLMs) as general-purpose reasoning engines has led to the development of dedicated software frameworks that encapsulate, coordinate, and leverage LLMs' capabilities for autonomous task execution [11]. These frameworks abstract the complexity of prompt engineering, memory management, tool integration, and multi-step planning so that developers can build sophisticated agentic applications [12].

Table 1 provides a comparative overview of six representative LLM-based agent frameworks, highlighting their core architectural paradigms, workflow models, multi-agent capabilities, and tool integration.

Table 1: Comparative Overview of Representative LLM-Based Agent Frameworks.

Ref	Framework	Core Paradigm	Workflow Type	Multi-Agent	Tool Use
[13]	LangChain	Modular composition + RAG orchestration	DAG/LCEL pipelines; stateless default	Supported through LangGraph	200+ integrations (APIs, DBs, vector stores)
[14]	AutoGPT	Goal-driven autonomous loop	Recursive think-act loop	Limited (plugin-based)	Web browser, code exec, file I/O
[15]	AutoGen	Conversational multi-agent actor model	Static + dynamic conversation	Native (N agents)	Code execution, function calling
[16]	MetaGPT	SOP-driven role-based meta-programming	Assembly-line SOP chains	Role-specialized teams	Tool Server; file I/O
[17]	CAMEL	Role-playing communicative agents	Role-playing conversational workflows	Native role-playing multi-agent interaction	Limited (no code execution)
[18]	CrewAI	Role-assigned crew automation	Configurable team workflows	Native	Extensive plugin ecosystem

3.1 LangChain: Modular Composition

LangChain has become a popular framework for developing LLM-based applications, providing modular abstractions for model interaction, retrieval, tool integration, memory management, and agent-oriented workflow orchestration[19]. This modular design allows complex LLM applications to be built from reusable, interchangeable components, with workflows ranging from Retrieval-Augmented Generation (RAG) pipelines to multi-step agentic systems incorporating reasoning, external tools, and contextual state [20].

Structurally, the main functional components of the LangChain ecosystem can be categorized into four interrelated layers: (1) Model I/O: standardized interfaces for communication with different language models and prompt components; (2) Retrieval: document loaders, text splitters, retrievers, and vector stores for retrieval of external knowledge; (3) Workflow Orchestration: coordination of sequential, parallel, conditional, and graph-based execution, exemplified by LCEL pipelines and LangGraph; and (4) Agent Components: reasoning, tool selection, memory, and action planning to support goal-oriented execution [21] Cross-cutting components include memory, which stores contextual information, and external tools, which allow interaction with external resources during the execution process.

As depicted in **Figure 2**, these components constitute an iterative workflow where user instructions are conveyed through model interfaces and orchestration mechanisms, augmented by retrieval, memory, and external tools that offer additional information and execution capabilities. Intermediate states and tool outputs can be returned to the orchestration layer for reasoning and action selection until a predetermined termination condition is met. This modular structure allows for the construction of simpler LLM pipelines as well as more complex stateful agent workflows.

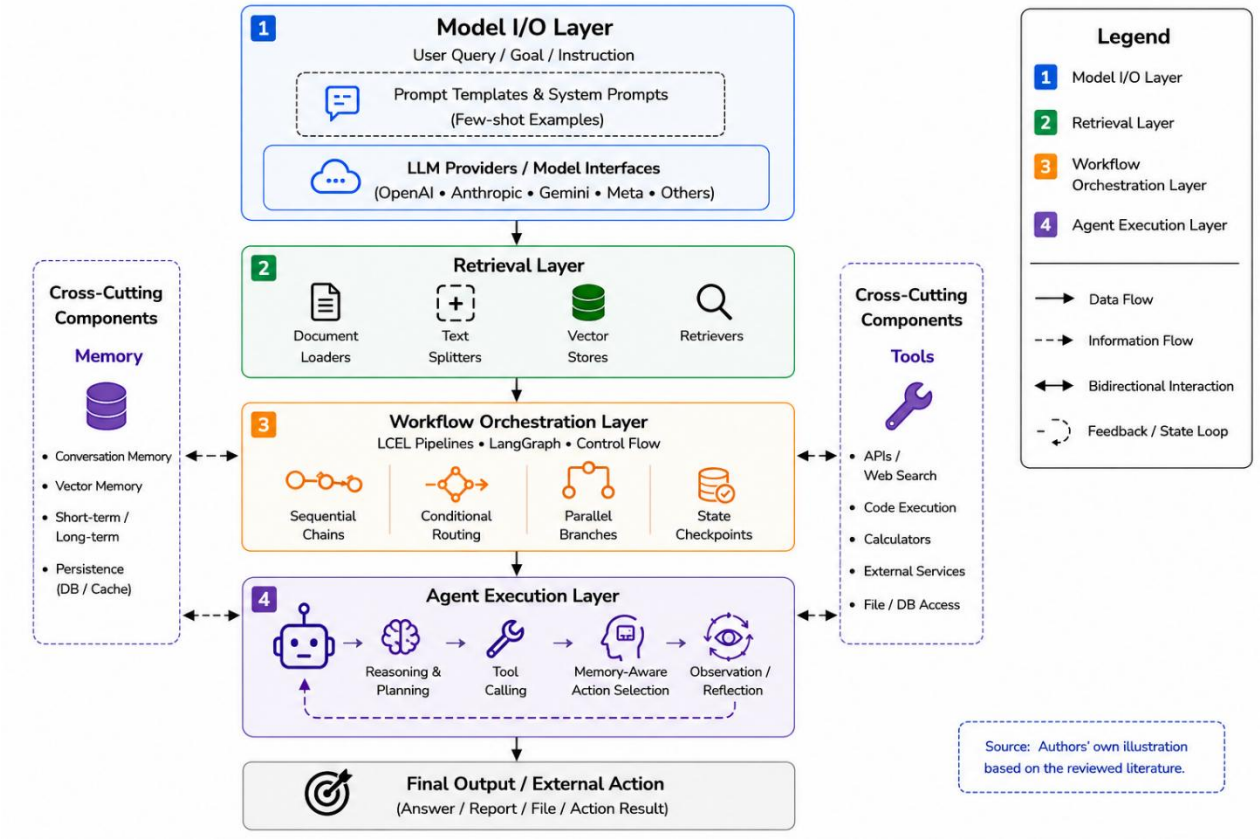


Figure 2: layered modular architecture of the LangChain framework

3.1.1 LangChain Expression Language (LCEL) and Pipeline Composition

LangChain introduced the LangChain Expression Language (LCEL) as a declarative way of composing reusable components into structured LLM pipelines. LCEL links together components like prompt templates, language models, retrievers, tools, and output parsers using a single runnable interface, which is often expressed using pipe-style composition. This abstraction makes it easier to build sequential and parallel execution workflows and provides a more flexible composition mechanism than previous class-based chain implementations [22].

LCEL is particularly well suited for deterministic pipeline composition where individual Runnable components are linked together to form structured graphs of execution. Pipelines can be sequential; have parallel branches, conditional branches, retrieval components, and external functions. The pipeline composition does not inherently provide state management; thus, conversational history or persistent application state needs to be explicitly added when needed[23]. This separation of pipeline execution from state management enables modular testing and component reuse and lets developers choose state management mechanisms according to application needs.

In the broader LangChain ecosystem, conversational context can be preserved using memory and message-history mechanisms that retain information across interactions. Existing implementations of LangChain have provided ways such as Conversation Buffer Memory, Conversation Buffer Window Memory, and Conversation Token Buffer Memory to control the amount and form of conversational history maintained during execution[24]. These mechanisms allow applications to trade off between contextual continuity and token consumption and computational overhead.

For deployment, LangServe offers mechanisms to expose LangChain runnable components and LCEL-based applications as REST APIs, making them usable in external applications and production-oriented service architectures [22]. The combination of LCEL-based composition, explicit state management, and deployment support has resulted in a modular pipeline architecture that can serve as the basis for more complex agentic workflows.

3.1.2 LangGraph: From DAGs to Stateful Cyclic Graphs

LangGraph brings graph-based orchestration to the LangChain ecosystem, enabling agentic workflows that need persistent state, conditional control flow, iterative execution, and human intervention. Unlike purely acyclic pipeline structures, LangGraph allows execution paths to return to earlier defined nodes via cycles, enabling workflows that incorporate planning, tool execution, evaluation, reflection, and iterative refinement[13]. This makes it particularly well suited to long-running, multi-step agentic applications that must track intermediate information and execution state across multiple interactions.

LangGraph models workflows as graphs of nodes, edges, and shared state. Nodes perform computational operations such as LLM-based reasoning, tool invocation, state transformation, or human-in-the-loop interaction, and edges specify the transitions between them. Conditional routing selects the next execution step based on the current state or intermediate results. Cyclic transitions allow for returning to earlier stages in a workflow, thus supporting iterative patterns such as reflection, correction, replanning, and repeated tool use [25]. The graph state is updated during execution and can store relevant information such as conversation history, intermediate outputs, tool results, contextual variables, and decisions.

Figure 3 illustrates an example of a stateful agent workflow, where a user-defined objective is followed by planning, tool execution, evaluation, and state-update stages. Then there is a conditional routing mechanism that decides if the current objective has been met or if more iterations are needed. If further refinement is needed, execution can return to a previous node, establishing a feedback loop. Otherwise, the workflow proceeds toward termination and generation of the final output. A persistent state allows for continuity through these transitions so that ensuing nodes can operate on information gathered during prior steps of execution.

Thus, the main architectural distinction shown in **Figure 3** is not stateful vs. stateless execution but acyclic pipeline

composition vs. graph-based orchestration with explicit support for cycles, conditional transitions, and persistent execution state. LCEL works well for composable pipeline operations, while LangGraph provides more control-flow capabilities for workflows that need iterative reasoning, recovery, human-in-the-loop checkpoints, or repeated interactions with external tools. These features make LangGraph particularly relevant for complex agentic and multi-agent workflows where execution cannot always be encoded as a predetermined linear sequence.

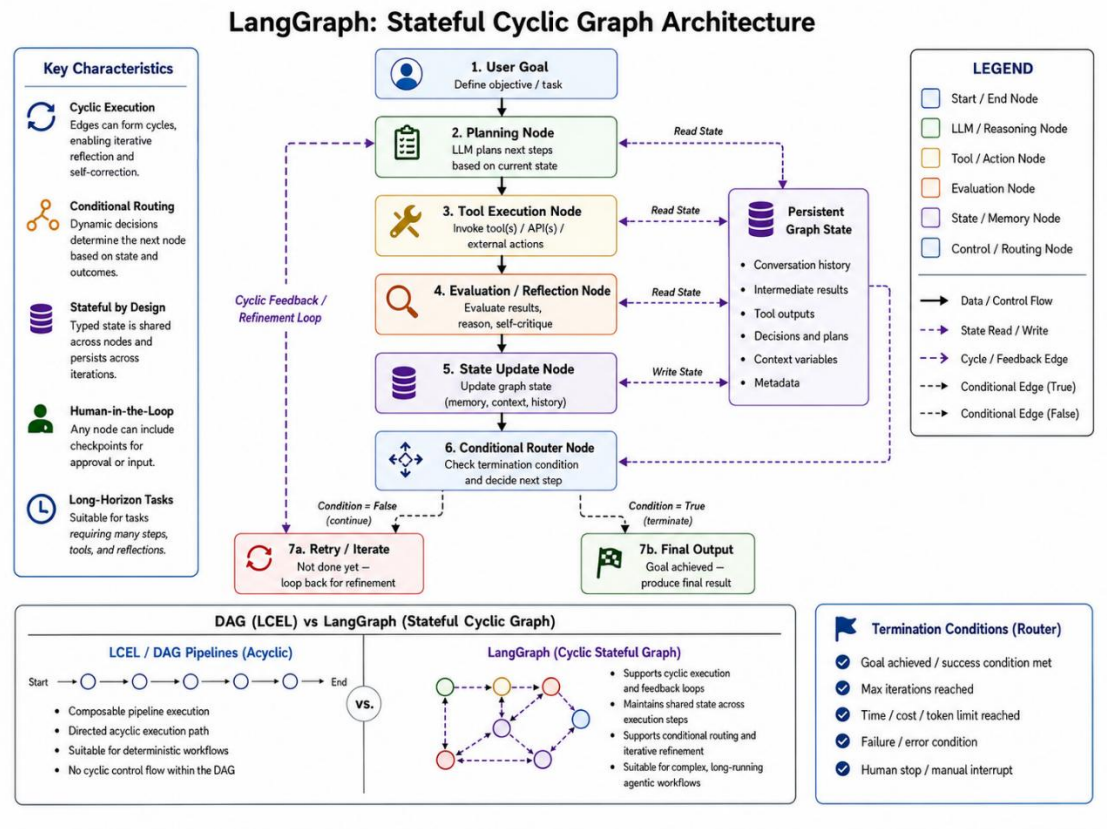


Figure 3: LangGraph Stateful Cyclic Execution Architecture

3.1.3 Evolution from Linear Chains to Graph-Based Orchestration

The persistent, cyclic graph structure described above is not merely a feature of one framework; it marks a broader architectural pivot in how LLM-based systems are built. Framing this shift at the ecosystem level, rather than the framework level, clarifies why independently built agent projects keep converging on similar designs [13].

The evolution of the LangChain ecosystem is indicative of a larger architectural shift in LLM-based applications from prescribed sequential pipelines to more flexible graph-based orchestration. The early chain-driven workflows were mostly built around pre-determined sequences of execution that ordered calls to models and processing components. Such an approach is well suited for jobs with stable and predictable processing stages. However, increasing complexity of agentic apps demands more control mechanisms for conditional execution, iterative refinement, state persistence, and recovery from intermediate failures [26].

This transition highlights an important distinction among modern orchestration approaches. Comparative analyses of LangChain and LangGraph emphasize the shift from chain-oriented composition toward explicit graph-based control of workflow state and execution transitions [27]. Conversational multi-agent frameworks such as AutoGen, in contrast, emphasize configurable agent-to-agent interaction and coordination [15]. These approaches are complementary orchestration abstractions rather than necessarily competing architectural models.

3.1.4 Performance Evaluation and System Limitations

Empirical evidence on LangChain-based agent systems indicates that their practical effectiveness is driven not only by the flexibility of the framework but also by the reliability of the underlying models, tools, retrieval components, and orchestration logic. The authors of the 2024 LangChain State of AI Agents Report observe increasing adoption of AI agents in production environments. Output quality and performance reliability are key deployment concerns [28]. Such observations suggest that production-oriented agent systems must balance architectural adaptability with predictable execution, output consistency, and operational dependability.

Experimental work on LLM-based agents has also revealed limitations in multi-step reasoning and task performance. Errors can be introduced during intermediate reasoning or tool execution steps and propagate to later steps, especially when workflows do not have explicit validation, recovery, or correction mechanisms [15] . Completing each stage of the pipeline successfully does not guarantee that the agent behaves reliably from beginning to end. The limitation is more severe for long-horizon workflows, where repeated calls to the model, interactions with external tools, and the accumulation of intermediate states increase the chances of error propagation. These results motivate the use of verification, reflection, retry, and state-aware recovery mechanisms for more sophisticated architectures of agent orchestration.

Security is yet another limitation for tool-augmented LangChain applications. When agents perform actions on retrieved documents, external APIs, databases, or other tools, the inputs they receive may be untrusted and can affect their subsequent reasoning and actions. Prompt-injection attacks can inject malicious instructions through external content and possibly change agent behavior in Retrieval-Augmented Generation (RAG) or tool-enabled workflows [29]. This potential impact is even greater when the agent is allowed to take actions in the outside world, rather than just producing textual responses. Thus, reliable deployment requires application-level orchestration mechanisms as well as safeguards such as input and output validation, limited tool permissions, execution isolation, access control, and monitoring.

Overall, the existing evidence indicates that LangChain provides substantial flexibility for the composition of LLM-based applications, but the flexibility of the framework alone does not ensure reliable agent execution. Long-horizon consistency, error propagation, recovery mechanisms, security of tool interactions, and reproducible evaluation are important limitations that remain. These issues motivate the move to stateful orchestration approaches like LangGraph, where explicit state management and conditional control flow enable more structured recovery and iterative refinement.

3.2 AutoGPT: Autonomous Goal-Driven Execution

AutoGPT is one of the first popular approaches for autonomous, goal-driven LLM agents. Unlike developer-specified pipeline frameworks where the execution steps are largely predefined, AutoGPT emphasizes iterative autonomous execution with the language model being involved in task planning, action selection, tool usage, and evaluation of intermediate outcomes [30]. The agent repeatedly selects the actions that are appropriate for the goal at a high level. The observations from previous steps are used to guide decisions in the next steps. This establishes an iterative goal-directed execution loop.

The conceptual AutoGPT workflow integrates several interacting capabilities: goal decomposition and task planning, LLM-based reasoning and decision-making, memory management, external tool execution, and iterative evaluation of intermediate outcomes [31].

Short-term contextual information supports the current execution cycle. Longer-term memory mechanisms can hold information that might be useful in future steps. The tool-integration layer allows the agent to go beyond text generation by interacting with external resources such as web services, files, APIs, and executable environments. The observations obtained by each action can then be stored in memory and used to determine whether the current goal has been achieved or whether further planning and execution are required.

This architecture can be described as an iterative plan-reason-act-observe-evaluate cycle. The first step is to identify an objective, which is decomposed into actionable tasks. The LLM then decides on an action and calls upon an external tool if needed. The observations are added to the agent's working context and memory, and then the agent's current progress is evaluated. If the goal has not been achieved, the agent can replan and go through another execution cycle; otherwise, the process terminates and the result is returned. This iterative structure decouples autonomous goal-driven execution from fixed sequential LLM pipelines.

The AutoGPT ecosystem grew with AutoGPT Forge, which focused on a more modular approach to building and testing agent implementations [32]. Related standardization activities, including an agent protocol interface, also aim for consistent interaction between agents, evaluation environments, and external applications. Such mechanisms can enhance interoperability and allow for more systematic benchmarking of agent behavior [33]. However, standard interfaces do not solve the larger challenges of autonomous execution, such as long-horizon reliability, error propagation, evaluation consistency, and safety of interacting with external tools.

The conceptual workflow of an AutoGPT-style autonomous agent is shown in **Figure 4** and includes goal decomposition, LLM-based reasoning, tool execution, memory management, and iterative evaluation. If the goal is not reached, the agent continuously monitors its progress and replans through the feedback loop, or if the goal is reached, it terminates the process by generating the final output.

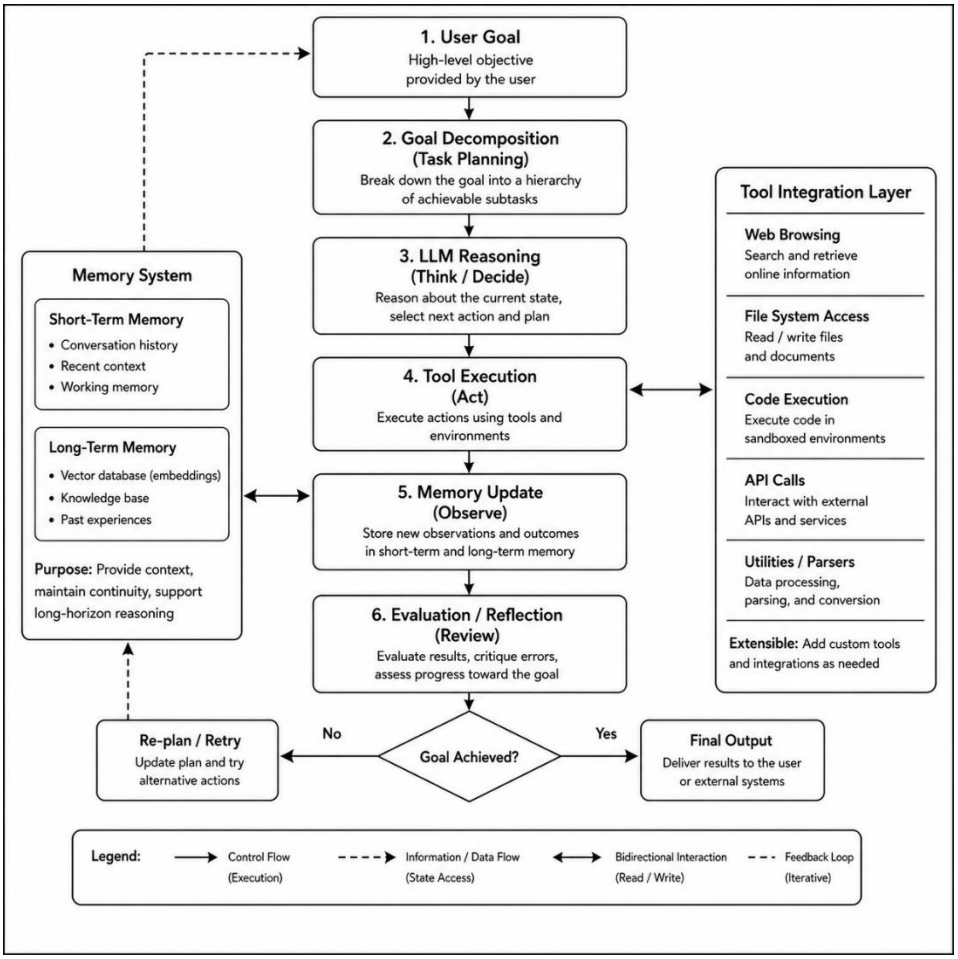


Figure 4: AutoGPT Agent Architecture

3.2.1 Benchmarking and Empirical Evaluation

Empirical evaluations of AutoGPT-style agents have shown a continued gap between autonomous task execution and reliable long-horizon performance. Studies assessing autonomous agents in interactive decision-making environments indicate that, although AutoGPT-style architectures can generalize their behavior across various task settings, performance tends to degrade as tasks necessitate longer sequences of reasoning, action selection, tool interaction, and feedback processing. Existing empirical evidence suggests that increasing agent autonomy does not necessarily lead to reliable task completion, particularly when errors at intermediate stages propagate through subsequent execution steps. Recent surveys of agent harnesses further highlight this challenge, showing that the reliability of long-running LLM agents depends not only on the underlying language model but also on the surrounding execution infrastructure, including execution loops, tool management, context handling, state storage, and evaluation mechanisms [34].

The reported results show large variations in performance across underlying language models and agent configurations, suggesting that agent performance is not only dependent on the orchestration framework but also on the underlying model capabilities and its interaction with the surrounding agent infrastructure. Therefore, performance comparisons should not solely focus on the agent framework but also consider the model-framework configuration and the orchestration architecture. Aside from benchmark performance, several recurring reliability limitations in AutoGPT-style autonomous loops affect their suitability for unconstrained deployment [35].

Four failure patterns that are especially relevant and have been reported throughout the literature are goal drift, repetitive or non-terminating execution loops, invalid or hallucinated tool actions, and uncontrolled computational or API cost [36] Goal drift is when successive planning and execution steps slowly drift away from the original goal. When the agent is unable to identify sufficient termination conditions or keeps repeating unsuccessful actions, repetitive loops form. Invalid tool actions mean the agent generates inappropriate tool calls, parameters, or unsupported actions, and long-running execution not limited by explicit iteration, time, or cost limits can lead to overconsumption of resources [37].

Together, these limitations suggest that autonomous execution requires more than unfettered iterative prompting. Thus, practical agent systems can use explicit termination criteria, bounded execution budgets, tool validation, state monitoring, recovery mechanisms, and human intervention for high-risk or persistently unsuccessful actions. In this sense, AutoGPT has played an important role in demonstrating the promise of goal-oriented autonomous LLM agents, while the well-documented limitations of AutoGPT have also helped motivate subsequent research toward more controllable, observable, and reliable forms of agent orchestration [28].

3.2.2 Architectural and Operational Comparison with Contemporary Frameworks

Table 2 compares four widely used LLM agent frameworks such as AutoGPT, AutoGen, MetaGPT, and LangChain, and reveals large differences in execution model, multi-agent support, reasoning performance, software-engineering orientation, production reliability, and typical failure modes.

Table 2: Architectural and Operational Comparison of Representative LLM Agent Frameworks

Dimension	AutoGPT [14]	AutoGen [15]	MetaGPT [16]	LangChain [38]
Execution Model	Autonomous recursive self-prompting loop	Conversational multi-agent orchestration	SOP-driven role-specialized collaboration	Modular chain-based orchestration
Multi-Agent Support	Limited	Native multi-agent collaboration	Native role-specialized teams	Supported via LangGraph

Reasoning Capability	Limited long-horizon stability	Collaborative reasoning through configurable multi-agent interaction	Optimized for workflow decomposition	Dependent on orchestration design
Code Generation	Experimental support	Integrated code execution	Strong software engineering orientation	Tool-assisted coding pipelines
Production Reliability	Susceptible to loop instability	Enhanced through iterative verification	Structured execution through SOP-based role constraints	Dependent on workflow implementation
Primary Failure Modes	Goal drift, recursive loops	Communication overhead	Workflow rigidity	Prompt injection and output variability

4. Multi-Agent Collaboration Architectures

The move from single-agent systems to LLM-based Multi-Agent Systems (MAS) introduces a new architectural paradigm where reasoning, planning, and task execution can be distributed among several interacting agents. Instead of a single agent to address all levels of a complex task, multi-agent architecture allows agents to take on specialized roles, share intermediate information, coordinate actions, and assess or improve each other's output [39]. This distribution of duties can lead to task decomposition, parallel execution, role specialization, and collaborative problem-solving.

Multi-agent collaboration, however, also introduces challenges that are less prominent in single-agent settings. As the number of interacting agents grows, the performance of the system greatly depends on efficient coordination, communication efficiency, role allocation, shared-context management, and conflict resolution [40]. Adding multiple agents can provide potential benefits, but these benefits can be reduced or even negated by communication overhead, inconsistent decisions by agents, the propagation of incorrect information, redundant interactions, and failures in coordination. The success of an LLM-based MAS depends on both the agents' capabilities and the architecture of their interactions.

To systematically analyze these design choices, this section investigates LLM-based multi-agent systems from three complementary dimensions: collaboration architecture, communication mechanisms, and framework-level implementation. Collaboration architectures are described in terms of hierarchical, peer-to-peer, and role-based models. Communication and interoperability mechanisms are reviewed, and a comparative study of representative multi-agent frameworks is presented.

This taxonomy gives a systematic framework to characterize the pros, cons, and design trade-offs of different multi-agent coordination approaches.

4.1 Collaboration Models

4.1.1 Taxonomy of Multi-Agent Collaboration

Based on the reviewed literature, the LLM-based multi-agent collaboration architecture can be broadly categorized into three representative categories: hierarchical, peer-to-peer, and role-based collaboration models, according to the organizational structure, agent roles, interaction patterns, and task-allocation mechanisms. The main differences

among these categories lie in how decision-making authority is distributed, how tasks are allocated among agents, how information is exchanged, and how agents coordinate their outputs toward a common goal. The architectural features, benefits, limitations, and application scenarios of each collaboration model are discussed in the following subsections.

4.1.1.1 Hierarchical (Orchestrator-Subordinate) Models

Hierarchical multi-agent architectures involve collaboration around a central coordinating agent that decomposes a high-level objective into subtasks, delegates them to specialized agents, and integrates the resulting intermediate outputs. A centralized structure provides explicit control over task assignment, task order, and information flow. This makes it especially suitable for workflows with well-defined stages and specialized responsibilities. A prime illustration of this paradigm of organization is MetaGPT, where software engineering workflows are simulated with role-specialized agents (product managers, architects, engineers, and reviewers) orchestrated by established Standard Operating Procedures (SOPs) [16].

In this architecture, different agents are responsible for various stages of the workflow. The Product Manager can craft a Product Requirements Document (PRD), the Architect can translate these requirements into a system design, and engineering agents can then produce implementation artifacts. Then reviewer agents can review the generated output before it is finished. This specialization of roles and structured information exchange can reduce ambiguity between workflow stages and can provide explicit checkpoints for evaluation of intermediate artifacts [16].

However, as with all architectural decisions, hierarchical coordination involves important trade-offs. The central orchestrator might become a coordination bottleneck or a single point of failure as the number of agents and task dependencies grows. Mistakes in the decomposition, routing, or decision-making of the orchestrator can also be passed on to subordinate agents and influence downstream outputs. Predefined roles and SOP-driven workflows can also further limit flexibility where tasks require decentralized negotiation or dynamically changing responsibilities. Hierarchical architecture thus provides a strong structure to the workflow with centralized coordination, but the effectiveness of such architecture depends on the availability of reliable orchestration, proper assignment of roles, and mechanisms for detecting and recovering from failures of coordination.

Figure 5 shows a typical hierarchical multi-agent architecture where an orchestrator processes a user-defined objective and distributes it to role-specialized agents. The specialized agents produce intermediate artifacts that are then integrated in the engineering and review stages. This SOP-oriented workflow exhibits centralized task decomposition, role specialization, and sequential coordination.

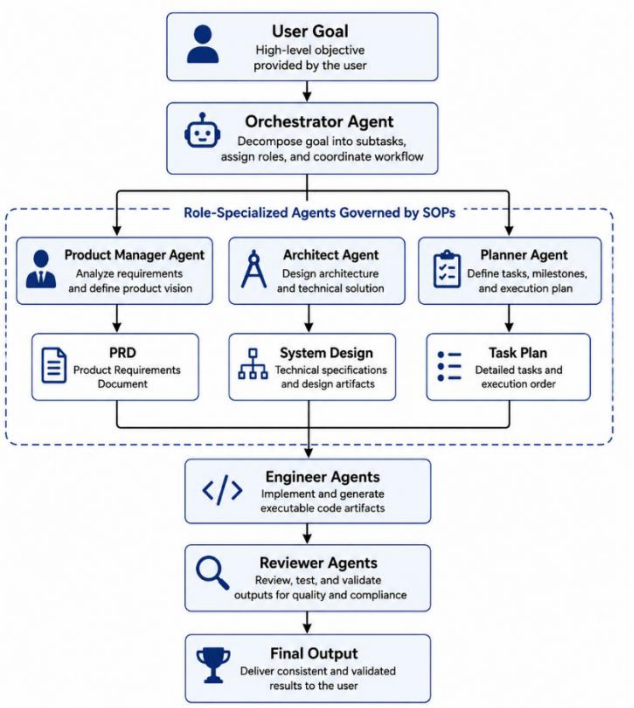


Figure 5: Hierarchical Multi-Agent Architecture with SOP-Governed Role Specialization

4.1.1.2 Peer-to-Peer (Collaborative Discourse) Models

Peer-to-peer multi-agent architectures organize cooperation among agents that have similar decision-making authority, instead of a static hierarchical controller. Agents are interacting directly, passing proposals, intermediate results, critiques, and revisions so that collective decisions emerge iteratively. This distributed design allows collaborative reasoning, information sharing, and consensus-based problem solving, especially when different agents can independently evaluate alternative solutions or opinions.

Conversational multi-agent frameworks such as AutoGen can support this form of interaction through configurable agent-to-agent conversations and group-chat mechanisms in which agents exchange messages and respond according to predefined or dynamically selected interaction patterns [15]. Related multi-agent debate approaches have reported improvements over single-agent baselines on selected reasoning and factuality tasks by allowing multiple agents to independently generate responses, critique competing outputs, and iteratively refine candidate solutions [41]. However, these benefits are task- and configuration-dependent and should not be interpreted as a general guarantee that increasing the number of interacting agents will improve performance.

Peer-to-peer collaboration also poses several architectural challenges. With more agents and interaction rounds, communication overhead and token consumption may grow prohibitively large. Agents might repeatedly exchange redundant information, reinforce incorrect conclusions, fail to reach consensus, or propagate errors through the communication network. Moreover, the lack of global control can make termination, conflict resolution, assignment of responsibility, and global monitoring of the workflow more complex. Therefore, the peer-to-peer architectures are more decentralized and allow for more flexible collaborative reasoning but require effective communication policies, stopping criteria, and mechanisms for handling inconsistent or conflicting agent outputs.

We show a typical peer-to-peer collaboration architecture in **Figure 6**, where agents communicate directly and also maintain shared contextual memories. The bidirectional message exchange enables decentralized information sharing. The iterative critique loops allow the agents to evaluate and improve each other's intermediate outputs.

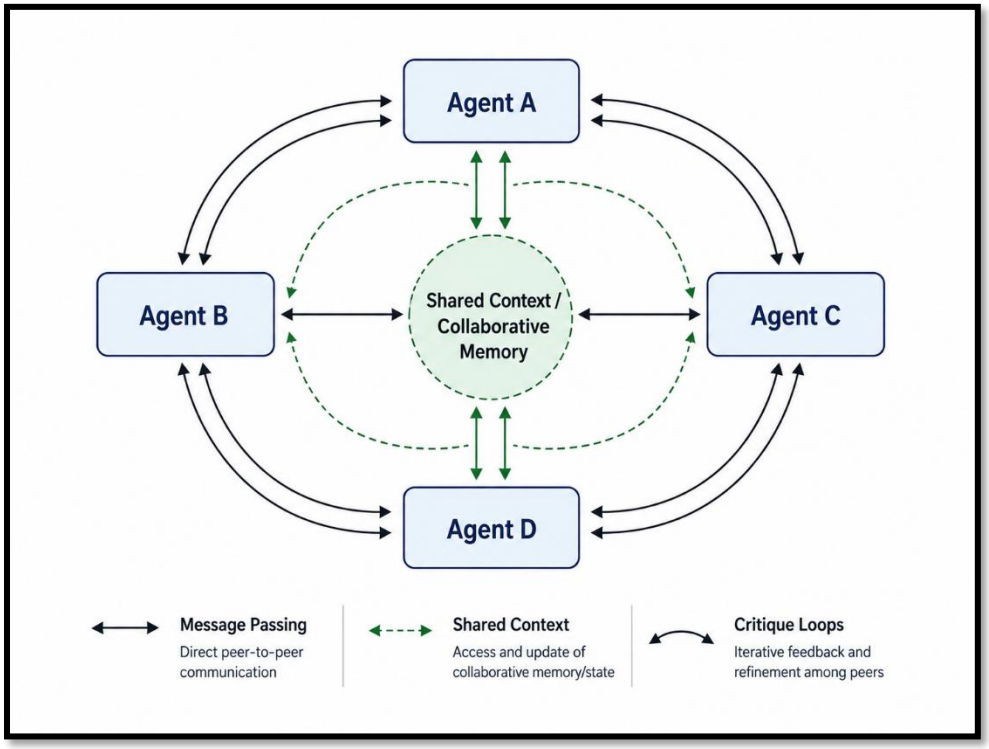


Figure 6: Peer-to-Peer Model

4.1.1.3 Role-Based Ensemble Models

Role-based multi-agent architectures organize collaboration by assigning agents different functional roles, expertise profiles, responsibilities, or behavioral instructions, which govern their reasoning and interaction. Role-based collaboration differs from topology-based classifications because it emphasizes functional specialization rather than a particular communication structure. It can therefore coexist with hierarchical, peer-to-peer, sequential, or graph-based coordination.

CAMEL suggested role-playing as a formalized mechanism for autonomous agent cooperation, in which role-specific prompting was applied to drive interactions between agents with complementary responsibilities [17]. Similarly, CrewAI organizes agents into pre-defined roles, goals, and task responsibilities, facilitating teamwork among specialized agents in coordinated team workflows [42]. Role-based collaboration can be combined with hierarchical coordination, sequential execution, or collaborative interaction, depending on the configuration, so role specialization should be considered an orthogonal design dimension, not a mutually exclusive communication topology.

Role specialization improves task decomposition by adding subtasks to the explicitly defined responsibilities of agents and by adding evaluation checkpoints via critic or reviewer roles. But its success relies very heavily on the quality of the role definitions and the coordination mechanisms linking the agents. Redundant work or conflicting outputs may result from overlapping or ambiguous roles. Conversely, overly rigid role definitions may hinder adaptation when task requirements change. Furthermore, the use of specialized agents leads to increased communication, computational, and task overhead. Role-based ensembles thus need the careful design of roles and responsibilities, as well as mechanisms for combining possibly conflicting intermediate outputs.

Figure 7 is an example of a role-based multi-agent workflow where a manager controls a set of functionally specialized agents. The researchers, engineers, and critic agents perform complementary functions (information gathering, implementation, and validation, respectively), and their intermediate artifacts are assembled into a final goal-aligned output.

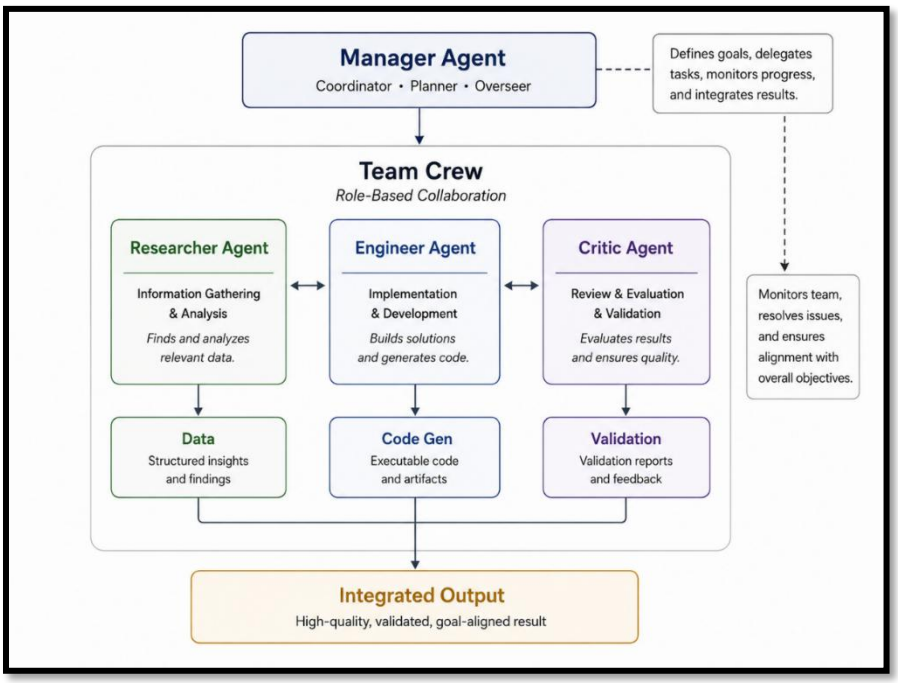


Figure 7: Role-Based Ensemble Models

4.1.1.4 Framework-Level Comparison of Collaboration Architectures

Table 3 summarizes representative LLM-based multi-agent frameworks and emphasizes their different architectural orientations, task domains, and reported contributions. In general, the comparison indicates that recent frameworks

have shifted from flexible conversation-based coordination, as seen in AutoGen, to more structured role-specialized and workflow-oriented frameworks, like MetaGPT and ChatDev.

Table 3: Comparative Overview of Representative LLM-Based Multi-Agent Frameworks

Ref	Framework	Task Domain	Reported Findings	Comparison / Reference Baseline	Main Contribution / Advantage
[15]	AutoGen	Mathematical reasoning, ALFWorld, OptiGuide	Demonstrated improved multi-agent task solving through configurable conversations, tool use, code execution, and human-in-the-loop interaction	GPT-4 standalone, LangChain/ReAct-style baselines, ChatGPT tools, contemporary agent frameworks	Flexible multi-agent conversation programming with human/tool/code integration
[16]	MetaGPT	Collaborative software engineering; code-generation benchmarks	Improves collaborative software generation by encoding SOPs into multi-agent workflows and verifying intermediate outputs	Single-agent LLMs and existing LLM-agent frameworks	SOP-based role specialization provides structured task decomposition and intermediate artifact coordination
[17]	CAMEL	Role-playing agents; communicative cooperation	Introduced role-playing and inception prompting to study autonomous cooperation among communicative agents	Single-agent or less structured conversational settings	Role differentiation enables autonomous cooperation and analysis of agent society behavior
[43]	MegaAgent	Large-scale LLM-based multi-agent cooperation; software/game development; policy simulation	Demonstrates dynamic agent generation, task decomposition, parallel execution, monitoring, and scalability to large agent populations	AutoGen, MetaGPT, CAMEL, AgentVerse	Supports large-scale autonomous cooperation without relying exclusively on predefined SOPs
[44]	ChatDev	Software development lifecycle	Uses specialized LLM agents guided by chat chains and communicative dehallucination across design, coding, testing, and documentation	Single-agent or fragmented phase-specific development approaches	Demonstrates language-based communication as a unifying mechanism for autonomous software development
[18]	CrewAI	Production-oriented multi-agent workflow	Provides role-based orchestration of agents, crews, flows, tools, memory, and observability for practical agent	Framework-level engineering characterization; no directly comparable peer-	Practical framework for building and deploying role-specialized autonomous agent

	automation	deployment	reviewed benchmark reported in this survey.	teams

4.2 Communication Mechanisms and Interoperability Protocols

Communication is a key component in LLM-based multi-agent systems, as agents need to share information, coordinate actions, maintain shared context, and integrate intermediate results during collaborative execution. However, communication in modern agentic systems takes place at different architectural levels. Agents may communicate through direct message exchange, graph-controlled transitions, or shared-memory mechanisms at the intra-system level. Standardised protocols are designed to enable communication between heterogeneous agents, tools, services, and frameworks at the interoperability level. It is important to make a distinction between those levels since an internal coordination mechanism and an interoperability protocol address different architectural requirements.

4.2.1 Principal Communication Paradigms

Based on the communication patterns identified across the reviewed frameworks, three main paradigms can be distinguished: structured message passing, graph-based communication and control, and shared-state or blackboard communication.

Message passing with structure. In message-oriented architectures, agents communicate by explicitly exchanging structured messages containing information such as sender or role, message content, task context, and tool-related information. AutoGen offers a representative conversation model, where agents communicate by exchanging messages over customizable interaction mechanisms [15]. Explicit message passing allows us to inspect interaction histories and enables conversational coordination, whereas rich agent-to-agent communication may scale context size, token throughput, latency, and coordination overhead with the number of agents or interaction rounds.

Graph communication and control. Graph-based frameworks model agentic workflows as nodes, states, and explicitly defined transitions. Nodes in LangGraph can be agents, tools, or computational operations, and edges and conditional routing are used to determine the next execution step based on the current graph state [13]. Cyclic transitions make it possible for execution to revisit previous nodes, thus enabling iterative reasoning, reflection, replanning, and recovery. Thus, communication and coordination are tightly coupled with the control-flow structure of the workflow, rather than conventional message-passing architectures.

Blackboard or shared-state communication. In shared-state architectures, coordination between agents is done by reading and writing to a common information space instead of direct peer-to-peer messages only. The information shared can include the status of the task, intermediate artifacts, memory, plans, observations, or outputs generated. MetaGPT exhibits a similar artifact-centric collaboration pattern where role-specialized agents generate and consume structured intermediate outputs (e.g., requirements, designs, and implementation artifacts) through various stages of the workflow [45]. Shared-state approaches can eliminate the need for repeated direct communication in some workflows and allow for asynchronous specialization but bring new problems of state consistency, synchronization, access control, and conflicting updates.

Importantly, these paradigms of communication are not mutually exclusive. One multi-agent system might incorporate direct messaging for negotiation, shared state for persistent context, and graph-based routing for execution control. Their suitability depends on variables such as collaboration topology, task complexity, number of agents, persistence needed, communication cost, and degree of centralized workflow control [46].

In attempting to present a comprehensive view on communication in LLM-based multi-agent systems, we divide the interaction architecture into three complementary levels, as illustrated in **Figure 8**. The first layer is the collaboration architecture, which defines how agents are organized in hierarchies, peer-to-peer, or role-based. The second layer is

the internal communication and coordination mechanisms. This includes structured message passing, graph-based control, and shared-state or blackboard communication. The third layer is cross-system interoperability protocols for interaction between heterogeneous agents, tools, services, and frameworks. These layers are impacted by cross-cutting requirements such as security, authentication, state consistency, observability, reliability, governance, scalability, cost-efficiency, oversight by humans, and privacy.

The layered perspective sees organizational topology, internal coordination, and external interoperability as distinct but interacting architectural dimensions. These dimensions are not mutually exclusive. For instance, a role-based system may combine hierarchical orchestration, structured message exchange, and an interoperability protocol. This separation allows for a more systematic comparison of agentic systems and highlights that effective multi-agent collaboration depends not only on agent roles or communication topology but also on the mechanisms used to manage state, control information flow, and support secure interaction across heterogeneous environments.

At the interoperability level, these internal coordination mechanisms should be distinguished from protocols designed for communication across system boundaries. MCP primarily standardizes access to tools, resources, and contextual data, whereas A2A focuses on agent-to-agent capability discovery, task delegation, and message exchange across heterogeneous systems. ACP represents an earlier agent communication effort that subsequently converged with A2A, while ANP explores decentralized agent discovery, addressing, and network-level interaction. These protocols are therefore complementary rather than interchangeable and operate at a different architectural level from framework-specific coordination mechanisms[47].

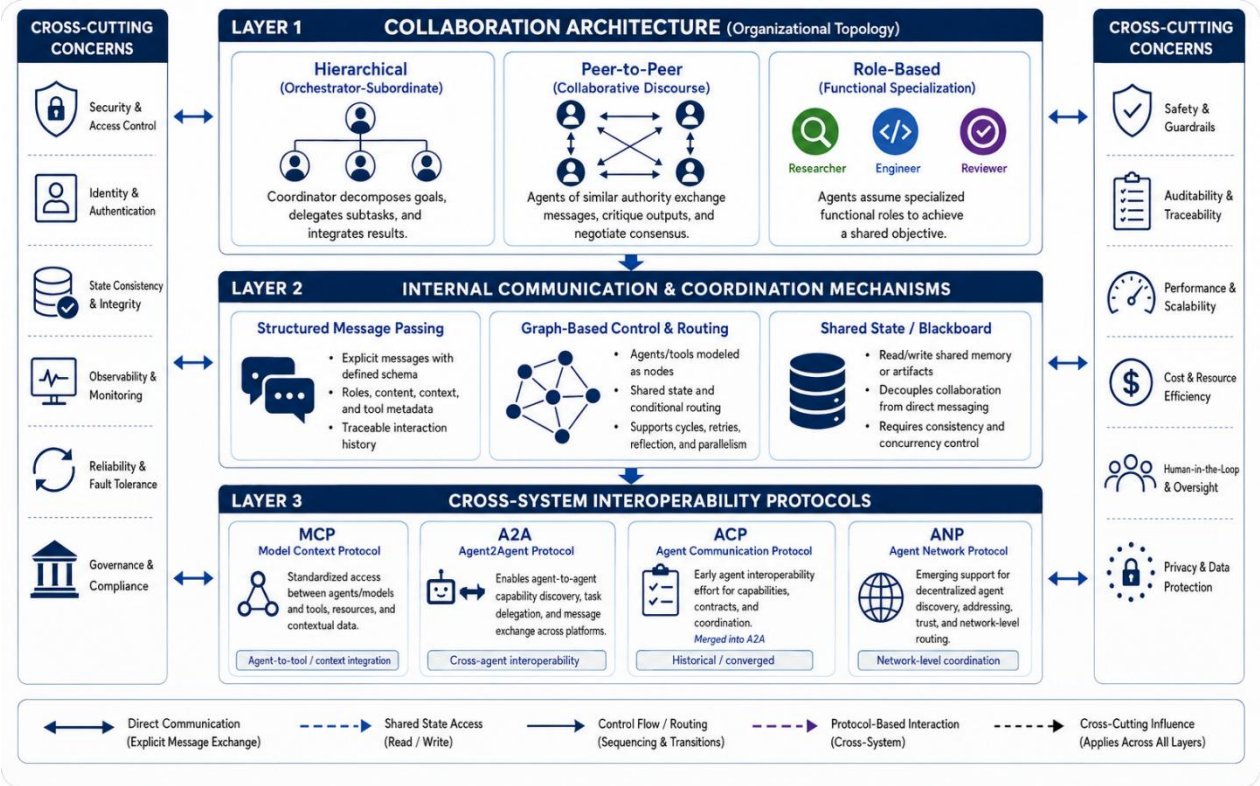


Figure 8: Layered Communication and Interoperability Architecture for LLM-Based Multi-Agent Systems

5. OPEN CHALLENGES AND FUTURE RESEARCH DIRECTIONS

Despite rapid advances in LLM-based agent and multi-agent systems, several technical and methodological challenges remain that limit their reliable deployment. The literature review suggests that these challenges are beyond the capabilities of individual language models and include evaluation standardization, long-horizon reliability, security, interoperability, communication efficiency, governance, and interpretability. To address these limitations,

research needs to consider the agent system as a holistic architecture, rather than evaluating the base LLM in isolation.

A key challenge is the absence of standardized evaluation methodologies that enable reproducible comparison of agent frameworks, underlying models, collaboration architectures, and deployment environments. There are many existing studies, but they use different tasks, metrics, model configurations, tool environments, and stopping conditions, so it is difficult to compare them directly. Future benchmark suites should therefore measure multiple dimensions at once, including accuracy of task completion, reliability of reasoning, recovery from intermediate failures, latency, token consumption, computational cost, accuracy of tool use, and compliance with safety [47].

Long-horizon evaluation is particularly important because failures not obvious during brief interactions can accumulate over long reasoning and execution sequences. Benchmarks such as REALM-Bench highlight the importance of evaluating LLM-based agents and multi-agent systems on real-world dynamic planning and scheduling tasks involving multi-agent coordination, inter-agent dependencies, and changing environmental conditions [48].

Future benchmarking protocols should also incorporate controlled perturbations, paraphrased instructions, tool failures, incomplete observations, and dynamically changing environments to evaluate robustness in conditions that better reflect real-world deployment. LLM-as-a-Judge approaches may offer a scalable supplement to human evaluation for the assessment of complex agent outputs. However, future work should explore judge-model bias, inter-model agreement, sensitivity to prompt formulation, and correlation with human assessment before such methods are used as primary evaluation mechanisms.

Autonomous agents can still accumulate errors with long execution trajectories. Erroneous intermediate reasoning, erroneous tool selection, corrupted state, or erroneous observations can propagate to the following stages and ultimately lead to task failure. Multi-agent architectures introduce an additional risk, namely that wrong information generated by one agent may affect other agents that collaborate with it.

Addressing this requires state-aware fault detection and recovery that goes beyond simple retry mechanisms. Promising directions include intermediate verification checkpoints, confidence-aware routing, rollback mechanisms, redundant agent validation, adaptive replanning, and explicit termination policies. Evaluation should not only measure if an agent eventually completes a task but also how well it detects, isolates, and recovers from intermediate failures.

The attack surface of agent systems expands as they become more autonomous and increasingly connected to external tools, services, shared memory, and other agents. Agents can be exposed to vulnerabilities like prompt injections, malicious tool outputs, agent impersonation, and manipulation of shared context through interaction with untrusted documents, external tools, APIs, shared memory, or other agents[29].

The diversity of agent frameworks is increasing, which causes significant problems with interoperability. Agents built on LangChain/LangGraph, AutoGen, MetaGPT, CrewAI, and other ecosystems may have differing message representation, state management, tool interfaces, capability descriptions, and execution semantics. This fragmentation hinders the development of heterogeneous multi-agent systems in which independently developed agents can cooperate in a reliable way.

Emerging interoperability efforts such as MCP, A2A, and ANP address complementary aspects of heterogeneous agent ecosystems. MCP primarily supports standardized access to tools, resources, and contextual data, whereas A2A focuses on interoperable agent-to-agent communication and task coordination. ACP should be regarded as an earlier interoperability initiative that has subsequently converged with A2A rather than as an independently evolving protocol. Evaluating these protocols under heterogeneous, large-scale agent deployments is a priority, along with issues such as capability discovery, identity management, protocol translation, state synchronization, version compatibility, failure recovery, and safe communication across frameworks. One key direction is to develop protocol-agnostic interoperability layers to mediate between agents that use different internal orchestration frameworks [49].

Future work should explore adaptive communication policies to decide when it is necessary, with which agents, and how much context information should be exchanged. Other promising directions are dynamic agent selection, communication compression, hierarchical routing, context summarization, early termination, and cost-aware

orchestration. Hence, future evaluations should report efficiency in addition to task performance instead of considering accuracy as the only optimization objective.

Interpretability becomes increasingly difficult when multiple interacting agents make the final decision rather than a single invocation of a model. Looking at individual prompts and responses may not be enough to understand how local interactions, critiques, shared-memory updates, and routing decisions resulted in the ultimate outcome [50].

Future research should therefore develop system-level interpretability methods that can reconstruct interaction trajectories, identify influential agents and messages, trace information propagation, and detect the origin of erroneous collective decisions. These approaches might incorporate communication graphs, execution traces, state histories, and causal attribution techniques to provide post-hoc explanations of multi-agent behavior without the need for complete exposure of internal modelling reasoning.

These challenges indicate that the next phase of agentic AI research should move beyond increasing agent autonomy or the number of collaborating agents. There is a need to put more emphasis on measurable reliability, secure interaction, efficient coordination, interoperability, governance, and system-level interpretability. Advances in these directions will be pivotal for making LLM-based multi-agent systems move from experimental orchestration frameworks to reliable architectures that can operate in heterogeneous and real-world settings.

6. CONCLUSIONS

This survey investigates the evolution of agentic artificial intelligence (AI) systems based on large language models (LLMs), from modular single-agent architectures to autonomous and collaborative multi-agent systems. The analysis indicates a shift from composable pipelines and goal-oriented single-agent execution toward stateful graph-based orchestration and collaborative multi-agent systems. It classified representative multi-agent architectures into hierarchical, peer-to-peer, and role-based models, each with different trade-offs in terms of coordination, specialization, flexibility, and communication overhead. The survey also analyzed the main internal communication mechanisms such as structured message passing, graph-based control, and shared-state communication. Interoperability emerged as a distinct architectural requirement for heterogeneous agent ecosystems, requiring a clear distinction between internal coordination mechanisms and cross-system protocols for agent-to-agent and agent-to-tool interaction. While there has been a lot of progress in recent years, current agentic artificial intelligence systems still face challenges around long-horizon reliability, error propagation, security and scalability, evaluation consistency, communication and computational cost, governance and interoperability. The results suggest that building robust LLM-based multi-agent systems will require not only better underlying language models but also more reliable orchestration, standardized evaluation, safe communication, efficient coordination, and effective cross-system interaction mechanisms. Solving these challenges is key to moving agentic AI from experimental systems to reliable, scalable real-world applications.

ACKNOWLEDGEMENTS

During the preparation of this review, the authors used ChatGPT for language editing and textual refinement. The authors reviewed and edited all generated content and took full responsibility for the final manuscript.

FUNDING

This research received no external funding.

DISCLOSURE STATEMENT

The authors have no conflicts of interest regarding the publication of this manuscript.

REFERENCES

[1] V. I. Slyusar, Z. Gomolka, and Y. P. Kondratenko, “General Characteristics of the Large Language Models and Comparative Analysis,” *AI in Education Systems: Successful Cases and Perspectives*, pp. 1–20, Oct. 2025, doi: 10.1201/9788743809258-

1/GENERAL-CHARACTERISTICS-LARGE-LANGUAGE-MODELS-COMPARATIVE-ANALYSIS-SLYUSAR-GOMOLKA-KONDRATENKO.

[2] T. Ma and X. Chen, "Emerging Trends in LLM Agent Development," Jan. 2026, doi: 10.2139/SSRN.6119146.

[3] M. Abou Ali, F. Dornaika, and J. Charafeddine, "Agentic AI: a comprehensive survey of architectures, applications, and future directions," *Artificial Intelligence Review 2025 59:1*, vol. 59, no. 1, pp. 11-, Nov. 2025, doi: 10.1007/S10462-025-11422-4.

[4] Y. Mendoza Juan, "Development of a multi-agent, LLM-driven system to enhance human-machine interaction: integrating DSPy with modular agentic strategies and logical reasoning layers for the autonomous generation of smart contracts," Jul. 15, 2024, *Universitat Politècnica de Catalunya*. Accessed: May 18, 2026. [Online]. Available: https://hdl.handle.net/2117/414255

[5] E. Davis, "Building Custom AI Workflows Using LangChain Tools," *ThinkTide Global Research Journal*, vol. 5, no. 4, pp. 54–62, Nov. 2024, Accessed: May 18, 2026. [Online]. Available: http://thinktidejournal.com/index.php/TGRJ/article/view/53

[6] W. Cugunov, Unlocking the Power of Auto-GPT and Its Plugins: Implement, Customize, and Optimize Auto-GPT for Building Robust AI Applications. Packt Publishing, Sep. 13, 2024.[7] E. Oliveira, K. Fischer, and O. Stepankova, "Multi-agent systems: which research for which applications," *Rob. Auton. Syst.*, vol. 27, no. 1–2, pp. 91–106, Apr. 1999, doi: 10.1016/S0921-8890(98)00085-2.

[8] U. Eswaran, V. Eswaran, V. Eswaran, and K. Murali, "Agentic AI in Space Exploration: Autonomous Decision-Making Beyond Earth," pp. 111–126, 2025, doi: 10.1007/978-3-031-89424-4_8.

[9] S. H. Shaikh, "LLM-Based Multi-agent Systems: Frameworks, Evaluation, Open Challenges, and Research Frontiers," *Communications in Computer and Information Science*, vol. 2827 CCIS, pp. 149–170, 2026, doi: 10.1007/978-3-032-15632-7_9/SAVE-RESEARCH.

[10] T. Guo *et al.*, "Large Language Model based Multi-Agents: A Survey of Progress and Challenges," *IJCAI International Joint Conference on Artificial Intelligence*, pp. 8048–8057, Jan. 2024, Accessed: Aug. 27, 2026. [Online]. Available: https://arxiv.org/pdf/2402.01680

[11] J. Li *et al.*, "Fundamental Capabilities and Applications of Large Language Models: A Survey," *ACM Comput. Surv.*, vol. 58, no. 2, Sep. 2025, doi: 10.1145/3735632.

[12] J. Schneider, "Agentic AI Prompt Engineering: Advancing Generative AI Patterns Conceptually," pp. 1–8, Jan. 2026, doi: 10.1109/ICITDA68167.2025.11332344.

[13] SapkotaRanjan, ShresthaRashik, RijalMadhav, and KarkeeManoj, "LangChain vs. LangGraph vs. LangSmith: Taxonomies of Agentic AI Toolchains for End-to-End Orchestration," Sep. 2025, doi: 10.36227/TECHRXIV.175695645.52670060/V1.

[14] M. Fırat and S. Kuleli, "What if GPT4 Became Autonomous: The Auto-GPT Project and Use Cases," *Journal of Emerging Computer Technologies*, vol. 3, no. 1, pp. 1–6, Mar. 2024, doi: 10.57020/JECT.1297961.

[15] Q. Wu *et al.*, "AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation," Aug. 2023, Accessed: Aug. 10, 2026. [Online]. Available: https://arxiv.org/pdf/2308.08155

[16] S. Hong et al., "MetaGPT: Meta Programming for a Multi-Agent Collaborative Framework," in Proc. 12th Int. Conf. Learn. Representations (ICLR), 2024.

[17] Guohao Li∗ HasanAbedAlKaderHammoud* HaniItani* DmitriiKhizbullin, "CAMEL:Communicative Agents for 'Mind' Exploration of Large Language Model Society", Accessed: May 22, 2026. [Online]. Available: https://arxiv.org/pdf/2303.17760

[18] G. I. Rashed, A. O. M. Bahageel, H. A. I. Gony, A. Badjan, and H. I. Shaheen, "AI-Driven Multi-Agent Demand Response Framework for Residential Load Optimization Using CrewAi," *2025 IEEE 20th Conference on Industrial Electronics and Applications, ICIEA 2025*, 2025, doi: 10.1109/ICIEA65512.2025.11149013.

[19] Y. Lei *et al.*, "Large Language Model Agents: A Comprehensive Survey on Architectures, Capabilities, and Applications," Dec. 2025, doi: 10.20944/PREPRINTS202512.2119.V1.

[20] A. A. Jafari, C. Ozcinar, and G. Anbarjafari, "A Lightweight Modular Framework for Constructing Autonomous Agents Driven by Large Language Models: Design, Implementation, and Applications in AgentForge," Jan. 2026, Accessed: May 20, 2026. [Online]. Available: https://arxiv.org/pdf/2601.13383

[21] O. Topsakal and T. C. Akinci, "Creating Large Language Model Applications Utilizing LangChain: A Primer on Developing LLM Apps Fast," *International Conference on Applied Engineering and Natural Sciences*, vol. 1, no. 1, pp. 1050–1056, Jul. 2023, doi: 10.59287/ICAENS.1127.

[22] L. Kuligin, J. Zaldívar, and M. Tschochohei, Generative AI on Google Cloud with LangChain: Design Scalable Generative AI Solutions with Python, LangChain, and Vertex AI on Google Cloud. Packt Publishing, 2024.

[23] D. Tolis, J. Rytilahti, O. Weerakoon, P. Puhtila, E. Kaila, and T. Mäkilä, "Democratizing AI Development: A Feature-Based Categorization of API Platforms, Development Frameworks, LCNC and AIaaS Platforms for LLM-Based Applications," *Journal of Software Engineering Research and Development*, vol. 14, no. 1, pp. 62–87, May 2026, doi: 10.5753/JSERD.2026.5969.

[24] M. Isham, A. Raj, K. A. Shetty, and K. Adamsab, "Branch LLM: A Local Branching Memory Interface for Exploratory Conversations," pp. 572–577, Dec. 2025, doi: 10.1109/INSPIRE67328.2025.11300653.

[25] C. Z. Chang Yang†, Z. Y. Zijin Hong, and J. S. Ninghao Liu, "Graph-based Agent Memory: Taxonomy, Techniques, and Applications", Accessed: May 22, 2026. [Online]. Available: https://arxiv.org/pdf/2602.05665

[26] O. Elizarov, "Architecture of Applications Powered by Large Language Models," 2024, Accessed: May 20, 2026. [Online]. Available: http://www.theseus.fi/handle/10024/868581

[27] Z. Lei, "LangGraph vs LangChain: Which Framework Is Best for AI Agents?," *MetaVision Journal of Multidisciplinary Studies (MVJMS)*, vol. 1, no. 4, pp. 16–26, Nov. 2024, Accessed: May 20, 2026. [Online]. Available: https://sinojournals.com/index.php/mvjms/article/view/10

[28] L. Reddy Alva, B. Pandey, L. R. Alva, and B. Pandey, "Agentic AI systems in the age of generative models: architectures, cloud scalability, and real-world applications," *Artificial Intelligence Review 2025 59:3*, vol. 59, no. 3, pp. 88-, Jan. 2026, doi: 10.1007/S10462-025-11458-6.

[29] M. Leo, F. Tan, T. Miao, and G. Anand, "From threat to trust: assessing security risks of agentic AI systems," *International Journal of Information Security 2026 25:1*, vol. 25, no. 1, pp. 23-, Jan. 2026, doi: 10.1007/S10207-025-01185-Y.

[30] D. S *et al.*, "AGENTIC AI: AUTONOMOUS INTELLIGENT AGENTS USING LARGE LANGUAGE MODELS AND CLOUD COMPUTING FOR REAL-WORLD TASK AUTOMATION," *JETIR*, vol. 13, no. 5, pp. d5–d9, 2026, Accessed: May 20, 2026. [Online]. Available: https://www.jetir.org/view?paper=JETIR2605302

[31] L. Lin and Y. Liu, "Multimodal Large Models Toward AGI," *Multimodal Large Models*, pp. 263–336, 2026, doi: 10.1007/978-981-95-4929-0_5.

[32] "GitHub - Significant-Gravitas/AutoGPT: AutoGPT is the vision of accessible AI for everyone, to use and to build on. Our mission is to provide the tools, so that you can focus on what matters. · GitHub." Accessed: Aug. 27, 2026. [Online]. Available: https://github.com/Significant-Gravitas/AutoGPT?utm_source=chatgpt.com

[33] A. Bandi, B. Kongari, R. Naguru, S. Pasnoor, and S. V. Vilipala, "The Rise of Agentic AI: A Review of Definitions, Frameworks, Architectures, Applications, Evaluation Metrics, and Challenges," *Future Internet 2025, Vol. 17, Page 404*, vol. 17, no. 9, p. 404, Sep. 2025, doi: 10.3390/FI17090404.

[34] Q. Meng *et al.*, "Agent Harness for Large Language Model Agents: A Survey," Apr. 2026, doi: 10.20944/PREPRINTS202604.0428.V3.

[35] S. Brady, "Springdrift: An Auditable Persistent Runtime for LLM Agents with Case-Based Memory, Normative Safety, and Ambient Self-Perception," Apr. 2026, Accessed: May 21, 2026. [Online]. Available: https://arxiv.org/pdf/2604.04660

[36] L. Wang *et al.*, "A survey on large language model based autonomous agents," *Frontiers of Computer Science 2024 18:6*, vol. 18, no. 6, pp. 186345-, Mar. 2024, doi: 10.1007/S11704-024-40231-1.

[37] B. Zhang *et al.*, "Breaking Agents: Compromising Autonomous LLM Agents Through Malfunction Amplification," *EMNLP 2025 - 2025 Conference on Empirical Methods in Natural Language Processing, Proceedings of the Conference*, pp. 34964–34976, 2025, doi: 10.18653/V1/2025.EMNLP-MAIN.1771.

[38] D. Goyal and A. Gautam, "Introduction to LangChain Framework," *Textual Intelligence Large Language Models and Their Real-World Applications.*, pp. 253–285, Jan. 2025, doi: 10.1002/9781394287499.CH11.

[39] Z. Feng *et al.*, "Multi-agent embodied AI: advances and future directions," *Science China Information Sciences 2026 69:5*, vol. 69, no. 5, pp. 151202-, Mar. 2026, doi: 10.1007/S11432-025-4820-4.

[40] Z. Deng *et al.*, "Multi-Agent Systems and Social Intelligence: A Comprehensive Survey on Collaboration, Communication, and Emergent Behaviors," Dec. 2025, doi: 10.2139/SSRN.5957495.

[41] Z. Zhang, Y. Zhang, D. Zeng, K. Liu, and J. Zhao, "Beware of the Woozle Effect: Exploring and Mitigating Hallucination Propagation in Multi-Agent Debate," *IEEE Trans. Audio Speech Lang. Process.*, 2026, doi: 10.1109/TASLPRO.2026.3675803.

[42] S. Sarkar, "Enterprise Usage of AI Agents: A Comprehensive Survey on Applications, Frameworks, and Future Productivity Impact," Mar. 2026, doi: 10.2139/SSRN.6405998.

[43] Q. Wang *et al.*, "MegaAgent: A Large-Scale Autonomous LLM-based Multi-Agent System Without Predefined SOPs," pp. 4998–5036, Accessed: May 23, 2026. [Online]. Available: https://github.com/

[44] C. Qian *et al.*, "ChatDev: Communicative Agents for Software Development," vol. 1, pp. 15174–15186, Accessed: May 23, 2026. [Online]. Available: https://github.com/OpenBMB/ChatDev.

[45] N. Krishnan, "Advancing Multi-Agent Systems Through Model Context Protocol: Architecture, Implementation, and Applications," 2025.

[46] C. Li *et al.*, "ACPS: Agent Collaboration Protocols for the Internet of Agents," *Proceedings of 2025 9th IEEE International Conference on Network Intelligence and Digital Content, IC-NIDC 2025*, pp. 342–346, 2025, doi: 10.1109/IC-NIDC67200.2025.11390349.

[47] T. Kehkashan *et al.*, "From benchmarks to deployment: a comprehensive review of agentic AI evaluation," *Artificial Intelligence Review 2026 59:8*, vol. 59, no. 8, pp. 167-, Apr. 2026, doi: 10.1007/S10462-026-11571-0.

[48] L. Geng and E. Y. Chang, "REALM-Bench: A Benchmark for Evaluating Multi-Agent Systems on Real-world, Dynamic Planning and Scheduling Tasks," *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, vol. 1-A, pp. 2652–2662, Apr. 2026, doi: 10.1145/3770854.3785692;SUBPAGE:STRING:BASIC.

[49] Y. Wang *et al.*, "Internet of Agents: Fundamentals, Applications, and Challenges," *IEEE Trans. Cogn. Commun. Netw.*, vol. 12, pp. 4476–4501, 2026, doi: 10.1109/TCCN.2025.3623369.

[50] K. Jiang *et al.*, “KoMA: Knowledge-Driven Multi-Agent Framework for Autonomous Driving with Large Language Models,” *IEEE Transactions on Intelligent Vehicles*, vol. 10, no. 10, pp. 4655–4668, 2025, doi: 10.1109/TIV.2024.3488793.