

Open-Source AI Agent Frameworks: A Comparative Performance Analysis

Jyoti Amit Kumar Dhamecha^{1*}, Tulsi Ashokbhai Javandhra²

Abstract

The rapid growth of agentic AI has led to a diverse ecosystem of open-source frameworks for developing LLM-based agents. These frameworks differ in their approaches to state management, multi-agent coordination, tool orchestration, memory, and workflow execution. This paper presents a comparative analysis of prominent open-source AI agent frameworks, including LangChain, LangGraph, CrewAI, AutoGen, Semantic Kernel, and related systems. The study synthesizes a common architectural model to identify the core components and design patterns shared across these frameworks while highlighting their architectural differences. It further examines reported performance characteristics using findings from 2026 benchmark studies, with particular attention to latency, token efficiency, execution overhead, and orchestration costs across standardized tasks. The analysis discusses how architectural choices influence framework performance, scalability, flexibility, and suitability for different agentic AI applications. In addition, the paper examines practical design trade-offs involving centralized versus distributed orchestration, state persistence, agent communication, and tool integration. Finally, it identifies key challenges that remain in the development and evaluation of agentic AI frameworks, including inconsistencies in benchmarking methodologies, limited cross-framework interoperability, reproducibility concerns, production governance, and the absence of widely accepted evaluation standards. The study provides a structured foundation for understanding the evolving landscape of open-source agent frameworks and identifies directions for future research toward interoperable, efficient, and reliable agentic AI systems.

Keywords: AI agents, open-source software, LangChain, AutoGen, CrewAI, multi-agent systems, benchmarking.

INTRODUCTION

Large language models (LLMs) have evolved from systems primarily designed for single-turn interactions into components capable of planning, iterative reasoning, tool use, and interaction with external environments. This development has contributed to the emergence of AI agent frameworks, which provide software abstractions for tool invocation, state management, workflow execution, memory, and multi-agent coordination [1, 2]. Unlike conventional LLM applications based mainly on prompt-response interactions, agentic systems require mechanisms for maintaining intermediate states, executing iterative reasoning loops, processing tool outputs, handling failures, and coordinating specialized agents to accomplish complex tasks [1, 3].

*Author For Correspondence

Jyoti AmitKumar Dhamecha
E-mail: jyotidhamecha69@gmail.com

¹Assistant Professor, Department of Computer Science and Engineering, Sardar Patel College of Administration and Management (SPEC Campus), Gujarat, India

²Student, Department of Computer Science and Engineering, Sardar Patel College of Administration and Management (SPEC Campus), Gujarat, India

Received Date: August 29, 2026

Accepted Date: September 26, 2026

Published Date: September 28, 2026

Citation: Jyoti Amit Kumar Dhamecha, Tulsi Ashokbhai Javandhra. Open-Source AI Agent Frameworks: A Comparative Performance Analysis. Journal of Open Source Developments. 2026; 13(2): 22–33p.

The open-source ecosystem for agent development has expanded rapidly, with

frameworks adopting different architectural approaches. LangChain provides reusable components and agent abstractions, while LangGraph introduces graph-based workflow orchestration with support for stateful and long-running agent applications [4, 5]. CrewAI adopts an agent-and-task model in which specialized agents collaborate through Crews and Flows [6]. AutoGen provides a multi-agent conversation framework that enables configurable agents to communicate and collaborate to solve complex tasks [7]. Semantic Kernel provides agent abstractions and multiple orchestration patterns, including sequential, concurrent, handoff, group-chat, and magnetic orchestration [8]. Recent surveys have highlighted that these frameworks differ in their approaches to memory, communication, planning, orchestration, and interoperability [9].

As agent frameworks have matured, evaluation has increasingly moved beyond qualitative comparisons toward controlled empirical benchmarking. A 2026 unified benchmark study reported substantial differences in latency, planning accuracy, and coordination success resulting from framework-level architectural choices, demonstrating that orchestration design can significantly influence system performance [10]. A separate large-scale study of open-source multi-agent systems analyzed more than 42,000 commits and 4,700 resolved issues across eight systems, identifying recurring concerns related to bugs, infrastructure, coordination, testing, documentation, and long-term maintenance [11]. These findings indicate that framework selection involves considerations beyond task accuracy, including execution overhead, maintainability, reliability, and development practices.

However, benchmark results must be interpreted carefully because performance depends on the underlying LLM, task characteristics, number of agents, tool calls, execution environment, and evaluation methodology [10]. Consequently, numerical results from different studies cannot always be treated as directly comparable.

RELATED WORK

Research on LLM-based agents has expanded rapidly as language models have acquired stronger reasoning, planning, tool use, and interaction capabilities. Early work established the foundation for agentic systems by combining language model reasoning with external actions. Yao et al. introduced ReAct, which integrates reasoning traces with action execution and demonstrated how an LLM can interact with external environments through iterative reasoning and tool use [1]. This approach established an important foundation for subsequent agent architectures, in which reasoning, action selection, observation, and subsequent decision-making form an iterative execution loop.

Several surveys have examined the broader development of autonomous LLM agents. Xi et al. analyzed the emerging capabilities, architectures, applications, and challenges of LLM-based agents, emphasizing components such as planning, memory, tool use, and interaction [3]. Wang et al. provided a comprehensive survey of autonomous agents and categorized major components and application scenarios, highlighting the increasing importance of planning, memory, perception, action, and multi-agent collaboration [2]. These studies provide the conceptual foundation for understanding agent frameworks; however, they primarily focus on agent architectures and capabilities rather than detailed empirical comparisons of implementation frameworks.

The development of open-source frameworks has introduced different approaches to agent orchestration. LangChain provides reusable abstractions for connecting LLMs with prompts, tools, retrievers, and agents, while LangGraph extends these capabilities through graph-based stateful workflows designed for more controllable and persistent agent execution [4, 5]. CrewAI adopts a role-based multi-agent architecture in which agents are assigned specialized responsibilities and coordinated through Crews and Flows [6]. AutoGen focuses on multi-agent conversations, allowing configurable agents to communicate with one another and with humans or external tools [7]. Semantic Kernel provides another approach based on modular agent components and orchestration patterns, including sequential, concurrent, handoff, group-chat, and magnetic workflows [8, 12].

Recent research has increasingly examined these frameworks from a comparative perspective. Derouiche et al. reviewed agentic AI frameworks, protocols, architectural approaches, and interoperability challenges, identifying the need for standardized mechanisms for communication and coordination among heterogeneous agents [9]. Such interoperability is particularly important as organizations increasingly combine different agent architectures and external tools within larger workflows.

Empirical benchmarking has also become an important research direction. Orogat et al. presented a unified benchmark for evaluating multi-agent LLM frameworks and examined factors including task performance, execution characteristics, and framework-level overhead [10]. Ahmad and Marathe compared LangGraph, CrewAI, and AutoGen using controlled experimental tasks and reported differences in execution and coordination characteristics among the frameworks [13]. These studies demonstrate the importance of evaluating frameworks using measurable performance indicators rather than relying exclusively on feature comparisons.

Beyond performance, recent research has examined the engineering challenges associated with developing and maintaining multi-agent systems. Liu et al. conducted a large-scale analysis of multi-agent AI systems and identified recurring issues involving coordination, infrastructure, testing, documentation, and system maintenance [11]. These findings suggest that framework evaluation should include software engineering and operational considerations in addition to task-level performance.

Research has also explored methods for improving the efficiency of agentic workflows. Wang et al. examined optimization techniques for LLM-based agents, covering approaches related to planning, execution, memory, and resource utilization [14]. Dennis et al. investigated methods for compiling agentic workflows into model parameters to reduce repeated inference and execution costs [15]. Such work indicates that future agent frameworks will need to address not only functional capabilities but also computational efficiency and deployment costs.

Overall, the existing literature provides substantial coverage of individual agent architectures, framework implementations, and emerging benchmarking methodologies. However, there remains a need for a structured comparison that connects framework architecture with measurable performance characteristics, such as latency, token consumption, and orchestration overhead, while also considering governance, maintainability, and interoperability. This study addresses this gap by synthesizing architectural and benchmarking evidence across major open-source agent frameworks and organizing the comparison around common architectural components and measurable evaluation criteria.

COMMON ARCHITECTURAL MODEL OF AI AGENT FRAMEWORKS

The major open-source AI agent frameworks considered in this study differ in implementation details, programming abstractions, and orchestration mechanisms. However, their core functionality can be represented through a common architectural model consisting of several interconnected components: input and task interpretation, planning and reasoning, memory and state management, tool orchestration, execution and observation, and multi-agent coordination. (Figure 1) This common model provides a consistent basis for comparing frameworks such as LangChain/LangGraph, CrewAI, AutoGen, and Semantic Kernel [4–9].

Task and Input Layer

The task and input layer receives instructions from the user or an external application. The input may range from a simple question to a complex objective requiring multiple reasoning and execution steps. The agent interprets the objective and determines the actions required to achieve the requested outcome. Unlike conventional prompt-based applications, agent frameworks can maintain task context across multiple execution cycles [1, 2].

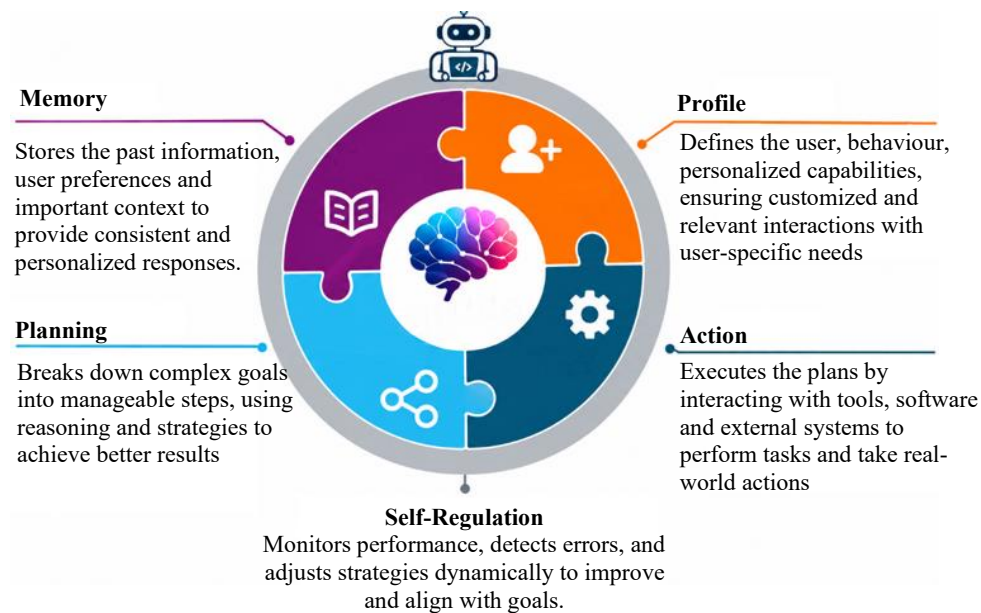


Figure 1. Common architectural model of an LLM-based AI agent framework.

Planning and Reasoning Layer

The planning and reasoning layer determines how the agent decomposes a task into smaller steps and selects appropriate actions. ReAct demonstrated an important approach in which reasoning and external actions are interleaved within an iterative process [1]. Modern frameworks extend this concept through structured workflows, graph-based execution, task decomposition, and agent-to-agent coordination [5, 7].

The planning process may be explicit or implicit. In an explicit approach, the framework maintains a structured representation of the workflow or task state. In an implicit approach, planning decisions are primarily generated by the underlying LLM. These differences can influence execution control, observability, and computational overhead [9, 10].

Memory and State Management

Memory and state management enable an agent to preserve information across multiple execution steps. The state may include the original user request, intermediate results, tool outputs, messages exchanged between agents, and execution status. Persistent state is particularly important for long-running workflows because the agent may need to resume execution after an interruption or incorporate information generated during previous steps [2, 5].

Frameworks differ considerably in how they represent and persist in the state. Graph-oriented architectures can explicitly define state transitions, whereas multi-agent conversation frameworks may maintain state through message histories. These differences affect scalability, debugging, reproducibility, and resource consumption [5, 7, 9].

Tool Orchestration Layer

Tool orchestration connects the LLM to external capabilities, such as databases, APIs, search systems, code interpreters, file systems, and other software services. The agent selects an appropriate tool, generates the required parameters, receives the tool output, and incorporates the result into the subsequent execution cycle [1, 4].

The number and complexity of tool calls can directly affect execution latency and token consumption. Consequently, tool orchestration represents an important factor when evaluating agent frameworks and comparing their execution overhead [10, 13].

Execution and Observation Layer

After selecting an action or tool, the framework executes the operation and returns an observation to the agent. The observation may contain a successful result, an error, structured data, or information requiring additional reasoning. The agent can then determine whether the task has been completed or whether another execution cycle is required [1, 3].

This iterative process can be represented as:

Input → Reasoning → Action → Tool Execution → Observation → Updated State → Reasoning

The cycle continues until the termination condition is satisfied. Such iterative execution distinguishes agentic systems from conventional single-pass LLM applications [1, 2].

Multi-Agent Coordination Layer

Complex tasks can be divided among multiple specialized agents. Each agent can perform a particular role, such as planning, research, coding, verification, or data analysis. Frameworks, including CrewAI and AutoGen, explicitly support multi-agent interaction and coordination [6, 7].

Multi-agent execution can improve task decomposition and specialization; however, it may also introduce additional communication, synchronization, and orchestration overhead. Consequently, increasing the number of agents does not necessarily result in proportional improvements in performance. Recent benchmarking research has considered coordination cost and execution efficiency when evaluating multi-agent frameworks [10, 11, 13].

Governance and Monitoring Layer

Production-oriented agent systems also require monitoring, logging, access control, error handling, human intervention, and evaluation mechanisms. These capabilities are increasingly important because agents can independently invoke tools and perform multiple actions. Recent studies have identified testing, infrastructure, documentation, and maintenance as important challenges in multi-agent system development [11].

COMPARATIVE PERFORMANCE AND BENCHMARK ANALYSIS

The performance of an AI agent framework depends not only on the underlying large language model but also on the framework's orchestration strategy, state management mechanism, number of tool calls, communication between agents, and execution environment. Therefore, framework benchmarking requires multiple performance indicators rather than a single measure of task accuracy [9–11, 13].

Evaluation Metrics

For this study, four primary performance dimensions were considered: latency, token efficiency, orchestration overhead, and task execution performance.

Latency represents the time required to complete an agent task. It can be influenced by model inference time, network communication, tool execution, state updates, and coordination between multiple agents. Frameworks that introduce additional orchestration steps may increase end-to-end execution time, even when the underlying LLM remains unchanged [10, 13].

Token efficiency refers to the amount of input and output tokens consumed to complete a task. Agentic workflows may generate additional tokens because of planning steps, intermediate reasoning, tool descriptions, observations, and inter-agent messages. Consequently, a framework that successfully completes a task with fewer model interactions may reduce computational and operational costs [10, 14].

Orchestration overhead represents the additional processing introduced by the framework for managing agent execution, state transitions, tool calls, message exchange, and workflow control. This overhead becomes particularly relevant in multi-agent systems where several agents repeatedly communicate before reaching a final result [9–11, 13].

Task execution performance measures whether the framework successfully completes the assigned task. Depending on the benchmark, this may include task success rate, accuracy, completion time, or other task-specific measures [10, 13].

Framework-Level Performance Considerations

Recent benchmark studies indicate that agent frameworks can exhibit different performance characteristics even when evaluated using similar underlying language models [10, 3]. These differences are primarily associated with architectural design rather than LLM alone. For example, graph-based execution can provide explicit control over state transitions and workflow paths, whereas conversational multi-agent architectures may rely more heavily on message-based coordination [5, 7].

LangGraph emphasizes explicit stateful workflow execution and therefore provides mechanisms for controlling complex agent loops and persistent execution states [5]. CrewAI organizes agents according to roles, tasks, and collaborative workflows, introducing a coordination layer between specialized agents [6]. AutoGen emphasizes agent-to-agent communication through conversational interactions, which can result in additional messages when several agents participate in a task [7]. Semantic Kernel provides multiple orchestration patterns, allowing developers to select different execution strategies according to application requirements [8, 12].

These architectural differences make it inappropriate to interpret framework performance using a single universal ranking. Instead, performance should be examined according to the characteristics of the workload and the selected orchestration pattern.

Latency and Execution Overhead

Latency is particularly important for applications requiring interactive or near-real-time responses. An agent may require several sequential operations before producing a final response. Each additional LLM invocation, tool call, state transition, or inter-agent message can contribute to the overall execution time [10].

Multi-agent workflows can introduce additional coordination overhead because agents may need to exchange intermediate results. This overhead becomes more significant when the workflow contains sequential dependencies, repeated communication, or unnecessary agent interactions [10–12]. Conversely, parallel execution can reduce the elapsed time for independent tasks, although it may increase resource consumption and coordination complexity.

Token Efficiency

Token consumption is another important consideration because agentic systems frequently perform multiple LLM calls. A single user request can generate planning instructions, tool selection messages, tool results, intermediate agent communications, and final responses.

Frameworks that maintain extensive conversation histories or repeatedly pass intermediate states to agents may consume more tokens. Structured state management can potentially reduce unnecessary context transmission, although the actual effect depends on the implementation and workload [5, 10, 14].

Token efficiency should therefore be evaluated using a consistent measurement procedure. A suitable benchmark should report at least:

- Total input tokens
- Total output tokens

Table 1 summarizes the recommended benchmark evaluation criteria for assessing AI agent frameworks, including token consumption and other execution-related metrics.

Comparative Interpretation of Benchmarks

The available 2026 studies provide useful evidence that framework architecture affects execution characteristics [10–11, 13]. However, these studies should not be interpreted as a universal ranking of frameworks. Different benchmarks may use different LLMs, prompts, datasets, task complexities, hardware configurations, and stopping conditions.

COMPARATIVE ANALYSIS OF OPEN-SOURCE AI AGENT FRAMEWORKS

The frameworks examined in this study provide overlapping capabilities but differ in their architectural abstractions, state management mechanisms, orchestration strategies, and approaches to multi-agent collaboration (Table 2). To enable a consistent comparison, the frameworks are evaluated using the common architectural model presented in Section III and the performance dimensions discussed in Section IV [4–15].

Framework Comparison

The frameworks differ in architecture, memory handling, tool integration, multi-agent coordination, and workflow control (Table 2).

Architectural Differences

LangChain provides a broad set of abstractions for constructing LLM applications and agents. Its component-oriented architecture allows developers to combine models, prompts, tools, retrievers, and other application components [4]. LangGraph builds on this ecosystem by representing agent execution as a graph containing nodes and state transitions. This approach provides more explicit control over execution paths, persistence, and long-running workflows [5].

Table 1. Recommended benchmark evaluation criteria for AI agent frameworks.

Metric	Measurement	Relevance
Latency	Total task execution time	Measures responsiveness
Token Efficiency	Input + output tokens	Indicates computational and usage efficiency
LLM Calls	Number of model invocations	Measures reasoning/execution cycles
Tool Calls	Number of external operations	Indicates tool use complexity
Orchestration Overhead	Framework processing and coordination time	Measures framework-level cost

Source: Compiled by the authors based on the evaluation dimensions discussed in [9–11, 13].

Table 2. Comparison of major open-source AI agent frameworks.

Framework	Primary architectural approach	State/Memory management	Tool orchestration	Multi-agent support	Workflow control	Key consideration
LangChain	Component-based agent framework	Memory and application-managed state	Tools, APIs, retrievers, and external services	Supported	Agent-based workflows	Broad ecosystem and reusable components [4]
LangGraph	Graph-based stateful orchestration	Explicit persistent state and checkpoints	Integrated tool nodes and workflow edges	Supported	Graph-based, conditional and cyclic workflows	Suitable for controllable and long-running workflows [5]
CrewAI	Role-based agent collaboration	Task and workflow context	Tools assigned to individual agents	Strong multi-agent orientation	Crews and Flows	Emphasizes role-based collaboration [6]

Source: Compiled by the authors from framework documentation and related research [4–9, 12].

Table 3. Performance dimensions and expected framework-level influences.

Performance dimension	Factors influencing performance	Framework consideration
Latency	LLM calls, tool calls, sequential steps, agent communication	Workflow design and orchestration depth
Token Consumption	Prompt size, conversation history, intermediate reasoning, agent messages	State and context-management strategy
Orchestration Overhead	State transitions, message routing, coordination	Complexity of execution architecture
Task Success	Planning, tool selection, coordination, error handling	Agent and workflow design

Source: Synthesized from [9–11, 14].

CrewAI uses comparatively high-level abstraction based on agents, tasks, Crews, and Flows. Agents can be assigned specific roles and responsibilities, allowing complex objectives to be divided among specialized components [6]. This design makes multi-agent collaboration a central part of the framework architecture.

AutoGen approaches multi-agent coordination primarily through agent communication and conversations. Agents can be configured with different capabilities and interact with other agents, humans, and tools [7]. This architecture supports flexible collaboration but may introduce additional communication overhead when tasks require multiple interaction cycles.

Semantic Kernel provides a modular approach in which agents and tools can be combined with several orchestration strategies [8, 12]. Its orchestration capabilities allow developers to select different coordination patterns depending on whether tasks require sequential execution, parallel processing, agent handoff, group interaction, or more complex coordination.

Comparison of Performance Considerations

The architectural differences described above can influence measurable performance characteristics (Table 3). However, framework architecture alone cannot determine performance because the actual result depends on the LLM, prompt design, task complexity, tool configuration, number of agents, and execution environment [10–11, 13].

Interpretation of Benchmark Results

Recent benchmark studies demonstrate that framework performance varies according to the task and experimental configuration [10, 13]. Consequently, the results should be interpreted as workload-specific observations rather than a universal ranking of frameworks.

For example, a framework that provides explicit state transitions and extensive workflow control may introduce additional orchestration operations but can offer greater control over complex execution. Conversely, a simpler agent architecture may require fewer orchestration steps for straightforward tasks but may provide less explicit control for complex, persistent workflows [5, 9].

Multi-agent frameworks introduce another trade-off. Dividing a complex task among specialized agents can provide modularity and specialization; however, communication between agents can increase token consumption and execution time [7, 10]. Therefore, the number of agents should be treated as an experimental variable rather than automatically interpreted as an indicator of system capability.

Framework Selection Considerations

The comparison indicates that framework selection should be based on application requirements rather than a single performance metric. Applications requiring explicit state transitions and controllable long-running workflows may emphasize graph-based orchestration capabilities [5]. Applications centered on role-based collaboration may use agent-team abstractions [6], while

applications requiring flexible conversational coordination may use multi-agent communication architectures [7]. Applications requiring multiple standardized orchestration patterns may benefit from the orchestration mechanisms provided by Semantic Kernel [8, 12].

OPEN CHALLENGES AND FUTURE RESEARCH DIRECTIONS

Although open-source AI agent frameworks have progressed rapidly, several technical and methodological challenges remain.

Benchmarking Standardization

One of the major challenges is the absence of a universally accepted benchmarking methodology for AI agent frameworks. Existing studies may use different language models, prompts, datasets, hardware configurations, task complexities, numbers of agents, and tool environments [9–11, 13]. Consequently, reported latency, token consumption, and task success measurements may not be directly comparable.

Reproducibility of Framework Comparisons

Future comparative studies should record framework versions, model versions, system configurations, prompts, tool definitions, hardware specifications, random seeds where applicable, and complete experimental procedures. Publishing benchmark scripts and configuration files would further improve reproducibility and enable independent researchers to validate reported findings.

Token and Orchestration Cost

Agentic workflows can require substantially more model interactions than conventional single-response applications. Planning, tool selection, observations, state updates, and agent-to-agent communication can each generate additional processing and token consumption [10, 14].

Multi-Agent Coordination Overhead

Multi-agent architecture provides opportunities for specialization, task decomposition, and collaborative problem solving. However, communication among agents introduces additional messages, synchronization requirements, and intermediate contexts [7, 10, 11].

Production Governance and Human Oversight

Research prototypes and production systems have different operational requirements. A production-oriented agent system may require authentication, authorization, logging, monitoring, error recovery, human approval mechanisms, and controlled access to external tools.

As agents become capable of performing actions autonomously, governance mechanisms become increasingly important. Human-in-the-loop mechanisms can be incorporated for sensitive actions, while monitoring systems can record agent decisions, tool calls, state transitions, and execution outcomes. Recent research on multi-agent system development has also identified infrastructure, testing, documentation, and maintenance as important engineering concerns [11].

Security and Tool Use Risks

Future frameworks should therefore provide stronger mechanisms for permission management, tool isolation, input validation, execution limits, and audit logging. Security evaluation should be incorporated into agent benchmarks rather than treated solely as an application-level concern.

Interoperability Between Frameworks

The growing number of agent frameworks creates a heterogeneous development environment. Applications may use one framework for workflow orchestration, another for specialized agents, and external systems for tools, memory, or monitoring. This creates interoperability challenges [9].

Table 4. Major challenges and suggested research directions.

Challenge	Key issue	Suggested research direction
Benchmarking	Different evaluation conditions	Standardized benchmark protocols
Reproducibility	Changing models and framework versions	Open configurations and reproducible experiments
Token Cost	Repeated LLM interactions	Cost-aware workflow optimization
Coordination	Inter-agent communication overhead	Efficient communication and agent selection
Governance	Autonomous tool execution	Human oversight and access control
Security	Tool and data access risks	Isolation, validation, and auditing
Interoperability	Heterogeneous frameworks	Standard communication and state interfaces

Source: Compiled by the authors based on [5, 9–15].

Standardized interfaces for agent communication, tool invocation, state exchange, and workflow representation could reduce dependence on individual framework implementations. Interoperability would also make it easier to replace or combine framework components without redesigning the entire application.

Observability and Debugging

Traditional software systems generally provide deterministic execution paths that can be inspected using conventional debugging techniques. Agentic systems are more difficult to debug because execution may involve probabilistic model outputs, dynamic planning, changing tool selections, and multiple interacting agents.

Reliability of Long-Running Agent Workflows

Long-running agent workflows present additional reliability challenges. An agent may need to execute many sequential steps, interact with several tools, maintain state over extended periods, and recover from temporary failures.

Therefore, state persistence, checkpointing, retry mechanisms, termination conditions, and human intervention are important architectural capabilities [5]. Future research should evaluate framework reliability over extended workflows rather than focusing primarily on short benchmark tasks.

Future Research Directions

Based on the challenges identified above, several research directions can be considered:

1. *Standardized benchmarking*: Development of common benchmark suites covering single-agent, multi-agent, tool use, and long-running workflows.
2. *Framework-level measurement*: Separation of LLM inference cost, tool execution time, and framework orchestration overhead.
3. *Reproducible evaluation*: Publication of benchmark code, configurations, prompts, framework versions, and experimental environments.
4. *Interoperable agent architectures*: Development of standardized interfaces for communication, tools, memory, and state exchange [9].
5. *Cost-aware orchestration*: Development of methods for reducing unnecessary LLM calls, redundant agent communication, and excessive context transmission [14, 15].

Overall, future research should move toward a multidimensional evaluation model in which agent frameworks are assessed according to both functional performance and software engineering characteristics (Table 4). Such an approach would provide a more reproducible and application-oriented basis for selecting and improving open-source agent frameworks.

CONCLUSION

The rapid development of agentic artificial intelligence has resulted in a growing ecosystem of

open-source frameworks for building LLM-based agents. These frameworks provide different approaches to planning, reasoning, state management, tool orchestration, workflow execution, and multi-agent coordination. This study presents a comparative analysis of major frameworks, including LangChain, LangGraph, CrewAI, AutoGen, and Semantic Kernel, using a common architectural and performance-oriented perspective.

The analysis demonstrated that although these frameworks share several fundamental components, their architectural abstractions differ significantly. LangChain provides a broad component-based ecosystem, whereas LangGraph emphasizes graph-based stateful workflow execution. CrewAI focuses on role-based agent collaboration, AutoGen provides conversation-oriented multi-agent coordination, and Semantic Kernel provides multiple orchestration patterns. These differences influence how developers design workflows, manage state, coordinate agents, integrate tools, and monitor execution.

This study also examined important performance dimensions, including latency, token consumption, orchestration overhead, task success, scalability, observability, and maintainability. Existing benchmark studies indicate that framework-level architecture can influence execution characteristics; however, performance measurements are strongly dependent on the underlying LLM, task complexity, prompt configuration, number of agents, tool usage, and execution environment. Therefore, framework comparisons should not be reduced to a single universal performance ranking.

The review further identified several open challenges. These include the lack of standardized benchmarking procedures, reproducibility limitations, multi-agent communication overhead, token and execution costs, security and governance requirements, interoperability between frameworks, observability difficulties, and reliability of long-running workflows. Addressing these challenges will require evaluation methodologies that combine technical performance measurements with software engineering and operational considerations.

Future research should focus on standardized and reproducible benchmarks that evaluate agent frameworks under controlled conditions. Such benchmarks should separately measure model inference, tool execution, and framework orchestration costs while reporting task success, latency, token usage, communication overhead, and reliability. Greater attention is also required for interoperability, security, human oversight, and long-running agent execution.

In conclusion, open-source agent frameworks provide diverse architectural approaches rather than a single universal solution. The appropriate framework depends on application requirements, workflow complexity, state management needs, coordination patterns, performance constraints, and operational requirements. A multidimensional and reproducible evaluation methodology can therefore provide a stronger foundation for understanding the strengths, limitations, and future evolution of agentic AI frameworks.

REFERENCES

1. Yao S, Zhao J, Yu D, Du N, Shafran I, Narasimhan K, et al. ReAct: synergizing reasoning and acting in language models [Preprint]. 2022. arXiv:2210.03629.
2. Wang L, Ma C, Feng X, Zhang Z, Yang H, Zhang J, et al. A survey on large language model based autonomous agents. *Front Comput Sci.* 2024;18(6):186345. doi:10.1007/s11704-024-40231-1.
3. Xi Z, Chen W, Guo X, He W, Ding Y, Hong B, et al. The rise and potential of large language model based agents: a survey. *Sci China Inf Sci.* 2025;68(2):121101. doi:10.1007/s11432-024-4222-0.
4. LangChain. (2024). LangChain documentation [Online]. LangChain. Available from: <https://python.langchain.com/docs/introduction/>.
5. CrewAI. (2026). CrewAI documentation: agents, crews and flows [Online]. CrewAI. Available from: <https://docs.crewai.com/>.
6. Wu Q, Bansal G, Zhang J, Wu Y, Li B, Zhu E, et al. AutoGen: enabling next-gen LLM applications via multi-agent conversation [Preprint]. 2023. arXiv:2308.08155.

7. Microsoft. (2026). Semantic Kernel Agent Architecture [Online]. Microsoft Learn. Available from: <https://learn.microsoft.com/en-us/semantic-kernel/frameworks/agent/agent-architecture>
8. Derouiche H, Brahmi Z, Mazeni H. Agentic AI frameworks: architectures, protocols, and design challenges [Preprint]. 2025. arXiv:2508.10146.
9. Orogat A, Rostam A, Mansour E. Understanding multi-agent LLM frameworks: a unified benchmark and experimental analysis [Preprint]. 2026. arXiv:2602.03128.
10. Liu D, Upadhyay K, Chhetri V, Siddique AB, Farooq U. A large-scale study on the development and issues of multi-agent AI systems. 2025 IEEE International Conference on Big Data (BigData), Macau, China. 2025. p.7785–92. doi:10.1109/BigData66926.2025.11402104.
11. Ahmad MSJ, Marathe K. Benchmarking multi-agent frameworks: LangGraph vs. CrewAI vs. AutoGen. J Acad Trends Innov Res. 2026;2:719–731.
12. Chowa SS, Alvi R, Rahman SS, Rahman MA, Raiaan MAK, Islam MR, et al. From language to action: a review of large language models as autonomous agents and tool users. Artif Intell Rev. 2026;59(2):71. doi:10.1007/s10462-025-11471-9.
13. Du S, Zhao J, Shi J, Xie Z, Jiang X, Bai Y, et al. A survey on the optimization of large language model-based agents. ACM Comput Surv. 2026;58(9):1–37. doi:10.1145/3789261.
14. Dennis S, Patil R, Shabahang K, Guo H. Compiling agentic workflows into LLM weights: near-frontier quality at two orders of magnitude less cost [Preprint]. 2026. arXiv:2605.22502.
15. Gupta A, Goyal M. A comprehensive review of Microsoft Semantic Kernel: Architecture, capabilities, and research directions. 2025 IEEE DELCON - International Conference on Recent Smart Technologies in Engineering for Sustainable Development, New Delhi, India. 2025. p. 1–6. doi:10.1109/DELCON68055.2025.11400282.