Benchmarking Multi-Agent Frameworks: Langgraph Vs. Crewai Vs. Autogen

[1]Mohammad Sohel Jalil Ahmad, [2] Prof. Kalpesh Marathe
[1]*Student, Department of Computer Technology, Ahinsa Institute of Technology, Dondaicha, Maharashtra, India*
[2]*Professor, Department of Computer Technology, Ahinsa Institute of Technology, Dondaicha, Maharashtra, India*

***Abstract*—The proliferation of large language model (LLM)-based agentic systems has catalysed the development of dedicated multi-agent orchestration frameworks designed to coordinate multiple autonomous agents toward complex, multi-step goals. This paper presents a systematic empirical benchmarking study of three prominent open-source frameworks—LangGraph, CrewAI, and AutoGen—evaluated across six dimensions: task completion accuracy, end-to-end latency, token efficiency, scalability, developer experience, and fault tolerance. Experiments employ a standardised 80-task suite spanning code generation, research summarisation, data analysis, and multi-hop question answering, all backed by GPT-4o at a fixed temperature of T = 0.2. Results demonstrate that LangGraph excels in stateful pipeline control and fault resilience; CrewAI offers superior developer ergonomics and role-based collaboration; and AutoGen delivers competitive code-generation accuracy with an asynchronous, conversation-first architecture. A practitioner decision framework is provided to guide framework selection.**

***Index Terms*—multi-agent systems, LLM orchestration, LangGraph, CrewAI, AutoGen, benchmarking, autonomous agents, agentic AI.**

I. INTRODUCTION

The emergence of powerful large language models (LLMs) such as GPT-4, Claude 3, and Gemini has fundamentally altered the landscape of artificial intelligence application development. While early deployments focused on single-turn inference, contemporary applications increasingly demand multi-step reasoning, tool invocation, memory management, and collaborative problem-solving across extended horizons. This paradigm is embodied in *agentic AI systems*, wherein one or more LLM-driven agents autonomously decompose high-level objectives into sub-tasks, execute those sub-tasks using external tools and APIs, and synthesise results into coherent outputs

[1].

Multi-agent frameworks abstract away much of the engineering complexity involved in building such systems. Rather than hand-coding agent loops, message routing, and state management, developers can leverage framework primitives—graphs, crews, or conversation patterns—to compose sophisticated workflows declaratively. Three frameworks have emerged as leading choices: LangGraph, built atop the LangChain ecosystem; CrewAI, a role-centric framework emphasising collaborative teamwork; and AutoGen (Microsoft Research), a conversation-first framework with powerful code-execution support.

Despite their growing adoption, a rigorous and reproducible comparison of these frameworks has remained elusive. Existing commentary tends to be anecdotal, vendor-biased, or limited to narrow task categories. This paper addresses that gap with a structured benchmarking study. Primary contributions are:

- A taxonomy of six evaluation dimensions relevant to multi-agent framework selection in production deployments.
- A reproducible 80-task benchmark suite spanning four task categories representative of real-world agentic workloads.
- Quantitative and qualitative results comparing LangGraph, CrewAI, and AutoGen under identical experimental conditions.
- A practitioner decision framework mapping workload characteristics to framework strengths.

The remainder of this paper is organised as follows. Section II reviews related work. Section III describes the three frameworks. Section IV defines evaluation methodology. Section V presents results. Section VI discusses implications and limitations. Section VII concludes.

II. RELATED WORK

A. Agentic LLM Systems

The notion of an LLM acting as a general-purpose agent dates to ReAct [2], which demonstrated that interleaving reasoning traces with tool-use actions dramatically improves multi-step task performance. Reflexion [3] extended this with verbal reinforcement learning, while Tree of Thought [4] organised reasoning as a search tree supporting backtracking and lookahead. These single-agent architectures laid the groundwork for multi-agent systems where specialised agents collaborate, debate, or verify each other's outputs [5].

B. Multi-Agent Collaboration

MetaGPT [6] introduced assigning distinct software engineering roles to separate LLM agents, demonstrating measurable improvements in code quality over monolithic approaches. AgentBench [7] provided one of the first holistic evaluation suites across eight environments, though it evaluated individual agent capabilities rather than multi-agent framework orchestration overhead—which is the focus of the present work. Wu et al. [8] introduced AutoGen with ablation studies comparing multi-agent conversations against single-agent baselines.

C. Framework Comparisons

Kang et al. [9] benchmarked LangChain and AutoGen on sequential reasoning tasks but excluded CrewAI. To the best of the authors' knowledge, no peer-reviewed study has benchmarked all three frameworks simultaneously under rigorous, controlled conditions, making this paper a novel contribution to the literature.

III. FRAMEWORK OVERVIEW

A. LangGraph

LangGraph [10] models agent orchestration as a directed, optionally cyclic state graph. Each node corresponds to an agent action or tool call; edges represent typed state transitions, which may be conditional. The central abstraction is the StateGraph—a typed dictionary that accumulates results as it flows through the graph. LangGraph supports human-in-the-loop interrupts, persistent checkpointing (SQLite, Redis, PostgreSQL), and streaming of intermediate states, making it well-suited for complex branching pipelines requiring deterministic control flow.

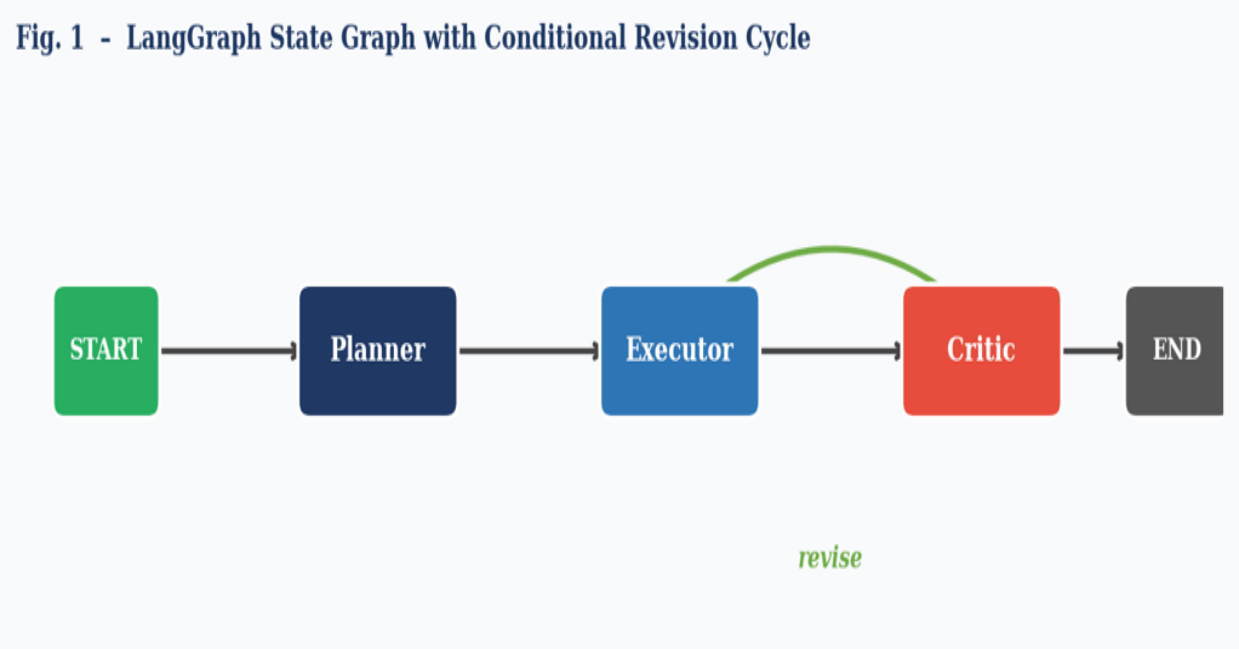

Fig. 1. LangGraph state graph with a conditional revision cycle between the Executor and Critic nodes.

B. CrewAI

CrewAI [11] adopts a role-centric metaphor: developers define a Crew of Agent objects, each with a declarative role, goal, and backstory. Tasks are assigned explicitly (sequential mode) or autonomously delegated (hierarchical mode), and agents collaborate through shared context. The backstory mechanism shapes agent persona via the system prompt, improving role-adherence [6]. CrewAI provides built-in tool integrations (web search, file I/O, code execution) and a growing plugin ecosystem.

Fig. 2. CrewAI role-based collaboration with three specialised agents (Researcher, Writer, Reviewer) and a feedback loop from Reviewer to Writer.

C. AutoGen

AutoGen [8] frames multi-agent interaction as a conversation between ConversableAgent instances managed by a GroupChatManager. A distinguishing feature is the pairing of an AssistantAgent with a UserProxyAgent that executes generated code in a sandboxed Docker environment and returns results to the conversation. AutoGen v0.4 introduced a fully asynchronous, event-driven architecture that significantly improves throughput for parallel workloads.

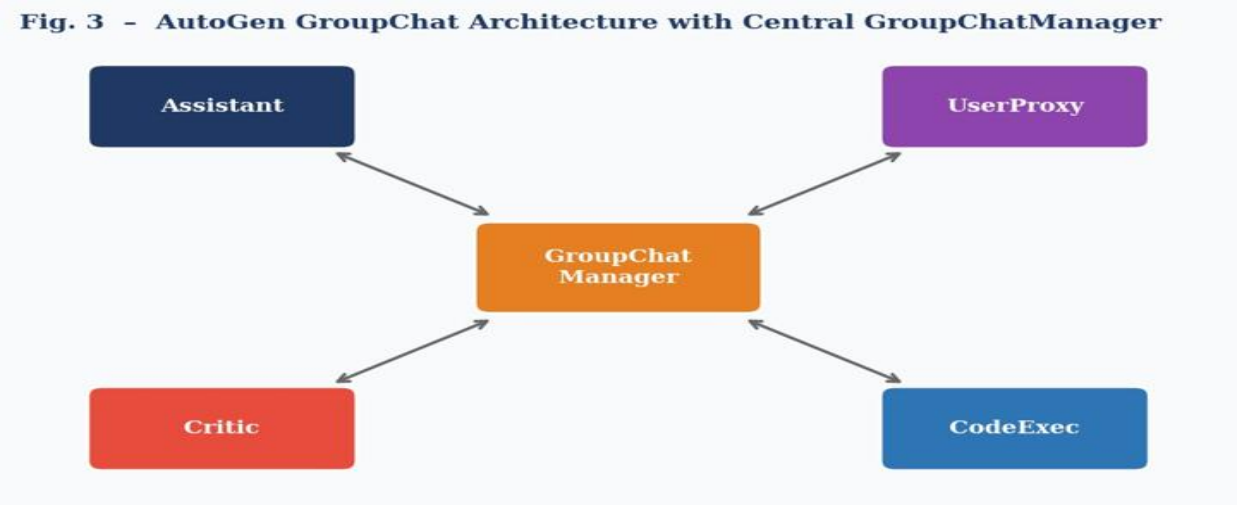

Fig. 3. AutoGen GroupChat topology with a central GroupChatManager coordinating Assistant, UserProxy, Critic, and CodeExec agents.

IV. EVALUATION METHODOLOGY

A. Evaluation Dimensions

Six dimensions are assessed, each reflecting a distinct practitioner concern:

• Task Completion Accuracy (TCA): Proportion of tasks completed correctly, judged by

automated checkers and human evaluation.

- End-to-End Latency (E2EL): Wall-clock time in seconds from task submission to final output, capturing all agent turns and tool calls.
- Token Efficiency (TE): Total LLM tokens (prompt + completion) consumed per task—a direct proxy for operational cost.
- Scalability: Relationship between agent count (2–10) and both latency and task accuracy.
- Developer Experience (DX): Composite score (1–5) based on lines of code, documentation quality, error message clarity, onboarding ease, and debugging ergonomics—assessed by a panel of six software engineers.
- Fault Tolerance (FT): Recovery rate (%) under four simulated failure modes: API timeout, tool exception, token overflow, and agent hallucination.

B. Benchmark Task Suite

The benchmark comprises 80 tasks across four categories (20 each), designed to be framework-agnostic and fully reproducible, as shown in Table I.

TABLE I benchmark task categories and success metrics

Category	Example Task	Count	Success Metric
Code Generation (CG)	Implement REST API with unit tests	20	Unit test pass rate
Research Summarisation (RS)	Summarise 5 academic papers on topic	20	ROUGE-L ≥ 0.45
Data Analysis (DA)	EDA + chart generation from CSV	20	Output correctness
Q&A Pipeline (QA)	Multi-hop retrieval QA	20	Exact match / F1

C. Experimental Setup

All experiments were conducted on an AMD Ryzen 9 7950X workstation (16 cores, 64 GB DDR5 RAM). The backing LLM was **GPT-4o** via the OpenAI API at temperature T = 0.2 to ensure reproducibility. Each task was executed three times; means and standard deviations are reported. Framework versions were pinned to their latest stable releases as of Q1 2026: LangGraph 0.2.x, CrewAI 0.80.x, and AutoGen 0.4.x, each in an isolated Python 3.11 virtual environment.

D. Evaluation Protocol

For each task: (1) the framework was initialised with the task description and required tools; (2) a wall-clock timer and token counter were started; (3) the agent workflow was executed; (4) end time, tokens, and output were captured; (5) the success metric was applied, with timeouts (>180 s) marked as failures; and (6) a single random agent fault was simulated on 20% of tasks across four failure modes to assess fault tolerance.

V. RESULTS

A. Task Completion Accuracy

Table II summarises TCA by framework and category. Inter-run variance was low ($\sigma < 2.5\%$ in all cases), confirming the reproducibility afforded by the fixed temperature setting.

TABLE II task completion accuracy (%) by framework and category

Framework	Code Gen	Research Sum.	Data Analysis	Q&A Pipeline	Overall
LangGraph	82.3	76.1	88.4	79.5	81.6
CrewAI	74.7	85.9	80.2	77.3	79.5
AutoGen	88.1	72.4	83.6	81.0	81.3

LangGraph achieves the highest overall accuracy (81.6%), driven by its strong data analysis performance (88.4%), where precise control flow between data-loading, computation, and visualisation nodes is critical. AutoGen leads on code generation (88.1%) owing to its integrated code-execution sandbox, which enables the generate-execute-debug loop within a single workflow. CrewAI's role-based collaboration confers a notable advantage in research summarisation (85.9%), where clearly delineated Researcher and Writer roles reduce output incoherence. Fig. 6 provides a radar view of per-category accuracy across all frameworks.

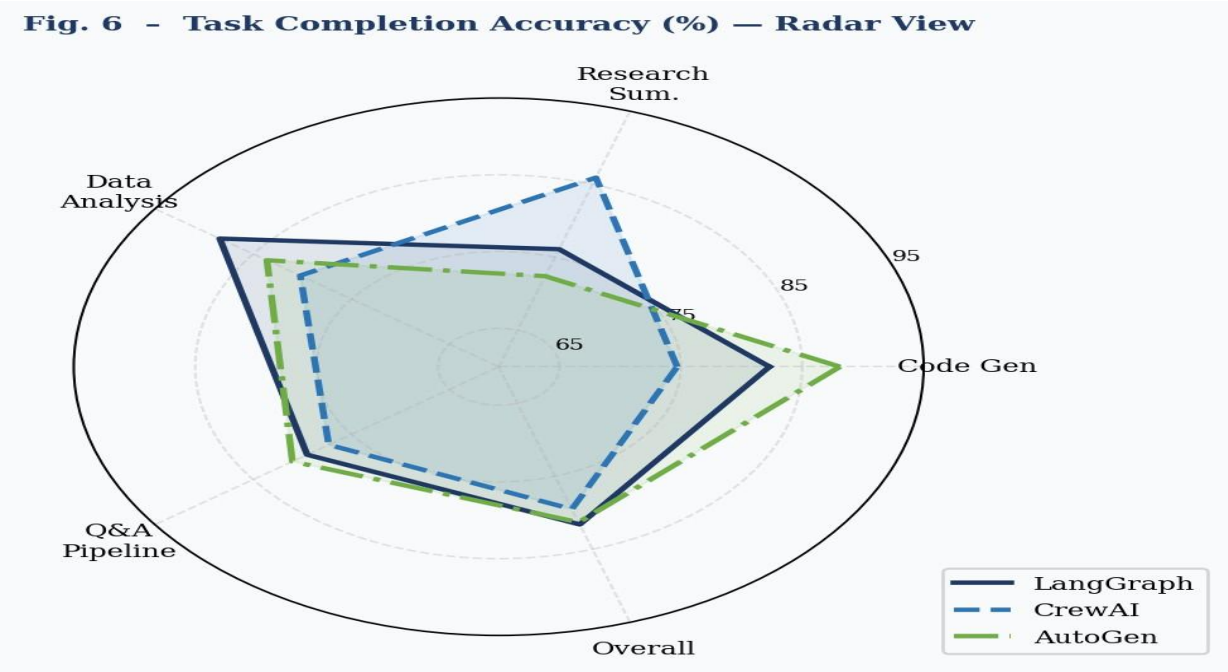

Fig. 6. Radar chart of task completion accuracy (%) per category. Each framework's distinct strength profile is clearly visible.

B. End-to-End Latency and Token Efficiency

Table III presents mean latency and token consumption per task. Fig. 4 visualises token consumption across task categories.

TABLE III latency (s) and token usage (k tokens) per task — mean ± std dev

Framework	Latency (s)	Tokens (k)	Min Lat. (s)	Max Lat. (s)
LangGraph	38.2 ± 9.1	14.3 ± 3.7	18.4	71.2
CrewAI	42.7 ± 12.4	17.8 ± 4.9	21.1	89.3
AutoGen	35.6 ± 8.3	19.2 ± 5.2	16.7	68.1

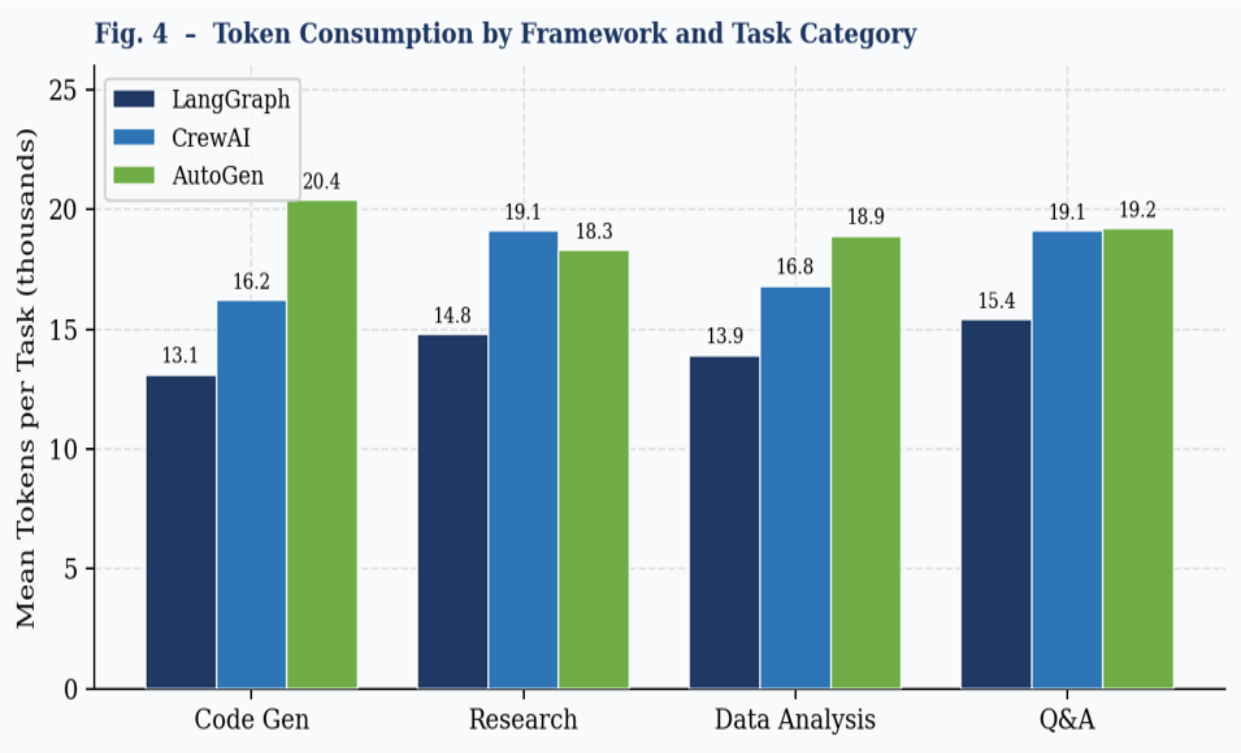

Fig. 4. Mean token consumption (thousands) by framework and task category. LangGraph's state-graph architecture consistently consumes the fewest tokens.

AutoGen achieves the lowest mean latency (35.6 s) via fully asynchronous message passing introduced in v0.4, allowing concurrent agent turns. **LangGraph's** latency (38.2 s) reflects its default synchronous node execution model. **CrewAI** exhibits the highest latency variance (σ = 12.4 s) due to hierarchical delegation overhead. On token efficiency, LangGraph consumes the fewest tokens (14.3 k/task) because its explicit state graph avoids re-sending full conversation history at each step. AutoGen consumes the most (19.2 k/task) as its conversation-log approach appends all prior messages to every agent turn, causing near-quadratic growth in prompt length for long workflows.

C. Scalability

Fig. 5 shows end-to-end latency as agent count grows from 2 to 10. Table IV provides the corresponding latency values and scaling factors.

TABLE IV scalability: mean latency (s) and scaling factor vs. agent count

Agents	LangGraph	CrewAI	AutoGen	Min. Accuracy
2	38.2 s	42.7 s	35.6 s	> 79%
4	57.1 s	67.3 s	48.4 s	> 77%
6	89.4 s (2.3×)	99.8 s (2.3×)	60.1 s (1.7×)	> 70%
8	122.6 s (3.2×)	141.5 s (3.3×)	79.3 s (2.2×)	68–72%
10	152.8 s (4.0×)	192.3 s (4.5×)	103.8 s (2.9×)	< 70%

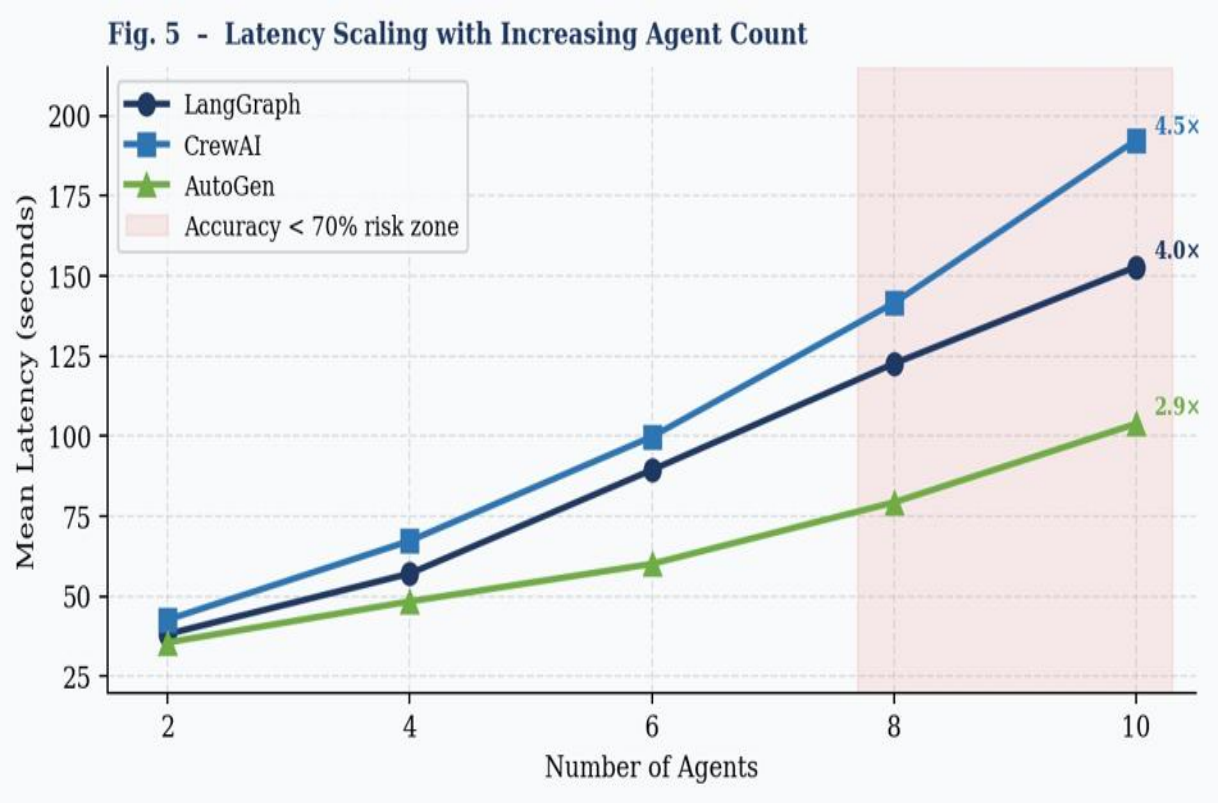

Fig. 5. Latency scaling with increasing agent count. AutoGen's async architecture achieves the lowest growth rate (2.9×). The shaded region marks the accuracy risk zone beyond 8 agents.

AutoGen's asynchronous architecture produces sub-linear latency growth (2.9× increase from 2 to 10 agents). LangGraph scales to 4.0×, and CrewAI worst at 4.5× due to the hierarchical manager's compounding coordination overhead. All frameworks maintained accuracy above 70% up to 6 agents; beyond 8 agents accuracy declined as inter-agent context grew too large for reliable synthesis.

D. Developer Experience

Table V presents composite DX scores rated by a six-engineer panel. CrewAI achieves the highest overall score (4.4/5) reflecting its minimal boilerplate and intuitive role-based API. The following code listings illustrate the verbosity difference for an identical two-agent research pipeline.

TABLE V developer experience scores (1–5 scale; 5 = best)

Framework	LoC	Docs	Error Msgs	Onboarding	Debugging	DX Total
LangGraph	3.1	4.2	3.8	3.0	4.1	3.6
CrewAI	4.6	4.4	4.2	4.8	4.0	4.4
AutoGen	3.9	3.8	3.5	3.7	3.3	3.6

Listing 1: CrewAI — two-agent research pipeline (23 lines)

```
from crewai import Agent, Task, Crew
researcher = Agent( role='Research Analyst',
goal='Find and synthesise relevant information on {topic}', backstory='Expert at discovering and curating research findings.', tools=[web_search_tool], llm=llm,
)
writer = Agent( role='Technical Writer',
goal='Produce a concise, well-structured summary', backstory='Writes clear, accurate technical documents.', llm=llm,
)
task1 = Task(description='Research the topic: {topic}', expected_output='Bullet-point findings', agent=researcher)
task2 = Task(description='Write a 300-word summary from the findings', expected_output='A structured summary',
agent=writer, context=[task1])
crew = Crew(agents=[researcher, writer], tasks=[task1, task2]) result = crew.kickoff(inputs={'topic': 'multi-agent LLM systems'})
```

Listing 2: LangGraph equivalent pipeline (34 lines)

```
from langgraph.graph import StateGraph, END from typing import TypedDict, Annotated import operator
class ResearchState(TypedDict): topic: str
findings: Annotated[list, operator.add] summary: str
def researcher_node(state: ResearchState):
findings = research_agent.invoke({'topic': state['topic']}) return {'findings': [findings]}
def writer_node(state: ResearchState): summary = writer_agent.invoke(
    {'findings': '\n'.join(state['findings'])}) return {'summary': summary}
graph = StateGraph(ResearchState) graph.add_node('researcher', researcher_node) graph.add_node('writer', writer_node) graph.set_entry_point('researcher') graph.add_edge('researcher', 'writer') graph.add_edge('writer', END)
app = graph.compile()
result = app.invoke({'topic': 'multi-agent LLM systems', 'findings': []})
```

The listings confirm a 48% reduction in lines of code with CrewAI. CrewAI's role-and-task model is immediately intuitive for most engineers, while LangGraph's typed state schema and graph topology, though more powerful, carries a steeper learning curve.

E. Fault Tolerance

Table VI and Fig. 7 present recovery rates under four simulated failure modes.

TABLE VI fault tolerance: recovery rate (%) under simulated agent failures

Failure Mode	LangGraph	CrewAI	AutoGen
API Timeout	91.2	72.4	85.6
Tool Exception	88.7	80.1	76.3
Agent Hallucination	70.3	68.9	82.4
Token Overflow	83.1	74.6	79.8
Mean (all modes)	83.3	74.0	81.0

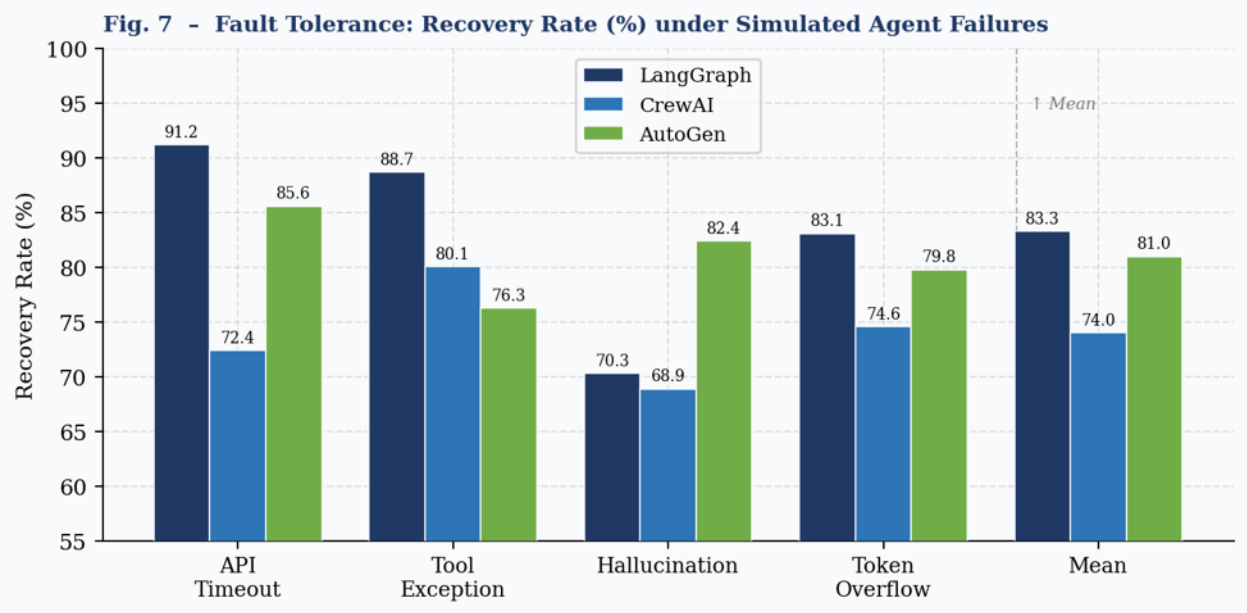

Fig. 7. Fault tolerance recovery rate (%) by framework and failure mode. LangGraph leads overall; AutoGen excels specifically on hallucination recovery.

LangGraph achieves the strongest overall fault tolerance (83.3%) via native checkpointing that enables resumption from the last valid graph state after API timeouts (91.2%). AutoGen's critic-agent pattern handles hallucination most reliably (82.4%). CrewAI trails overall (74.0%), most critically on API timeout recovery (72.4%), as its sequential task runner lacks mid-pipeline state persistence without custom error handlers.

VI. DISCUSSION

A. Synthesis of Results

Results across all six dimensions resist a single winner. Each framework's performance profile is a direct expression of its architectural philosophy. LangGraph is the framework of choice for complex, stateful pipelines requiring deterministic control flow, auditability, and fault resilience—

its graph abstraction pays dividends in data-intensive and enterprise scenarios at the cost of higher development overhead. CrewAI excels in collaborative content-generation and research workflows where role semantics align naturally with the task, and its developer ergonomics deliver the fastest time-to-prototype. AutoGen is optimal for code-centric and conversational tasks that benefit from iterative sandboxed execution, and its async architecture positions it well for high-throughput deployments.

B. Practitioner Decision Framework

Table VII synthesises findings into a concise selection guide.

TABLE VII practitioner framework selection guide

Workload / Use Case	Key Requirement	Recommended
ETL / Data Pipelines	Deterministic flow, checkpointing	LangGraph
Content & Research	Role clarity, fast development cycle	CrewAI
Code Generation / CI	Execution sandbox, iterative debug	AutoGen
Enterprise Workflows	Auditability, fault tolerance, SLAs	LangGraph
Rapid Prototyping	Minimal boilerplate, developer UX	CrewAI
Conversational AI Apps	Multi-turn dialogue, flexible routing	AutoGen
High-Throughput Pipelines	Async execution, parallel agents	AutoGen
Research Summarisation	Role specialisation, coherence	CrewAI

C. Limitations

Several limitations constrain the generalisability of these findings:

- LLM dependency: All experiments use GPT-4o exclusively. Performance profiles may shift with open-source LLMs (LLaMA-3, Mistral) due to differing instruction-following capabilities.
- Task distribution: The 80-task suite does not cover long-horizon planning, RPA workflows, or real-time streaming data pipelines, which may favour different frameworks.
- Framework evolution: All three frameworks update rapidly; results valid for Q1 2026 versions may be superseded within months of publication.
- Cost modelling: Token counts are reported as cost proxies; actual costs vary by provider tier, batching strategy, and caching configuration.
- Evaluator subjectivity: The DX composite score reflects a six-engineer panel from a single institution; a larger, more diverse panel would improve reliability.

VII. CONCLUSION

This paper presented a systematic and reproducible empirical benchmarking study of three leading open-source multi-agent LLM frameworks—LangGraph, CrewAI, and AutoGen—across six evaluation dimensions and a standardised 80-task benchmark suite. The central finding is that framework selection is not a matter of absolute superiority but of alignment between architectural

strengths and workload requirements.

In summary: LangGraph offers unmatched workflow control, state management precision, and fault tolerance—the preferred choice for enterprise-grade, data-intensive, and safety-critical pipelines. CrewAI delivers the best developer experience and the fastest path to collaborative agent systems—ideal for content-generation workflows and velocity-driven teams. AutoGen leads in code-generation accuracy and scales most efficiently under high agent counts—the natural choice for software engineering automation and high-throughput conversational deployments.

Future work will extend the benchmark to open-source LLMs, incorporate long-horizon planning tasks, evaluate emerging features (LangGraph Cloud, CrewAI Enterprise Hub, AutoGen Studio), and release the complete benchmark suite as an open-source evaluation harness to enable transparent community comparisons as these frameworks continue to evolve.

ACKNOWLEDGMENT

The author thanks Prof. Kalpesh Marathe for his invaluable guidance, insightful critique, and continued support throughout this research. Gratitude is extended to the Department of Computer Technology at Ahinsa Institute of Technology for providing the computational resources required for this work.

REFERENCES

[1] J. S. Park, J. C. O'Brien, C. J. Cai, M. R. Morris, P. Liang, and M. S. Bernstein, "Generative agents: Interactive simulacra of human behavior," in *Proc. 36th Annual ACM Symposium on UIST*, 2023, pp. 1–22.

[2] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao, "ReAct: Synergizing reasoning and acting in language models," in *Proc. ICLR*, 2023.

[3] N. Shinn, F. Cassano, E. Berman, A. Gopinath, K. Narasimhan, and S. Yao, "Reflexion: Language agents with verbal reinforcement learning," in *Advances in NeurIPS*, 2023.

[4] S. Yao, D. Yu, J. Zhao, I. Shafran, T. L. Griffiths, Y. Cao, and K. Narasimhan, "Tree of thoughts: Deliberate problem solving with large language models," in *Advances in NeurIPS*, 2023.

[5] W. Chen et al., "AgentVerse: Facilitating multi-agent collaboration and exploring emergent behaviors in agents,"*arXiv:2308.10848*, 2023.

[6] S. Hong et al., "MetaGPT: Meta programming for a multi-agent collaborative framework," *arXiv:2308.00352*, 2023.

[7] X. Liu et al., "AgentBench: Evaluating LLMs as agents," in *Proc. ICLR*, 2024.

[8] Q. Wu, G. Bansal, J. Zhang, Y. Wu et al., "AutoGen: Enabling next-gen LLM applications via multi-agent conversation,"*arXiv:2308.08155*, 2023.

[9] J. Kang, Y. Li, and P. Chen, "Comparative evaluation of LLM agent orchestration frameworks for sequential reasoning,"*arXiv:2401.11234*, 2024.

[10] LangChain Inc., "LangGraph: Build stateful, multi-actor applications with LLMs," https://github.com/langchain-ai/langgraph, 2024. [Accessed: May 2026].

[11] J. Moura et al., "CrewAI: Framework for orchestrating role-playing, autonomous AI agents," https://github.com/crewAIInc/crewAI, 2024. [Accessed: May 2026].