Review

LLM-Based Multi-Agent Orchestration: A Survey of Frameworks, Communication Protocols, and Emerging Patterns

Yiwen Zhu [*], Lihe Liu [†], Jiaqian Yu [†], Di Zhang [†]

Posted Date: 30 April 2026

doi: 10.20944/preprints202604.2147.v1

Keywords: large language models; multi-agent systems; agent orchestration; communication protocols; model context protocol; agent-to-agent protocol; LangGraph; CrewAI; AutoGen

 doi:10.20944/preprints202604.2147.v1

Review

LLM-Based Multi-Agent Orchestration: A Survey of Frameworks, Communication Protocols, and Emerging Patterns

Yiwen Zhu [1,*], Lihe Liu [2,†], Jiaqian Yu [3,†] and Di Zhang [4,†]

[1] Department of Physics and Astronomy, Rice University, Houston, TX 77005, USA
[2] Department of Computer Science, University of Wisconsin–Madison, Madison, WI 53706, USA
[3] Department of Statistics, The George Washington University, Washington, DC 20052, USA
[4] Daniel J. Epstein Department of Industrial and Systems Engineering, University of Southern California, Los Angeles, CA 90089, USA
* Correspondence: yz167@rice.edu
† These co-authors are listed in alphabetical order by surname; their individual contributions are detailed in the Author Contributions section.

Abstract

The proliferation of large language model (LLM) agents has enabled increasingly complex multi-step automation; however, composing multiple agents into coherent systems introduces significant orchestration challenges that remain poorly documented. This survey examines LLM-based multi-agent orchestration from 2023 through early 2026 (literature cutoff: March 2026). We propose a three-topology, one-adaptivity taxonomy—centralized, decentralized, and hierarchical coordination topologies, each optionally augmented with a dynamic/adaptive control axis—grounded in classical multi-agent systems theory and recent empirical evidence. We compare four leading frameworks (LangGraph, CrewAI, AutoGen/Microsoft Agent Framework, and OpenAI Agents SDK) along axes directly relevant to practitioners: state-management granularity, token cost structure, failure-recovery options, and design philosophy. The emerging protocol stack is examined in terms of why MCP (agent-to-tool) and A2A (agent-to-agent) occupy complementary layers, how the ACP–A2A merger signals protocol convergence, and where ANP's decentralized-discovery design fits. Production design considerations—state management, task planning, error handling, scalability, and security—are evaluated with reference to published benchmarks. We close by identifying five open challenges and proposing a six-dimension evaluation framework for multi-agent coordination quality. This paper provides practitioners with a decision framework spanning taxonomy, framework selection, protocol adoption, and production deployment.

Keywords: large language models; multi-agent systems; agent orchestration; communication protocols; model context protocol; agent-to-agent protocol; LangGraph; CrewAI; AutoGen

1. Introduction

Research in artificial intelligence has accelerated sharply over the past three years, driven by large language models (LLMs) such as GPT-4, Claude, Gemini, and LLaMA. Any one of these models can parse a legal contract, produce working code, or distill a technical paper into a readable summary. Real-world workflows, however, are seldom so tidy. A product launch, a clinical trial analysis, a cross-repository refactor: each demands sustained reasoning across many steps, access to live external data, and the capacity to revise earlier conclusions when fresh evidence undermines them [1,2]. Under this pressure, the field has moved toward agentic AI, in which an LLM is wrapped with tools, persistent memory, and planning scaffolds so that it can act, observe, iterate, and self-correct [3,4].

Single-turn completions evolved into chain-of-thought prompting [5], which Wei et al. showed to be an emergent ability of large models. Retrieval-augmented generation [6] broke the closed-book

 doi:10.20944/preprints202604.2147.v1

constraint by coupling models with external knowledge. Tool use and function calling turned text generators into agents capable of querying databases and executing code [3,7]. Once individual agents became viable, a follow-on question became unavoidable: what happens when several such agents are asked to collaborate?

The answer, LLM-based multi-agent orchestration, is a paradigm in which specialized agents assume distinct roles, exchange information through structured protocols, and coordinate their actions to solve problems beyond the reach of any single agent [8,9]. In this survey, orchestration covers five interrelated mechanisms: (1) task decomposition and allocation, (2) inter-agent communication and context sharing, (3) state management and persistence, (4) control-flow sequencing, and (5) error detection and recovery [10,11]. Without the orchestration layer, a multi-agent system collapses into a collection of independent programs that duplicate effort, contradict one another, or loop without termination [12].

Enterprise adoption reflects this urgency. LangChain's 2025 State of AI Agents report (1,340 respondents) shows 57.3% of organizations with agents in production, with customer service (26.5%) and research/data analysis (24.4%) as the leading use cases (vendor-reported†) [13]. McKinsey's 2025 State of AI survey reports that 23% of organizations are scaling agentic AI systems, with an additional 39% experimenting (industry survey†) [14]. Gartner projects 40% enterprise application penetration by end of 2026, yet separately predicts that over 40% of agentic AI projects will be canceled by end of 2027, citing spiraling costs, unclear business value, and inadequate risk management (analyst projection†) [15]. The gap between ambition and delivery is precisely where orchestration research matters most.

Multi-agent systems grow harder to manage past a handful of agents: five agents yield ten pairwise interaction channels; ten agents yield forty-five. Debugging, monitoring, and testing burdens increase super-linearly as coordination complexity compounds [16]. Guo et al. [8] identify coordination failures—agents that contradict one another, duplicate effort, or produce inconsistent shared state—as the dominant cause of system-level degradation, distinct from the individual model errors that single-agent benchmarks measure. These observations motivate a principled treatment of orchestration that goes beyond cataloging available tools.

Contributions. This paper presents a comprehensive survey of LLM-based multi-agent orchestration, examining orchestration patterns, framework design philosophies, communication protocols, production deployment concerns, and evaluation methodology. Specifically, we make five concrete contributions:

1. A taxonomy of orchestration patterns comprising three coordination topologies (centralized, decentralized, hierarchical) and one orthogonal adaptivity axis (dynamic), rooted in classical MAS theory and grounded in LLM-era empirical evidence, with a decision framework for selection based on task structure, agent count, fault-tolerance requirements, and cost budget (Section 4).
2. A comparative framework analysis going beyond feature checklists to contrast design philosophies, state-management models, token-cost profiles, and failure modes across LangGraph, CrewAI, AutoGen/Microsoft Agent Framework, OpenAI Agents SDK, MetaGPT, and DSPy (Section 5).
3. An integrated treatment of the emerging protocol stack (MCP, A2A, ANP) structured around why two coordination layers are architecturally necessary, why collapsing them would be harmful, and what the ACP–A2A merger and AAIF formation mean for long-term convergence (Section 6).
4. A discussion of production-grade design considerations—state management, task planning, error handling, scalability, and security—illustrated with quantitative results from published work (Section 7).
5. A review of evaluation gaps with a proposed six-dimension evaluation framework for coordination quality, application domains spanning six sectors, and five open challenges grounded in benchmark data (Sections 8–10).

 doi:10.20944/preprints202604.2147.v1

The survey spans 2023 through early 2026, with a literature cutoff of March 2026; framework capabilities, protocol adoption status, and benchmark scores reflect this date. LLM-based multi-agent orchestration is an active design direction rather than a settled paradigm; results remain highly task-dependent, benchmark-dependent, and sensitive to implementation details. Readers consulting this survey after the March 2026 cutoff should verify current framework and protocol status directly. We concentrate on LLM-backed systems, touching classical multi-agent research [17] and single-agent LLM surveys [18,19] only where they supply essential context. The remainder of this paper is organized as follows: Section 2 provides background on LLM agents. Section 3 describes the survey methodology. Section 4 presents the orchestration taxonomy and decision framework. Section 5 surveys frameworks. Section 6 examines protocols. Section 7 discusses design considerations. Section 8 reviews evaluation. Section 9 covers applications. Sections 10 and 11 identify open challenges and conclude. Figure 1 summarizes this organization.

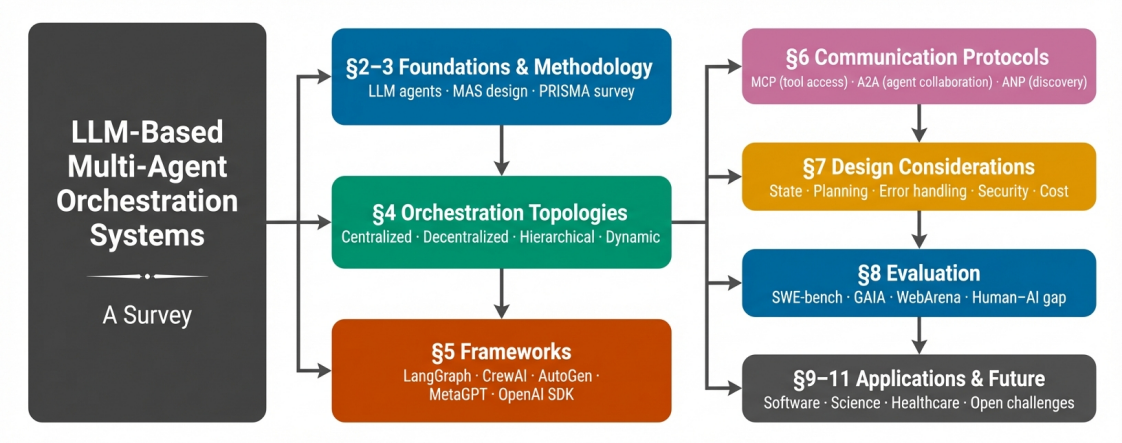

Figure 1. Organization of this survey. The paper progresses from foundations and methodology through orchestration topologies, frameworks, and communication protocols, to design considerations, evaluation, and applications.

2. Background

2.1. From LLMs to LLM-Based Agents

At bottom, an LLM is a neural network trained on massive text corpora to generate and reason about language [20]. The capabilities are substantial; the constraints are equally real. Training-data cutoffs freeze knowledge at a point in time, there is no native mechanism for live information access, and finite context windows impose a hard ceiling on reasoning scope. Wang et al. [18] decompose an LLM-based agent into four components: a Brain (LLM core handling reasoning and planning), Perception (input modules for instructions and observations), Memory (short-term via context window; long-term via external vector stores [21]), and Action (output modules for API calls, code execution, and inter-agent messaging).

Two foundational paradigms deserve emphasis. The ReAct framework [3] interleaves reasoning traces with action execution; by generating a thought, acting on it, observing the result, and reasoning again, a ReAct agent grounds chain-of-thought in real-world feedback. On HotpotQA, ReAct's error profile differs qualitatively from pure chain-of-thought: where chain-of-thought hallucinates, ReAct tends to err in search strategy—a distinction that matters for multi-agent systems where different error types demand different recovery mechanisms. On ALFWorld, ReAct achieves a 34% absolute improvement over imitation-learning baselines using one or two in-context examples versus 10^5 training samples. Reflexion [4] adds a self-critique loop: after a failed attempt, the agent generates a verbal reflection, stores it in episodic memory, and consults it on subsequent tries. Reflexion reaches 91% pass@1 on HumanEval, surpassing GPT-4's 80%, demonstrating that agent-level self-critique can exceed the raw capability of the underlying model—a finding with direct implications for multi-agent design where failure experiences can be shared across agents.

 doi:10.20944/preprints202604.2147.v1

Tree of Thoughts (ToT) [22] generalizes chain-of-thought by enabling exploration over multiple reasoning paths. LATS [23] pushes further by unifying reasoning, acting, and planning within a Monte Carlo Tree Search framework, achieving 92.7% on HumanEval with GPT-4. The progression from ReAct through Reflexion to LATS illustrates how agent reasoning has matured: each step trades additional token cost for higher reliability, a tradeoff that orchestration frameworks must expose and manage.

2.2. Multi-Agent Systems: From Classical to LLM-Based

Multi-agent systems trace back to distributed-AI work in the 1980s [17]. Classical research contributed agent communication languages (FIPA-ACL [24]), coordination mechanisms such as the contract-net protocol, and organizational models including holarchies. LLM-based MAS depart from these foundations in three ways. First, agents communicate primarily in natural language rather than formal ontologies, making interactions more flexible but harder to verify [8]. Second, behavior can be steered at runtime through prompts and context, which is both a feature (rapid adaptation) and a risk (prompt sensitivity, injection attacks) [9]. Third, LLM reasoning enables emergent capabilities that rule-based agents cannot replicate; Du et al. [25] demonstrated that multi-agent debate significantly improves factuality and reasoning, suggesting that the interaction itself is a source of quality.

Key design dimensions include agent specialization through role prompts or tool assignment [26,27], communication topology (broadcast, point-to-point, publish-subscribe), coordination mechanism (centralized, decentralized, or hierarchical), and memory architecture (shared, private, or hybrid) [21]. The choice of communication topology has direct consequences for cost: broadcast architectures expose every agent to every message, while selective subscription filters at the source. MetaGPT's publish-subscribe mechanism, in which each agent subscribes only to role-relevant messages, offers a concrete alternative to full-broadcast approaches and halves token consumption per line of code in software-development benchmarks [26].

2.3. The Orchestration Challenge

Scale makes orchestration unavoidable. A software development crew—product manager, architect, developer, tester, reviewer—requires precise dependency management: the developer needs the architect's specification, the reviewer needs the developer's code, and all agents must remain aligned on the same requirements throughout. Orchestrating these dependencies—ordering, artifact sharing, failure recovery, and consistency—is the specific problem that frameworks and protocols exist to solve. MetaGPT's Standardized Operating Procedures attack this problem with structured intermediate outputs (requirements documents, UML diagrams, interface specifications) at each stage, cutting cascading hallucinations and achieving 85.9% on HumanEval at 124.3 tokens per line versus ChatDev's 248.9 [26].

Without a dedicated orchestration layer, agents exhibit three canonical failure modes: task duplication (multiple agents independently solving the same subtask), contradictory outputs (agents producing inconsistent artifacts from shared premises), and convergence failure (the system cycles indefinitely without reaching a terminal state). The frequency and severity of these failures increase with agent count and task complexity, which is why orchestration is a first-class engineering concern and not an implementation afterthought.

3. Survey Methodology

This paper is a structured narrative survey rather than a fully systematic review. It follows a structured search-and-screening process appropriate for a rapidly evolving field where preprints, framework documentation, and industry reports carry substantial weight alongside peer-reviewed publications. The narrative synthesis approach is chosen because the field's pace of development makes strict systematic-review methodology (registered protocol, dual-reviewer screening, GRADE evidence grading) impractical at the March 2026 cutoff.

 doi:10.20944/preprints202604.2147.v1

Search process. We searched six databases: Scopus, Web of Science, IEEE Xplore, ACM Digital Library, arXiv, and Google Scholar. The search window spanned January 2022 through March 2026. A representative Boolean query was: `("LLM agent" OR "AI agent" OR "autonomous agent") AND ("multi-agent" OR "orchestration" OR "agent collaboration") AND ("framework" OR "protocol" OR "coordination")`, with minor adaptations per database. Citation-chain snowballing (forward and backward) from seminal papers [3,8,26,28] supplemented keyword searches.

Screening flow (PRISMA-style). (1) Initial retrieval: approximately 1,200 records (database search counts were logged per session but not merged into a single deduplicated total at time of search). (2) After automated deduplication and removal of clearly off-topic entries: approximately 400 unique entries assessed. (3) After title-and-abstract screening: approximately 200 survivors forwarded to full-text review. (4) After full-text review against inclusion and exclusion criteria: 120 records passed. (5) Final corpus: approximately 80 primary references cited in the manuscript plus additional contextual references screened but not cited. All stage counts are estimates; exact per-database counts are available from the corresponding author.

Inclusion criteria. Peer-reviewed publications or high-citation preprints (>=10 citations) addressing orchestration patterns, agent communication protocols, framework design, evaluation, or production deployment of LLM-based multi-agent systems.

Exclusion criteria. Non-English publications; workshop papers shorter than four pages without an adopted system; blog posts lacking empirical evidence; single-agent-only work without a multi-agent dimension; duplicate publications (venue version retained over preprints).

Source-quality tiers. To distinguish evidence strength, we apply three tiers throughout: **Tier 1** — peer-reviewed publications at major venues (NeurIPS, ICML, ICLR, ACL, IEEE, ACM); **Tier 2** — high-citation preprints (>=10 citations) and official specifications from standards bodies (Linux Foundation, W3C); **Tier 3** — industry reports, vendor announcements, and SDK documentation, used only for adoption and deployment facts and explicitly attributed throughout. Claims sourced from Tier 3 are marked with a dagger (†).

Primary reference breakdown. Multi-agent frameworks and systems (22); orchestration patterns and coordination mechanisms (14); communication protocols and standards (8); evaluation benchmarks (11); foundational LLM and agent research (11); industry reports (7); security, safety, and alignment (7). These categories sum to 80 and are non-overlapping by primary topic; references spanning multiple categories are counted under their primary contribution.

Limitations. All screening was performed by a single reviewer (Y.Z.), which may introduce selection bias. Inclusion and exclusion decisions for borderline cases were documented. The rapid pace of development in this field means that preprints and framework releases that post-date the March 2026 cutoff are not covered.

4. Taxonomy of Orchestration Patterns

The way agents are coordinated shapes every downstream property of a system: scalability limits, fault tolerance, debugging difficulty, token cost, and task suitability. Drawing on classical MAS foundations [17] and recent LLM-MAS literature [8,10,11], we organize orchestration approaches into four patterns. Figure 2 depicts these patterns with their representative communication topologies. This taxonomy is an analytical lens for reasoning about design choices, not an exhaustive classification; many production systems blend multiple patterns (Section 4.5), and the taxonomy captures the dominant coordination structure rather than every runtime detail.

 doi:10.20944/preprints202604.2147.v1

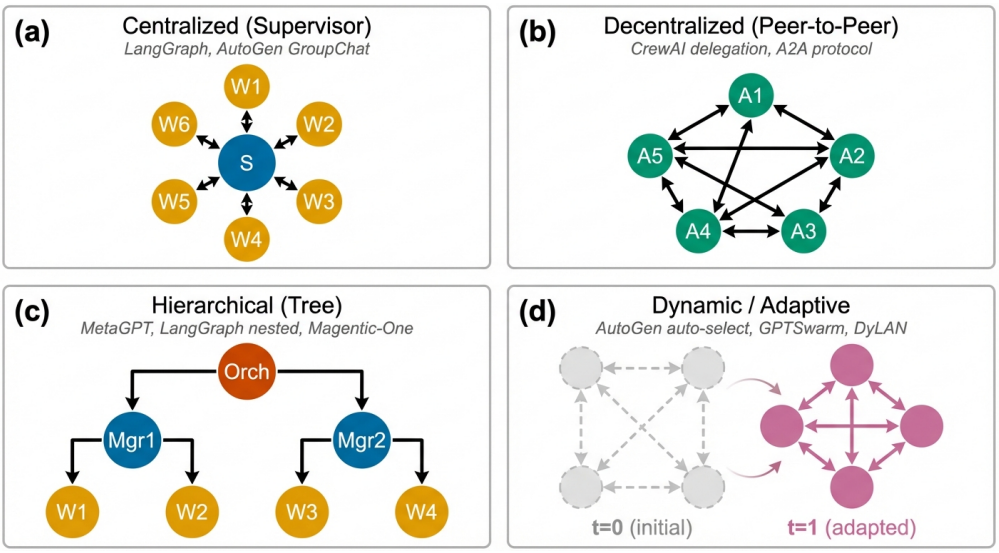

Figure 2. Four orchestration topologies for multi-agent systems: (a) centralized, with a single supervisor routing all communication; (b) decentralized, with peer-to-peer negotiation; (c) hierarchical, with tree-structured delegation; and (d) dynamic, with runtime-adaptive topology reconfiguration.

4.1. Taxonomy Structure Note

The first three patterns—centralized, decentralized, and hierarchical—describe **coordination topology**: the structural relationship between agents in terms of control authority and communication flow. The fourth pattern, dynamic/adaptive, describes **control adaptivity**: whether the topology evolves at runtime. These dimensions are orthogonal; any base topology can incorporate runtime adaptation. Dynamic-centralized systems (AutoGen auto-select), dynamic-hierarchical systems (Magentic-One re-planning), and dynamic-decentralized systems (AgentVerse) all exist in practice. We retain all four as named patterns because they correspond to distinct framework primitives and practitioner design decisions, but the decision framework in Section 4.5 treats adaptivity as a modifier rather than a fourth exclusive category.

4.2. Centralized (Supervisor) Pattern

A single supervisor agent owns the workflow. It decomposes the incoming task, assigns subtasks to worker agents, collects their outputs, and decides what happens next.

Architecture. Communication follows a hub-and-spoke topology: every message passes through the supervisor, which maintains a global view of task progress [29]. The supervisor is typically an LLM that consults current state to select the next worker.

Framework implementations. In LangGraph, a supervisor node invokes an LLM to route execution across worker nodes based on the shared state object; conditional edges handle branching without embedding routing logic inside the workers [30]. AutoGen's GroupChatManager fills an analogous role, selecting the next speaker using one of four policies (auto, round_robin, random, or manual) and broadcasting the response to all participants [28]. A fifth custom-function option accepts a callback that receives the last speaker and GroupChat object and returns the next speaker, providing flexibility when built-in modes are insufficient.

When to choose it. The centralized pattern works well when task structure is known in advance, agent count is modest (roughly 3–7), and auditability matters—for instance, in compliance-sensitive financial workflows where every routing decision must be traceable.

Strengths and limitations. Clear control flow; easy to trace and debug; the supervisor's global view enables efficient resource allocation. The supervisor is also a single point of failure and a throughput bottleneck. Its context window fills as agent count and conversation length grow; AutoGen's auto speaker-selection mode processes the full conversation history through a nested chat at every turn, a pattern whose token cost scales linearly with dialogue length. GroupChatManager provides Trans-

 doi:10.20944/preprints202604.2147.v1

form Messages utilities (MessageHistoryLimiter, MessageTokenLimiter, TextMessageCompressor) to mitigate this, but they require manual tuning.

4.3. Decentralized (Peer-to-Peer) Pattern

No central coordinator exists. Agents communicate directly, each deciding autonomously when to act, whom to consult, and how to contribute.

Architecture. Task allocation emerges from negotiation and self-selection; each agent holds local state and makes its own routing decisions [31].

Framework implementations. CrewAI supports a delegation mechanism (allow_delegation=True) that lets one agent hand a subtask to a peer without passing through a central manager [32]. The A2A protocol is designed for peer-to-peer scenarios: agents publish Agent Cards advertising their capabilities, and other agents discover and delegate to them through structured task objects without centralized authority [33].

When to choose it. The decentralized pattern suits open-ended exploration tasks—brainstorming, adversarial red-teaming, hypothesis generation—where the problem space is too large or too poorly understood for a single coordinator to decompose effectively. It also fits scenarios where organizational boundaries make centralization impractical.

Strengths and limitations. No single point of failure; adding agents does not overload a coordinator; emergent solutions can arise from unconstrained interaction. However, coherent global behavior is hard to guarantee. LLM-Deliberation experiments [31] show that even state-of-the-art models (GPT-4, Llama-3 70B) struggle with multi-party negotiation tasks requiring arithmetic reasoning and strategic planning. Distributed execution traces make post-hoc debugging laborious.

4.4. Hierarchical Pattern

Agents are organized in a tree. Manager agents delegate downward; results percolate upward. Each manager handles a domain or subtask scope and may further decompose its assignment.

Architecture. Top-level managers receive the task and delegate to mid-level managers, who delegate to leaf-level workers. Communication scales linearly within the tree rather than quadratically across the full agent set [34].

Framework implementations. LangGraph's nested-graph facility allows a parent graph to invoke sub-graphs as nodes, each encapsulating its own agents and logic, with the parent's state passing through defined interfaces [30]. CrewAI provides a hierarchical process mode in which a manager agent automatically delegates tasks based on capability assessment, validates intermediate outputs, and synthesizes the final answer; the manager makes runtime delegation decisions rather than pre-assigning tasks, introducing a dynamic element within the hierarchical structure [32]. MetaGPT encodes an explicit software-company hierarchy—product manager, architect, project manager, and engineer—coordinating through Standardized Operating Procedures that prescribe structured artifacts at each stage. This structured-document approach forces agents to produce verifiable artifacts rather than chat messages, achieving 85.9% on HumanEval and cutting token consumption to 124.3 tokens per line versus ChatDev's 248.9 [26]. Magentic-One [35] extends the pattern with a lead Orchestrator that plans, tracks progress, and re-plans to recover from errors, directing specialized agents for web browsing, file navigation, and code execution.

When to choose it. The hierarchical pattern is a natural fit for large, decomposable tasks with clear domain boundaries—enterprise workflows with distinct functional areas, or software projects with well-defined phases—and for systems that must scale beyond 10–15 agents where flat topologies become unwieldy.

Strengths and limitations. Balances control and scalability; natural modularity permits independent development and versioning of sub-teams; communication overhead is bounded by tree depth. Multi-level message passing introduces latency, intermediate managers can bottleneck if the tree is poorly balanced, and the rigid tree structure may not suit tasks demanding dynamic reorganization mid-execution.

 doi:10.20944/preprints202604.2147.v1

4.5. Dynamic/Adaptive Pattern

The orchestration topology is not fixed at design time; it changes as the task unfolds. A meta-level mechanism observes progress and reconfigures the agent network, creating agents on demand, reassigning roles, and rerouting communication [36].

Framework implementations. AutoGen's auto speaker selection uses the LLM to choose the next speaker at each turn based on conversation history; optional speaker_transitions constraints impose partial structure on an otherwise adaptive process [28]. LangGraph's conditional edges evaluate state predicates at runtime, enabling branches, loops, and dynamic routing without a predetermined path [30]. DyLAN [37] introduces a two-stage paradigm using an Agent Importance Score to select optimal agent teams, achieving up to 25% accuracy improvement on MMLU subjects. GPTSwarm [38] models the entire agent system as an optimizable computation graph, then uses REINFORCE to optimize edge probabilities, automatically discovering effective collaboration structures. On MMLU, GPTSwarm achieves comparable accuracy to DyLAN (83.01% vs. 83.66%) at roughly 1/20th the cost ($5.32 vs. $105.93), demonstrating that learned orchestration topologies can be dramatically more token-efficient than hand-designed ones.

When to choose it. The dynamic pattern is appropriate when the task is genuinely novel, when required capabilities cannot be enumerated in advance, or when the system must operate in non-stationary environments.

Strengths and limitations. Handles unpredictable tasks; resources are allocated only as needed. The adaptation mechanism itself is a failure source: AutoGen's auto speaker selection can misroute when conversation histories grow long or when agent role descriptions are ambiguous. Without safeguards (maximum iteration counts, token budgets, cycle detection), dynamic systems can burn through API credits on unproductive exploration.

4.6. A Decision Framework for Pattern Selection

Table 1 summarizes the tradeoffs. The decision criteria below translate them into actionable guidance.

Table 1. Comparison of multi-agent orchestration patterns.

Dimension	Centralized	Decentralized	Hierarchical	Dynamic
Control flow	Single supervisor	Peer negotiation	Tree-structured	Runtime-adaptive
Scalability	Limited (3–7 agents)	High	Moderate–High (10+)	Variable
Fault tolerance	Low (SPOF)	High	Moderate	Moderate
Debugging ease	High	Low	Moderate	Low
Token cost profile	Grows with conversation length	Grows with agent count	Grows with tree depth	Highly variable
Task suitability	Well-defined workflows	Open-ended exploration	Complex, decomposable	Novel, unpredictable
Example frameworks	LangGraph supervisor, AutoGen GroupChat	CrewAI delegation, A2A peers	LangGraph nested graphs, MetaGPT, Magentic-One	AutoGen auto-select, GPTSwarm, DyLAN

Decision criteria:

1. Is the task structure known at design time? If yes, prefer centralized or hierarchical. If not, prefer dynamic.
2. Are there more than ~10 agents? Hierarchical is likely necessary; flat centralized topologies saturate the supervisor's context window.
3. Must the system tolerate individual agent failures? Decentralized or hierarchical patterns offer natural redundancy; centralized patterns require explicit failover logic.

 doi:10.20944/preprints202604.2147.v1

4. Is token cost a binding constraint? Hierarchical patterns with structured outputs (MetaGPT-style SOPs) offer the best cost-per-task ratios. GPTSwarm's learned topologies achieve 20x cost reduction over hand-designed dynamic alternatives.
5. Does the task require cross-organizational collaboration? A2A-based decentralized patterns are the only viable option when agents span trust boundaries.

Most production systems blend patterns. Figure 7 summarizes this decision flow and shows how the Dynamic modifier layers over any of the three base topologies.

Note on the dynamic dimension. Practitioners should first select a base topology (centralized, decentralized, or hierarchical) based on task structure, agent count, and fault-tolerance requirements, and then determine whether runtime adaptation is needed. Dynamic behavior is a modifier, not a fourth mutually exclusive option. A hierarchical backbone might use dynamic speaker selection inside each team; a centralized supervisor might hand off sub-problems to decentralized peer groups. The right combination depends on task structure, agent count, fault-tolerance requirements, and cost budget.

5. Frameworks for Multi-Agent Orchestration

The taxonomy in Section 4 is intentionally orthogonal to the framework survey that follows. A single framework may support multiple orchestration patterns, and the same pattern can be implemented through very different runtime architectures.

Note on Benchmark Comparability.

All framework-specific benchmark numbers cited in this section (HumanEval scores, SWE-bench rates, tokens-per-line metrics) are drawn from each framework's original publication, tested under different model versions, task subsets, and evaluation dates. Direct cross-framework numeric comparison is not supported. These figures are cited to characterize each framework's design intent and the conditions under which it was evaluated, not to establish a performance ranking. Section 8.2 discusses this limitation in detail; for this reason we deliberately refrain from ranking frameworks by benchmark score throughout this survey.

The framework ecosystem in early 2026 is differentiated not by feature checklists—every major framework supports tool use, memory, and multi-agent coordination—but by design philosophy. The fundamental question each framework answers differently is: what is the right primitive for organizing agent interaction? LangGraph answers "a stateful graph." CrewAI answers "a role-playing team." AutoGen answers "a multi-turn conversation." The OpenAI Agents SDK answers "a handoff." These choices ramify through every aspect of the developer experience, from how state is managed to how failures propagate. Figure 3 illustrates the distinct profiles of each framework across seven evaluation dimensions.

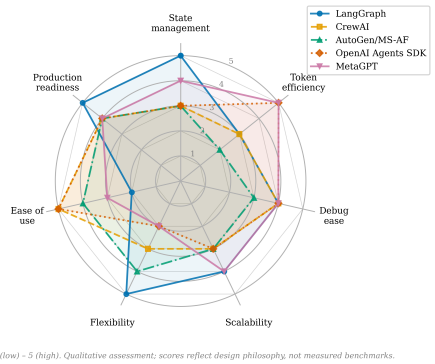

Figure 3. Framework comparison across seven evaluation dimensions. Each framework exhibits a distinctive profile reflecting its design philosophy: LangGraph excels in state management and debugging, CrewAI in ease of use, AutoGen in flexibility, and the OpenAI Agents SDK in minimal overhead.

 doi:10.20944/preprints202604.2147.v1

5.1. LangGraph: The Graph as State Machine

LangGraph, developed by LangChain, models agent workflows as stateful directed graphs [30]. Its positioning is deliberately low-level: explicit about control flow, opinionated about state, and unapologetic about the learning curve that entails.

Core abstractions. The StateGraph is the central construct. Nodes are computation units (LLM calls, tool invocations, Python functions) that receive the current state and return a delta. Conditional edges evaluate state predicates to define execution flow. State travels along edges as a typed data structure (TypedDict or Pydantic model); LangGraph merges node outputs back into global state via reducer functions defined in the schema, enabling concurrent agents to contribute to the same state field without clobbering each other (per LangGraph documentation [30]). This is functional-programming thinking applied to agent orchestration.

Multi-agent patterns. LangGraph natively supports supervisor (a routing node delegates to workers), hierarchical (nested sub-graphs invoked as nodes), and swarm (agent-driven handoffs through dynamic edges) patterns, covering three of the four categories in our taxonomy. The conditional-edge mechanism supports the dynamic pattern as well.

State management. Built-in checkpointing persists graph state to configurable backends (in-memory, SQLite, PostgreSQL, and other pluggable options per LangGraph documentation [30]). Every state mutation creates a checkpoint, enabling pause/resume, human-in-the-loop approval gates, replay, and rollback to any prior state—among the strongest state-management guarantees in the current open-source ecosystem.

Strengths and limitations. Transparency: the graph is inspectable, visualizable, and debuggable. Fine-grained control suits complex, mission-critical workflows (LangChain lists Klarna, LinkedIn, and Elastic among reported enterprise users (vendor-reported†) [30]). The framework is verbose for simple tasks: even a two-agent handoff requires defining a state schema, nodes, edges, and a compilation step. Conditional-edge graphs with many branches become hard to read, and the learning curve is steeper than more declarative alternatives.

5.2. CrewAI: The Role-Playing Team

CrewAI is a high-level, role-based framework for multi-agent collaboration, independent of LangChain, with the design philosophy of minimizing ceremony and maximizing the metaphor [32].

Core abstractions. An Agent is defined by role, goal, and backstory—fields that shape behavior through role-playing prompts. A Task specifies what needs to be done, the expected output format, and inter-task dependencies. A Crew groups agents and tasks under a process model. Flows add an enterprise-oriented layer with event-driven control and conditional branching for production deployments.

Process models. Sequential mode executes tasks in list order, each receiving the prior task's output as context. Hierarchical mode uses a manager agent to delegate based on runtime capability assessment, validate intermediate results, and aggregate the final output. The manager makes dynamic delegation decisions at each step, incurring a token cost premium proportional to crew size.

Memory. CrewAI exposes a unified memory interface spanning short-term, long-term, entity, and user memory, backed by a vector store with a composite retrieval score weighting semantic similarity, recency, and importance (per CrewAI documentation [32]). Specific backend choices and similarity thresholds are implementation details that vary across framework versions.

Strengths and limitations. The role-based metaphor is immediately intuitive to non-specialists; the project reports substantial enterprise adoption (vendor-reported, unverified†) [32]. Non-standard workflows that do not fit the crew-and-task metaphor can be awkward to express, and there is no low-level graph control.

 doi:10.20944/preprints202604.2147.v1

5.3. AutoGen/Microsoft Agent Framework: The Conversation as Coordination

AutoGen, from Microsoft Research [28], introduced conversation-driven multi-agent orchestration. Its core insight is that natural-language dialogue can serve as the primary coordination mechanism. In October 2025, Microsoft merged AutoGen with Semantic Kernel to form the Microsoft Agent Framework, combining AutoGen's multi-agent patterns with Semantic Kernel's enterprise features: session-based state management, type safety, middleware pipeline, OpenTelemetry instrumentation, and Azure AI Foundry integration [39].

AutoGen architecture. The ConversableAgent base class supports LLM generation, tool execution, and human input, configured via system_message, llm_config, human_input_mode (NEVER/ALWAYS/TERMINATE), and code_execution_config. Specialized subclasses include AssistantAgent (LLM-backed) and UserProxyAgent (code execution and human solicitation). GroupChat manages multi-agent turn-taking through a GroupChatManager supporting four speaker-selection policies (auto, round_robin, random, manual) plus custom selection functions.

A note on auto speaker selection. The auto mode—LLM-based selection over the full conversation history—is the most powerful and the most problematic. As conversation length grows, token cost rises linearly and selection reliability degrades: ambiguous role descriptions or overlapping agent capabilities can cause consistent misrouting. AutoGen provides Transform Messages utilities to control context length, but these are manual mitigations, not architectural solutions. Many production deployments use custom selection functions that encode domain-specific routing logic, effectively reimplementing a lightweight supervisor within the conversational paradigm.

Microsoft Agent Framework. The unified framework adds graph-based workflow orchestration, native Azure AI Foundry integration, Magentic orchestration (a manager agent that dynamically constructs a task ledger), and support for Python and .NET with MCP and A2A as first-class citizens [39].

Strengths and limitations. The conversation paradigm is natural for tasks that benefit from multi-turn dialogue: brainstorming, iterative refinement, adversarial review. AutoGen's original paper [28] has amassed thousands of citations per Google Scholar, making it the most referenced framework in the field. Conversation-based coordination is token-expensive for structured workflows; a sequential pipeline expressed as a GroupChat conversation burns tokens on turn-selection overhead that LangGraph's graph or CrewAI's sequential process would avoid.

5.4. OpenAI Agents SDK: The Handoff as Primitive

Released in March 2025, the Agents SDK is the production successor to OpenAI's experimental Swarm project [40]. Its design philosophy is radical minimalism: as few primitives as possible, with escape hatches for everything else.

Core primitives. Four constructs cover the essentials: (1) Agent, an LLM with instructions and tools; (2) Handoff, a mechanism for transferring the conversation and its full context to another agent, implemented as a transfer_to_ tool call; (3) Guardrails, input and output validation checks that run in parallel with agent execution; and (4) Tracing, built-in observability for debugging and monitoring.

Handoff mechanics. Input filters control what conversation history the receiving agent can see. The on_handoff callback enables side effects at the moment of transfer. The nest_handoff_history option compresses prior transcript into a single assistant message, addressing context-window bloat after multiple sequential handoffs. And input_type defines a schema for handoff parameters, making handoffs structured rather than purely implicit.

Strengths and limitations. Minimal API surface; onboarding takes hours, not days. Guardrails elevate safety to a first-class design concern. Direct MCP server integration confirms that MCP has become a de facto standard for tool connectivity. Sessions with pluggable backends (SQLite, Redis, Dapr) provide persistent memory. However, the handoff pattern alone may be too thin for complex multi-agent scenarios requiring fine-grained coordination; there is no way to express that two agents must both complete before a third starts without building that logic outside the SDK.

 doi:10.20944/preprints202604.2147.v1

5.5. Other Notable Frameworks

AgentScope (Alibaba) uses a message-exchange communication model augmented with an actor-based distribution framework [41]. The Actor model maps naturally onto agents, and the framework exploits this to offer seamless local-to-distributed deployment without code changes. Built-in retry with configurable backoff targets production robustness.

CAMEL [27] introduced inception prompting: system prompts that bootstrap autonomous two-agent dialogue while preventing failure modes such as role flipping, instruction repetition, and infinite loops. CAMEL demonstrated that structured role-playing could sustain high-quality collaboration without continuous human steering.

MetaGPT [26] encodes a software company's SOPs into LLM prompts, requiring structured artifacts at each stage. Its publish-subscribe mechanism, in which each agent subscribes only to role-relevant information, yields 100% task completion at 124.3 tokens per line versus ChatDev's 248.9 tokens per line and 2.5 human corrections per task.

DSPy [42] treats LLM pipelines as optimizable programs; its compiler automatically tunes prompts and few-shot examples to maximize a user-defined metric, achieving 25–65% improvement over standard few-shot prompting on GPT-3.5 and Llama2-13b. Published as a spotlight at ICLR 2024, DSPy's declarative philosophy points toward a future where multi-agent coordination strategies are compiled, not hand-coded.

OpenHands (formerly OpenDevin) [43] provides an open platform for AI software developers as generalist agents, supporting sandboxed code execution, web browsing, and integration with 15+ evaluation benchmarks. The CodeActAgent v1.8 with claude-3.5-sonnet reports a 26% resolve rate on SWE-Bench Lite [43]; its CodeAct architecture expresses agent actions as executable code rather than natural language instructions, enabling tighter feedback loops between action generation and execution result. Subsequent systems built on this platform, as tracked by the SWE-bench leaderboard [44], have reached substantially higher resolution rates on SWE-bench Verified.

5.6. Comparative Analysis

Table 2 compares the major frameworks along nine dimensions. The comparison that matters most, however, is philosophical.

Structured approaches (LangGraph, MetaGPT) prioritize deterministic control: the developer specifies the coordination graph or SOP, and the framework executes it. We interpret this predictability as a likely driver of adoption in production settings where reliability and auditability are non-negotiable, consistent with framework-vendor positioning (vendor-reported†) [30]; we do not claim independent evidence that reliability was the decisive factor in selection decisions. Emergent approaches (AutoGen, CrewAI) let coordination patterns arise from conversation or role-playing; they reduce upfront design effort and handle tasks whose structure is not fully known at design time, but they trade auditability for flexibility. The OpenAI Agents SDK sidesteps the debate by reducing coordination to its minimal form (handoffs) and leaving everything else to the developer, a choice that keeps the framework tractable but demands that developers re-implement patterns such as parallel execution barriers or conditional delegation that other frameworks provide out of the box.

A secondary tension concerns the unit of state. LangGraph and the Microsoft Agent Framework treat state as a first-class typed object with explicit persistence and reducer semantics. CrewAI and AutoGen treat state as an emergent property of conversation history, which can be simpler to reason about for short tasks but may become brittle for long-running workflows where context grows beyond the model's effective window. The choice of state model therefore constrains what failure-recovery strategies are available: rollback is straightforward in LangGraph because every state transition is checkpointed, but non-trivial in AutoGen where the only persistent artifact is the conversation transcript.

The asymmetry has direct operational consequences for what an operator can do when something goes wrong. AutoGen's ConversableAgent "maintains its internal context based on sent and received

 doi:10.20944/preprints202604.2147.v1

messages" and performs "implicit state inference and progress making conditioned on conversation history" [28]; the conversation log is therefore the system of record, and an attempt to roll back is necessarily a truncation of that log. Side effects produced inside an agent step—a tool that has already written a row to a database, an external API that has already been billed, a file that has already been overwritten—are not part of the truncated log and must be reversed by mechanisms outside the framework. LangGraph's checkpoint-per-mutation discipline [30] makes this distinction explicit: each mutation is a typed delta merged through a reducer, so replay can target a specific node and either re-execute or skip its side-effecting branches. The Microsoft Agent Framework occupies an intermediate position by adding session-scoped persistence on top of AutoGen's conversation paradigm [39]. For practitioners working under compliance regimes that demand precise audit trails (HIPAA-aligned clinical workflows, SOX-governed financial pipelines, or the human-oversight obligations imposed by the EU AI Act on high-risk systems [58]), the state model is therefore not an internal implementation detail but a binding architectural choice: conversation-as-state rules out fine-grained rollback by construction, whereas explicit graph state preserves the option even if it costs additional engineering up front.

Table 2. Comparison of multi-agent orchestration frameworks.

Feature	LangGraph	CrewAI	AutoGen / MS Agent Framework	OpenAI Agents SDK	AgentScope	MetaGPT
Core abstraction	Stateful graph	Role-based crew	Conversable agents	Agent + handoff	Message-based actors	SOP-based roles
Design philosophy	Graph as state machine	Role-playing team	Conversation as coordination	Handoff as primitive	Actor model distribution	Documents replace dialogue
Orchestration patterns	All four	Sequential, hierarchical	Conversation-driven, dynamic	Swarm/handoff	Flexible	Hierarchical
State management	Checkpointed graph state with reducers	Unified memory with composite scoring (vector-backed)	Conversation history + sessions	Sessions (pluggable backends)	Distributed actor state	Shared message pool (pub-sub)
Token cost profile	Moderate	Moderate–high (hierarchical manager overhead)	High (full-broadcast, nested-chat selection)	Low (minimal overhead)	Moderate	Low (structured outputs cut ~50%)
Language support	Python, JS	Python	Python, .NET	Python, JS/TS	Python	Python
LLM agnostic	Yes	Yes	Yes	Provider-agnostic, OpenAI-optimized	Yes	Yes
Human-in-loop	Interrupt and approve via breakpoints	Task input gates	UserProxyAgent (three modes)	Guardrails (parallel validation)	Yes	Limited
Best for	Complex custom workflows, mission-critical systems	Role-based team tasks, rapid prototyping	Conversational refinement, enterprise	Lightweight handoffs, quick onboarding	Distributed deployments at scale	Software development pipelines

†*Vendor-reported, not independently verified. Tier 3 deployment claims are marked with † throughout.*

Framework comparison limitations. Table 2 reflects framework documentation as of the March 2026 literature cutoff. Framework capabilities change frequently across minor versions; benchmark results were obtained primarily with GPT-4 or GPT-4o and may not generalize to other backbones; token cost profiles depend on task structure as well as framework architecture; qualitative assessments (learning curve, debugging ease) vary by team background.

Table 7 reduces the same comparison to a feature checklist along the twelve dimensions practitioners most often ask about when choosing between frameworks. Yes/No/Preview entries reflect

 doi:10.20944/preprints202604.2147.v1

each framework's documented support as of the March 2026 cutoff; "(via X)" entries indicate that the feature is delivered through an external integration rather than a built-in primitive.

Table 7. Framework feature checklist. "Yes" indicates a built-in primitive; "Preview" indicates partial or beta support; "(via X)" indicates the feature is reachable through an external integration. Production-deployment claims that depend on vendor self-report are marked with † and explained in the row notes.

Feature	LangGraph [29,30]	CrewAI [32]	AutoGen [28]	OpenAI Agents SDK [40]	MS Agent Framework [39]
Stateful checkpointing	Yes (per-mutation; in-memory / SQLite / PostgreSQL backends)	Limited (no native checkpoint; memory store)	Limited (conversation history is the persisted artifact)	Sessions persistence (SQLite/Redis/Dapr); not arbitrary-state replay	Sessions persistence (Azure backends); not arbitrary-state replay
Built-in supervisor primitive	Yes (routing node)	Yes (manager agent in hierarchical mode)	Yes (GroupChatManager)	No (must be coded)	Yes (Magentic orchestrator)
Native multi-agent conversation	Yes	Yes	Yes (core paradigm)	Yes (handoff chain)	Yes
Handoff primitive	Yes (nested graphs + dynamic edges)	Yes (delegation primitive)	Yes (speaker selection)	Yes (transfer_to_<agent_name>)	Yes
Tool calling	Yes	Yes	Yes	Yes	Yes
Agent-to-agent messaging via standard protocol (A2A or equivalent)	Preview (community integrations)	Preview (community integrations)	No (not native to original AutoGen)	No (third-party only)	First-class (A2A)
MCP integration	Via LangChain ecosystem [29,30]	Via community connectors [32]	Via community connectors	Yes (direct MCP server integration)	First-class (MCP)
Streaming responses	Yes	Yes	Yes	Yes	Yes
Built-in memory layer (short + long term)	Yes (checkpoints + vector store)	Yes (short-term, long-term, entity, user; vector-backed composite scoring)	Conversation history (no native long-term store)	Yes (Sessions short-term; long-term via external vector store)	Yes (sessions + Azure memory services)
Failure recovery (rollback)	Yes (checkpoint replay to any prior state)	Limited (max_iter, max_retry_limit; no state rollback)	Limited (truncate conversation; no state rollback)	Limited (sessions persist; no checkpoint replay)	Partial (session persistence; not arbitrary checkpoint)
Observability built-in	Yes (LangSmith integration)	Limited (basic logging)	Limited (user-supplied)	Yes (Tracing built-in)	Yes (OpenTelemetry)
Production-deployment story	Yes (vendor-reported† users include Klarna, LinkedIn, Elastic [30])	Yes (vendor-reported†) [32]	Research-grade (Microsoft Research; superseded by MS Agent Framework Oct 2025)	Yes (March 2025 release)	Yes (October 2025 launch)

The checklist exposes three structural patterns that the prose comparison alone makes harder to see. First, only LangGraph and the Microsoft Agent Framework offer first-class, mutation-level state semantics and the corresponding rollback story (Section 5.6); CrewAI, AutoGen, and the OpenAI Agents SDK can persist state but cannot replay an arbitrary prior state without external scaffolding. Second, only the MS Agent Framework treats both A2A and MCP as first-class citizens at the framework level [39]; the others surface MCP through ecosystem connectors and treat A2A as a community integration, which has implications for cross-vendor multi-agent deployments. Third, observability is genuinely uneven: LangGraph's LangSmith integration, OpenAI's first-class Tracing, and the MS Agent Framework's OpenTelemetry pipeline are built into the framework; CrewAI and the original AutoGen rely on user-supplied logging or external monitors. Together these three patterns reduce the apparent five-way choice to a smaller set of consequential decisions: how strict the state model needs

 doi:10.20944/preprints202604.2147.v1

to be, whether cross-organization protocol speak is required, and how much production observability is acceptable to bolt on after the fact.

6. Communication Protocols for Multi-Agent Systems

Frameworks solve orchestration within a single application boundary. Protocols solve it across boundaries: between tools, between agents, and across organizations. This section examines the emerging protocol stack around the central question: why does the stack need two coordination layers (agent-to-tool and agent-to-agent), and what would go wrong if we tried to collapse them into one?

6.1. Model Context Protocol (MCP): The Vertical Layer

Anthropic open-sourced MCP in November 2024 to standardize how AI assistants connect to external data sources and tools [45]. The analogy to the Language Server Protocol (LSP) is architectural: just as LSP gave every code editor a uniform interface to language-specific analysis backends, MCP gives every LLM application a uniform interface to tools, data, and prompt templates.

Architecture. MCP is built on JSON-RPC 2.0 with a strict client-server model: Hosts (LLM applications) initiate connections; Clients reside inside hosts and maintain one-to-one sessions with Servers; Servers expose three primitives—resources (structured data), tools (callable functions), and prompts (reusable instruction templates). Two client-side primitives, sampling (server requests host LLM completion) and elicitation (server requests user input), enable bidirectional interaction. Sessions progress through initialization (capability negotiation), operation (tool calls, resource reads), and shutdown phases.

Scope as strategy. MCP is an agent-to-tool protocol and explicitly does not handle agent-to-agent communication. This sharp scoping is a strategic choice: by solving one problem well (tool and data integration), MCP achieved rapid adoption without competing against agent coordination protocols. In December 2025, Anthropic reported that within roughly its first year MCP had reached millions of monthly SDK downloads and thousands of community servers, with first-class support in ChatGPT, Claude, Cursor, Gemini, and Microsoft Copilot (vendor-reported†) [46], and announced that MCP was being donated to the Agentic AI Foundation (AAIF) under the Linux Foundation, with OpenAI and Block as co-founders [46].

6.2. Agent-to-Agent Protocol (A2A): The Horizontal Layer

Google introduced A2A in April 2025, launching it with more than 50 technology partners including Atlassian, Salesforce, SAP, and ServiceNow [33]. Where MCP connects agents to tools (vertical integration), A2A connects agents to each other (horizontal collaboration), enabling autonomous agents to discover peers, negotiate capabilities, and delegate work regardless of underlying framework or vendor.

Architecture. A2A is built on JSON-RPC 2.0 over HTTP(S) with SSE for streaming. Core abstractions include: Agent Cards—JSON documents published at /.well-known/agent.json that list an agent's name, endpoint, skills, and supported authentication flows; Tasks—the fundamental work unit with a six-state lifecycle (submitted → working → input-required → completed/failed/canceled), where the input-required state enables multi-turn interactions within a single task; Messages supporting text, structured data, and files through a typed Part system; and Artifacts—output objects produced during execution.

Governance. In June 2025, Google donated A2A to the Linux Foundation, launching with more than 100 technology partners [47]. Version 0.3 (July 2025) added gRPC support and signed security cards. As of March 2026, the protocol's A2A community repository reports a broad partner ecosystem (vendor-reported†) [33].

6.3. Why Two Layers? The Vertical-Horizontal Complementarity

The distinction between MCP and A2A reflects a genuine architectural boundary. Tools are stateless capabilities: a database query, an API call, a file read. They do not have goals, do not negotiate,

 doi:10.20944/preprints202604.2147.v1

and do not push back. Agents are stateful entities with goals, partial knowledge, and the ability to accept, refuse, or renegotiate a task. Connecting to a tool is fundamentally different from collaborating with a peer; the former is a function call, the latter is a negotiation.

Collapsing the two layers would force one of two bad outcomes: either the tool-integration protocol would need task lifecycles, capability negotiation, and multi-turn interaction (bloating a simple problem), or the agent collaboration protocol would need to handle low-level resource reads and prompt templates (diluting its focus). The MCP/A2A split avoids both by letting each protocol do one thing well. An agent can use MCP to invoke a weather API and A2A to delegate a research subtask to a peer—two fundamentally different interaction patterns handled by two purpose-built protocols sharing the same underlying transport (JSON-RPC 2.0). Figure 4 illustrates how these layers compose; Figure 5 illustrates a typical interaction sequence involving both MCP tool invocation and A2A task delegation.

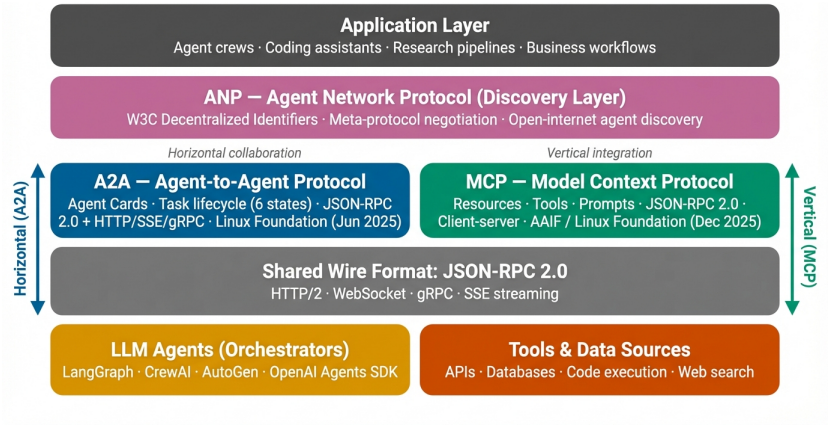

Figure 4. Emerging protocol stack for LLM-based multi-agent systems. MCP handles vertical integration (agent-to-tool), A2A handles horizontal collaboration (agent-to-agent), and ANP targets decentralized discovery on the open internet.

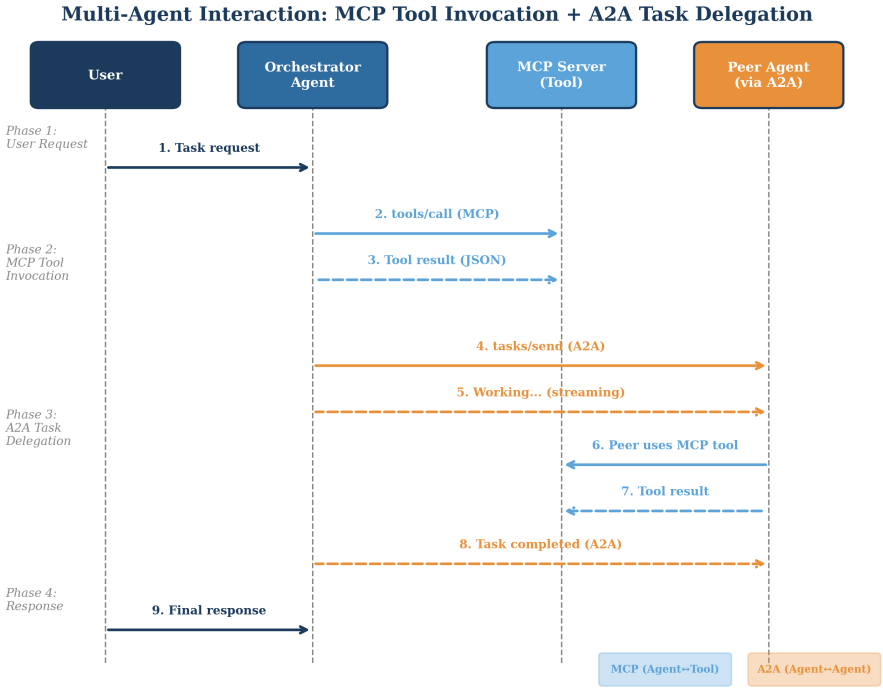

Figure 5. Representative interaction sequence combining MCP (tool invocation) and A2A (peer-agent task delegation) within a single coordinating agent's workflow.

 doi:10.20944/preprints202604.2147.v1

6.4. ACP Merger and Protocol Convergence

IBM Research launched the Agent Communication Protocol (ACP) in March 2025 for its BeeAI platform, defining a RESTful HTTP-based protocol with MIME-typed multipart messages [48]. Rather than maintain competing standards, IBM and Google merged ACP into A2A under the Linux Foundation in August 2025 [49], with the BeeAI platform transitioning to A2A compliance. In December 2025, the Agentic AI Foundation (AAIF) was established under the Linux Foundation, governing MCP while A2A is hosted by a separate Linux Foundation project, with three contributed artifacts: MCP (Anthropic), goose (Block's open-source agent framework), and AGENTS.md (OpenAI's developer instruction file standard, reported as adopted by 60,000+ open-source projects (vendor-reported†)) [46]. This convergence pattern—competitors cooperating on standards while competing on implementations—echoes successful precedents in networking (TCP/IP) and containerization (OCI).

6.5. Agent Network Protocol (ANP): The Discovery Layer

ANP targets decentralized agent discovery on the open internet, without centralized registries or pre-established trust [50]. Its three-layer design uses W3C Decentralized Identifiers (DIDs) for identity and secure communication, meta-protocol negotiation for agents to dynamically agree on communication formats, and domain-specific application protocols riding on top. Presented at a W3C WebAgents Community Group meeting in February 2025, ANP fills a genuine gap—enabling agent ecosystems spanning organizational boundaries without central coordination—but faces the classic chicken-and-egg problem of network protocols. Adoption remains early-stage.

6.6. Protocol Comparison

Table 3 compares the three active protocols. A pragmatic adoption path, as Ehtesham et al. [51] recommend: start with MCP for tool access, add A2A when structured agent collaboration becomes necessary, and extend to ANP when open-internet discovery is required.

Table 3. Comparison of agent communication protocols.

Dimension	MCP	A2A (incl. former ACP)	ANP
Primary scope	Agent-to-tool	Agent-to-agent	Agent-to-network
Wire protocol	JSON-RPC 2.0	JSON-RPC 2.0; HTTP/SSE/gRPC	DID + JSON-LD
Communication model	Client-server	P2P (6-state task lifecycle)	Decentralized P2P
Discovery mechanism	Server configuration	Agent Cards (well-known URL)	DID documents
Authentication	Bearer tokens, API keys	OAuth 2.0; signed cards (v0.3)	DID cryptography
Async support	Limited	Full (SSE, webhooks)	Full
Governance	AAIF/Linux Foundation (Dec 2025)	Linux Foundation (June 2025)	W3C CG (proposed)
Industry supporters	OpenAI, Anthropic, Block, AWS, Google, MS	100+ organizations (as of June 2025 Linux Foundation launch)	Open-source community
Maturity	High (broad vendor support)	Growing (v0.3, July 2025)	Early stage
Best for	Tool and API integration	Multi-agent collaboration	Open-internet ecosystems

Table 6 narrows the comparison to a feature-by-feature matrix that practitioners use to decide which protocol(s) a given system needs to speak. The cells are normative summaries of each protocol's design intent as documented in its own specification or in the Ehtesham et al. survey [51]; cells

 doi:10.20944/preprints202604.2147.v1

marked "(not specified)" reflect deliberate scope decisions by the protocol authors rather than missing information.

Table 6. Protocol feature matrix. Cells reflect each protocol's documented design intent as of the March 2026 cutoff; ACP is shown as a separate column for historical clarity but its successor posture is folded into A2A as of the August 2025 merger [49].

Feature	MCP [45,46,51]	A2A (incl. former ACP) [33,47,51]	ANP [50,51]	ACP (pre-merger) [48,49,51]
Transport layer	JSON-RPC 2.0	JSON-RPC 2.0 over HTTP(S); SSE for streaming; gRPC added in v0.3 (July 2025)	DID + JSON-LD over HTTP(S); compatible with OpenAPI / JSON-RPC / WebRTC	RESTful HTTP with MIME-typed multipart messages
Discovery mechanism	Server configuration; capability negotiation at session init	Agent Cards published at /.well-known/agent.json	W3C DID documents (e.g., did:wba); JSON-LD application descriptions	Runtime APIs, offline packaging, manifest-based metadata
Authentication / security	Transport-level (bearer tokens, API keys)	OAuth 2.0, API keys; signed Agent Cards (v0.3+)	DID-anchored cryptography; multi-DID privacy strategy	Transport- and session-level credentials
Streaming support	Asynchronous server notifications (limited)	Full: SSE, webhooks, push notifications	Full: peer-to-peer streams via meta-protocol layer	Both synchronous and asynchronous invocation
State management	Session-scoped (initialization → operation → shutdown)	Task object with explicit 6-state lifecycle	Stateless application layer; stateful sessions inside meta-protocol layer	Stateful sessions
Multi-turn task lifecycle	(Not in scope; tool calls are single-shot or notification-driven)	Yes: submitted → working → input-required → completed/failed/canceled	(Left to application protocol layer)	Yes: session-scoped multi-turn
Open-internet vs intranet posture	Intranet-leaning (typically inside a single trust boundary)	Both—designed for cross-org peer interaction but works intra-org	Open-internet by design (no central registries, no pre-established trust)	Both
Governance (March 2026)	AAIF under Linux Foundation	Linux Foundation A2A Project	W3C WebAgents Community Group (proposed)	Folded into A2A under LF AI & Data, August 2025

Two design choices in this table deserve explicit discussion. The first is the streaming asymmetry between MCP and A2A: MCP's notification mechanism supports asynchronous server-side updates but is described in the protocol survey as "limited" relative to A2A's first-class SSE and push-notification interfaces [51]. This is a direct consequence of the vertical-versus-horizontal scoping discussed in Section 6.3—tool calls are typically request-response, while peer collaboration is intrinsically long-running. The second is the open-internet posture of ANP, which replaces both centralized registries and pre-established OAuth flows with W3C decentralized identifiers and JSON-LD application descriptions [50]. This makes ANP the only one of the three currently active protocols that is structurally compatible with cross-organizational discovery without prior trust establishment, and it is also the reason ANP cannot simply be "bolted on" as a discovery layer over A2A—the trust model differs at the root.

7. Design Considerations

Building multi-agent systems that survive contact with production requires grappling with concerns that cut across frameworks and protocols.

 doi:10.20944/preprints202604.2147.v1

7.1. State Management

Multiple agents operating concurrently need a consistent view of shared data while maintaining private state not leaked to peers. State grows: accumulated context can overflow LLM context windows, requiring summarization or selective retrieval. Long-running workflows demand durable persistence. LangGraph checkpoints every state mutation via reducer functions, enabling replay and rollback to any prior point (per LangGraph documentation [30]). CrewAI exposes a unified memory interface spanning short-term, long-term, entity, and user memory, backed by a vector store with composite-scoring retrieval weighting semantic similarity, recency, and importance (per CrewAI documentation [32]). The Microsoft Agent Framework adds session-based state management with pluggable backends (per Microsoft Agent Framework documentation [39]). External vector databases remain the dominant choice for long-term agent memory across all frameworks [21].

MetaGPT's publish-subscribe mechanism offers an instructive solution to the shared-versus-private tension: each agent subscribes only to role-relevant messages, so the architect sees requirements but not test results, and the tester sees code but not stakeholder feedback [26]. This selective visibility prevents information overload that degrades LLM reasoning quality.

7.2. Task Planning and Decomposition

Planning strategies span a spectrum. Static planning fixes the task graph at design time (LangGraph's explicit graph structure). LLM-based dynamic planning generates decompositions at runtime (CrewAI's optional planning agent). Between these poles sit iterative refinement (ReAct-style [3]), hierarchical decomposition (MetaGPT's SOP cascade [26]), and search-based planning (LATS's Monte Carlo Tree Search [23]).

The choice has direct cost implications. Static plans are cheap but inflexible. Dynamic LLM-based plans add planning overhead but adapt to unexpected results. Search-based plans (LATS) achieve higher reliability (92.7% HumanEval) at substantially higher token cost. GPTSwarm offers a middle path: learn an efficient topology offline, execute it cheaply at inference time. OPTIMA [52] demonstrates that training-based optimization of multi-agent communication can yield up to 2.8x performance gains with fewer than 10% of the tokens on information-exchange-heavy tasks.

For task allocation, three strategies dominate: role-based (match tasks to agents by declared role [27,32]), capability-based (route via A2A Agent Card skill descriptors [33]), and dynamic (assign based on availability and past performance [28]). DSPy's compiler-based approach [42] points toward a fourth—optimized allocation—in which the system automatically tunes task-agent assignments, showing 25–65% improvement over manual prompt engineering.

7.3. Error Handling and Recovery

Failures in multi-agent systems arrive in layers: agent-level (hallucination, tool crash, context overflow), communication-level (timeout, serialization error), orchestration-level (deadlock, infinite loop, wrong routing), and infrastructure-level (network outage, resource exhaustion). Production systems combine multiple defenses into a defense-in-depth posture: AgentScope provides configurable retry with exponential backoff [41]; Reflexion enables agents to learn from failures through verbal self-critique [4]; LangGraph's checkpoints enable rollback to a known-good state [30]; the OpenAI Agents SDK's guardrails catch bad inputs proactively [40]; and CrewAI's max_iter (default 20) and max_retry_limit (default 2) impose hard boundaries to prevent runaway loops.

7.4. Scalability

Hierarchical orchestration addresses communication scaling by confining most interactions within sub-teams; tree-structured communication scales with depth rather than breadth. AgentScope's actor-based distribution deploys agents across machines transparently [41]. MetaGPT's publish-subscribe mechanism filters messages at the source, preventing broadcast storms in flat communication topologies.

 doi:10.20944/preprints202604.2147.v1

The most overlooked scalability bottleneck is the LLM context window itself. As agent count grows, the accumulated context that coordinating agents must process grows proportionally. Mitigation strategies include MetaGPT's structured documents (compressing information into denser formats than raw conversation), CrewAI's respect_context_window flag (triggering automatic summarization), and AutoGen's Transform Messages utilities. Recent work on trajectory reduction [53] demonstrates that automatically removing useless, redundant, and expired information from agent trajectories can reduce input tokens by 40–60% while maintaining performance within 1–2% of the original agent.

7.5. Security and Trust

Multi-agent systems introduce attack surfaces beyond those of single-agent applications: prompt injection cascading through inter-agent messages [54], agent impersonation, data exfiltration through tool calls, privilege escalation via handoff chains, and supply-chain attacks on MCP servers. The MAESTRO framework (Cloud Security Alliance, February 2025) provides a seven-layer threat modeling architecture specifically designed for agentic AI systems [55].

HackAPrompt [54], which collected over 600,000 adversarial prompts against three state-of-the-art models, demonstrated systemic LLM vulnerability to prompt manipulation. In a multi-agent system, compromising one agent via prompt injection can propagate through inter-agent messages. Lee et al. [56] formalize this as "Prompt Infection," a novel attack where malicious prompts self-replicate across interconnected agents. Their experiments show that multi-agent systems are highly susceptible even when agents do not publicly share all communications—the most distinctive security concern in the field.

Mitigations include the OpenAI Agents SDK's guardrails [40], MCP's capability negotiation limiting tool access to explicitly granted permissions [45], A2A's OAuth 2.0 and signed security cards [33], and sandboxed execution via AutoGen's DockerCommandLineCodeExecutor. The 2025 AI Agent Index [57] finds that 25 of 30 surveyed deployed agentic systems disclose no internal safety evaluation, a gap between deployment pace and safety infrastructure. The EU AI Act (Regulation 2024/1689), in force from August 2024, mandates risk assessment and human oversight for high-risk AI systems [58], requirements that are particularly challenging in multi-agent architectures where reasoning paths traverse multiple agents.

8. Evaluation Methodologies

Evaluating multi-agent systems is harder than evaluating single models. The stochastic nature of LLM outputs, emergent interaction effects, and the cost of running multi-agent experiments all contribute to an evaluation gap: the disconnect between orchestration framework sophistication and the primitiveness of the metrics used to assess them. As Figure 6 reveals, significant performance gaps persist between AI agents and human baselines across all major evaluation suites.

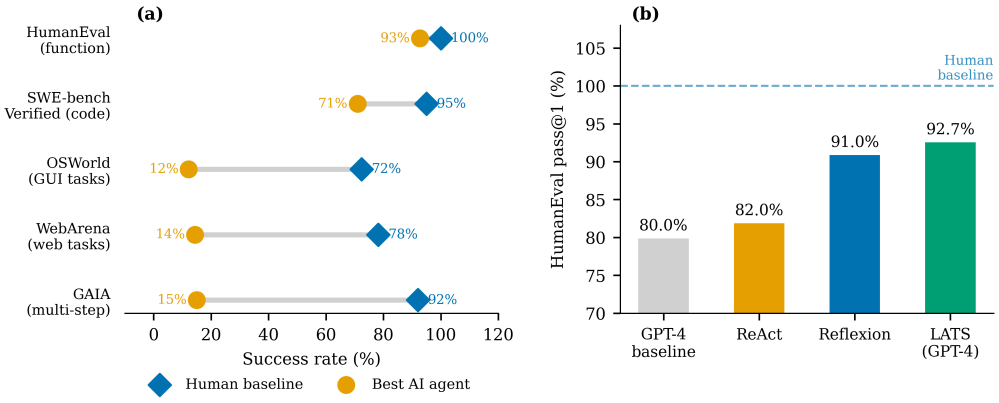

Figure 6. Best-reported agent performance versus human baselines on selected multi-step agent benchmarks (SWE-bench Verified, GAIA, WebArena, OSWorld), as of the March 2026 literature cutoff. Numbers are sourced from the primary benchmark papers and their public leaderboards; see Sections 5.6 and 8.1 for per-system citations.

 doi:10.20944/preprints202604.2147.v1

8.1. Key Benchmarks

SWE-bench [59] uses real GitHub issues and asks agents to generate correct patches, verified against test suites. When the original SWE-bench launched, Claude 2 resolved only 1.96% of 2,294 issues. SWE-Agent [60] pushed resolution on the original benchmark to 12.5% through custom agent-computer interfaces. SWE-bench Verified [61] (500 human-validated instances) and SWE-Bench Lite (a smaller curated subset, used by many open-source systems for cost reasons) are related but distinct subsets: numbers on one are not directly comparable to numbers on another, and cross-system leaderboard positions shift with the underlying model and prompting scheme. OpenHands' CodeActAgent reports 26% on SWE-Bench Lite [43]; as of late 2025–early 2026 the public SWE-bench leaderboard [44] reported top systems above 70% on SWE-bench Verified. The progression illustrates the compound effect of better agent architectures, tool interfaces, and underlying model capabilities—but also the importance of citing the specific subset and evaluation date when quoting a number.

GAIA [62] targets multi-step reasoning with tool use. Humans score 92%; GPT-4 with plugins scores 15%—a 77-percentage-point gap that reveals the cost of the orchestration deficit on tasks requiring sustained multi-tool coherence.

WebArena [63] creates a realistic web environment with 812 long-horizon tasks. The best GPT-4-based agent achieves 14.41% end-to-end success versus human performance of 78.24%, revealing the difficulty of sustained web interaction.

OSWorld [64] benchmarks multimodal agents in real computer environments. Humans accomplish 72.36% of tasks; the best model achieves only 12.24%, primarily struggling with GUI grounding and operational knowledge.

HumanEval is the baseline single-function coding benchmark; the progression from ReAct through Reflexion (91% pass@1) to LATS (92.7%) illustrates how agent reasoning advances through systematic evaluation.

AgentBench [65] evaluates LLM-as-agent across eight interactive environments (operating systems, databases, knowledge graphs, digital card games, web browsing, web shopping, and household tasks) at ICLR 2024. A finding of note: commercial models significantly outperform open-source alternatives, with the gap widest in tasks demanding sustained multi-step reasoning. A subtler finding has received less attention: code-focused training has ambivalent impacts on agent performance, improving some task categories while degrading others.

tau-bench [66] evaluates tool-agent-user interaction in real-world domains (retail and airline customer service), testing agents' ability to follow domain-specific policies while interacting with simulated users. Even state-of-the-art function-calling agents (GPT-4o) succeed on fewer than 50% of tasks, with reliability (pass^8) below 25% in the retail domain—a gap between single-call accuracy and consistent multi-turn performance.

8.2. The Evaluation Gap

No widely adopted benchmark specifically targets multi-agent orchestration—evaluating coordination patterns, protocol compliance, and system-level emergent behaviors as distinct from task outcomes. The field evaluates what agents produce, not how well they collaborate. Missing dimensions include coordination efficiency (messages per task, ratio of productive to redundant communication), scalability behavior (latency and throughput as agent count increases), robustness under adversarial conditions, cost efficiency (total token expenditure per task), and emergent behavior quality. The progression from ReAct to Reflexion (91% HumanEval pass@1) to LATS (92.7%) demonstrates that systematic evaluation drives progress; the absence of equivalent multi-agent benchmarks leaves orchestration research guided more by intuition than evidence.

The current state—where MetaGPT reports its own token efficiency, GPTSwarm reports its own cost savings, and no independent benchmark verifies either—is not sustainable. Cross-system comparisons are rarely apples-to-apples: systems differ in LLM call budgets, tool access, human-curated scaffolding, and evaluation dates (a result with GPT-4 from March 2023 is not comparable to

 doi:10.20944/preprints202604.2147.v1

one with GPT-4o from May 2024). Most benchmark results were obtained with GPT-4 or GPT-4o as the backbone; performance with other models may differ substantially both in absolute terms and in the relative ranking of orchestration approaches.

Illustrative application. Table 4a applies the six dimensions qualitatively to three representative systems, illustrating how the framework discriminates between approaches rather than merely benchmarking task performance.

Table 4a. Illustrative application of the six-dimension framework to three orchestration systems. Every cell is qualitative unless a specific number and citation is given; quantitative cells are marked with their evidence tier in parentheses.

Dimension	LangGraph	AutoGen / MS Agent Framework	MetaGPT
Task performance	Qualitative: strong on structured workflows; enterprise adoption reported by vendor (vendor-reported†) [30]	Qualitative: strong on conversational multi-turn tasks	HumanEval 85.9%, MBPP 87.7% (peer-reviewed) [26]
Coordination efficiency	Qualitative: high—explicit graph eliminates redundant messages (framework-architectural argument)	Qualitative: lower—full-broadcast GroupChat; Transform Messages needed as manual mitigation	~124 tokens/line vs ChatDev's 248.9 (peer-reviewed) [26]
Scalability	Qualitative: moderate—nested graphs scale; conditional edges unwieldy at large agent counts	Qualitative: limited—cost scales linearly with conversation length	Qualitative: high—tree hierarchy plus pub-sub filtering
Robustness	Qualitative: strong—checkpoint rollback plus human-in-loop breakpoints (per LangGraph documentation [30])	Qualitative: moderate—max_turns limits; session persistence in Microsoft Agent Framework	Qualitative: moderate—max_retry plus structured documents constrain hallucination propagation
Cost efficiency	Qualitative: moderate—no automatic conversation compression	Qualitative: lower—full history re-processed each turn	~50% token reduction vs ChatDev on software tasks (peer-reviewed) [26]
Emergent behavior	Qualitative: low risk—explicit graph limits off-script paths	Qualitative: higher risk—open conversation enables off-script behaviors	Qualitative: low risk—rigid SOPs constrain interaction

All qualitative cells are judgments from our reading of framework documentation and published case studies; absolute performance depends on model version, task distribution, and implementation details. Only cells with a citation include quantitative claims.

 doi:10.20944/preprints202604.2147.v1

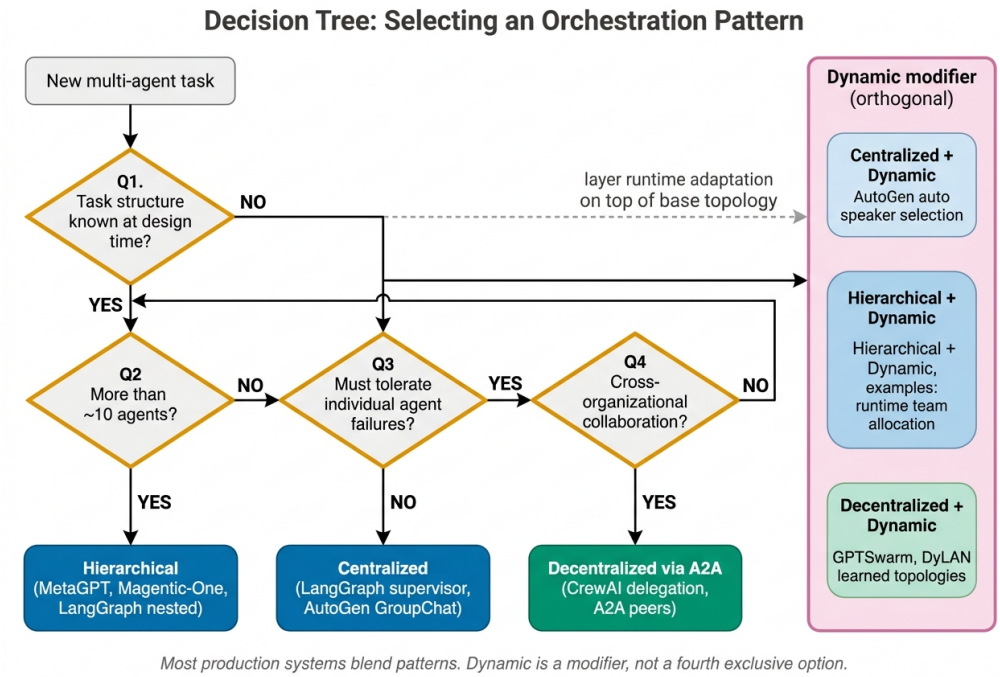

Figure 7. Decision tree for selecting an orchestration pattern (cross-referenced from Section 4.5). Four questions route a task to a base topology (Centralized, Decentralized, or Hierarchical); the Dynamic modifier is orthogonal and can be layered over any base choice.

9. Applications

Multi-agent orchestration has moved beyond proof-of-concept demonstrations into deployed systems with measurable impact. Table 5 maps domains to orchestration patterns, representative systems, key metrics, and maturity levels.

Software development is the flagship domain. MetaGPT achieves 85.9% on HumanEval and 87.7% on MBPP with 100% task completion at 124.3 tokens per line, versus ChatDev's 248.9 tokens/line and 2.5 human corrections [26,36]. OpenHands' CodeActAgent reports 26% on SWE-Bench Lite [43]; the public SWE-bench leaderboard [44] reports leading systems above 70% on SWE-bench Verified by late 2025. Adjacent agentic coding tools—Cursor, Claude Code [67], Windsurf [68], and Amazon Q Developer [69]—have transitioned from autocomplete to autonomous multi-step coding across entire codebases (these tools are primarily single-agent or minimally-orchestrated; Amazon has reported (vendor-reported†) that Q Developer generated a substantial share of new internal Amazon code during 2024 [69]). Explicitly multi-agent coding systems include MetaGPT and OpenHands, which coordinate specialized agents for planning, coding, and testing.

Scientific research systems coordinate literature review, hypothesis generation, experiment design, and manuscript preparation. The AI Scientist [70] produces full research papers at approximately $15 each; ChemCrow [71] integrates 18 chemistry tools to synthesize compounds autonomously.

Business process automation shows measurable ROI. Klarna's AI assistant handled two-thirds of support chats in early 2024—2.3 million conversations—reducing average resolution time from 11 minutes to under 2 minutes (vendor-reported†) [72]. A Heathrow deployment of Salesforce Agentforce has been reported to resolve around 90% of passenger inquiries without human escalation in early production use (vendor-reported†) [73]; ServiceNow reports AI agent-driven IT incident deflection in early deployments (vendor-reported†) [74]. Deloitte's 2025 survey (self-reported organizational data) finds organizations deploying agentic systems report 15–30% cost reduction in targeted processes, with 6–18 month payback periods [75].

Healthcare multi-agent systems apply hierarchical orchestration to clinical decision support, medical coding, and pharmacovigilance. A triage agent assesses clinical queries; specialized agents handle differential diagnosis, drug interaction checking, and guideline retrieval; a synthesis agent

 doi:10.20944/preprints202604.2147.v1

integrates findings into structured recommendations. A critical constraint is that agents must provide traceable reasoning chains that clinicians can audit, making centralized and hierarchical patterns strongly preferred over decentralized alternatives. Current systems operate in advisory modes under regulatory constraints (FDA, EU AI Act [58]).

The shape of these clinical workflows maps cleanly onto the orchestration taxonomy in Section 4. Clinical decision support is a paradigmatic centralized-with-hierarchy case (Sections 4.1, 4.3): the workflow structure is known in advance (intake → differential → safety check → synthesis), agent count is small enough to fit comfortably inside a supervisor's context window, and auditability is non-negotiable, which is exactly the regime in which Section 4.5 recommends a single supervisor over decentralized peer negotiation. Medical coding follows a hierarchical decomposition (encounter parsing → specialty-specific code candidate generation → cross-coder reconciliation → final code assignment) in which structured artifacts at each stage—analogous to MetaGPT's standardized intermediate documents [26]—become the primary mechanism for limiting cascading hallucination and producing the documentation a regulator would expect to inspect. Pharmacovigilance signal review is structurally similar but adds a stronger conservative-bias requirement: false negatives are more costly than false positives, which favors panel-of-experts patterns where multiple agents independently assess the same case before a synthesis agent integrates their findings. Across all three workflows, the state-management implications discussed in Section 5.6 apply directly: only frameworks with explicit checkpointed state (LangGraph-style [30]) can support the kind of mutation-level rollback that a clinician auditor would expect when investigating an adverse event, whereas conversation-as-state architectures cannot reconstruct the precise pre-decision system state by design.

A scoping note: the present survey does not identify a peer-reviewed report of a deployed multi-agent clinical system with measurable patient-outcome data within its included references, and the EU AI Act's high-risk classification of clinical decision-support tools [58] is itself relatively new (in force from August 2024), so quantitative outcome metrics for multi-agent healthcare deployments are not yet established at the time of this survey. The maturity entry "Pilot" in Table 5 reflects this. What is clear is the regulatory direction of travel: EU AI Act obligations on transparency, human oversight, and risk management [58], combined with the capability-versus-safety transparency gap documented for general agentic systems by the 2025 AI Agent Index (25/30 deployed systems disclose no internal safety evaluation [57]), imply that healthcare multi-agent deployments will be among the first domains where regulators demand process-level audit trails and not just outcome-level performance numbers—a requirement that the orchestration choices made today will determine whether tomorrow's systems can satisfy.

Education applications use hierarchical tutoring architectures where a student model agent tracks learner knowledge, a pedagogical agent selects teaching strategies, and a content agent retrieves and adapts learning materials. Automated grading systems use a panel-of-experts pattern—multiple evaluator agents independently assess student work against different criteria, with a synthesis agent aggregating scores. MT-Bench [76] is cited as an adjacent evaluation methodology (LLM-as-judge, not a multi-agent orchestration system per se) that validates the grading-panel approach at over 80% agreement with human annotators.

Three primitives from elsewhere in the survey map directly onto these education patterns. First, the panel-of-experts grading workflow is structurally identical to AgentVerse's Expert Recruitment + Collaborative Decision-Making + Evaluation stages [34], in which expert descriptions are generated dynamically against the goal and the group composition is then adjusted based on feedback from the evaluation stage. Translated into a grading context, this means the rubric criteria themselves drive expert recruitment, and disagreement among graders triggers re-recruitment of additional specialist evaluators rather than a forced consensus. Second, DSPy's compiler-based optimization [42] gives a principled answer to the long-standing question of how to write the grader prompts: the rubric becomes a metric, the panel becomes a declarative pipeline of typed modules with natural-language signatures, and the compiler tunes prompts and few-shot demonstrations to maximize agreement with

 doi:10.20944/preprints202604.2147.v1

a small set of expert-graded exemplars. The reported 25–65% improvement over hand-written few-shot prompting on math, multi-hop QA, and complex pipelines [42] suggests substantial headroom for grading panels currently relying on manually authored rubric prompts. Third, Generative Agents [12] supplies a model for tutoring social dynamics: their three-component architecture of memory stream, reflection, and planning gave 25 LLM-based agents the ability to form relationships, propagate information, and coordinate group activities without being explicitly programmed to do so. The same primitives map directly onto a tutoring agent that must remember a learner's prior misconceptions (memory stream), synthesize them into a current model of what the learner does and does not understand (reflection), and select an instructional move accordingly (planning).

Two caveats temper these mappings. First, MT-Bench's 80%+ agreement rate with human annotators [76] was measured on single-judge LLM evaluation of pairwise model-output comparisons rather than on multi-agent grading panels operating against a structured rubric; its limitations—position bias, verbosity bias, self-enhancement bias [76]—transfer directly into the panel setting and are arguably amplified when several judges share a common model family. Second, the included references do not contain a peer-reviewed report of a production-scale education deployment with measurable learning-outcome data; current evidence is from research demonstrations and small-scale pilots, which is the maturity tag assigned in Table 5. The combination of strong evaluation-methodology grounding (MT-Bench), strong compilation primitives (DSPy), and strong social-dynamics primitives (Generative Agents, AgentVerse) means the building blocks exist; assembling them into deployed K–12 or university tutoring systems with rigorous outcome studies remains an open task.

Autonomous systems. VOYAGER [77] achieves 3.3x unique item acquisition and up to 15.3x milestone speed versus prior methods in Minecraft using only GPT-4 prompting with no fine-tuning. Its skill library—executable code stored and composed incrementally—offers a model for multi-agent knowledge sharing where multiple agents contribute to and draw from a shared skill repository. Claude Computer Use [78], released October 2024, enables agents to interpret screenshots and generate GUI interactions, pointing toward agent teams operating computers as humans do with specialized agents handling different applications coordinated by a task-level orchestrator.

Table 5. Application domains for multi-agent orchestration: representative systems, dominant patterns, key metrics, and deployment maturity.

Domain	Dominant pattern(s)	Representative systems	Key metrics (with evidence tier)	Maturity
Software development	Hierarchical, dynamic	MetaGPT, OpenHands (multi-agent); Cursor, Claude Code† (adjacent agentic)	MetaGPT HumanEval 85.9%, MBPP 87.7% (peer-reviewed) [26]; OpenHands 26% SWE-Bench Lite (peer-reviewed) [43]; SWE-bench Verified leaders >70% (leaderboard, as of March 2026) [44]	Production
Scientific research	Hierarchical, centralized	AI Scientist [70], ChemCrow [71]	~$15 per generated paper (author-reported, small-sample) [70]; 18-tool chemistry integration (peer-reviewed) [71]	Experimental
Business / customer service	Centralized, hierarchical	Klarna AI [72], Salesforce Agentforce [73], ServiceNow [74]	Klarna: two-thirds of support chats automated, resolution time reduced from 11 min to <2 min (vendor-reported†) [72]; Deloitte survey: 15–30% cost reduction in targeted processes (industry survey†) [75]	Production
Content creation	Sequential (centralized)	CrewAI content crews, news-analysis pipelines	No widely accepted quantitative benchmark at survey cutoff; qualitative productivity claims by adopters are vendor-reported† and not independently verified	Early production
Healthcare	Hierarchical, centralized	Clinical decision support, medical coding, pharmacovigilance	Advisory-only deployments under FDA and EU AI Act constraints (regulatory†) [58]; multi-agent outcome metrics not yet standardized at survey time (author interpretation)	Pilot

 doi:10.20944/preprints202604.2147.v1

Domain	Dominant pattern(s)	Representative systems	Key metrics (with evidence tier)	Maturity
Education	Hierarchical, decentralized	Multi-agent tutoring; panel-of-experts grading	MT-Bench LLM-as-judge reaches >80% agreement with human annotators (peer-reviewed) [76]; transfer of this agreement rate to full multi-agent tutoring deployments is not yet established	Pilot
Autonomous systems	Hierarchical, decentralized	VOYAGER [77], Claude Computer Use [78]	VOYAGER: 3.3× unique-item acquisition, up to 15.3× milestone speed vs prior methods in Minecraft (peer-reviewed) [77]	Experimental–Pilot

Maturity levels: Experimental (research demonstrations), Pilot (limited real-world deployment), Production (scaled commercial deployment).

10. Open Challenges and Future Directions

10.1. Protocol Bridging and Convergence

MCP, A2A, and ANP address complementary layers, but bridging them remains ad hoc. There is no standardized way to expose an MCP-connected tool as an A2A-compatible agent service, or to surface an A2A agent's capabilities through an ANP-discoverable endpoint [51]. AAIF oversees MCP while a separate Linux Foundation project hosts A2A, and ANP targets the W3C WebAgents Community Group; all three protocols evolve rapidly with breaking changes between versions and no conformance test suites. The ACP–A2A merger (Section 6.4) is an encouraging precedent—two competing standards recognized their overlap and unified under neutral governance. Extending that spirit to MCP–A2A bridging, and eventually to ANP integration, is the protocol community's most pressing task. Priorities include: bridge specifications defining MCP–A2A–ANP interoperation patterns, versioned specifications with explicit backward-compatibility guarantees, and open-source conformance test suites that frameworks can validate against.

Concrete bridging gaps follow directly from the architectural differences cataloged in Table 6. MCP's session model assumes a single trust boundary with transport-level authentication [45,51]; A2A's task model assumes peer agents that may sit on opposite sides of an organizational firewall, authenticated through OAuth 2.0 or signed Agent Cards in v0.3 [33,51]; ANP's three-layer design assumes neither central registries nor pre-established trust, replacing both with W3C decentralized identifiers and JSON-LD application descriptions [50]. Translating an MCP tool invocation into an A2A task therefore requires both (i) lifting a stateless function call into a stateful six-state task and (ii) re-authenticating the caller across a trust boundary that MCP did not anticipate. Translating an A2A Agent Card into an ANP-discoverable endpoint requires re-encoding the OpenAPI-style skill list as a DID-anchored JSON-LD graph and replacing OAuth flows with did:wba authentication [50]. None of these translations is conceptually hard, but the absence of canonical mappings means each integration is currently bespoke, with predictable interoperability failures at the seams. The Ehtesham et al. survey [51] sketches a phased adoption roadmap (MCP $\rightarrow$ A2A $\rightarrow$ ANP), but does not yet provide the bridge specifications themselves; producing them is a tractable near-term task that would disproportionately benefit cross-vendor deployments.

10.2. Self-Organizing Multi-Agent Systems

Current systems rely on human designers to pre-specify roles, topologies, and coordination rules. GPTSwarm [38] demonstrated that effective collaboration topologies can be learned via REINFORCE, automatically discovering structures resembling Tree-of-Thought and Reflexion at 1/20th the cost of hand-designed systems. Generative agents [12] showed that 25 LLM-based entities could autonomously organize social behaviors—spreading event invitations, forming relationships, coordinating arrival times—without explicit programming. AgentVerse [34] demonstrated that dynamically adjusting agent group composition during task execution outperforms static teams on collaborative problem-solving. OPTIMA [52] shows 2.8x performance gains with under 10% of tokens through training-based communication optimization. Translating these research results into

 doi:10.20944/preprints202604.2147.v1

reliable self-organization in production remains an open problem, because the conditions under which learned topologies generalize to novel task types are not yet understood. Promising directions include meta-learning for coordination optimization, evolutionary team composition algorithms, multi-agent reinforcement learning for role assignment, and DSPy-style compilation of multi-agent coordination strategies from user-defined metrics.

Two design ideas in particular sharpen the picture for what reliable self-organization would have to look like. GPTSwarm's edge-optimization formulation models the agent system as a composite graph in which each node is an LLM, tool, or decision operation, and each edge e_i carries a probability θ_i that is trained against a downstream utility via REINFORCE [38]. This makes topology selection itself a learnable variable rather than a hand-tuned hyper-parameter; the same paper shows that the optimizer can route around adversarial agents in mixed truthful-versus-adversarial swarm configurations, suggesting an additional safety dividend from learned topologies [38]. DyLAN's temporal feed-forward network [37] offers a complementary primitive: a two-stage protocol in which an unsupervised Agent Importance Score is used to prune the team during a Team Optimization stage, then the pruned team executes the task with optional early stopping. The 25.0% accuracy gain reported on certain MMLU subjects [37] comes from removing weak agents rather than from adding capacity, which inverts the usual scaling assumption that more agents are uniformly better. Both ideas are still tested at small agent counts on academic benchmarks; whether the same selection pressure remains stable under non-stationary user populations or in the presence of adversarial agents inserted across organizational boundaries (Section 7.5) is a question the field has not answered.

10.3. The Evaluation Vacuum

The field needs: standardized multi-agent benchmarks that evaluate coordination quality (communication efficiency, delegation appropriateness) rather than just task accuracy; process-oriented metrics assessing how agents collaborate; scalability testing frameworks that systematically vary agent count; reproducibility standards managing LLM stochasticity; and cost-adjusted evaluation reporting performance per dollar. Without standardized evaluation, comparing orchestration approaches remains largely anecdotal, and the field risks optimizing for metrics that do not matter while ignoring ones that do.

The benchmarks that already exist illustrate, almost by accident, the shape of what is missing. AgentBench [65] places LLM-as-Agent in 8 distinct environments spanning Operating System, Database, Knowledge Graph, Digital Card Game, Lateral Thinking Puzzles, House-Holding, Web Shopping, and Web Browsing tasks; the same authors observe that "code training could be a double-edged sword", with code-tuning improving some agent-task categories while degrading others [65]. This kind of cross-environment finding cannot be recovered from any single-environment leaderboard. τ-bench [66] adds a complementary axis by introducing the pass^k metric, which measures whether an agent solves the same task on every one of k independent trials; even gpt-4o achieves only $\sim$61% pass^1 on τ-retail and drops to $\sim$25% pass^8, exposing a reliability gap that a one-shot accuracy number entirely hides [66]. OSWorld [64] shows the gap from another angle: humans complete 72.36% of its 369 tasks while the best evaluated model reaches 12.24%, with the residual error attributed primarily to GUI grounding and operational knowledge [64]. None of these benchmarks evaluates inter-agent coordination as such—they measure a single agent in a complex environment—so the evaluation vacuum for true multi-agent metrics persists even though the ingredients (multi-environment coverage, reliability metrics, human baselines) already exist in the single-agent literature and could be recombined.

10.4. Cost Optimization at Scale

Multi-agent systems multiply LLM API costs. At enterprise scale, cost can kill a project; Gartner's 40%-cancellation prediction [15] is driven partly by runaway expenses. Yet the cost picture is not uniformly bleak: MetaGPT's structured outputs halve token consumption versus ChatDev; GPTSwarm achieves 20x cost reduction at comparable accuracy; DSPy's automatic optimization achieves 25–65%

 doi:10.20944/preprints202604.2147.v1

improvement without additional inference cost [42]. Active research areas include token-budget management across agents, communication compression via summarization agents and structured message formats, model-selection routing of routine subtasks to smaller models [79], and caching to avoid redundant LLM calls.

Although none of these levers are new in isolation, the combination opens an underexplored design space. DSPy's compiler-driven approach treats prompt content and few-shot demonstrations as parameters that are tuned against a user-defined metric rather than written by hand [42]; the reported 25–65% improvement is obtained by changing what is sent into the model, not how often it is called, which makes the gain stack with rather than substitute for the upstream architecture choices made in Section 5. Trajectory reduction [53] reports 40–60% input-token reductions with performance maintained within 1–2% of the original agent, by removing useless, redundant, and expired context from agent histories. Combined with MetaGPT's publish-subscribe filtering [26], which already restricts each agent's view to role-relevant messages, these techniques attack the same bottleneck—context-window growth—from different layers of the stack. A practical research priority is to characterize how these layers compose: does compiler-level prompt optimization survive trajectory pruning, and do publish-subscribe-filtered conversations still benefit from REINFORCE-trained edge probabilities [38]? Cost-adjusted multi-agent benchmarks (Section 10.3) are a precondition for answering these questions empirically rather than anecdotally.

10.5. Trust, Safety, and Alignment

Individually aligned agents can collectively produce misaligned outcomes—a multi-agent analog of the alignment problem [80]. When a multi-agent system produces a harmful output, attributing responsibility to a specific agent requires causal tracing through potentially long execution histories; current frameworks lack robust tools for this. Prompt Infection [56] demonstrates that adversarial prompts can self-replicate across interconnected agents, and the 2025 AI Agent Index [57] documents that only 4 of 30 surveyed agents provide agent-specific system cards. The EU AI Act [58] demands transparency, explainability, and human oversight in high-risk AI systems, requirements that are challenging when reasoning paths traverse multiple agents each with its own context. Priorities include formal verification of multi-agent behavior against safety specifications, standardized audit frameworks capturing full causal chains across agents, and adversarial red-teaming methodologies tailored to agent networks.

The empirical picture from the 2025 AI Agent Index sharpens what "trust gap" means concretely [57]. Across 30 indexed deployed agentic systems, Staufer et al. report that 25/30 disclose no internal safety evaluation, 21/30 do not disclose their AI nature to end users or third parties by default, and 10/30 publish detailed execution traces (with another subset publishing only summarized reasoning), with scope and granularity varying widely across systems [57]. Even capability disclosure is asymmetric: 9/30 systems publish capability benchmarks (typically GUI/computer-use or coding) but the same systems often lack any safety-evaluation disclosure, a pattern Staufer et al. characterize as a "weaker form of safety washing" [57]. Set against the threat model of Prompt Infection [56], in which a single malicious prompt self-replicates through inter-agent messages even when agents do not publicly share all communications, this disclosure asymmetry is concerning: the systems most likely to need agent-level audit are the ones least likely to publish what they evaluated. The deeper alignment-theoretic concern is that the failure modes Ngo et al. catalog for individual RLHF-trained systems—situationally-aware reward hacking, internally-represented misaligned goals that generalize beyond the fine-tuning distribution, and unwanted power-seeking behaviors during deployment [80]—compound rather than cancel when multiple such systems are coupled through orchestration. The EU AI Act's high-risk-system obligations (transparency, explainability, human oversight) [58] thus need a multi-agent-aware operational interpretation that current SDK observability primitives—even tracing-first ones such as the OpenAI Agents SDK [40]—only partially provide.

 doi:10.20944/preprints202604.2147.v1

11. Conclusion

This paper has examined LLM-based multi-agent orchestration across taxonomy, framework design, communication protocols, and production deployment. Systematic methodology (Section 3) identified approximately 80 primary references from an initial pool of approximately 1,200, ensuring comprehensive yet focused coverage. The findings go beyond cataloging.

On orchestration patterns: the three coordination topologies—centralized, decentralized, and hierarchical—and the dynamic adaptivity modifier are complementary building blocks, not competing alternatives. The decision framework in Section 4.5 translates abstract tradeoffs into concrete selection criteria based on task structure, agent count, fault-tolerance requirements, and cost budget. Production systems routinely blend patterns.

On frameworks: the ecosystem has differentiated along design-philosophy lines. LangGraph treats the graph as a state machine, providing maximum control at the cost of verbosity. CrewAI treats the team as a role-playing group, providing intuitive abstraction at the cost of flexibility. AutoGen (now Microsoft Agent Framework) treats conversation as coordination, providing natural multi-turn interaction at the cost of token efficiency. The OpenAI Agents SDK treats the handoff as the sole primitive, providing minimal overhead at the cost of coordination expressiveness. MetaGPT's documents-replace-dialogue approach and DSPy's compilation paradigm point toward a future where coordination strategies are optimized automatically. GPTSwarm's learned topologies—20x cost reduction over hand-designed alternatives—offer early evidence that this future is achievable.

On protocols: the stack is consolidating. MCP handles tool integration, A2A handles agent collaboration (now encompassing ACP, with a growing open-source partner ecosystem under the Linux Foundation (vendor-reported†)), and ANP targets open-internet discovery (early stage). The vertical-horizontal complementarity between MCP and A2A is architecturally principled, and the field should resist pressure to collapse the two layers.

On benchmarks: the sobering numbers—SWE-bench's initial 1.96% (on the original full benchmark), now surpassed by systems exceeding 70% on SWE-bench Verified (a distinct and smaller, human-validated subset); GAIA's 92% vs. 15% gap; WebArena's 78% vs. 14% gap; OSWorld's 72% vs. 12% gap—confirm that the gap between LLM capabilities on isolated tasks and performance on real-world multi-step problems remains substantial even as it narrows. Numbers from different SWE-bench subsets are not directly comparable. On applications, the evidence base ranges from mature (customer service with documented ROI) to experimental (autonomous scientific research), but the breadth confirms that multi-agent orchestration has moved beyond proof-of-concept. The 57.3% production deployment rate reported by LangChain [13] and the measurable ROI documented in customer service and business process automation represent a meaningful shift from research demonstrations to operational systems.

Progress in this field is driven less by any single architecture than by three interacting factors: explicit state management that survives agent failures and context overflow, disciplined coordination protocols that separate tool integration from inter-agent collaboration, and task-aligned evaluation that measures coordination quality rather than only task outcomes. Near-term advances are most likely from interoperable, auditable systems with bounded autonomy—where humans remain in the decision loop for high-stakes actions—rather than from fully autonomous multi-agent ecosystems. The path from research prototype to production system runs through the challenges documented in this survey: orchestration patterns must be chosen deliberately rather than by default, frameworks must be evaluated on design-philosophy fit rather than feature checklists, protocols must be bridged, costs must be controlled, and safety must be engineered in from the start. Closing the five open challenges identified in Section 10—protocol bridging, self-organization, evaluation methodology, cost optimization, and trust/safety—will determine whether that adoption delivers durable value.

Author Contributions: Conceptualization, Y.Z., L.L., J.Y., and D.Z.; methodology, Y.Z., L.L., J.Y., and D.Z.; investigation, Y.Z., L.L., J.Y., and D.Z.; writing—original draft preparation, Y.Z., L.L., J.Y., and D.Z.; writing—review and editing, Y.Z., L.L., J.Y., and D.Z.; validation, L.L.; formal analysis, J.Y.; data curation, D.Z.; visualization,

 doi:10.20944/preprints202604.2147.v1

Y.Z.; project administration, Y.Z. All four authors contributed equally to the conceptualization, methodology, investigation, and drafting of this work; co-authors L.L., J.Y., and D.Z. are listed in alphabetical order by surname. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable. This study is a literature survey and did not involve human participants or animal subjects.

Informed Consent Statement: Not applicable. This study is a literature survey and did not involve human participants.

Data Availability Statement: No new empirical data were created. Supplementary materials (PRISMA screening log, database query set, per-reference evidence-tier ratings, and a machine-readable serialization of the Section 4.5 decision framework) are deposited at Zenodo, DOI: 10.5281/zenodo.19774440, under the CC-BY-4.0 license. All primary sources are publicly available via the URLs and arXiv identifiers listed in the bibliography.

Acknowledgments: The authors thank the anonymous reviewers of *Future Internet* for comments that improved the manuscript.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. T. Brown et al., "Language models are few-shot learners," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 33, 2020, pp. 1877–1901.
2. OpenAI, "GPT-4 technical report," arXiv preprint arXiv:2303.08774, 2023.
3. S. Yao et al., "ReAct: Synergizing reasoning and acting in language models," in *Proc. International Conference on Learning Representations (ICLR)*, 2023. arXiv:2210.03629.
4. N. Shinn et al., "Reflexion: Language agents with verbal reinforcement learning," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2023. arXiv:2303.11366.
5. J. Wei et al., "Chain-of-thought prompting elicits reasoning in large language models," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 35, 2022. arXiv:2201.11903.
6. P. Lewis et al., "Retrieval-augmented generation for knowledge-intensive NLP tasks," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 33, 2020, pp. 9459–9474.
7. C. Qu et al., "Tool learning with large language models: A survey," *Frontiers of Computer Science*, 2024. arXiv:2405.17935.
8. T. Guo et al., "Large language model based multi-agents: A survey of progress and challenges," in *Proc. International Joint Conference on Artificial Intelligence (IJCAI)*, 2024. arXiv:2402.01680.
9. Y. Talebirad and A. Nadiri, "Multi-agent collaboration: Harnessing the power of intelligent LLM agents," arXiv preprint arXiv:2306.03314, 2023.
10. K.-T. Tran et al., "Multi-agent collaboration mechanisms: A survey of LLMs," arXiv preprint arXiv:2501.06322, 2025.
11. S. Chen et al., "A survey on LLM-based multi-agent system: Recent advances and new frontiers in application," arXiv preprint arXiv:2412.17481, 2024.
12. J. S. Park et al., "Generative agents: Interactive simulacra of human behavior," in *Proc. ACM Symposium on User Interface Software and Technology (UIST)*, 2023. arXiv:2304.03442.
13. LangChain, "State of AI Agents 2025," December 2025. Available: https://www.langchain.com/state-of-agent-engineering
14. McKinsey & Company, "The state of AI in 2025: Agents, innovation, and transformation," McKinsey Global Survey, November 2025.
15. Gartner, "Gartner predicts 40% of enterprise apps will feature task-specific AI agents by 2026," Gartner Newsroom, August 2025; Gartner, "Gartner predicts over 40% of agentic AI projects will be canceled by end of 2027," Gartner Newsroom, June 2025; Gartner, "Gartner predicts agentic AI will autonomously resolve 80% of common customer service issues by 2029," Gartner Newsroom, March 2025.
16. LangChain Blog, "How and when to build multi-agent systems," 2024. Available: https://blog.langchain.com/how-and-when-to-build-multi-agent-systems/
17. M. Wooldridge, *An Introduction to MultiAgent Systems*, 2nd ed. Chichester: John Wiley & Sons, 2009.

 doi:10.20944/preprints202604.2147.v1

18. L. Wang et al., "A survey on large language model based autonomous agents," *Frontiers of Computer Science*, vol. 18, no. 6, 2024. arXiv:2308.11432.
19. Z. Xi et al., "The rise and potential of large language model based agents: A survey," arXiv preprint arXiv:2309.07864, 2023.
20. J. Zhao et al., "A survey of large language models," arXiv preprint arXiv:2303.18223, 2023.
21. Z. Zhang et al., "A survey on the memory mechanism of large language model based agents," arXiv preprint arXiv:2404.13501, 2024.
22. S. Yao et al., "Tree of Thoughts: Deliberate problem solving with large language models," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2023. arXiv:2305.10601.
23. A. Zhou et al., "Language Agent Tree Search unifies reasoning, acting, and planning in language models," arXiv preprint arXiv:2310.04406, 2024.
24. Foundation for Intelligent Physical Agents (FIPA), "FIPA ACL Message Structure Specification," 2002. Available: http://www.fipa.org/specs/fipa00061/
25. Y. Du et al., "Improving factuality and reasoning in language models through multiagent debate," in *Proc. International Conference on Machine Learning (ICML)*, 2024. arXiv:2305.14325.
26. S. Hong et al., "MetaGPT: Meta programming for a multi-agent collaborative framework," in *Proc. International Conference on Learning Representations (ICLR)*, 2024. arXiv:2308.00352.
27. G. Li et al., "CAMEL: Communicative agents for 'mind' exploration of large language model society," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2023. arXiv:2303.17760.
28. Q. Wu et al., "AutoGen: Enabling next-gen LLM applications via multi-agent conversation," arXiv preprint arXiv:2308.08155, 2023.
29. LangChain, "LangGraph multi-agent workflows," LangChain Blog, 2024. Available: https://blog.langchain.com/langgraph-multi-agent-workflows/
30. LangChain, "LangGraph: Agent orchestration framework for reliable AI agents," 2024–2025. Available: https://www.langchain.com/langgraph
31. S. Abdelnabi et al., "LLM-deliberation: Evaluating LLMs with interactive multi-agent negotiation games," arXiv preprint arXiv:2309.17234, 2023.
32. CrewAI, "The leading multi-agent platform," 2024–2025. Available: https://crewai.com/
33. Google, "Agent2Agent protocol (A2A)," 2025. Available: https://github.com/a2aproject/A2A
34. J. Chen et al., "AgentVerse: Facilitating multi-agent collaboration and exploring emergent behaviors," in *Proc. International Conference on Learning Representations (ICLR)*, 2024. arXiv:2308.10848.
35. A. Fourney et al., "Magentic-One: A generalist multi-agent system for solving complex tasks," arXiv preprint arXiv:2411.04468, 2024.
36. X. Qian et al., "Communicative agents for software development," in *Proc. Association for Computational Linguistics (ACL)*, 2024. arXiv:2307.07924.
37. Z. Liu et al., "A dynamic LLM-powered agent network for task-oriented agent collaboration," in *Proc. Conference on Language Modeling (COLM)*, 2024. arXiv:2310.02170.
38. M. Zhuge et al., "GPTSwarm: Language agents as optimizable graphs," in *Proc. International Conference on Machine Learning (ICML)*, 2024. arXiv:2402.16823.
39. Microsoft, "Introducing Microsoft Agent Framework: The open-source engine for agentic AI apps," Microsoft Foundry Blog, October 2025. Available: https://devblogs.microsoft.com/foundry/introducing-microsoft-agent-framework-the-open-source-engine-for-agentic-ai-apps/
40. OpenAI, "OpenAI Agents SDK," 2025. Available: https://openai.github.io/openai-agents-python/
41. D. Gao et al., "AgentScope: A flexible yet robust multi-agent platform," arXiv preprint arXiv:2402.14034, 2024.
42. O. Khattab et al., "DSPy: Compiling declarative language model calls into self-improving pipelines," in *Proc. International Conference on Learning Representations (ICLR)*, 2024. arXiv:2310.03714.
43. X. Wang et al., "OpenHands: An open platform for AI software developers as generalist agents," in *Proc. International Conference on Learning Representations (ICLR)*, 2025. arXiv:2407.16741.
44. SWE-bench, "SWE-bench leaderboard," Available: https://www.swebench.com, accessed March 2026.
45. Anthropic, "Introducing the Model Context Protocol," Anthropic News, November 2024. Available: https://www.anthropic.com/news/model-context-protocol
46. Anthropic, "Donating the Model Context Protocol and establishing the Agentic AI Foundation," Anthropic News, December 2025. Available: https://www.anthropic.com/news/donating-the-model-context-protocol-and-establishing-of-the-agentic-ai-foundation

 doi:10.20944/preprints202604.2147.v1

47. Linux Foundation, "Linux Foundation launches the Agent2Agent Protocol project to enable secure, intelligent communication between AI agents," Linux Foundation Press Release, June 23, 2025. Available: https://www.linuxfoundation.org/press/linux-foundation-launches-the-agent2agent-protocol-project-to-enable-secure-intelligent-communication-between-ai-agents
48. IBM Research, "Agent Communication Protocol (ACP)," 2025. Available: https://research.ibm.com/projects/agent-communication-protocol
49. LF AI & Data Foundation, "ACP joins forces with A2A under the Linux Foundation's LF AI & Data," August 2025. Available: https://lfaidata.foundation/communityblog/2025/08/29/acp-joins-forces-with-a2a-under-the-linux-foundations-lf-ai-data/
50. G. Chang, R. Ai, T. Shi, G. Hsu, W. Xu, H. Wang, and C. Liu, "Agent Network Protocol technical white paper," arXiv preprint arXiv:2508.00007, 2025. Available: https://agent-network-protocol.com/
51. A. Ehtesham et al., "A survey of agent interoperability protocols: MCP, ACP, A2A, and ANP," arXiv preprint arXiv:2505.02279, 2025.
52. W. Chen et al., "Optima: Optimizing effectiveness and efficiency for LLM-based multi-agent system," in *Findings of the Association for Computational Linguistics (ACL)*, 2025. arXiv:2410.08115.
53. H. Xu et al., "Reducing the cost of LLM agents through trajectory reduction (AgentDiet)," arXiv preprint arXiv:2509.23586, 2025.
54. S. Schulhoff et al., "Ignore this title and HackAPrompt: Exposing systemic weaknesses of LLMs through a global-scale prompt hacking competition," in *Proc. Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2023. arXiv:2311.16119.
55. Cloud Security Alliance, "MAESTRO: Agentic AI threat modeling framework," CSA Blog, February 2025. Available: https://cloudsecurityalliance.org/blog/2025/02/06/agentic-ai-threat-modeling-framework-maestro
56. D. Lee et al., "Prompt Infection: LLM-to-LLM prompt injection within multi-agent systems," arXiv preprint arXiv:2410.07283, 2024.
57. L. Staufer et al., "The 2025 AI Agent Index," arXiv preprint arXiv:2602.17753, 2026.
58. European Parliament and Council, "Regulation (EU) 2024/1689 laying down harmonised rules on artificial intelligence (AI Act)," *Official Journal of the European Union*, August 2024.
59. C. E. Jimenez et al., "SWE-bench: Can language models resolve real-world GitHub issues?" in *Proc. International Conference on Learning Representations (ICLR)*, 2024. arXiv:2310.06770.
60. J. Yang et al., "SWE-agent: Agent-computer interfaces enable automated software engineering," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2024. arXiv:2405.15793.
61. OpenAI, "Introducing SWE-bench Verified," OpenAI Blog, August 2024. Available: https://openai.com/index/introducing-swe-bench-verified/
62. G. Mialon et al., "GAIA: A benchmark for general AI assistants," in *Proc. International Conference on Learning Representations (ICLR)*, 2024. arXiv:2311.12983.
63. S. Zhou et al., "WebArena: A realistic web environment for building autonomous agents," in *Proc. International Conference on Learning Representations (ICLR)*, 2024. arXiv:2307.13854.
64. T. Xie et al., "OSWorld: Benchmarking multimodal agents for open-ended tasks in real computer environments," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2024. arXiv:2404.07972.
65. X. Liu et al., "AgentBench: Evaluating LLMs as agents," in *Proc. International Conference on Learning Representations (ICLR)*, 2024. arXiv:2308.03688.
66. S. Yao et al., "tau-bench: A benchmark for tool-agent-user interaction in real-world domains," arXiv preprint arXiv:2406.12045, 2024.
67. Anthropic, "Introducing Claude Code," Anthropic Blog, February 2025. Available: https://www.anthropic.com/news/claude-code
68. Codeium, "Windsurf: The first agentic IDE," 2024. Available: https://codeium.com/windsurf
69. Amazon Web Services, "Amazon Q Developer product page and developer productivity claims," 2024–2025. Available: https://aws.amazon.com/q/developer/ (Quantitative deployment figures attributed to Q Developer in this paper are self-reported by AWS in connected press materials and blog posts linked from this page.)
70. C. Lu et al., "The AI Scientist: Towards fully automated open-ended scientific discovery," arXiv preprint arXiv:2408.06292, 2024.
71. A. M. Bran et al., "ChemCrow: Augmenting large-language models with chemistry tools," *Nature Machine Intelligence*, vol. 6, pp. 525–535, 2024. arXiv:2304.05376.

 doi:10.20944/preprints202604.2147.v1

72. Klarna, "Klarna AI assistant handles two-thirds of customer service chats in its first month," Klarna Press Release, February 2024.
73. Salesforce, "Heathrow Airport taps Salesforce Agentforce to enhance passenger experience," Salesforce UK Newsroom, June 11, 2025. Available: https://www.salesforce.com/uk/news/press-releases/2025/06/11/heathrow-airport-agentforce-passenger-experience/ (A 90% inquiry-resolution figure is self-reported by the Heathrow customer in this press release; the Agentforce product page at https://www.salesforce.com/agentforce/ is the general product reference.)
74. ServiceNow, "Now Assist and AI Agents for IT Service Management," ServiceNow Product Materials and Press Room, 2025. Available: https://www.servicenow.com/company/media/press-room.html (Incident-deflection figures cited here are self-reported by ServiceNow in customer case studies linked from this press room.)
75. Deloitte, "The state of AI in the enterprise, 2026 AI report," Deloitte US, 2025.
76. L. Zheng et al., "Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2023. arXiv:2306.05685.
77. G. Wang et al., "VOYAGER: An open-ended embodied agent with large language models," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 36, 2023. arXiv:2305.16291.
78. Anthropic, "Introducing computer use: Claude 3.5 Sonnet," Anthropic, October 2024. Available: https://www.anthropic.com/news/3-5-models-and-computer-use
79. T. Masterman et al., "The landscape of emerging AI agent architectures for reasoning, planning, and tool calling: A survey," arXiv preprint arXiv:2404.11584, 2024.
80. J. Ngo et al., "The alignment problem from a deep learning perspective," arXiv preprint arXiv:2209.00626, 2023.