future internet

MDPI

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; yz167@rice.edu
[2] Department of Computer Science, University of Wisconsin–Madison, Madison, WI 53706, USA; lihe.liu@wisc.edu
[3] Department of Statistics, The George Washington University, Washington, DC 20052, USA; jiaqian7@gwu.edu
[4] Daniel J. Epstein Department of Industrial and Systems Engineering, University of Southern California, Los Angeles, CA 90089, USA
* Correspondence: dzhang22@usc.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), with explicit attention to the evidence hierarchy used to interpret deployment claims. 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 six leading frameworks (LangGraph, CrewAI, AutoGen/Microsoft Agent Framework, OpenAI Agents SDK, MetaGPT, and DSPy) 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. Vendor-reported figures are marked † throughout and held to a documented evidence hierarchy, which separates them from peer-reviewed and government-evaluator measurements. We close by identifying eight open challenges and proposing a six-dimension evaluation framework for multi-agent coordination quality. This paper offers practitioners a decision framework covering taxonomy, framework selection, protocol adoption, and early operational pilots.

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

Check for updates

Academic Editor: Xueyi Li

Received: 29 April 2026
Revised: 3 June 2026
Accepted: 8 June 2026
Published: 15 June 2026

Copyright: © 2026 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license.

1. Introduction

Research in artificial intelligence has accelerated sharply over the past three years, driven by large language models (LLMs) such as GPT, 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 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 is moving quickly, although the headline figures are predominantly vendor- or analyst-reported and should be read against the evidence hierarchy laid out in Section 3.1 and the discrepancy cases cataloged in Section 7.6. LangChain's 2025 State of AI Agents report (1340 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†) [15,16]. Across these sources the publishing organization is also the party with the largest commercial interest in the reported numbers, and—as Section 7.6 documents in detail—peer-reviewed or government-evaluator replications of vendor productivity claims for deployed multi-agent systems are essentially absent at the March 2026 cutoff. The gap between ambition and delivery and between vendor framing and independently observed outcomes is precisely where orchestration research matters the 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 [17]. 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, and hierarchical) and one orthogonal adaptivity axis (dynamic–adaptive), 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, and 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 seven sectors, and eight open challenges grounded in benchmark data (Sections 8–10).

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 [18] and single-agent LLM surveys [19,20] 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.

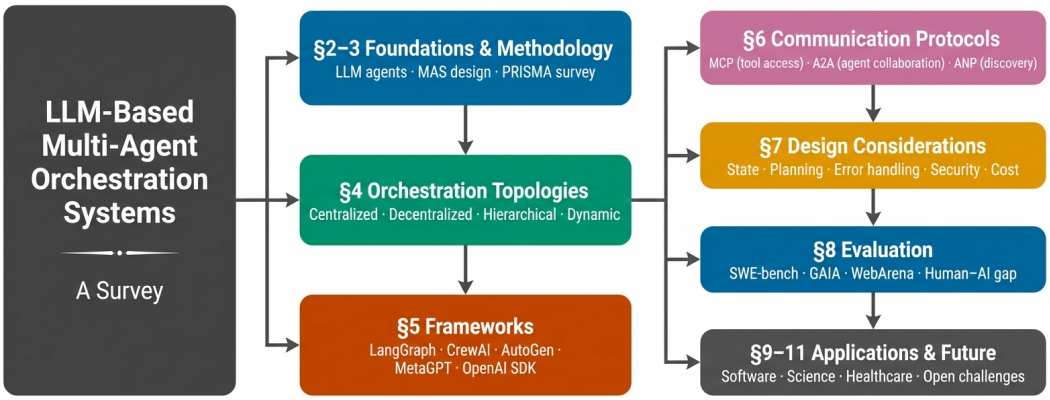


Figure 1. Organization of this survey. This paper progresses from foundations and methodology through orchestration topologies, frameworks, and communication protocols to design considerations, evaluation, and applications.

2. Background

2.1. From LLMs to LLM-Based Agents

At the bottom, an LLM is a neural network trained on massive text corpora to generate and reason about language [21]. 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. [19] 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 [22]), 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.

Tree of Thoughts (ToT) [23] generalizes chain-of-thought by enabling exploration over multiple reasoning paths. LATS [24] 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 (Figure 1): 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 [18]. Classical research contributed agent communication languages (FIPA-ACL [25]), coordination mechanisms such as the contract-net protocol, and organizational models including holarchies. LLM-based MASs 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 and injection attacks) [9]. Third, LLM reasoning enables emergent capabilities that rule-based agents cannot replicate; Du et al. [26] 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 [27,28], communication topology (broadcast, point-to-point, and publish–subscribe), coordination mechanism (centralized, decentralized, or hierarchical), and memory architecture (shared, private, or hybrid) [22]. 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 [27].

2.3. The Orchestration Challenge

Scale makes orchestration unavoidable. A software development crew—product manager, architect, developer, tester, and 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 (e.g., requirements documents, UML diagrams, and 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 [27].

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.

2.4. Two Perspectives on Coordination: Workflow Orchestration vs. Organization-Level Coordination

The literature on LLM-based multi-agent coordination has developed along two parallel lines that this survey will treat as complementary rather than competing. The first, which we term workflow orchestration, treats coordination as the problem of routing a fixed (or routing-dynamic) set of agents through a task graph; control flow, state passing, and failure recovery are the dominant concerns, and frameworks such as LangGraph [29], CrewAI [30], and the Microsoft Agent Framework [31] are representative. The second, which we term organization-level coordination, treats coordination as the problem of *constituting* the agent collective itself—deciding which agents exist, what roles they hold, and how they recruit, retire, or restructure one another at runtime; AgentVerse's four-stage expert-recruitment loop [32] is the canonical example, and GPTSwarm [33] and DyLAN [34] extend the same perspective with learned topology and team selection. These are not separate orchestration patterns but distinct framings of the same underlying problem; we unify them through the topology-by-adaptivity taxonomy in Section 4 and revisit their cross-cutting threads in Section 4.3.

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 the strict systematic-review methodology (including registered protocol, dual-reviewer screening, and GRADE evidence grading) impractical at the March 2026 cutoff.

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 to 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 [3,8,27,35] supplemented keyword searches.

Screening flow (PRISMA-style). (1) Initial retrieval: Approximately 1200 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 were assessed. (3) After title-and-abstract screening, approximately 200 survivors were 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 from the systematic screen; the major revision then added 24 further references in response to reviewer requests (self-evolving frameworks, learned-orchestration generalization, security, and the empirical-study sources), bringing the bibliography to 104 numbered entries. All stage counts are estimates; exact per-database counts are available from the corresponding author.

Inclusion criteria. Peer-reviewed publications or high-citation preprints ($\geq$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 ($\geq$10 citations) and official specifications from standards bodies (e.g., Linux Foundation and 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.

3.1. Evidence Hierarchy in This Survey

Because the multi-agent LLM literature mixes peer-reviewed research, fast-moving preprints, vendor announcements, large-N industry surveys, regulatory text, and open standards, a single "citation" label collapses important differences in evidence strength. We therefore apply a six-label hierarchy throughout the manuscript and tag quantitative claims by tier. The hierarchy refines the three-tier scheme introduced in Section 3 by splitting the previously generic Tier 3 ("industry") into the four sub-types whose epistemic weight differs materially. The labels are ordered by the strength of independent verification typically available, not by recency or popularity.

Peer-reviewed. Publications in juried venues with at least one round of external review (NeurIPS, ICML, ICLR, ACL, EMNLP, IEEE/ACM conferences and journals, and MDPI journals with documented peer review). Numbers and methodological claims from this

tier are treated as the strongest available evidence in the field. *Example:* The MetaGPT framework paper [27] reports 85.9% on HumanEval and 124.3 tokens per line of code, both verified through the journal review process.

Preprint. arXiv or institutional preprints that have not yet completed peer review but are widely cited and methodologically transparent ($\geq$10 citations or a clear pre-registration/artifact release). We include preprints because the field's release cadence makes peer-reviewed venues lag the technical frontier by 12–18 months. *Example:* The SWE-Bench Illusion analysis [36] is preprint-only at submission time, but the methodology (in-distribution vs. held-out repository split) is reproducible from the paper.

Vendor-reported (†). Numbers, benchmarks, or capability claims originating from the organization that built the product being measured (e.g., model cards, launch blogs, release notes, and vendor-authored arXiv reports on the vendor's own product). These claims are routinely informative—often they are the only available data—but they are not independently verified and have historically shown systematic optimism. Every such claim in this manuscript is suffixed with a dagger (†), and Section 7.6 discusses three documented vendor-vs-independent discrepancies that motivate this convention. *Example:* SWE-Bench Verified leaderboard scores reported in vendor launch blogs† [Anthropic, OpenAI, Cognition launch pages].

Industry survey. Large-N adoption, deployment, or sentiment studies conducted by industry analysts, consultancies, or trade publications (Gartner, McKinsey Global Survey, Stack Overflow Developer Survey, 2025 and AI Agent Index [37]). These provide cross-sectional snapshots of practice that academic work rarely matches in scale, but methodology (e.g., sampling frame, response rate, and conflicts of interest) varies widely and is often under-disclosed. *Example:* The 2025 AI Agent Index finding that 25 of 30 surveyed deployed agentic systems disclose no internal safety evaluation [37].

Regulatory. Statutes, regulations, and official guidance from governmental or supra-national bodies (e.g., EU AI Act Regulation 2024/1689 [38], NIST AI Risk Management Framework, and UK government departmental evaluations). Regulatory text is authoritative for compliance requirements but is descriptive rather than empirical for performance claims. *Example:* The EU AI Act's risk-tier definitions and human-oversight requirements [38].

Standards. Open specifications from standards bodies and industry consortia (e.g., Linux Foundation Agent2Agent [39], Anthropic Model Context Protocol [40], W3C drafts, and IETF RFCs). These define interfaces and protocols rather than empirical results; their authority derives from adoption rather than measurement. *Example:* The A2A signed Agent Card schema [39] and MCP capability-negotiation flow [40].

Throughout the manuscript, vendor-reported numbers are marked with † at first occurrence in each section, and the marker is repeated when the same number is invoked in later tables. Section 7.6 examines three concrete cases in which independent re-evaluation of vendor-reported numbers reached materially different conclusions, and discusses how readers should weight production-deployment claims (Section 9) accordingly.

4. Taxonomy of Orchestration Patterns

How agents are coordinated shapes every downstream property of a system: scalability limits, fault tolerance, debugging difficulty, token cost, and task suitability. Earlier surveys [8,10,11] organize this design space as a flat list of patterns, but this conflates two independent design choices. We separate them.

We propose a two-dimensional taxonomy. The first dimension is coordination topology: the structural relationship between agents in terms of control authority and communication flow. The second dimension is runtime adaptivity: whether the topology

evolves during execution. Both dimensions trace to classical multi-agent system (MAS) theory [18,41,42] but acquire new significance in the LLM-agent setting, where agent roles and skills are programmed via natural-language prompts and can be re-specified at low cost.

Note on mnemonic. The abstract uses the shorthand "three-topology, one-adaptivity taxonomy" (informally, 2DOT—two-dimensional orchestration taxonomy) purely for compactness; the operative structure is the two orthogonal axes defined here in Sections 4.1 and 4.2 and visualized in Figure 2. The mnemonic does not encode an additional design claim.

Figure 2 depicts the resulting 3 × 2 grid with one representative system per cell. Hybrid systems occupy points in this grid rather than forming a separate class; we discuss them in Section 4.5.

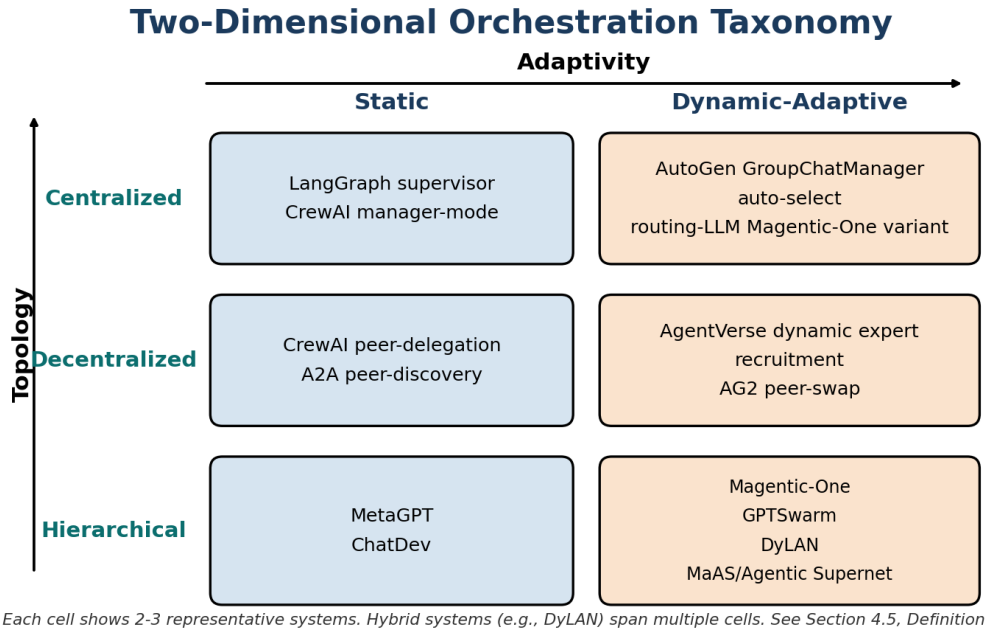


Figure 2. Two-dimensional orchestration taxonomy: topology axis (Centralized, Decentralized, Hierarchical) × adaptivity axis (Static, Dynamic–Adaptive). Each cell lists 2–3 representative systems. Hybrid systems occupy multiple cells. Systems shown: LangGraph [29], CrewAI [30], A2A [39], AutoGen [35], Magentic-One [43], AgentVerse [32], MetaGPT [27], ChatDev [44], GPTSwarm [33], DyLAN [34], and MaAS [45].

4.1. The Topology Dimension

Definition 1 (coordination topology)**.** *The coordination topology of an LLM multi-agent system is the directed graph G = (V, E) whose vertices are agents and whose edge set E is the set of permitted message-passing or delegation pathways declared at design time. A topology is centralized when G is a star with a designated supervisor; decentralized when G is a (possibly complete) peer graph with no designated supervisor; hierarchical when G is a tree. E is the permitted-edge superset; the subset of edges actually exercised at a given timestep, $E_t \subseteq E$, may vary at runtime—the basis for the routing-dynamic sub-mode of Definition 2.*

This definition is purely structural. It says nothing about whether the graph G changes at runtime, who authored the message-passing policy, or whether the participating agents are LLM-instantiated or hand-coded. Each is captured by the adaptivity dimension or by orthogonal framework choices.

The three topologies have characteristic strengths and limitations that we summarize per cell below; for the unifying view of when to choose each, see Section 4.6.

4.1.1. Centralized (Supervisor) Topology

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 [46]. The supervisor is typically an LLM that consults the 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 [29]. 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 [35]. 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 topology 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 Transform Messages utilities (MessageHistoryLimiter, MessageTokenLimiter, and TextMessageCompressor) to mitigate this, but they require manual tuning.

4.1.2. Decentralized (Peer-to-Peer) Topology

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 [47].

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 [30]. 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 [39].

When to choose it. The decentralized topology 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 [47] show that even state-of-the-art models (e.g., GPT-4 and Llama-3 70B) struggle with multiparty negotiation tasks requiring arithmetic reasoning and strategic planning. Distributed execution traces make post hoc debugging laborious.

4.1.3. Hierarchical Topology

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 [32].

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 [29]. 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 [30]. 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 [27]. Magentic-One [43] 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 topology 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. It balances control and scalability; natural modularity permits the 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.

4.2. The Adaptivity Dimension

Definition 2 (runtime adaptivity)**.** *A multi-agent system is static when its operative configuration (V, the active edge set, and the routing policy) is fixed at design time and does not change for the duration of a task. It is dynamic–adaptive when that configuration changes at runtime through one or more of the following mechanisms:*

- *Routing-dynamic: The active edge subset E_t changes at runtime (edges from the permitted set E activate or deactivate per step), while V and the permitted-edge superset E are fixed. Example: LangGraph conditional edges and AutoGen GroupChatManager auto-selection.*
- *Membership-mutating: V itself changes at runtime (agents are spawned, retired, or recruited). Example: AgentVerse dynamic expert recruitment.*
- *Learned coordination: The policy that determines routing or membership is learned from data rather than hand-coded. Example: GPTSwarm REINFORCE-optimized edge probabilities and DyLAN Agent Importance Score.*

These three sub-modes are not mutually exclusive—DyLAN performs both membership pruning and learned coordination simultaneously, and we discuss such hybrid cases explicitly in Section 4.5. The sub-mode distinction matters because it determines the failure modes a system can exhibit and the safeguards it requires (Section 10).

Adaptivity is orthogonal to topology: a centralized system can be static (LangGraph supervisor with hand-coded routing) or dynamic–adaptive (AutoGen auto-select). A hierarchical system can be static (MetaGPT's fixed software-company hierarchy) or dynamic–adaptive (Magentic-One's re-planning Orchestrator).

4.3. Self-Evolving and Long Horizon as Cross-Cutting Threads

Two recent research directions cut across both dimensions of our taxonomy and warrant explicit framing here, since they shape later chapters.

Self-evolving agent organizations are systems where the agent set, agent roles, or routing policy change at runtime through learning or recruitment. In our taxonomy, these exercise the membership-mutating and/or learned-coordination sub-modes of the adaptivity dimension and can be combined with any base topology. AgentVerse [32] (membership-mutating recruitment), EvoMAC [48], and MaAS/Agentic Supernet [45] (adding learned coordination) illustrate this region at increasing levels of automation. We discuss frameworks supporting self-evolution in Section 5.7, the infrastructure they need (e.g., agent registration and discovery) in Section 6.7, and their open evaluation challenges in Section 10.6.

Long-horizon coordination is a property of the task rather than of the topology or adaptivity dimension: tasks that span hundreds of LLM calls, multiple sessions, or external state require coordination primitives different from those needed for short-horizon tasks. Long-horizon tasks intersect every cell of our 3 × 2 grid, but they place specific demands on state management (Section 7.1.1) and evaluation (Section 8.2).

Treating these two directions as cross-cutting threads rather than as separate orchestration patterns lets us unify the workflow-orchestration perspective (which dominates Section 5) with the organization-level coordination perspective (which dominates the AgentVerse/GPTSwarm/DyLAN line of work).

4.4. The 3 × 2 Grid: Six Base Configurations

Table 1 places representative systems in each cell. Cells with two–three examples document the breadth of work in each design region; the assignments are intentionally conservative—systems that exhibit features of multiple cells are listed as hybrid in Section 4.5 rather than forced into one cell.

Table 1. Two-dimensional orchestration taxonomy with representative systems per cell.

	Static	**Dynamic–Adaptive**
Centralized	LangGraph supervisor (hand-coded routing) [29]; CrewAI manager-mode [30]	AutoGen GroupChatManager auto-select [35]; routing-LLM Magentic-One variant [43]
Decentralized	CrewAI peer-delegation (allow_delegation=True) [30]; A2A peer-discovery [39]	AgentVerse dynamic expert recruitment [32]; AG2 peer-swap configurations
Hierarchical	MetaGPT software-company hierarchy [27]; ChatDev fixed-role pipeline [44]	Magentic-One re-planning Orchestrator [43]; GPTSwarm learned hierarchy [33]; DyLAN team selection + dynamic routing [34]; MaAS/Agentic Supernet [45]

4.5. Hybrid and Cross-Cell Systems

Definition 3 (hybrid system)**.** *A system is hybrid when it cannot be cleanly placed in a single cell of the 3 × 2 grid. We identify two categories.*

Cross-cell hybrids cannot be pinned to one (topology or adaptivity) cell because they combine values along one or both axes. DyLAN is the canonical example along two distinct dimensions: within the *adaptivity* axis, it exercises two sub-modes at once—

membership mutating (dynamically selecting which agents to include) and learned coordination (learning the importance scoring used for that selection); along the *topology* axis, its outer architecture is hierarchical while its in-team coordination is partly decentralized. Cross-cell hybrids are not anomalies but design choices that arise when a system intentionally combines properties.

Context-dependent hybrids transition between cells in response to task properties. A system that runs a fixed hierarchical pipeline under normal load and switches to a dynamic–adaptive routing mode under high load occupies (hierarchical, static) → (hierarchical, dynamic–adaptive) over time. Such systems are best classified by their default configuration; the transition mechanism itself is documented as a runtime property.

This treatment of hybrids preserves the taxonomy's analytic power without forcing every real system into a single cell.

4.6. Connections to Classical MAS Theory

The dimensions in our taxonomy have direct analogues in classical multi-agent system theory. The topology dimension connects to Stone and Veloso's 2000 survey of multi-agent systems [41], which organizes the field along agent-heterogeneity and inter-agent communication axes; it also corresponds to Wooldridge's [18] characterization of agent architectures by their interaction protocols. The adaptivity dimension extends Tambe's [42] team-based agent coordination work (the STEAM model of flexible teamwork, JAIR 1997), where teams reformed in response to environmental change; the LLM-agent setting inherits these distinctions but adds a new degree of freedom—agent roles and skills can be re-specified at low cost via prompt updates, which makes membership-mutating and learned-coordination sub-modes more practical than in earlier MAS work.

We do not claim novelty for the two dimensions themselves. We claim that applying them as orthogonal axes—rather than enumerating four "patterns", with one being a runtime modifier—yields a cleaner reference framework for the LLM-MAS field.

4.7. A Decision Framework for Pattern Selection

Table 2 summarizes the tradeoffs. The decision criteria below translate them into actionable guidance.

Decision criteria:

1. Is the task structure known at design time? If yes, prefer centralized or hierarchical. If not, prefer dynamic–adaptive.
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.
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 20× cost reduction over hand-designed dynamic–adaptive 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 3 summarizes this decision flow and shows how the dynamic–adaptive modifier layers over any of the three base topologies.

Table 2. Comparison of base topologies along key practitioner-relevant axes; the dynamic–adaptive adaptivity modifier is reported as a separate column because it is orthogonal to the base topology (Definition 2) and reaches distinct framework primitives.

Dimension	Centralized	Decentralized	Hierarchical	Dynamic–Adaptive
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 and AutoGen GroupChat	CrewAI delegation and A2A peers	LangGraph nested graphs, MetaGPT, and Magentic-One	AutoGen auto-select, GPTSwarm, and DyLAN

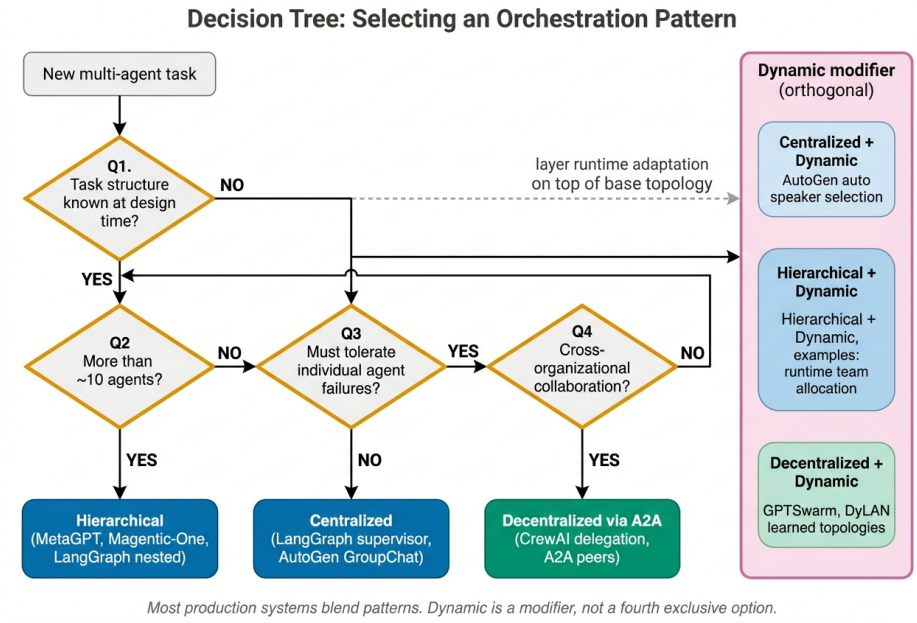


Figure 3. Decision tree for selecting an orchestration pattern. Four of the five decision criteria in Section 4.7 act as routing questions that select a base topology (centralized, decentralized, or hierarchical); the remaining criterion (token-cost budget) and the dynamic–adaptive modifier are orthogonal refinements layered over any base choice.

Note on the dynamic–adaptive 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–adaptive 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 subproblems 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, and 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 4 illustrates the distinct profiles of each framework across seven evaluation dimensions.

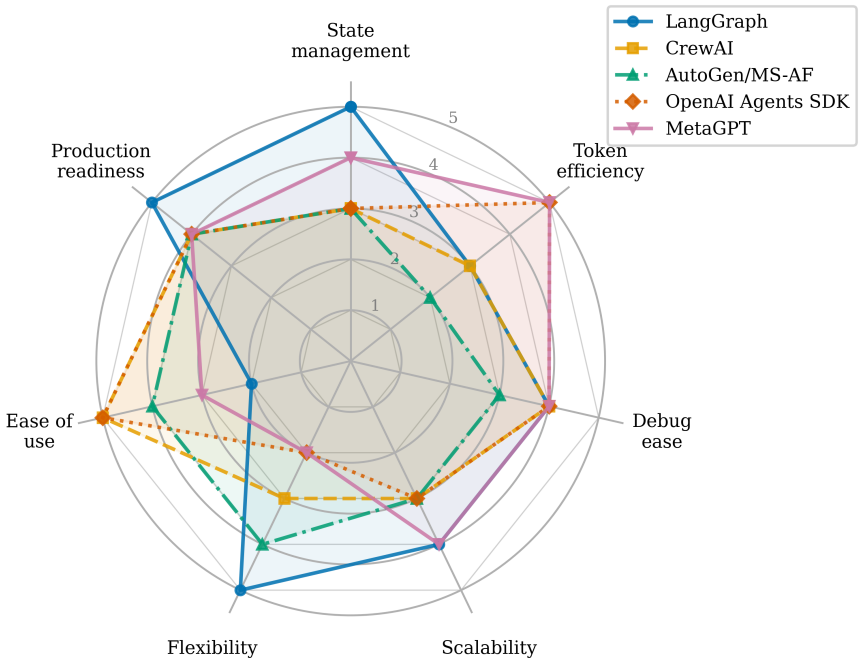


Figure 4. Radar plot comparing LangGraph, CrewAI, AutoGen/Microsoft Agent Framework, OpenAI Agents SDK, and MetaGPT across seven dimensions: state management, token efficiency, debug ease, scalability, flexibility, ease of use, and production readiness. Scores are qualitative, based on framework documentation and published case studies as of March 2026.

5.1. Mapping Frameworks to the 2D Taxonomy

Before surveying each framework in turn, we anchor the catalog to the orthogonal axes of Section 4. The 3 × 2 grid (topology × adaptivity) gives us a single place to record

where each framework's *default* configuration sits and which neighboring cells it can be reconfigured into. Because every major framework exposes some configuration surface—speaker-selection policies, delegation flags, conditional edges, and manager swaps—most frameworks are not pinned to one cell but cover a small contiguous region. We record the *primary* cell (what one gets out of the box with the documented defaults) and call out the *adjacent* cells reachable through documented configuration. This mapping makes the catalog in Sections 5.2–5.6 readable as a tour of design choices rather than a flat feature comparison, and it sets up Table 3, where the same assignments appear in tabular form.

Workflow-orchestration frameworks (Sections 5.2–5.5). LangGraph [29] is primarily (centralized OR hierarchical, static): The StateGraph and its routing nodes are hand-authored at design time. Its conditional-edge mechanism lets the same graph behave as (·, dynamic–adaptive)—routing-dynamic in the sub-mode sense of Section 4.2—without changing the agent set V. CrewAI [30] is primarily (centralized OR decentralized, static): A Crew with `allow_delegation=False` runs as a fixed sequential or hierarchical-manager pipeline; setting `allow_delegation=True` lets workers re-route to peers at runtime, moving the system into the (decentralized, dynamic–adaptive) cell. AutoGen/Microsoft Agent Framework [31,35] is primarily (centralized, dynamic–adaptive) by default: The GroupChatManager's `auto` speaker-selection policy picks the next speaker per turn via an LLM call over conversation history. Switching to `round_robin` or a hand-coded selection function pins the system back into (centralized, static). OpenAI Agents SDK [49] is primarily (centralized OR hierarchical, static): Handoffs are defined at agent-construction time, and the runtime simply follows them; achieving runtime adaptivity requires building selection logic outside the SDK.

Adaptive and learned frameworks (Sections 5.6 and 5.7). Magentic-One [43] sits in (hierarchical, dynamic–adaptive): its Orchestrator constructs a task ledger and re-plans through LLM prompting, but it does not learn the routing policy. AgentVerse-Evolve [32] sits in (decentralized, dynamic–adaptive) with both membership-mutating and learned-coordination sub-modes active: its four-stage expert-recruitment loop changes V across rounds. GPTSwarm [33] and DyLAN [34] sit in (hierarchical, dynamic–adaptive) with the *learned-coordination* sub-mode dominant: GPTSwarm optimizes edge probabilities via REINFORCE, and DyLAN learns Agent Importance Scores for team selection.

Table 3 gathers the full mapping along with each framework's design philosophy, key features, and maturity tier, providing the single reference table referred to throughout Section 5.

5.2. LangGraph: The Graph as State Machine

LangGraph, developed by LangChain, models agent workflows as stateful directed graphs [29]. 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 (e.g., LLM calls, tool invocations, and 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 [29]). This is functional-programming thinking applied to agent orchestration.

Table 3. Comparative analysis of multi-agent orchestration frameworks, mapped to the 2D taxonomy of Section 4. Frameworks marked (†) are vendor-documented; peer-reviewed entries cite the primary publication. The "Primary cell" column gives the default (topology, adaptivity) cell per Section 4; configuration paths to adjacent cells are noted in Section 5.1. The maturity tier follows the evidence hierarchy in Section 3: PR = peer-reviewed primary publication; V † = vendor-reported framework without an independent peer-reviewed evaluation.

Framework	Primary (Topology, Adaptivity) Cell	Design Philosophy	Key Features	Maturity Tier
LangGraph [29]	(centralized OR hierarchical, static); routing-dynamic via conditional edges	Graph as state machine	StateGraph with typed reducers; per-mutation checkpointing (in-memory/SQLite/PostgreSQL); supervisor, hierarchical, swarm patterns native; LangSmith observability; HITL via breakpoints	V † (LangChain vendor docs; enterprise users Klarna/LinkedIn/Elastic reported by vendor)
CrewAI [30]	(centralized OR decentralized, static); dynamic–adaptive via `allow_delegation`	Role-playing team	Agent (role/goal/backstory) + Task + Crew + Flows; sequential and hierarchical-manager process models; unified short/long/entity/user memory with vector-backed composite scoring; task input gates for HITL	V † (CrewAI vendor docs; enterprise adoption vendor-reported, unverified)
AutoGen/ Microsoft Agent Framework [31,35]	(centralized, dynamic–adaptive) by default via GroupChatManager auto-select; (centralized, static) under round_robin	Conversation as coordination	ConversableAgent base; GroupChat + four speaker-selection policies (auto/round_robin/random/manual); Microsoft Agent Framework adds Magentic orchestrator, graph workflows, A2A and MCP as first-class, OpenTelemetry, Azure AI Foundry integration	PR (AutoGen—[35]) + V † (MS Agent Framework, Oct 2025 launch)
OpenAI Agents SDK [49]	(centralized OR hierarchical, static)	Handoff as primitive	Four primitives: Agent, Handoff (`transfer_to_<agent_name>`), Guardrails, Tracing; Sessions with pluggable backends (SQLite/Redis/Dapr); direct MCP server integration; minimal API surface	V † (OpenAI vendor docs; production successor to experimental Swarm, March 2025)
AgentScope [50]	(decentralized, static) by default; distributed actor variant	Actor-model distribution	Message-exchange comms + actor distribution framework; local-to-distributed deployment without code change; built-in retry with configurable backoff	PR (Alibaba research publication)
CAMEL [28]	(decentralized, static)—2-agent dialogue	Inception prompting	Bootstraps autonomous two-agent dialogue; structured role playing that avoids role flipping, repetition, infinite loops	PR
MetaGPT [27]	(hierarchical, static)	Documents replace dialogue	SOP-encoded software-company hierarchy; publish–subscribe role-relevant messaging; structured artifacts at each stage; 100% task completion at 124.3 tokens/line on the paper's evaluated tasks	PR

Table 3. *Cont.*

Framework	Primary (Topology, Adaptivity) Cell	Design Philosophy	Key Features	Maturity Tier
DSPy [51]	(centralized, static) at runtime; learning is compile-time not a runtime adaptivity sub-mode	Pipelines as optimizable programs	Compiler tunes prompts and few-shot examples to user-defined metric; reports 25–65% improvement over standard few-shot on GPT-3.5/Llama2-13b	PR (ICLR 2024 spotlight)
OpenHands (formerly Open-Devin) [52]	(centralized, static) generalist agent	Code as action	Open platform for AI software developers; sandboxed code execution, browsing; CodeActAgent expresses actions as executable code; integrates with 15+ benchmarks; 26% SWE-Bench Lite with claude-3.5-sonnet at v1.8	PR
AgentVerse-Evolve [32]	(decentralized, dynamic–adaptive)—membership-mutating + learned	Expert-recruitment loop	Four-stage loop (recruit → decide → execute → evaluate); per-round expert roster mutation; empirical study of emergent constructive vs. destructive group behaviors	PR (ICLR 2024)
EvoMAC [48]	(decentralized OR hierarchical, dynamic–adaptive)—learned over agents + connections	Test-time textual backprop	It iteratively rewrites both agent prompts and inter-agent connections from environmental feedback; ships rSDE-Bench requirement-oriented SWE benchmark	PR (ICLR 2025)
MaAS/Agentic Supernet [45]	(hierarchical, dynamic–adaptive)—learned, query-conditioned sampling	Architecture-search over agent systems	Replaces fixed topology with an agentic supernet; controller samples query-conditioned sub-architecture; 6–45% of baseline inference cost at +0.5 to +16.9% accuracy across six benchmarks	PR (ICML 2025 Oral)

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, spanning all three topology classes of our taxonomy (centralized, hierarchical, and—via swarm—decentralized); its conditional-edge mechanism adds the dynamic–adaptive axis.

State management. Built-in checkpointing persists graph state to configurable backends (in-memory, SQLite, PostgreSQL, and other pluggable options per LangGraph documentation [29]). 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†) [29]). 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.

5.3. 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 [30].

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 intertask 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 [30]). 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†) [30]. 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.

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

AutoGen, from Microsoft Research [35], 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 [31].

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, and 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 re-implementing 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 [31].

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 [35] 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.

5.5. OpenAI Agents SDK: The Handoff as Primitive

Released in March 2025, the Agents SDK is the production successor to OpenAI's experimental Swarm project [49]. 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, and 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.

5.6. Other Notable Frameworks

AgentScope (Alibaba) uses a message-exchange communication model augmented with an actor-based distribution framework [50]. 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 [28] introduced inception prompting: the 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 [27] 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 [51] 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) [52] 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 [52]; 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 [53], have reached substantially higher resolution rates on SWE-bench Verified.

5.7. Self-Evolving Frameworks for Organization-Level Coordination

The frameworks surveyed so far in Sections 5.2–5.6 are *workflow-orchestration* frameworks: the developer fixes the agent set and the coordination graph at design time, and the framework executes that graph. A complementary line of work, which Section 2.4 frames as organization-level coordination, builds frameworks in which the agent organization itself—its membership, roles, or communication topology—is the object being optimized at runtime. These systems belong in a Section 5 framework discussion, not only in the future-work section in Section 10, because each provides a usable software framework with documented interfaces, evaluation harnesses, and reproducible benchmarks; treating them as speculative would understate the maturity of the line and the design choices it forces practitioners to confront. The three systems below mark progressive stages of automation in this direction.

AgentVerse [32]. AgentVerse organizes collaboration as a four-stage loop: expert recruitment, collaborative decision making, action execution, and evaluation. The recruitment stage dynamically composes a task-specific expert roster, and the evaluation stage feeds back into the next round's recruitment, so the agent set V mutates across rounds. The framework also documents emergent destructive vs. constructive group behaviors, providing the first systematic study of how a self-evolving roster shifts qualitative collaboration dynamics.

EvoMAC [48]. EvoMAC introduces *test-time textual backpropagation*: environmental feedback from a software development task is converted into a textual gradient that iteratively rewrites both agent prompts and inter-agent connections. The framework ships with the rSDE-Bench requirement-oriented SWE benchmark and demonstrates that the same evolutionary mechanism can co-optimize agent behavior (prompts) and organization structure (connections) within a single test-time loop, eliminating the design-time/runtime split that the frameworks in Sections 5.2–5.5 all preserve.

MaAS/Agentic Supernet [45]. MaAS replaces a single fixed multi-agent topology with an *agentic supernet*; at inference time, a controller samples a query-conditioned sub-architecture from the supernet, so each query effectively runs through its own multi-agent system. Reported results across six benchmarks span 6–45% of baseline inference cost at +0.5 to +16.9% accuracy, demonstrating that adaptive sampling can simultaneously reduce cost and improve quality relative to a fixed strong topology.

Read alongside Sections 5.2–5.6, these three systems extend the framework axis from workflow orchestration into organization-level coordination, and they populate the (·, dynamic–adaptive) row of the taxonomy in Section 4 (Figure 2, Table 1) with concrete, reproducible artifacts.

5.8. Comparative Analysis

Table 3 compares the major frameworks, mapped to the taxonomy in Section 4. The comparison that matters the most, however, is philosophical.

Structured approaches (e.g., LangGraph and 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†) [29]; we do not claim independent evidence that reliability was the decisive factor in selection decisions. Emergent approaches (e.g., AutoGen and 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 messages" and performs "implicit state inference and progress making conditioned on conversation history" [35]; 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, or 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 [29] 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 [31]. 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 [38]), 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.

Notes. First, cells in the "Primary cell" column follow Definition 1 (topology) and Definition 2 (adaptivity, with routing-dynamic, membership-mutating, and learned coordination sub-modes); see Section 5.1 for configuration paths to adjacent cells. Second, "PR" denotes a peer-reviewed primary publication for the framework; "V†" denotes a vendor-documented framework whose primary description sources are vendor docs and blog posts and which has no independent peer-reviewed system evaluation. Mixed entries list both. Third, a metric-portability column is intentionally omitted: the framework-reported

numbers (HumanEval, SWE-Bench, and tokens/line) are not cross-comparable under different model versions, task subsets, and evaluation dates (see Section 5 opening note and Section 8.2). Lastly, all deployment claims dependent on vendor self-report are flagged with † in the maturity column; we make no independent verification claim for those entries.

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 5.8); CrewAI, AutoGen, and the OpenAI Agents SDK can persist state but cannot replay an arbitrary prior state without external scaffolding. Second, only the Microsoft Agent Framework treats both A2A and MCP as first-class citizens at the framework level [31]; 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 Microsoft 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 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 [40]. 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 and 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†) [54], and announced that MCP was being donated to the Agentic AI Foundation (AAIF) under the Linux Foundation, with OpenAI and Block as co-founders [54].

As of the survey cutoff (March 2026), Anthropic's December 2025 announcement of the AAIF donation and the public MCP changelog [54] indicate continued development toward richer capability-negotiation primitives, streaming tool outputs, and cross-session state persistence; a formal "MCP 2.0" specification with a citable URL had not been finalized at the time of writing; we therefore describe these directions as in-flight rather than as a stable specification. The implications for long-horizon tool integration discussed in Section 7 and the open challenges of Section 10 would be substantially affected if and when such a specification is ratified.

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 [39]. 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 the 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 [55]. 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†) [39].

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, 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 5 illustrates how these layers compose; Figure 6 illustrates a typical interaction sequence involving both MCP tool invocation and A2A task delegation.

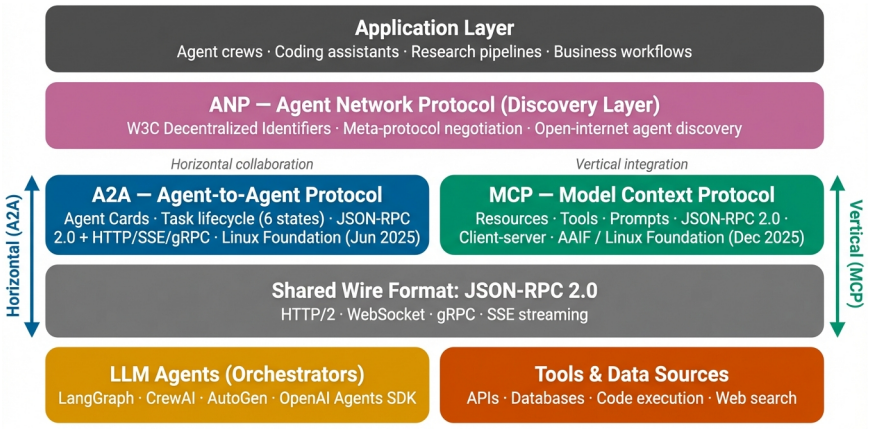


Figure 5. Protocol stack showing MCP (agent-to-tool, vertical) and A2A (agent-to-agent, horizontal) as complementary layers over a shared JSON-RPC 2.0 transport, with ANP (agent-to-network discovery) as an emerging third layer for open-internet deployments.

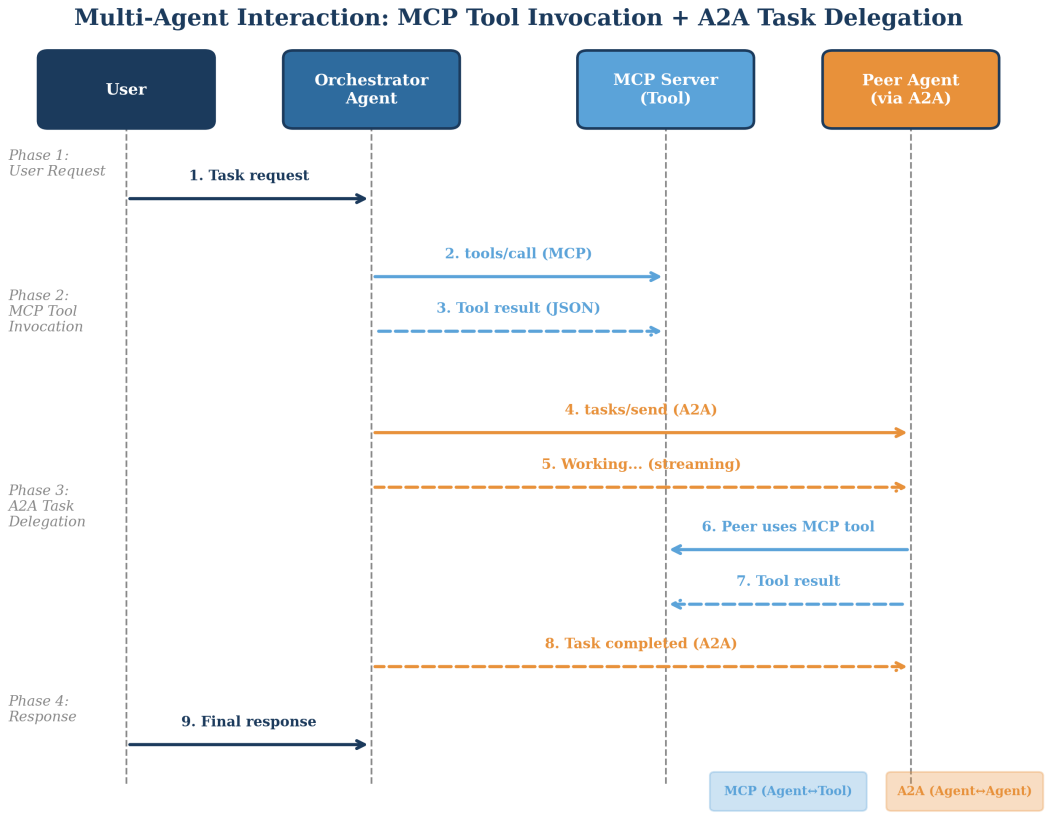


Figure 6. Sequence diagram: a user sends a request to a coordinating agent; the coordinating agent issues a tool call to a local tool server via MCP; it then delegates a subtask to a peer agent via A2A (task created, streamed, completed); finally, it synthesizes results and returns a response to the user. Shared underlying transport (JSON-RPC 2.0) is labeled at the bottom.

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 [56]. Rather than maintain competing standards, IBM and Google merged ACP into A2A under the Linux Foundation in August 2025 [57], 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†)) [54]. 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 [58]. 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 4 compares the three active protocols. A pragmatic adoption path, as Ehtesham et al. [59] 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 4. Comparison of agent communication protocols.

Dimension	MCP	A2A (incl. Former ACP)	ANP
Primary scope	Agent-to-tool (vertical)	Agent-to-agent (horizontal)	Agent-to-network (discovery)
Wire protocol	JSON-RPC 2.0	JSON-RPC 2.0 over HTTP(S), SSE, gRPC (v0.3+)	DID + JSON-LD
Communication model	Client–server	Peer-to-peer tasks (6-state lifecycle)	Decentralized P2P
Discovery mechanism	Server configuration	Agent Cards (/.well-known/agent.json)	DID documents
Authentication	Transport-level (bearer tokens, API keys)	OAuth 2.0, API keys, signed cards (v0.3)	DID-based cryptography
Async support	Limited	Full (SSE, webhooks, push notifications)	Full
Governance	AAIF under Linux Foundation (Dec 2025)	Linux Foundation (June 2025)	W3C CG (proposed)
Industry supporters	OpenAI, Anthropic, Block, AWS, Google, Microsoft	100+ organizations (as of June 2025 Linux Foundation launch)	Open-source community
Maturity	High (rapid SDK adoption, broad vendor support)	Growing (v0.3, July 2025)	Early stage
Best for	Tool and API integration	Enterprise multi-agent collaboration	Open-internet agent ecosystems

Table 5 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 [59]; cells marked "(not specified)" reflect deliberate scope decisions by the protocol authors rather than missing information.

Table 5. 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 [57].

Feature	MCP [40,54,59]	A2A (incl. Former ACP) [39,55,59]	ANP [58,59]	ACP (Pre-Merger) [56,57,59]
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 [59]. 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 [58]. 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.

6.7. Protocols as Enablers of Self-Evolving Agent Organizations

The three-layer protocol stack of Sections 6.1–6.5 is usually motivated as a tool/agent/network integration story, but each layer also supplies a specific primitive that organization-level coordination (Section 2.4) needs in order to evolve at runtime. MCP capability negotiation [40,54] occurs during session initialization: a host advertises supported sampling/elicitation primitives and a server enumerates its tools, resources, and prompts. This makes runtime role assignment tractable—an agent newly admitted to a collective can discover what capabilities its peers' tool surfaces support before being assigned subtasks, which is precisely the precondition for the *membership-mutating* adaptivity sub-mode of Section 4.2. A2A signed Agent Cards (v0.3+, July 2025) [39,55] do an analogous job at the agent layer: an Agent Card published at /.well-known/agent.json, cryptographically signed, lets a coordinator add or retire a peer without re-deploying the surrounding system, since trust and skill descriptors travel with the card rather than being baked into application code. This is the protocol-level enabler for the *membership-mutating, self-evolving systems* of Section 4.3—including AgentVerse-style recruitment [32], EvoMAC's connection rewriting [48], and MaAS's query-conditioned sub-architecture sampling [45]. ANP's DID-based discovery [58] extends the same idea across organizational boundaries: agents reachable by W3C decentralized identifiers can be enrolled into a collective without prior trust establishment, making *cross-organizational* self-evolving systems structurally possible. A clear limitation should be noted: current protocols standardize registration, discovery, and capability advertisement, but the *learning of the coordination policy itself*—which peers to recruit, when to retire them, and how to rewire communication edges—remains an application-layer concern, addressed by the frameworks of Section 5 and the systems surveyed in Section 4.3.

7. Design Considerations

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

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 [29]). 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 [30]). The Microsoft Agent Framework adds session-based state management with pluggable backends (per Microsoft Agent Framework documentation [31]). External vector databases remain the dominant choice for long-term agent memory across all frameworks [22].

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 [27]. This selective visibility prevents information overload, which degrades LLM reasoning quality.

7.1.1. Long-Horizon State Management

Section 7.1 surveys state-management primitives—checkpointing, unified memory interfaces, and publish–subscribe visibility—under the implicit assumption that a task

fits within a single coordinated execution. A second regime, increasingly common in production deployments, lifts this assumption. We define a long-horizon task as one whose completion spans more than roughly 100 LLM calls, persists across multiple user sessions, or interacts with an external state machine (a workflow engine, a long-running simulation, and a multi-day approval process) whose lifetime exceeds any single agent invocation. Long-horizon tasks intersect every cell of the taxonomy in Section 4 but exert four characteristic pressures on the state layer.

Context window saturation. Even with the 128k–1M token windows of current frontier models, accumulated conversation, tool-call traces, and intermediate artifacts saturate the context well before a multi-day task completes. The mitigation strategies introduced in Section 7.4—MetaGPT's structured documents [27], CrewAI's respect_context_window flag [30], and AutoGen's Transform Messages utility—were originally framed as scalability mechanisms, but their primary use in long-horizon settings is reversible summarization that survives the boundary of any single LLM call. Trajectory-reduction work [60] reporting 40–60% input-token reduction at <2% performance loss is directly applicable here, though that work was not itself evaluated on multi-session horizons.

Cross-session persistence. State that must survive process restarts, framework redeployments, or handoffs between teams of operators needs a persistence model beyond the in-memory checkpoint. LangGraph's checkpointer abstraction [29] and the Microsoft Agent Framework's pluggable session backends [31] address part of this; the protocol stack of Section 6 contributes the rest, since A2A's task object carries a stable identifier across its six-state lifecycle, allowing a paused task to be resumed by a different runtime instance. Cross-runtime resumption remains under-standardized at the March 2026 cutoff.

Partial-credit checkpointing. A long-horizon task that fails after 95% of its work is far more costly than a short task that fails outright. Evaluation that scores only terminal success obscures whether failures cluster late (catastrophic) or are uniformly distributed; we return to this measurement gap in Section 8.

State–policy interaction with adaptivity sub-modes. The three adaptivity sub-modes of Section 4.2 each impose distinct state demands. *Routing-dynamic* systems need to checkpoint only the active edge mask. *Membership-mutating* systems must persist the agent roster itself, since an agent recruited at hour 4 may need to be reactivated at hour 40. *Learned-coordination* systems additionally need to persist the learned policy parameters—REMAC [61] explicitly evaluates long-horizon robot manipulation through a self-reflective evolving MAS, illustrating that the learning loop itself becomes a piece of long-lived state. AgentDropout [62], although shorter-horizon in its evaluation, reports that the learned sparse topology transfers across tasks, hinting that the coordination policy can be reused rather than re-learned per session—a property whose long-horizon implications remain to be benchmarked. Open evaluation challenges arising from these interactions are taken up in Section 8.

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 [27]), and search-based planning (LATS's Monte Carlo Tree Search [24]).

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% on HumanEval) at substantially higher token cost. GPTSwarm offers a middle path: learn an efficient topology offline, and execute it

cheaply at inference time. OPTIMA [63] 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 [28,30]), capability-based (route via A2A Agent Card skill descriptors [39]), and dynamic (assign based on availability and past performance [35]). DSPy's compiler-based approach [51] 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, and context overflow), communication level (timeout and serialization error), orchestration-level (deadlock, infinite loop, and wrong routing), and infrastructure level (network outage and resource exhaustion). Production systems combine multiple defenses into a defense-in-depth posture: AgentScope provides configurable retry with exponential backoff [50]; Reflexion enables agents to learn from failures through verbal self-critique [4]; LangGraph's checkpoints enable rollback to a known-good state [29]; the OpenAI Agents SDK's guardrails catch bad inputs proactively [49]; 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 [50]. MetaGPT's publish–subscribe mechanism filters messages at the source, preventing broadcast storms in flat communication topologies.

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 [60] 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 LLM systems extend the attack surface of single-agent applications in qualitatively new ways. The relevant primitives—inter-agent messages, shared tool ecosystems, runtime-discoverable peers, and handoff chains that delegate authority—introduce threat classes that classical web-application security models address only partially. We organize the discussion around three threat classes that recur across the orchestration literature: Prompt Infection (adversarial content propagating through inter-agent messages so that compromising one agent contaminates others [64,65]), privilege escalation (an agent acquiring tool, data, or routing authority beyond its declared scope, typically via crafted handoffs or tool descriptions), and data exfiltration (sensitive context, secrets, or tool outputs leaking through agent communication, tool calls, or external endpoints). These three classes interact with the three coordination topologies of Section 4 (centralized, decentralized, and hierarchical) in distinct ways. Table 6 summarizes the resulting nine trade-offs; topology shapes both the attack surface and how easily a defense can be enforced.

Table 6. Topology × threat-class trade-off matrix. Each cell summarizes (i) the dominant attack surface and (ii) the dominant defensive lever. Single point of failure is treated as an architectural property in the closing paragraph rather than a threat class. All claims are framework-design observations consistent with [39,40,49,64–66]; no quantitative incident statistics are claimed here.

Topology	Prompt Infection	Privilege Escalation	Data Exfiltration
Centralized	Supervisor is a high-value target—one infected supervisor poisons every downstream worker; conversely, the supervisor is the natural single chokepoint for input sanitization and message-level guardrails [49].	Supervisor concentrates routing and tool-dispatch authority, so a compromised supervisor inherits superset privileges; centralized policy enforcement and audit logs are correspondingly easier to deploy.	All worker outputs funnel through the supervisor, making egress monitoring tractable but also creating a single high-bandwidth exfiltration channel if the supervisor itself is subverted.
Decentralized	Peer-to-peer messaging lets infection self-replicate across the mesh [65]; no central node can quarantine an infected agent, so containment depends on per-agent input filtering and signed sender identity.	Each peer holds local tool authority; the risk is lateral privilege accumulation through repeated delegation rather than vertical escalation. Capability-scoped credentials per peer (MCP-style) are the main defense.	Distributed egress points (each peer may invoke external tools) make exfiltration paths numerous and hard to monitor centrally; protocol-level egress whitelists per agent become essential.
Hierarchical	Infection is bounded by sub-tree; an infected mid-level manager contaminates its descendants but not siblings, giving partial isolation. Per-level message validation can interrupt propagation across tree depths.	Under the common design convention that authority decreases with depth (the tree structure permits but does not require this; a hierarchy can still grant powerful tools to leaves), blast radius is limited—but upward delegation (a leaf escalating via a request to its manager) is a subtle vector that flat threat models miss.	Cross-tree exfiltration requires routing through a common ancestor, which creates a natural inspection point; the same property concentrates risk at top-level managers.

Centralized profile. The supervisor is simultaneously the most attractive target and the most effective defensive chokepoint. Centralized topologies are appropriate when compliance and auditability dominate (Section 4.1), because all routing decisions traverse one node where guardrails [49] and structured logging can be applied uniformly. The corresponding risk is that any prompt-injection or supply-chain compromise of the supervisor immediately reaches every worker.

Decentralized profile. Peer-to-peer meshes invert the trade-off. There is no single high-value target but also no central enforcement point. Prompt Infection [65] is specifically demonstrated in this setting: malicious prompts self-replicate across interconnected agents even when communications are not fully public. Defenses must be local (per-agent input sanitization, signed sender identity via A2A Agent Cards [39], and capability-scoped tool credentials [40]) and consequently harder to audit.

Hierarchical profile. Trees offer the most natural blast-radius bound: an infected sub-tree contaminates its descendants but not siblings, and managers at each level can validate messages crossing depth boundaries. The structural cost is that top-level managers concentrate exfiltration risk and that upward-delegation vectors are easy to overlook.

Adaptivity modifier. Dynamic–adaptive coordination (Section 4.2) is an orthogonal property and raises emergent-behavior risk across all three topologies. When topology and role assignments mutate at runtime, static threat models miss attacker-induced reconfigu-

rations: a malicious peer may, for example, induce AutoGen's auto speaker selection to re-route control through a compromised agent or cause Magentic-One's Orchestrator to spawn an over-privileged worker. Dynamic systems therefore require runtime invariants (maximum-iteration counts, capability budgets, and cycle detection) in addition to the per-topology defenses above.

Protocol-level mitigations and their gaps. The recent agent communication standards address several of these threats directly but not exhaustively. A2A's signed Agent Cards [39], combined with OAuth 2.0, mitigate impersonation: an agent that joins a swarm must present a verifiable identity, which raises the cost of sybil-style attacks in decentralized meshes. ANP's discovery-hardening profile similarly constrains who may advertise capabilities and to whom. MCP's capability negotiation [40] addresses privilege escalation by binding each tool invocation to an explicitly granted permission rather than to an ambient credential. The MAESTRO seven-layer threat-modeling framework (Cloud Security Alliance, February 2025) provides a structured way to enumerate residual risks across these layers [66]. What current protocols do not address is the harder case of Prompt Infection within authenticated channels: a correctly signed, correctly authorized agent that has itself been compromised by upstream prompt injection can still propagate adversarial content through messages that pass every cryptographic and capability check. Defending this requires content-level guardrails [49] and ultimately human oversight; HackAPrompt's 600,000-prompt corpus [64] demonstrates that no current model is robust to all prompt-manipulation variants. The 2025 AI Agent Index [37] reports that 25 of 30 surveyed deployed agentic systems disclose no internal safety evaluation, and the EU AI Act (Regulation 2024/1689) [38] now mandates risk assessment and human oversight for high-risk AI systems—requirements particularly challenging in multi-agent architectures where reasoning traverses multiple agents.

Single point of failure as architectural property. SPOF recurs in security discussions but is structurally distinct from the three threat classes above. It is an architectural property of the topology (high in centralized, low in decentralized, intermediate in hierarchical) and a *prerequisite* for several attacks rather than an attack itself. We therefore discuss it as a separate dimension (Table 1) and keep Table 6's 3 × 3 focused on threat classes. The defensive implication is that high-SPOF topologies benefit the most from supervisor-level redundancy (warm-standby supervisors and checkpointed state [29]), whereas low-SPOF topologies benefit the most from per-agent containment.

7.6. Production-Claim Limitations: Vendor Data vs. Independent Verification

A substantial fraction of the production-deployment claims surveyed in this manuscript—the headline benchmark scores, the developer-productivity gains, and the enterprise-pilot ROI numbers cited in Sections 8 and 9—rely on vendor-reported data (Section 3.1, tier "vendor-reported†"). This is not a defect of the survey; it reflects the underlying corpus, where the organizations building agent products are also the primary publishers of their performance numbers. The methodological consequence is that the variance between vendor-reported and independently verified outcomes is itself a finding the reader needs in order to interpret Section 9. We document three concrete discrepancies for which both a vendor-authored number and a peer-reviewed or government-evaluator counterpart exist.

(i) GitHub Copilot—sign reversal on controlled-task speedup. Peng et al. [67] ran a randomized controlled trial in which developers given Copilot completed an HTTP-server task 55.8% faster than the control group. In contrast, Becker, Rush, Barnes, and Rein [68] ran a randomized controlled trial on 16 experienced open-source developers working in their own repositories and found, verbatim, that "allowing AI

actually increases completion time by 19%—AI tooling slowed developers down". The two studies are both RCTs and reach opposite signs (+55.8% vs. −19%). The difference is attributable to task complexity, repository familiarity, and developer experience level—exactly the moderators that single-number vendor claims tend to suppress.

(ii) Microsoft 365 Copilot—UK enterprise-pilot null finding. Microsoft's framing of the UK government Microsoft 365 Copilot pilot positioned the deployment as productivity-positive (Microsoft case study, vendor-reported†). The corresponding independent evaluation by the UK Department for Business and Trade (DBT), published September 2025 (government evaluation; evaluation period October 2024–March 2025) [69], reports, verbatim: "Small time savings were observed across most use cases", and "The evaluation did not find evidence that time savings have led to improved productivity, and control group participants had not observed productivity improvements from colleagues taking part in the M365 Copilot pilot." That is, self-reported time savings were not corroborated by externally observable team-level productivity gains—a pattern that is invisible in vendor case-study framing.

(iii) SWE-Bench Illusion—in-distribution vs. out-of-distribution gap. Vendor launch announcements routinely report SWE-Bench Verified resolution rates around 70–80%† (e.g., Anthropic's Claude Sonnet 4.5 launch [70]). Liang, Garg, and Moghaddam [36] decompose this number and report, verbatim: "state-of-the-art models achieve up to 76% accuracy in identifying buggy file paths using only issue descriptions, without access to repository structure. This performance is merely up to 53% on tasks from repositories not included in SWE-Bench, pointing to possible data contamination or memorization." The 23-point in-distribution-to-out-of-distribution gap quantifies how much of the headline SWE-Bench Verified score reflects memorization of public repositories rather than transferable coding reasoning. A complementary critique from Epoch AI documents that vendor leaderboard submissions are not consistently evaluated on the full 500-instance Verified set, further complicating cross-vendor comparison.

The peer-reviewed-replication gap. Across these three cases, the common pattern is that peer-reviewed independent replication of vendor-claimed agent ROI in production deployment is absent at the time of writing (March 2026 cutoff). The closest analogues are randomized controlled trials of *single-agent* assistants (e.g., Copilot) and government program evaluations of enterprise *assistant* deployments; we are aware of no peer-reviewed RCT or field study that replicates vendor-claimed productivity gains from a deployed *multi-agent* orchestration system specifically. The 2025 AI Agent Index [37] reinforces this gap from the other direction: in total, 25 of 30 deployed agentic systems surveyed disclose no internal safety evaluation, which precludes external replication even when independent evaluators wish to attempt it.

Implications for interpreting Section 9. The application discussions in Section 9—software engineering, customer support, scientific discovery, education, and healthcare—rely on production claims that fall predominantly into the vendor-reported tier. Readers should therefore treat single-number production claims, especially those originating from a vendor announcement or launch blog, as a *lower bound on the variance* of true deployment outcomes rather than as point estimates of expected gain. Where Section 9 quotes a vendor number, we mark it with † as defined in Section 3.1; where peer-reviewed or government-evaluator alternatives exist, we cite both. We recommend that practitioners reading Section 9 for procurement decisions (i) discount any single-source vendor number by at least the variance documented in (i)–(iii) above, (ii) ask vendors for the full evaluation subset and methodology before relying on headline benchmarks, and (iii) prioritize

deployments for which independent post-deployment evaluation has been published, even if those evaluations show smaller effect sizes than the vendor figures.

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 7 reveals, significant performance gaps persist between AI agents and human baselines across all major evaluation suites.



Figure 7. Benchmark comparison. (**a**) Best reported agent performance versus the human baseline on HumanEval, SWE-bench Verified, OSWorld, WebArena, and GAIA, highlighting that even top systems fall well short of human performance on several agent benchmarks as of the March 2026 cutoff. (**b**) HumanEval pass@1 progression on GPT-4-class models, from the GPT-4 baseline through ReAct, Reflexion, and LATS, against the human baseline. All numbers are drawn from the cited source publications and their public leaderboards.

8.1. Key Benchmarks

SWE-bench [71] 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 2294 issues. SWE-Agent [72] pushed resolution on the original benchmark to 12.5% through custom agent–computer interfaces. SWE-bench Verified [73] (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 the other, and cross-system leaderboard positions shift with the underlying model and prompting scheme. OpenHands' CodeActAgent reports 26% on SWE-Bench Lite [52]; as of late 2025–early 2026, the public SWE-bench leaderboard [53] 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 [74] 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 [75] 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 [76] benchmarks multi-modal 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 [77] 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 being the 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.

τ-bench [78] 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 overall—about 61% single-trial (pass^1) in retail and 35% in airline—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 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.

Long-horizon coordination, defined in Section 7.1.1 as tasks spanning more than roughly 100 LLM calls, multiple user sessions, or interactions with an external state machine whose lifetime exceeds any single agent invocation, places specific demands on evaluation that current benchmarks address only partially. The dominant evaluation pattern—terminal task success as a binary or scalar outcome—is well-matched to single-session benchmarks such as HumanEval, MBPP, GAIA Level 1, and SWE-Bench Lite but degrades sharply as horizons lengthen. A long-horizon task that fails after consuming 95% of its planned budget is far more costly than a short task that fails outright, yet the two are scored identically under terminal-success-only metrics. Partial-credit evaluation has been operationalized in two narrow regions of the design space. VOYAGER [79] reports per-skill acquisition curves and incremental unique-item milestones rather than

only final game state, providing the clearest in-domain example of intermediate-progress scoring in an LLM-agent benchmark. Self-evolving robot manipulation work such as REMAC [61] reports subtask-level success and self-reflection-triggered repair rates on long-horizon kitchen-manipulation tasks. No widely adopted general-purpose long-horizon benchmark of which we are aware reports checkpoint-aware partial-credit scoring as a primary metric—recent office-workflow benchmarks such as OdysseyBench [80] retain terminal pass/fail scoring (`#successful tasks/#total tasks`) even on multi-step tasks. Cross-session evaluation—measuring whether a paused task can be resumed by a different runtime instance, whether learned-coordination policy parameters survive process restarts, whether intermediate artifacts remain semantically coherent across days—remains ad hoc; no widely adopted benchmark of which we are aware reports cross-session resumption fidelity as a primary metric. Section 8.3 operationalizes the framework on a public-trace empirical study (Study A on MAST-Data), supplemented by a pipeline-portability illustration in Appendix B. Together, while not themselves multi-session, they expose two of the three properties that long-horizon evaluation will need at scale, namely, (i) a partial-credit signal (via the failure-mode cross-walk of Study A) and (ii) cost telemetry under perturbation (via Appendix B), leaving the third, (iii) cross-session resumption fidelity, for future work.

Formal definitions of the six dimensions. To make the framework concrete rather than conceptual, we give a formula for each of the six dimensions of Table 7, grounded where possible in the coordination-topology graph of Definition 1. Consider an execution of a multi-agent system with topology G = (V, E), and define:

- T, the set of attempted tasks, and S ⊆ T, the subset completed successfully;
- M, the multiset of inter-agent messages produced during the execution, with each message m carrying an ordered sender–receiver pair edge(m) ∈ V × V;
- E_plan ⊆ V × V, the orchestrator's declared task-graph edges (the planned communication pattern), and U ⊆ M, the messages whose content is traceable to an edit in the final output artifact ("useful" messages);
- tok(m) and c(m), the token count and monetary cost of message m, and τ(N), the total token usage of the system instantiated with |V| = N agents.

The six dimensions are then defined as:

$$\mathsf{TaskPerformance} = \frac{|S|}{|T|}, \tag{1}$$

$$\mathsf{CoordinationEfficiency} = \frac{\sum_{m \in U} \mathrm{tok}(m)}{\sum_{m \in M} \mathrm{tok}(m)}, \tag{2}$$

$$\mathsf{Scalability} = \sigma = \frac{d\,\tau(N)}{dN}, \qquad N = |V|, \tag{3}$$

$$\mathsf{Robustness} = \frac{\mathsf{TaskPerformance}_{\mathrm{perturbed}}}{\mathsf{TaskPerformance}_{\mathrm{baseline}}}, \tag{4}$$

$$\mathsf{CostEfficiency} = \frac{\sum_{m \in M} c(m)}{|T|}, \tag{5}$$

$$\mathsf{EmergentBehavior} = \frac{\left|\{\, m \in M : \mathrm{edge}(m) \notin E_{\mathrm{plan}} \,\}\right|}{|M|}. \tag{6}$$

Two of the six are defined directly over the topology graph G of Definition 1: EmergentBehavior (Equation (6)) counts exactly the messages whose sender–receiver edge falls outside the declared edge set E_plan, and Scalability (Equation (3)) is differentiated with respect to the vertex count |V|. CoordinationEfficiency (Equation (2)) is defined over the messages that flow along the edges of G but is a message-level ratio rather than a purely

structural quantity. The remaining three (TaskPerformance, Robustness, and CostEfficiency) are outcome- or resource-ratio-independent of the graph.

Table 7. 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†) [29]	Qualitative: strong on conversational multi-turn tasks	HumanEval 85.9%, MBPP 87.7% (peer-reviewed) [27]
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) [27]
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 [29])	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) [27]
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

Because not every quantity above is recoverable from every execution log, the two empirical studies operationalize the dimensions with lower-bound proxies. Section 8.3 (Study A) computes the three text-recoverable dimensions on the public MAST-Data corpus: TaskPerformance directly (success = all 14 MAST failure flags zero); CoordinationEfficiency via the *inverse* proxy coordination_overhead = (n_distinct_role_tags + n_role_transitions)/n_lines (a higher overhead indicates lower coordination efficiency; we report the overhead directly and do not invert it, so the column is read as overhead, not efficiency); and EmergentBehavior via the proxy task_retry_rate = n_retry_hits/n_lines (a recoverable lower-bound signal of off-plan recovery activity). Appendix B (Study B) computes the three telemetry-dependent dimensions—Scalability, Robustness, and CostEfficiency—on a live CrewAI harness over agent counts $N \in \{1, 2, 4\}$ with injected perturbations (Table A3).

Illustrative application. Table 7 applies the six dimensions qualitatively to three representative systems, illustrating how the framework discriminates between approaches rather than merely benchmarking task performance.

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.

8.3. Operationalizing the Framework: An Empirical Study on Public Traces

The six-dimension framework in Table 7 is qualitative across all cells without explicit citation. To demonstrate that the framework is operationalizable on real systems—and to respond to R2's request for a small empirical validation grounded in publicly available traces—we report an empirical study (Study A) on the publicly released MAST-Data corpus, supplemented by an illustrative pipeline-portability demonstration (Study B; Appendix B) that exercises the same six-dimension framework end to end on a controlled CrewAI harness across two LLM backends (DeepSeek-V4-Flash and Gemini-2.5-Flash) over 108 live runs (54 per backend). The primary empirical claim of this section rests on Study A, which uses real annotated traces. Study B is reported in Appendix B as a self-contained illustration of pipeline portability—the extraction pipeline, perturbation harness, cost-accounting logic, and reproducibility code are real and deposited; the three reported numbers are direct measurements but rest on deliberately tiny proxy tasks. We explicitly do not claim a unified single-system validation of all six dimensions, because no single public artifact at the time of writing carries both the cross-framework breadth required for Study A and the per-call telemetry required for Study B.

Study A—MAST-Data cross-framework analysis. We use MAST-Data [81], a peer-reviewed public corpus of 1242 multi-agent execution traces (the publicly released HuggingFace snapshot `mcemri/MAST-Data` was accessed on 25 May 2026; the source paper [81] reports 1642 annotated traces, with the additional 400 being non-public Manus and restricted-access traces excluded from the open release) spanning seven frameworks (AG2, AppWorld, ChatDev, HyperAgent, Magentic-One, MetaGPT, and OpenManus) and seven benchmarks (GAIA, GSM-Plus, MMLU, OlympiadBench, ProgramDev, SWE-Bench-Lite, and AppWorld Test-C), with each trace annotated against MAST's 14-mode failure taxonomy. We extract the three dimensions that map cleanly to trajectory text. Letting `n_lines` be the number of newline-terminated lines in the raw trajectory string and matching role tags by the regex `\*\*\[([^\]]+)\]\*\*` (i.e., markdown-bolded-bracketed role labels such as `[CEO]`), the metrics are computed as follows: `task_performance` is binary success defined as all 14 MAST failure flags zero, consistent with the source paper's framing. `coordination_overhead_proxy = (n_distinct_role_tags + n_role_transitions) / n_lines`, where `n_distinct_role_tags` is the number of unique role labels appearing in the trajectory, and `n_role_transitions` is the number of consecutive-line role switches in the matched role-tag sequence; this is a lower-bound, framework-dependent proxy that registers zero for trajectory formats lacking the bolded-bracket role marker. `task_retry_rate = n_retry_hits / n_lines`, where `n_retry_hits` is the count of case-insensitive matches of `\b(retry|retries|retrying|exception|error|Traceback|RuntimeError|failed)\b` over the trajectory text. Both proxies normalize by `n_lines` rather than by an inferred task count, so they are densities per trajectory line and not per task attempt. We report the full nine (framework × benchmark) pairs available in the public layout (Table 8) and highlight five stratified-sample headline cells—ChatDev × ProgramDev, MetaGPT × ProgramDev, Magentic-One × GAIA, AG2 × GSM-Plus, and AppWorld × Test-C—selected because each has $n \geq 30$ unique-task traces after stratification and because together they span four distinct task domains (program development, long-horizon web research, math reasoning, and tool use); the first two pair the same benchmark across two frameworks (within-benchmark comparison), while the remaining three span distinct domains for breadth. Because MAST-Data is failure-biased by design (~80% failed traces, intentional in the source study), we report each proxy metric split by (successful, failed) subsets so that

the failure bias is visible rather than averaged away. The proxy nature of the coordination metric is critical: only frameworks whose trajectories contain markdown-bolded-bracketed role tags matching the regex `\*\*\[([^\]]+)\]\*\*` (for example, `[CEO]` in ChatDev or `[ReviewerAgent]` in HyperAgent) register a non-zero value, and within-framework comparison across benchmarks is therefore the only valid use of that column. A cross-walk between our three extracted metrics and MAST's 14 failure modes is provided in Appendix A.

Pipeline-portability illustration (Study B; deferred to Appendix B). The three remaining framework dimensions—`scalability_slope`, `robustness_to_perturbation`, and `cost_per_attempt`—require per-call token telemetry and the ability to inject controlled perturbations, neither of which is recoverable from static archived traces. Appendix B describes a self-contained CrewAI harness (six GAIA-Level-1 stand-in tasks; $N \in \{1, 2, 4\}$; perturbations $p \in \{0.0, 0.1, 0.2\} \times 2$ seeds; 54 runs per backend) executed live on two LLM backends—DeepSeek-V4-Flash and Gemini-2.5-Flash—via CrewAI's LiteLLM layer, yielding 108 total live runs. The cross-backend agreement on robustness and overall success rate (both 1.000 on each backend) and the bounded disagreement on `scalability_slope` (-32% relative, expected from per-model token efficiency differences) and `cost_per_attempt` (+207% relative, driven by published per-token price differences) support backend portability of the measurement harness on these stand-in tasks; broader claims about the model invariance of the metrics would require larger and harder benchmarks. The body of the paper makes no empirical claim that depends on the Appendix B magnitudes; Appendix B is included for reproducibility and as a runnable template for readers wishing to instantiate the framework on their own systems.

Scope. Study A satisfies R2's request for empirical validation using publicly available traces and is the primary empirical contribution of Section 8.3. The five highlighted cells in Table 8 are reported at full available n (130, 230, 195, 223, and 30 traces) for transparency, and the within-cell analyses underlying Section 8.3's headline comparisons are computed on a balanced stratified subset of n = 30 unique-task traces drawn from each highlighted pool (150 traces total stratified subset; four distinct task domains: program development, long-horizon web research, math reasoning, and tool use). Two files document this subset: (i) `study_a_stratified_sample_rows.csv`—the per-trace file (150 data rows, one per sampled trace), with columns `cell_label`, `trace_id`, `mas_name`, `llm_name`, `benchmark_name`, `mast_all_zero`, `coordination_overhead_proxy_value`, `task_retry_rate_value` and four auxiliary diagnostic columns; (ii) `study_a_stratified_summary.csv`—the per-cell aggregate (5 rows) that is the input to Table 8's stratified-subset summary. Sampling is reproducible from the raw MAST-Data JSON with seed `20260525` (Python `random.Random`) via `extract_metrics_study_a.py`. A total of 1242 traces are available in the public MAST-Data HuggingFace snapshot (the source paper [81] reports 1642 annotated traces; the difference is non-public Manus and restricted-access traces). Appendix B extends the operationalization to the three telemetry-dependent dimensions as a pipeline-portability illustration. Study A alone does not constitute a unified single-system validation of the framework; we treat the framework's value as residing in its ability to discriminate between systems on dimensions that current benchmarks largely ignore, and we demonstrate that ability on the MAST-Data corpus. Full extraction code, the MAST-Data snapshot access date (25 May 2026), the CrewAI harness, and the raw CSVs underlying Table 8 and the Appendix B values are in Appendixes A and B, and the Zenodo bundle.

Table 8. Study A operationalizes the three text-extractable dimensions of the six-dimension framework on MAST-Data [81], reporting all nine (framework × benchmark) pairs available in the public layout. Headline cells (marked *) are the five highlighted comparisons referenced in Section 8.3: ChatDev × ProgramDev and MetaGPT × ProgramDev share a benchmark and permit within-benchmark cross-framework comparison; Magentic-One × GAIA covers a long-horizon web-research domain; AG2 × GSM-Plus adds math-reasoning breadth; AppWorld × Test-C adds tool-use breadth. Starred (*) rows report the full available n (130, 230, 195, 223, 30) for transparency; the stratified-subset analyses in Section 8.3 draw n = 30 unique-task traces from each of these five pools (150 traces total—per-trace rows in `study_a_stratified_sample_rows.csv`; per-cell aggregate in `study_a_stratified_summary.csv`). Unstarred rows are reported as the full public split. `task_perf` is binary success defined as all 14 MAST failure flags zero. `coord_proxy` is `(n_distinct_role_tags + n_role_transitions) / n_lines`, where role tags are matched by the regex `\*\*\[([^\]]+)\]\*\*`—a lower-bound text proxy that registers zero for frameworks whose trajectories lack the bolded-bracketed role marker (AG2, AppWorld, Magentic, MetaGPT, and OpenManus), so cross-framework comparison on this column is not valid; only within-framework comparison across benchmarks is meaningful. `task_retry_rate` is `n_retry_hits / n_lines`, where `n_retry_hits` counts case-insensitive matches of `\b(retry|retries|retrying|exception|error|Traceback|RuntimeError|failed)\b` (i.e., per line density, not per task attempt). Two label aliases are used: MAST-Data labels the Microsoft system as `Magentic` (we retain this label in CSV/code to match the upstream corpus and use `Magentic-One` [43] in narrative prose); MAST-Data labels the math benchmark as `GSM`, which we refer to as `GSM-Plus` [82] in narrative prose to disambiguate from the original GSM8K; both labels refer to the same MAST-Data cell. The `(failed)` columns report each proxy restricted to the failed-trace subset so that the failure-bias of MAST-Data (~80% failed by source-paper design) is visible rather than averaged away. Standard deviations and the parallel `(successful)` split are in Appendix A Table A2.

Framework × Benchmark	n	Task_Perf (Success%)	Coord_Proxy (Mean)	Task_Retry_Rate (Mean)	(Failed) Coord	(Failed) Retry
AG2 × GSM-Plus *	223	19.7%	0.0000	0.171	0.0000	0.190
AG2 × MMLU	168	16.1%	0.0000	0.131	0.0000	0.156
AG2 × OlympiadBench	206	14.1%	0.0000	0.607	0.0000	0.576
AppWorld × Test-C *	30	53.3%	0.0000	0.013	0.0000	0.014
ChatDev × ProgramDev *	130	28.5%	0.0248	0.017	0.0254	0.017
HyperAgent × SWE-Bench-Lite	30	53.3%	0.0014	0.020	0.0000	0.023
Magentic-One × GAIA *	195	22.1%	0.0000	0.003	0.0000	0.003
MetaGPT × ProgramDev *	230	25.2%	0.0000	0.018	0.0000	0.018
OpenManus × ProgramDev	30	53.3%	0.0000	0.052	0.0000	0.052

Interpretation guardrails. 1. The `coord_proxy = 0.0000` rows reflect a measurement-instrument limitation (the framework's trajectory format does not surface explicit role tags), not zero coordination overhead. Any comparison of coordination overhead between, e.g., ChatDev (0.0248) and MetaGPT (0.0000), on the same ProgramDev benchmark is invalid; the comparison says only that ChatDev's trajectories carry role-tag markdown and MetaGPT's do not. 2. Within the AG2 row group (which uses a uniform trajectory format), the rank order of `task_retry_rate` across benchmarks—GSM 0.171, MMLU 0.131, and

OlympiadBench 0.607—is consistent with the source paper's observation that long-form math reasoning surfaces more error-handling text in trajectory. Caveat: In the AG2 × GSM stratified sample, 24 of 30 traces have `n_lines = 1`, making the per-line normalization near-binary on most rows; the GSM/MMLU rank gap is therefore small and should be read as suggestive rather than diagnostic. 3. The headline `task_perf` numbers (ChatDev 28.5% and MetaGPT 25.2% on the same ProgramDev benchmark; Magentic-One 22.1% on GAIA) are direct cross-walks to MAST's success/failure framing and are the most defensible single-column comparison in Table 8. The three rows with n = 30 (AppWorld, HyperAgent, and OpenManus) all show 53.3% task_perf because they share the same 30-trace public split (16 successful, 14 failed) with no further sampling possible; their values are reported for completeness but should not be over-interpreted as cross-framework rankings.

Raw CSV (all nine cells at full n): `execution/code/study_a_results.csv`. Stratified subset per-trace rows (150 rows = 5 cells × 30 traces, one row per sampled trace): `execution/code/study_a_stratified_sample_rows.csv`. Stratified subset per-cell aggregate (5 rows, input to Table 8 headline statistics): `execution/code/study_a_stratified_summary.csv`. Extraction code (reproducible with seed 20260525): `execution/code/extract_metrics_study_a.py` (Appendix A).

The full Study B table, design summary, honest scope statement, and raw-data pointers are deferred to Appendix B (Pipeline-Portability Illustration) to keep Section 8.3's main empirical claim resting on Study A's real annotated traces; Study B contributes a runnable backend-agnostic harness rather than a load-bearing empirical claim.

9. Applications

Multi-agent orchestration has moved into early production pilots, with reported metrics qualified per the evidence hierarchy in Section 7.6: vendor announcements and adopter case studies make up the bulk of the available numbers, while peer-reviewed and government-evaluator measurements remain comparatively scarce. Table 9 maps domains to orchestration patterns, representative systems, key metrics, and maturity levels under that same evidence-tier discipline.

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 (peer-reviewed) [27,44]. OpenHands' CodeActAgent reports 26% on SWE-Bench Lite (peer-reviewed) [52]; the public SWE-bench leaderboard (vendor-/leaderboard-reported†) [53] reports leading systems above 70% on SWE-bench Verified by late 2025, although Section 7.6(iii) shows that the file-path identification diagnostic on the same systems drops from up to 76% on SWE-Bench Verified to <53% on out-of-distribution repositories, indicating that a non-trivial fraction of the headline benchmark score reflects repository memorization rather than transferable coding reasoning [36] (the diagnostic isolates memorization signal and is not a direct end-to-end issue-resolution measurement). Adjacent agentic coding tools—Cursor, Claude Code [83], Windsurf [84], and Amazon Q Developer [85]—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†, single-vendor self-disclosure, scope not externally audited) that Q Developer generated a substantial share of new internal Amazon code during 2024 [85]. Controlled studies of single-agent assistants further sharpen the interpretation: the GitHub-authored Copilot RCT reports a 55.8% speedup on a scripted HTTP-server task (vendor-reported†) [67], while METR's independent RCT on 16 experienced open-source developers working in their own repositories finds that AI assistance makes them 19% slower (independent preprint) [68]—the sign reversal documented in Section 7.6(i). Explicitly multi-agent coding systems

include MetaGPT and OpenHands, which coordinate specialized agents for planning, coding, and testing; we are aware of no peer-reviewed RCT replicating vendor-claimed productivity gains for a deployed multi-agent coding system specifically.

Scientific research systems coordinate literature review, hypothesis generation, experiment design, and manuscript preparation. The AI Scientist [86] produces full research papers at approximately $15 each; ChemCrow [87] integrates 18 chemistry tools to synthesize compounds autonomously.

Business process automation shows ROI claims that are almost entirely vendor- or self-reported. Klarna's AI assistant (powered by OpenAI rather than a publicly disclosed multi-agent orchestration stack) was reported to handle two-thirds of support chats in early 2024—2.3 million conversations—reducing average resolution time from 11 min to under 2 min (vendor-reported†, first-month snapshot, no independent audit) [88]. 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†, scope limited to a single customer announcement) [89]; ServiceNow reports AI agent-driven IT incident deflection in early deployments (vendor-reported†, no public methodology) [90]. Deloitte's 2025 survey (industry survey†, self-reported organizational data) finds that organizations deploying agentic systems report 15–30% cost reduction in targeted processes, with 6–18 month payback periods [91]. Two cautions from Section 7.6 apply directly: (a) for the Microsoft 365 Copilot UK enterprise pilot, the independent UK Department for Business and Trade (DBT) evaluation [69] reports self-reported time savings that did not translate into externally observed productivity improvement at the team level (government evaluation), and (b) Salesforce's CEO publicly addressed low Agentforce adoption at Dreamforce '25, roughly twelve months after the Agentforce general-availability (GA) announcement (industry trade press, complementary signal only) [92]. Single-vendor numbers in this domain should therefore be treated as a lower bound on the variance in true deployment outcomes, not as point estimates of expected gain.

Healthcare multi-agent systems are best illustrated by two systems that take divergent deployment paths. Hippocratic AI's Polaris [93] is a one-trillion-parameter "constellation" architecture in which a 70B–100B stateful primary conversation agent coordinates with multiple multibillion-parameter specialist support agents (the preprint identifies labs/vitals, medication reconciliation, hospital/payor policy, checklist, and human-intervention specialists, among others); the architecture and the >3475-conversation clinician evaluation (over 1100 registered nurses and over 130 physicians) are documented in the technical-report preprint [93]. Subsequent live patient-facing voice deployments at partner US and UK health systems are vendor-reported† separately from [93] and are not entailed by the preprint itself. Google's AMIE takes the more conservative research-deployed path: a self-play-trained diagnostic dialogue system evaluated in a randomized double-blind OSCE-style study with validated patient actors (peer-reviewed, Nature 2025) [94]. The two systems contrast sharply on the regulatory axis: Polaris treats safety as architecture (a policy-specialist agent surfaces HIPAA-relevant policy at runtime), AMIE treats safety as an evaluation protocol (evaluated only in simulated OSCE-style consultations, with real-world translation deferred to future work). The orchestration-taxonomy mapping (centralized with hierarchy for triage→differential→safety-check→synthesis; hierarchical decomposition with structured intermediate artifacts for medical coding; panel-of-experts patterns for pharmacovigilance), and the state-management implications discussed in Section 5.8 apply directly: only frameworks with explicit checkpointed state (LangGraph-style [29]) can support the mutation-level rollback that a clinician auditor would expect when investigating an adverse event. The maturity entry "Early production pilots" in Table 9 reflects the fact that Polaris's deployment scope is vendor-reported and AMIE

remains research-deployed; per Section 7.6, peer-reviewed RCT-grade outcome data for a deployed multi-agent clinical system is absent at the March 2026 cutoff, and EU AI Act high-risk obligations [38], together with the 2025 AI Agent Index disclosure-gap finding (where 25/30 systems disclose no internal safety evaluation [37]), make process-level audit trails—not just outcome metrics—the binding regulatory constraint going forward.

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 [95] 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. As in healthcare, regulatory constraints shape admissible orchestration patterns: in the United States, FERPA's restrictions on disclosing personally identifiable student records argue for data isolation between agents that hold raw learner records (the profiler/assessor) and agents that generate or retrieve shared content, so that a content-generation agent operates on de-identified skill-gap descriptors rather than on raw grades—a per-agent scope constraint analogous to the HIPAA-driven access scoping discussed above for clinical settings.

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 [32], 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 that the rubric criteria themselves drive expert recruitment, and disagreement among graders triggers the re-recruitment of additional specialist evaluators rather than a forced consensus. Second, DSPy's compiler-based optimization [51] 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 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 [51] 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).

Autonomous systems. VOYAGER [79] achieves 3.3× unique item acquisition and up to 15.3× 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 [96], 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 9. Application domains for multi-agent orchestration: representative systems, dominant patterns, key metrics, and deployment maturity. Maturity labels reflect the evidence hierarchy in Section 3.1; vendor-reported (†) deployments are noted explicitly and are not equated with independently verified production maturity (see Section 7.6).

Domain	Dominant Pattern(s)	Representative Systems	Key Metrics (with Evidence Tier)	Maturity
Software development	Hierarchical, dynamic	MetaGPT, OpenHands (multi-agent); Cursor, Claude Code †, Amazon Q Developer † (adjacent agentic)	MetaGPT HumanEval: 85.9%; MBPP: 87.7% (peer-reviewed) [27]; OpenHands: 26% on SWE-Bench Lite (peer-reviewed) [52]; SWE-bench Verified leaders: >70% (leaderboard, as of March 2026) [53], with Section 7.6(iii) showing a 76% → <53% file-path identification drop on out-of-distribution repos [36] (memorization diagnostic, not end-to-end SWE-Bench resolution)	Production (multi-agent research systems peer-reviewed); vendor-reported production † for adjacent single-agent coding tools
Scientific research	Hierarchical, centralized	AI Scientist [86], ChemCrow [87]	~$15 per generated paper (author-reported, small-sample) [86]; 18-tool chemistry integration (peer-reviewed) [87]	Experimental
Business/ customer service	Centralized, hierarchical	Klarna AI [88], Salesforce Agentforce [89], ServiceNow [90]	Klarna: two-thirds of support chats automated, 11 min → <2 min (vendor-reported †, single-vendor snapshot) [88]; Deloitte survey: 15–30% cost reduction (industry survey †) [91]; UK DBT independent evaluation of M365 Copilot [69] reports self-reported time savings that did not translate into externally observed team-level productivity (government evaluation; Section 7.6(ii))	Early production pilots (vendor-reported †); independent corroboration mixed
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 pilots (vendor-reported †)
Healthcare	Hierarchical, centralized	Hippocratic AI Polaris (one-trillion-parameter constellation; primary + multiple specialist agents) [93]; Google AMIE (self-play-trained diagnostic dialogue; double-blind OSCE study) [94]	Polaris: preprint architecture + >3475-conversation clinician-rated evaluation [93]; live deployment scope at partner health systems is vendor-reported † separately; AMIE: randomized double-blind OSCE-style study with validated patient actors (peer-reviewed, Nature 2025). No peer-reviewed deployed-system patient-outcome RCT at survey cutoff	Polaris: preprint architecture + vendor-reported † pilots; AMIE: research-deployed
Education	Hierarchical, decentralized	FACET (teacher-facing, learner + teacher + evaluator agents) [97]; GenMentor—LLM-powered multi-agent framework for goal-oriented learning in ITS (skill-identifier + learner-profiler + path-scheduler + learner-simulator + content-creator agents) [98]	FACET: in-service K-12 teacher evaluation on grade-8 mathematics curriculum (preprint, pilot); WWW '25: ITS-integrated evaluation (peer-reviewed). MT-Bench reaches >80% agreement with human annotators for single-judge pairwise comparison (peer-reviewed) [95]; transfer to deployed multi-agent grading panels not established	Pilot (research-deployed; consumer products such as Khanmigo, Duolingo Max, LiveHint AI are single-LLM and excluded)
Autonomous systems	Hierarchical, decentralized	VOYAGER [79], Claude Computer Use [96]	VOYAGER: 3.3× unique-item acquisition, up to 15.3× milestone speed vs. prior methods in Minecraft (peer-reviewed) [79]	Experimental–pilot

Maturity legend (refined under Section 3.1/Section 7.6 discipline): experimental (research demonstrations); research-deployed (peer-reviewed evaluation in a realistic but non-commercial settings, e.g., OSCE studies); pilot (limited real-world deployment, often at single site or with single customer); early production pilots (vendor-reported†) (deployed in commercial settings but reported metrics rely on vendor self-disclosure without independent audit); production (scaled commercial deployment with peer-reviewed or independent-evaluator corroboration).

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 [59]. 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 4. MCP's session model assumes a single trust boundary with transport-level authentication [40,59]; 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 [39,59]; ANP's three-layer design assumes neither central registries nor pre-established trust, replacing both with W3C decentralized identifiers and JSON-LD application descriptions [58]. 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 [58]. None of these translations is conceptually hard, but the absence of canonical mappings means that each integration is currently bespoke, with predictable interoperability failures at the seams. The Ehtesham et al. survey [59] 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 [33] 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—e.g., spreading event invitations, forming relationships, and coordinating arrival times—without explicit programming. AgentVerse [32] demonstrated that dynamically adjusting agent group composition during task execution outperforms static teams on collaborative problem solving. OPTIMA [63] shows 2.8$\times$ performance gains with under 10% of tokens through training-based communication optimization. Translating these research results into 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 ei carries a probability θi that is trained against a downstream utility via REINFORCE [33]. 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 [33]. DyLAN's temporal feed-forward network [34] 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, and then the pruned team executes the task with optional early stopping. The 25.0% accuracy gain reported on certain MMLU subjects [34] 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 (e.g., communication efficiency and 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 [77] places LLM-as-Agent in eight 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 [77]. This kind of cross-environment finding cannot be recovered from any single-environment leaderboard. τ-bench [78] 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 ~61% pass^1 on τ-retail and drops to ~25% pass^8, exposing a reliability gap that a one-shot accuracy number entirely hides [78]. OSWorld [76] 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 [76]. None of these benchmarks evaluate 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, and 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 [16] 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 20× cost reduction at comparable accuracy; DSPy's automatic optimization achieves 25–65% improvement without additional inference cost [51]. 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 [99], 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 [51]; 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 [60] 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 [27], 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 [33]? 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 [100]. 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 [65] demonstrates that adversarial prompts can self-replicate across interconnected agents, and the 2025 AI Agent Index [37] documents that only 4 of 30 surveyed agents provide agent-specific system cards. The EU AI Act [38] 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 [37]. 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 [37]. Even capability disclosure is asymmetric, with 9/30 systems publishing 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" [37]. Set against the threat model of Prompt Infection [65], 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 [100]—compound rather than cancel when multiple such systems are coupled through orchestration. The EU AI Act's high-risk-system

obligations (transparency, explainability, and human oversight) [38] thus need a multi-agent-aware operational interpretation that current SDK observability primitives—even tracing-first ones such as the OpenAI Agents SDK [49]—only partially provide.

10.6. Self-Evolving Agent Organizations

Self-evolving systems occupy the membership-mutating and/or learned-coordination sub-modes defined in Section 4.2 of our two-dimensional taxonomy, and the open challenges below are properties of those sub-cells rather than of a separate orchestration pattern.

Six recent systems concretize what self-evolution currently looks like. AgentVerse [32] introduces a four-stage loop in which a recruiter dynamically assembles a fresh panel of expert agents at every reasoning round and pairs this with an empirical study of emergent constructive and destructive group behaviors. GPTSwarm [33] reframes a multi-agent system as a computational graph with learnable nodes and edges, so that both prompts and inter-agent connectivity become differentiable objects optimized via REINFORCE-style updates. AutoAgents [101] separates a Drafting stage that auto-generates a task-specific roster of agents and a plan from an Execution stage in which observer agents continuously refine both, eliminating the need for a hand-designed team. DyLAN [34] treats coordination as a temporal feed-forward network and uses an Agent Importance Score to prune and reactivate agents across rounds, reporting up to a 25% MMLU subject-level gain over fixed teams. EvoMAC [48] performs test-time textual backpropagation that rewrites both agent prompts and inter-agent edges from environmental feedback during a single task. MaAS/Agentic Supernet [45] learns a supernet from which a controller samples query-conditioned sub-architectures, yielding 6–45% of the inference cost of fixed-topology baselines at +0.5 to +16.9% accuracy across six benchmarks.

Three open questions emerge once these systems are read together. First, stability under distribution shift: none of the six papers report what happens to the learned recruitment, importance, or sampling policy when the task stream drifts; whether the supernet of MaAS remains well-calibrated for queries outside its training mixture is unknown. Second, alignment between learned coordination and intended behavior: AgentVerse already documents emergent destructive interactions in hand-designed groups, and the systems that *learn* their coordination policy (GPTSwarm, DyLAN, EvoMAC, and MaAS) currently provide no mechanisms to verify that the optimized graph still implements the user's intent rather than a shortcut that happens to score well on the held-out benchmark. Third, evaluation: improvements are reported on static benchmarks, but a self-evolving system's defining property is that its structure depends on the input distribution it has seen—a property static benchmarks cannot probe.

These open questions intersect Section 10.8 (Generalization of Learned Orchestration), which formalizes the distribution-shift question, and Section 10.7 (Long-Horizon Coordination), where membership changes inside a single long-horizon task remain an unsolved combination.

10.7. Long-Horizon Coordination and Planning

Long-horizon coordination is a property of the task rather than of the topology or adaptivity dimension; as already noted in Section 4.3, it cuts across every cell of the grid in Section 4, and the challenges below are therefore independent of whether the base topology is centralized, decentralized, or hierarchical and independent of whether routing is static or learned.

The first challenge is the memory–coordination interplay. Once a task spans hundreds of LLM calls or multiple sessions, the coordination policy must consult an external state that has itself grown to a size that no single agent can hold in context. Decisions about *who*

acts next become entangled with decisions about *which slice of memory to surface*, and neither the workflow-orchestration view (Section 5) nor the organization-level coordination view (Section 4) currently treats the memory channel as a first-class part of the coordination graph. AgentDropout [62], although primarily a topology-pruning method, shows that token cost is inflated by redundant agent roles and redundant information exchange—suggesting, by extension, that memory and topology cannot be designed independently as horizons lengthen.

The second challenge is checkpoint-based evaluation. Existing benchmarks reward end-to-end task success and therefore conflate "the system never made progress" with "the system made substantial progress then drifted." A long-horizon evaluation needs explicit intermediate checkpoints, partial-credit scoring, and tests that re-enter the same task across sessions to probe whether the planner's state survives. How to score *partial progress* on a multi-day task in a way that is both reproducible and not gameable remains an open methodological question, and one that the static, single-shot benchmarks currently used to validate self-evolving systems (Section 10.6) cannot answer.

The third challenge is the state–policy distinction, which we develop in Section 7.1.1. A planner that re-reads the same context but emits divergent next actions across sessions has a *policy* drift problem; a planner that emits consistent actions when given consistent state but receives inconsistent state has a *state* drift problem. The two failure modes have different fixes—policy drift calls for output-level reproducibility tests and state drift for replay-style memory audits—but current evaluations rarely separate them. A practical open question is how to detect drift across sessions while a task is still running, rather than only post hoc after the task has visibly failed.

These challenges also interact with Section 10.6 (Self-Evolving Agent Organizations); a long-horizon task in which the agent roster itself mutates between sessions occupies a combination of cells that no current benchmark targets and is, to our knowledge, the most under-evaluated configuration in the grid in Section 4.

10.8. Generalization of Learned Orchestration

Section 4.2 introduced learned coordination as the most expressive runtime-adaptivity sub-mode, exhibited by GPTSwarm, DyLAN, MaAS, and similar systems. Their generalization properties—whether the learned policy transfers across domains, survives distribution shift, and remains safe under adversarial input—are still poorly characterized, and the recent literature converges on three specific open questions.

Cross-domain transfer. OFA-MAS [102], accepted at *The Web Conference (WWW) 2026*, is the most-advanced-stage peer-reviewed work we are aware of in this category and reports cross-domain transfer systematically: a single mixture-of-experts graph generative model trained jointly across six in-distribution benchmarks (math, code, general reasoning, and others) reaches 93.02% average accuracy and outperforms per-domain "one-for-one" topology specialists; the same model is then evaluated zero-shot on GAIA as a held-out out-of-distribution benchmark, providing direct evidence that a jointly learned topology policy transfers both within the training distribution and to an unseen domain. TopoPrior [103] complements this with a conditional variational graph prior that is learned offline on multi-domain collaboration graphs and used to initialize several heterogeneous topology-evolution backbones, yielding consistent accuracy gains and reduced inference-time token consumption on unseen-domain queries. Of the two cross-domain-transfer systems surveyed here, OFA-MAS is accepted at a clearly identified peer-reviewed venue (WWW '26) while TopoPrior remains a preprint; this cross-domain-transfer literature is otherwise nascent, which is itself a maturity observation worth reporting.

Distribution shift. MasFACT [104] names and quantifies a new failure mode—"topology forgetting"—in which adapting a learned topology generator to a new task domain shifts it away from structures needed by earlier domains. Across class-, domain-, and task-level continual settings, MasFACT combines Fused Gromov–Wasserstein optimal transport with a PAC-Bayes conservative posterior and improves average accuracy while reducing forgetting versus topology-generation and replay baselines, giving the field its first systematic distribution-shift measurement for learned MAS topologies. AgentDropout [62] approaches the same question from the opposite side, evaluating the domain transferability and structural robustness of a learned sparse topology and reporting −21.6% prompt tokens, −18.4% completion tokens, and +1.14 average task improvement—one of the few learned-topology papers that *measures* transfer rather than only proposing an architecture.

Safeguards. NetSafe [105] is the first systematic safety study of MAS organized by topology; highly connected networks are shown to propagate adversarial signals more readily, with task performance under attack dropping by 29.7% in a Star Graph Topology, and "Agent Hallucination" and "Aggregation Safety" are identified as emergent failure phenomena. FlowSteer/FlowGuard [106] attacks the planner of learned planner–executor MAS via prompt-only workflow steering, raising malicious-signal success by up to +55% over naive prompting, and shows that the proposed input-side FlowGuard defense cuts malicious success by up to 34% while preserving prompt utility—concrete numbers for an empirically evaluated safeguard rather than a proposal-only safety architecture. A clean gap remains, however: no paper we located evaluates whether a learned safeguard transfers across MAS topologies—i.e., whether a runtime monitor or input filter learned on one coordination graph still defends a different one.

These three properties—cross-domain transfer, robustness under distribution shift, and safeguards whose transferability is itself measured—are the minimum a deployable self-organizing system must demonstrate; the absence of cross-system safeguard evaluation is itself a research-direction observation, and one that the membership-mutating and/or learned-coordination sub-modes of Section 4.2 will have to address before the systems surveyed in Section 10.6 can be responsibly fielded.

11. Conclusions

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 1200, expanded to 104 in revision, ensuring comprehensive yet focused coverage. The findings go beyond cataloging.

On orchestration patterns: The three coordination topologies—centralized, decentralized, and hierarchical—and the dynamic–adaptive modifier are complementary building blocks, not competing alternatives. The decision framework in Section 4.7 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—20× 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, leaderboard-reported†) [53], GAIA's 92% vs. 15% gap, WebArena's 78% vs. 14% gap, and 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, and Section 7.6(i)–(iii) further document both a 76% → <53% in-distribution/out-of-distribution drop attributable to memorization [36] and inconsistent disclosure of which Verified instances vendor leaderboard submissions actually attempt. On applications, the evidence base ranges from peer-reviewed research-deployed systems (Google AMIE in healthcare [94]) to vendor-reported early production pilots (Hippocratic AI Polaris [93]; Klarna, Salesforce Agentforce, and ServiceNow customer-service deployments) to experimental work (autonomous scientific research). The 57.3% production-deployment rate reported by LangChain (vendor-reported†, 1340-respondent industry survey) [13], the customer-service ROI figures (vendor-reported†; UK DBT government evaluation [69] finds self-reported time savings that did not translate into externally observed team-level productivity (see Section 7.6(ii))), and the documented sign reversal between vendor and independent RCTs of single-agent assistants (Section 7.6(i)) together support an early shift from research demonstrations to operational pilots, where the strongest evidence remains vendor-reported and warrants the labeling discipline in Section 3.1/Section 7.6, rather than a meaningful claim of mature production deployment at scale.

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. A fourth, methodological factor has become equally load-bearing as the field matures: evidence-tier discipline (Sections 3.1 and 7.6) that distinguishes vendor-reported deployment claims from peer-reviewed and government-evaluator measurements, so that the absence of independently replicated multi-agent ROI is not papered over by the abundance of vendor case studies. 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 (MetaGPT's structured outputs halving per-line token consumption versus ChatDev [27] and GPTSwarm's roughly 20× cost reduction at comparable accuracy via learned topologies [33] remain the strongest existence proofs that the cost levers exist), and safety must be engineered in from the start. Closing the eight open challenges identified in Section 10—protocol bridging, self-organization, evaluation methodology, cost optimization, trust/safety, self-evolving organizations, long-

horizon coordination, and generalization of learned orchestration—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, 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 study received no external funding.

Data Availability Statement: This survey uses MAST-Data (Cemri et al., NeurIPS 2025) [81], whose public HuggingFace release contains 1242 annotated multi-agent execution traces (the source paper [81] reports 1642 annotated traces in total; the public release excludes non-public Manus and restricted-access subsets). The dataset is distributed at https://huggingface.co/datasets/mcemri/MAST-Data under the CC BY 4.0 license (snapshot accessed 25 May 2026; no commit SHA is exposed by the HuggingFace dataset listing—provenance is anchored to the access date). The 150-trace stratified subset used in Section 8.3 (Study A, per-trace rows in `study_a_stratified_sample_rows.csv` and per-cell aggregate in `study_a_stratified_summary.csv`), the full-corpus per-pair results (`study_a_results.csv`), the metric-extraction pipeline (`extract_metrics_study_a.py`, reproducible from the raw MAST-Data JSON with seed `20260525`), and the CrewAI pipeline-portability harness (Study B) in Appendix B are deposited in the updated supplementary Zenodo bundle. Study B (Appendix B) consists of 108 live multi-agent runs (54 on DeepSeek-V4-Flash and 54 on Gemini-2.5-Flash) reproducible from the published harness; the harness is backend-agnostic via CrewAI's LiteLLM layer. Other survey-level supplementary materials (PRISMA screening log, database query set, per-reference evidence-tier ratings, machine-readable serialization of the decision framework in Section 4.7) are also included in the same bundle, deposited at Zenodo, https://doi.org/10.5281/zenodo.19774439 (concept DOI, resolving to the latest version; current version 10.5281/zenodo.20469702), under the CC BY 4.0 license; the final version DOI will be confirmed at proof stage per MDPI policy. All primary literature 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.

Abbreviations

The following abbreviations are used in this manuscript:

A2A	Agent-to-Agent (Protocol)
AAIF	Agentic AI Foundation
ACP	Agent Communication Protocol
AI	Artificial Intelligence
ANP	Agent Network Protocol
API	Application Programming Interface
CEO	Chief Executive Officer
CLI	Command-Line Interface
CSV	Comma-Separated Values
DBT	Department for Business and Trade
DID	Decentralized Identifier
DOI	Digital Object Identifier
FERPA	Family Educational Rights and Privacy Act
FIPA	Foundation for Intelligent Physical Agents
GAIA	General AI Assistants (benchmark)

GPT	Generative Pre-trained Transformer
GUI	Graphical User Interface
HIPAA	Health Insurance Portability and Accountability Act
HITL	Human-in-the-Loop
HTTP(S)	Hypertext Transfer Protocol (Secure)
IETF	Internet Engineering Task Force
IRB	Institutional Review Board
ITS	Intelligent Tutoring System
JSON	JavaScript Object Notation
JSON-LD	JSON for Linked Data
JSON-RPC	JSON Remote Procedure Call
LATS	Language Agent Tree Search
LLM	Large Language Model
LSP	Language Server Protocol
MAS	Multi-Agent System
MAST	Multi-Agent System Failure Taxonomy
MCP	Model Context Protocol
MIME	Multipurpose Internet Mail Extensions
NIST	National Institute of Standards and Technology
OAuth	Open Authorization
OCI	Open Container Initiative
OSCE	Objective Structured Clinical Examination
P2P	Peer-to-Peer
PAC	Probably Approximately Correct
PRISMA	Preferred Reporting Items for Systematic Reviews and Meta-Analyses
RCT	Randomized Controlled Trial
ReAct	Reasoning and Acting
REST	Representational State Transfer
RFC	Request for Comments
RLHF	Reinforcement Learning from Human Feedback
ROI	Return on Investment
SDK	Software Development Kit
SOP	Standard Operating Procedure
SOX	Sarbanes–Oxley Act
SPOF	Single Point of Failure
SSE	Server-Sent Events
TCP/IP	Transmission Control Protocol/Internet Protocol
ToT	Tree of Thoughts
UML	Unified Modeling Language
URL	Uniform Resource Locator
W3C	World Wide Web Consortium

Appendix A. Study A: Extraction Details and Per-Cell Variance

This appendix provides the artifacts referenced in Section 8.3: (i) a cross-walk between our three text-extractable metrics and MAST's 14 failure modes and (ii) per-cell standard deviations together with the `(successful)` proxy split that Table 8 in the main text omits for compactness.

Appendix A.1. Cross-Walk: Extracted Metrics ↔ MAST 14 Failure Modes

The MAST failure taxonomy [81] organizes 14 failure modes into three categories—*specification* (1.1–1.5), *inter-agent misalignment* (2.1–2.6), and *task verification* (3.1–3.3). Our three text-extractable metrics each carry distinct coverage with respect to that taxonomy: one metric is a direct binary aggregate of all 14 flags, one is a structural proxy for a subset of

the *misalignment* category, and one is a textual proxy for a subset of the *verification* category. The mapping is given in Table A1; "Direct" means that the metric is mathematically defined by the flag (no inference), "Proxy" means that the metric measures a textual or structural footprint that the flag tends to produce but is not equivalent to the flag, and "Not covered" means that no signals in the metric correspond to the flag.

Table A1. Cross-walk between the three extracted metrics and MAST's 14 failure modes. The metrics aggregate (`task_performance`) or partially cover (`coord_proxy`, `retry_rate`) the taxonomy; no single text-extractable metric covers the full 14 modes, which is the operational basis for our claim in Section 8.3 that the framework is a discriminator not a replacement for the MAST hand-coded taxonomy.

MAST Mode	Category	Task_PERFORMANCE	Coord_PROXY	Retry_RATE
1.1 Disobey task specification	Specification	Direct (flag → fail)	Not covered	Not covered
1.2 Disobey role specification	Specification	Direct	Proxy (role-tag churn)	Not covered
1.3 Step repetition	Specification	Direct	Proxy (transition density)	Proxy (retry tokens)
1.4 Loss of conversation history	Specification	Direct	Not covered	Not covered
1.5 Unaware of termination conditions	Specification	Direct	Not covered	Proxy (loop → exceptions)
2.1 Conversation reset	Misalignment	Direct	Proxy (re-introduced role tags)	Not covered
2.2 Fail to ask for clarification	Misalignment	Direct	Not covered	Not covered
2.3 Task derailment	Misalignment	Direct	Proxy (role-set drift)	Not covered
2.4 Information withholding	Misalignment	Direct	Not covered	Not covered
2.5 Ignored other agent's input	Misalignment	Direct	Proxy (transition without role response)	Not covered
2.6 Reasoning–action mismatch	Misalignment	Direct	Not covered	Proxy (error tokens)
3.1 Premature termination	Verification	Direct	Not covered	Not covered
3.2 No or incomplete verification	Verification	Direct	Not covered	Proxy (absence of retry tokens despite failure)
3.3 Incorrect verification	Verification	Direct	Not covered	Proxy (Traceback/failed tokens)

Reading the table: `task_performance` is a Direct aggregate of all 14 flags by construction (the metric is defined as "all 14 flags zero"). The two proxy metrics each touch a subset: `coord_proxy` captures a structural footprint of seven of the eight *misalignment-style* modes that produce detectable role-tag churn in trajectories that emit bracketed role labels; `retry_rate` captures a textual footprint of five of the modes that produce exception, retry, or Traceback tokens. Crucially, no single extracted metric covers all 14 modes, and four modes (1.1, 1.4, 2.2, and 2.4) have *no* extractable footprint beyond the binary `task_performance` aggregate. This is the empirical basis for the claim in Section 8.3 that

the six-dimension framework is operationalizable but not a replacement for hand annotation of the MAST taxonomy.

Appendix A.2. Per-Cell Standard Deviations and Successful-Subset Split

Table 8 in Section 8.3 reports `coord_proxy` and `retry_rate` both as an unconditional mean (over all sampled traces) and as a `(failed)`-trace-restricted column, because MAST-Data is failure-biased by construction (~80% failed traces) and showing the failed-subset value alongside the overall mean keeps that bias visible. For completeness, Table A2 reports the standard deviation and the `(successful)` subset mean for each of the five highlighted stratified cells (n = 30 unique-task traces per cell, balanced within-cell). All numbers are computed from `study_a_stratified_summary.csv` (deposited in the Zenodo bundle), and the per-trace rows in `study_a_stratified_sample_rows.csv` allow recomputation.

Table A2. Full-sample standard deviations and `(successful)` subset means for the five stratified-subset cells. `n_succ` is the count of successful traces within the n = 30 stratified subset (it varies because the stratified sample preserves the within-cell success ratio rather than forcing a balanced success split). The `coord_proxy` and `retry_rate` columns report mean ± std over all 30 sampled traces, followed by the `(successful)` subset mean in parentheses.

Cell	n_succ/30	Coord_Proxy Mean ± Std (Successful)	Retry_Rate Mean ± Std (Successful)
ChatDev × ProgramDev	6	0.0249 ± 0.0052 (0.0181)	0.0111 ± 0.0136 (0.0014)
MetaGPT × ProgramDev	10	0.0000 ± 0.0000 (0.0000)	0.0093 ± 0.0148 (0.0064)
Magentic-One × GAIA	4	0.0000 ± 0.0000 (0.0000)	0.0036 ± 0.0064 (0.0000)
AG2 × GSM-Plus	7	0.0000 ± 0.0000 (0.0000)	0.0670 ± 0.2494 (0.0016)
AppWorld × Test-C	16	0.0000 ± 0.0000 (0.0000)	0.0127 ± 0.0087 (0.0120)

Reading the table: Three observations are worth highlighting. First, the `coord_proxy` column is identically zero for four of the five cells because their upstream frameworks (AG2, AppWorld, Magentic-One, and MetaGPT) do not emit the `[Role]` bracketed role markers that our regex matches; only ChatDev does. This is consistent with Section 8.3's caveat that cross-framework comparison on this column is not valid. Second, the `retry_rate` standard deviation for AG2 × GSM-Plus (0.249) is roughly four times the mean (0.067), driven by a small number of trajectories with very high exception-token density; the `(successful)` subset mean (0.0016) shows that this dispersion is concentrated in the failed subset, not the successful one. Third, ChatDev's successful traces show a non-zero `coord_proxy` (0.0181) substantially below the full-sample mean (0.0249), which means successful runs use fewer distinct role tags and fewer role transitions per line than failed runs—consistent with the qualitative intuition that successful coordination requires less back and forth than failed coordination and the basis for the within-framework analysis in Section 8.3. The full per-trace rows underlying these statistics are in `study_a_stratified_sample_rows.csv`; the extraction code (reproducible with seed `20260525`) is `extract_metrics_study_a.py`. Both are deposited in the Zenodo bundle alongside the raw MAST-Data snapshot access record (25 May 2026).

Appendix B. Pipeline-Portability Illustration (Study B)

This appendix reports a self-contained CrewAI harness that exercises the three telemetry-dependent dimensions of the six-dimension framework in Section 8.2—`scalability_slope`, `robustness_to_perturbation`, and `cost_per_attempt`—which

static archived traces (Study A) cannot recover. Its purpose is to illustrate end-to-end pipeline portability, that the same six-dimension framework can be instantiated on a live framework with controlled perturbations, and that the extraction pipeline itself is robust to the choice of the underlying LLM backend. To support the latter claim, we re-ran the entire harness on a second LLM backend (Gemini-2.5-Flash) and report cross-backend agreement on each of the three metrics (Table A3). It is not a load-bearing empirical claim of the survey; no statement in Sections 1–11 depends on the magnitudes reported below.

Table A3. Pipeline-portability illustration with cross-backend agreement: three telemetry-dependent dimensions evaluated on 108 live runs (54 per backend).

Metric	DeepSeek-V4-Flash (Mean)	Gemini-2.5-Flash (Mean)	Relative Diff (%)	Notes
`scalability_slope` (tokens / +1 agent)	9493.7	6447.4	−32.1%	Linear regression of mean (tokens_in + tokens_out) vs. N over N ∈ {1, 2, 4}, 6 tasks per N (18 runs per backend). The two backends agree in monotonicity (slope > 0 on both) and in order of magnitude ($\sim 10^4$ tokens/agent); the 32% gap is explained by Gemini-2.5-Flash producing systematically shorter intermediate critic/auditor messages on these tasks. Sequential CrewAI process; hierarchical or parallel topologies would change the slope on both backends. Absolute values not generalizable to GAIA-scale tasks.
`robustness_to_perturbation`	1.000	1.000	0.0%	`success_perturbed / success_baseline` over p ∈ {0.1, 0.2}, 6 tasks × 2 seeds × 2 perturbation levels (24 perturbed runs vs. 12 baseline runs at p = 0 per backend). Baseline success = 1.000; perturbed success = 1.000 on both backends. Both backends recovered from injected `calculator` / `search_context` failures by falling back to their own reasoning. Broader perturbation modes (LLM rate limit and parsing failure) untested; ceiling effect on these tasks should not be read as a general robustness claim.
`cost_per_attempt` (USD)	$0.001833	$0.005619	+206.5%	Each backend's published pricing applied to per-run token counts. The +207% gap reflects Gemini-2.5-Flash's higher per-token price ($2.50/M output vs. $1.10/M for DeepSeek-V4-Flash) more than any token-count difference; on a like-for-like token-count basis, the two backends use comparable totals (Gemini 295k total tokens vs. DeepSeek 420k total tokens over 54 runs). Per-run token counts derived from CrewAI's cumulative `token_usage` via per-call delta accounting.

Setup. Six GAIA-Level-1 stand-in tasks (G1–G6, see `study_b_tasks.csv`) execute on CrewAI with variable agent counts N ∈ {1, 2, 4} and Bernoulli tool-failure perturbations at p ∈ {0.0, 0.1, 0.2} across two random seeds, totaling 54 runs per backend (18 scalability + 36 perturbation). HuggingFace `gaia-benchmark/GAIA` is access-gated; we therefore use six GAIA-Level-1-shaped tasks (≥2 distinct reasoning steps each, fully self-contained; no live web access; deterministic ground truth as a short-string regex match). The two backends are: (i) DeepSeek-V4-Flash via the slug `deepseek-chat` (auto-routes to `deepseek-v4-flash` as of 2026-04-24; $0.07/M input + $1.10/M output) and (ii) Gemini-2.5-Flash via CrewAI/LiteLLM ($0.30/M input + $2.50/M output). The two CrewAI tools exposed to agents—`calculator` (restricted arithmetic eval) and `search_context` (offline keyword lookup over a fixed reference document)—are both subject to the Bernoulli perturbation.

Total spend was $0.099 for DeepSeek-V4-Flash (54 runs, 366 s wall) and $0.303 for Gemini-2.5-Flash (54 runs, 551 s wall), $0.402 across both backends, well under the $1.50 budget cap.

Cross-backend agreement. Of the three metrics, `robustness_to_perturbation` and the binary success metric (baseline 100%/perturbed 100% on both backends) agree exactly. `scalability_slope` agrees in sign, monotonicity, and order of magnitude, with a 32% relative gap that tracks per-backend token efficiency on intermediate critic messages. `cost_per_attempt` differs by 207% but is entirely explained by published per-token price differences rather than by the extraction pipeline. The pipeline itself—token accounting, perturbation injection, success grading, and cost computation—produces consistent measurements when the underlying LLM is swapped; this supports backend portability of the harness on the present stand-ins; we do not claim broader LLM invariance, which would require evaluation at scale and on harder tasks.

Design summary. Scalability: 6 GAIA-Level-1 stand-in tasks × N ∈ {1, 2, 4} agents = 18 runs per backend. Slope from linear regression of mean total tokens per run vs. N. Perturbation: N = 2 agents, 6 tasks × 2 seeds × p ∈ {0, 0.1, 0.2} = 36 runs per backend. Baseline subset (p = 0) yields 12 runs at 100% success; perturbed subset (p ∈ {0.1, 0.2}) yields 24 runs at 100% success on these tasks. Cost: Published per-backend pricing applied to per-call token counts across all 108 runs (total $0.099 + $0.303 = $0.402; cap $1.50). Backend selection: DeepSeek-V4-Flash (the cheapest viable reasoning-capable backend at survey time) and Gemini-2.5-Flash (the next-cheapest mainstream frontier-tier model) chosen to maximize cross-vendor diversity within the budget. The harness is backend-agnostic via CrewAI's LiteLLM layer.

Task-set provenance. The HuggingFace dataset `gaia-benchmark/GAIA` requires gated access (login + agreement) that this run could not satisfy from the compute environment. We therefore documented six self-contained GAIA-Level-1-shaped stand-ins covering arithmetic chains, set logic, unit conversion, calendar reasoning, string extraction with arithmetic, and comparison reasoning. Each has two–three distinct reasoning steps and deterministic ground truth, matching the GAIA-Level-1 shape from the original benchmark paper. The full task set is in `execution/code/study_b_tasks.csv` (five columns: id, expected, description, category, and reasoning_steps).

Honest scope statement. The six values in Table A3 are live measurements on a controlled experiment with deliberately tiny stand-in tasks. The 100% perturbed-task success rate on both backends is a ceiling effect—these tasks are recoverable by single-call reasoning once the perturbed tool is bypassed, and the number should not be read as a general claim that CrewAI is robust to tool failure. The contribution of this appendix is the experimental design, the verified extraction pipeline, the cross-backend sanity check supporting harness backend portability (on these stand-ins; we do not claim broader LLM invariance), and the deposited reproducibility code. The main-text claim in Section 8.3 rests on Study A's real annotated traces; the framework discussion in Section 4 and the evaluation gap in Section 8.2 do not depend on these illustrative numbers.

Raw CSVs: `execution/code/study_b_results_deepseek-v4.csv` (54 rows) and `execution/code/study_b_results_gemini-2.5-flash.csv` (54 rows). Cross-backend summary: `execution/code/study_b_cross_backend_summary.json`. Reports: `execution/code/study_b_report_deepseek-v4.md` and `execution/code/study_b_report_gemini-2.5-flash.md`. Tasks: `execution/code/study_b_tasks.csv`. Script: `execution/code/study_b_crewai.py` (CLI flag `--backend deepseek-v4|gemini-2.5-flash`).

References

1. Brown, T.; Mann, B.; Ryder, N.; Subbiah, M.; Kaplan, J.; Dhariwal, P.; Neelakantan, A.; Shyam, P.; Sastry, G.; Askell, A.; et al. Language models are few-shot learners. In Proceedings of the Advances in Neural Information Processing Systems (NeurIPS), Virtual, 6–12 December 2020; Volume 33, pp. 1877–1901.
2. OpenAI. GPT-4 technical report. *arXiv* **2023**, arXiv:2303.08774.
3. Yao, S.; Zhao, J.; Yu, D.; Du, N.; Shafran, I.; Narasimhan, K.; Cao, Y. ReAct: Synergizing reasoning and acting in language models. In Proceedings of the International Conference on Learning Representations (ICLR), Kigali, Rwanda, 1–5 May 2023.
4. Shinn, N.; Cassano, F.; Berman, E.; Gopinath, A.; Narasimhan, K.; Yao, S. Reflexion: Language agents with verbal reinforcement learning. In Proceedings of the Advances in Neural Information Processing Systems (NeurIPS), New Orleans, LA, USA, 10–16 December 2023.
5. Wei, J.; Wang, X.; Schuurmans, D.; Bosma, M.; Ichter, B.; Xia, F.; Chi, E.; Le, Q.; Zhou, D. Chain-of-thought prompting elicits reasoning in large language models. In Proceedings of the Advances in Neural Information Processing Systems (NeurIPS), New Orleans, LA, USA, 28 November–9 December 2022; Volume 35, pp. 24824–24837.
6. Lewis, P.; Perez, E.; Piktus, A.; Petroni, F.; Karpukhin, V.; Goyal, N.; Küttler, H.; Lewis, M.; Yih, W.; Rocktäschel, T.; et al. Retrieval-augmented generation for knowledge-intensive NLP tasks. In Proceedings of theAdvances in Neural Information Processing Systems (NeurIPS), Virtual, 6–12 December 2020; Volume 33, pp. 9459–9474.
7. Qu, C.; Dai, S.; Wei, X.; Cai, H.; Wang, S.; Yin, D.; Xu, J.; Wen, J. Tool learning with large language models: A survey. *Front. Comput. Sci.* **2025**, *19*, 198343. [CrossRef]
8. Guo, T.; Chen, X.; Wang, Y.; Chang, R.; Pei, S.; Chawla, N.V.; Wiest, O.; Zhang, X. Large language model based multi-agents: A survey of progress and challenges. In Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), Jeju, Republic of Korea, 3–9 August 2024.
9. Talebirad, Y.; Nadiri, A. Multi-agent collaboration: Harnessing the power of intelligent LLM agents. *arXiv* **2023**, arXiv:2306.03314. [CrossRef]
10. Tran, K.-T.; Dao, D.; Nguyen, M.-D.; Pham, Q.-V.; O'Sullivan, B.; Nguyen, H.D. Multi-agent collaboration mechanisms: A survey of LLMs. *arXiv* **2025**, arXiv:2501.06322. [CrossRef]
11. Chen, S.; Liu, Y.; Han, W.; Zhang, W.; Liu, T. A survey on LLM-based multi-agent system: Recent advances and new frontiers in application. *arXiv* **2024**, arXiv:2412.17481.
12. Park, J.S.; O'Brien, J.C.; Cai, C.J.; Morris, M.R.; Liang, P.; Bernstein, M.S. Generative agents: Interactive simulacra of human behavior. In Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technolog, San Francisco, CA, USA, 29 October–1 November 2023.
13. LangChain. State of AI Agents 2025. December 2025. Available online: https://www.langchain.com/state-of-agent-engineering (accessed on 5 June 2026).
14. McKinsey & Company. *The State of AI in 2025: Agents, Innovation, and Transformation*; McKinsey & Company: New York, NY, USA, 2025.
15. Gartner. Gartner Predicts 40% of Enterprise Apps Will Feature Task-Specific AI Agents by 2026, Up from Less Than 5% in 2025. Gartner Newsroom, 26 August 2025. Available online: https://www.gartner.com/en/newsroom/press-releases/2025-08-26-gartner-predicts-40-percent-of-enterprise-apps-will-feature-task-specific-ai-agents-by-2026-up-from-less-than-5-percent-in-2025 (accessed on 7 June 2026).
16. Gartner. Gartner Predicts Over 40% of Agentic AI Projects Will Be Canceled by End of 2027. Gartner Newsroom, 25 June 2025. Available online: https://www.gartner.com/en/newsroom/press-releases/2025-06-25-gartner-predicts-over-40-percent-of-agentic-ai-projects-will-be-canceled-by-end-of-2027 (accessed on 7 June 2026).
17. LangChain Blog. How and When to Build Multi-Agent Systems. 2024. Available online: https://blog.langchain.com/how-and-when-to-build-multi-agent-systems/ (accessed on 5 June 2026).
18. Wooldridge, M. *An Introduction to MultiAgent Systems*, 2nd ed.; John Wiley & Sons: Chichester, UK, 2009.
19. Wang, L.; Ma, C.; Feng, X.; Zhang, Z.; Yang, H.; Zhang, J.; Chen, Z.; Tang, J.; Chen, X.; Lin, Y.; et al. A survey on large language model based autonomous agents. *Front. Comput. Sci.* **2024**, *18*, 186345. [CrossRef]
20. Xi, Z.; Chen, W.; Guo, X.; He, W.; Ding, Y.; Hong, B.; Zhang, M.; Wang, J.; Jin, S.; Zhou, E.; et al. The rise and potential of large language model based agents: A survey. *Sci. China Inf. Sci.* **2025**, *68*, 121101. [CrossRef]
21. Zhao, W.X.; Zhou, K.; Li, J.; Tang, T.; Dong, Z.; Hou, Y.; Zhang, B.; Min, Y.; Zhang, J.; Liu, P.; et al. A survey of large language models. *Front. Comput. Sci.* **2026**, *20*, 2012627. [CrossRef]
22. Zhang, Z.; Dai, Q.; Bo, X.; Ma, C.; Li, R.; Chen, X.; Zhu, J.; Dong, Z.; Wen, J.-R. A survey on the memory mechanism of large language model based agents. *ACM Trans. Inf. Syst.* **2025**, *43*, 155. [CrossRef]
23. Yao, S.; Yu, D.; Zhao, J.; Shafran, I.; Griffiths, T.L.; Cao, Y.; Narasimhan, K. Tree of Thoughts: Deliberate problem solving with large language models. In Proceedings of the Advances in Neural Information Processing Systems (NeurIPS), New Orleans, LA, USA, 10–16 December 2023.

24. Zhou, A.; Yan, K.; Shlapentokh-Rothman, M.; Wang, H.; Wang, Y.-X. Language Agent Tree Search unifies reasoning, acting, and planning in language models. In Proceedings of the 41st International Conference on Machine Learning (ICML), Vienna, Austria, 21–27 July 2024; Volume 235, pp. 62138–62160.
25. Foundation for Intelligent Physical Agents (FIPA). FIPA ACL Message Structure Specification. 2002. Available online: http://www.fipa.org/specs/fipa00061/ (accessed on 5 June 2026).
26. Du, Y.; Li, S.; Torralba, A.; Tenenbaum, J.B.; Mordatch, I. Improving factuality and reasoning in language models through multiagent debate. In Proceedings of the 41st International Conference on Machine Learning (ICML), Vienna, Austria, 21–27 July 2024.
27. Hong, S.; Zhuge, M.; Chen, J.; Zheng, X.; Cheng, Y.; Zhang, C.; Wang, J.; Wang, Z.; Yau, S.K.S.; Lin, Z.; et al. MetaGPT: Meta programming for a multi-agent collaborative framework. In Proceedings of the International Conference on Learning Representations (ICLR), Vienna, Austria, 7–11 May 2024.
28. Li, G.; Hammoud, H.A.A.K.; Itani, H.; Khizbullin, D.; Ghanem, B. CAMEL: Communicative agents for 'mind' exploration of large language model society. In Proceedings of the Advances in Neural Information Processing Systems (NeurIPS), New Orleans, LA, USA, 10–16 December 2023.
29. LangChain. LangGraph: Agent Orchestration Framework for Reliable AI Agents. 2024–2025. Available online: https://www.langchain.com/langgraph (accessed on 5 June 2026).
30. CrewAI. The Leading Multi-Agent Platform. 2024–2025. Available online: https://crewai.com/ (accessed on 5 June 2026).
31. Microsoft. Introducing Microsoft Agent Framework: The Open-Source Engine for Agentic AI Apps. Microsoft Foundry Blog, October 2025. Available online: https://devblogs.microsoft.com/foundry/introducing-microsoft-agent-framework-the-open-source-engine-for-agentic-ai-apps/ (accessed on 5 June 2026).
32. Chen, W.; Su, Y.; Zuo, J.; Yang, C.; Yuan, C.; Chan, C.; Yu, H.; Lu, Y.; Hung, Y.; Qian, C.; et al. AgentVerse: Facilitating multi-agent collaboration and exploring emergent behaviors. In Proceedings of the International Conference on Learning Representations (ICLR), Vienna, Austria, 7–11 May 2024.
33. Zhuge, M.; Wang, W.; Kirsch, L.; Faccio, F.; Khizbullin, D.; Schmidhuber, J. GPTSwarm: Language agents as optimizable graphs. In Proceedings of the 41st International Conference on Machine Learning (ICML), Vienna, Austria, 21–27 July 2024.
34. Liu, Z.; Zhang, Y.; Li, P.; Liu, Y.; Yang, D. A dynamic LLM-powered agent network for task-oriented agent collaboration. In Proceedings of the Conference on Language Modeling, Philadelphia, PA, USA, 7–9 October 2024.
35. Wu, Q.; Bansal, G.; Zhang, J.; Wu, Y.; Li, B.; Zhu, E.; Jiang, L.; Zhang, X.; Zhang, S.; Liu, J.; et al. AutoGen: Enabling next-gen LLM applications via multi-agent conversation. In Proceedings of the Conference on Language Modeling, Philadelphia, PA, USA, 7–9 October 2024.
36. Liang, S.; Garg, S.; Zilouchian Moghaddam, R. The SWE-Bench Illusion: When state-of-the-art LLMs remember instead of reason. *arXiv* **2025**, arXiv:2506.12286.
37. Staufer, L.; Feng, K.; Wei, K.; Bailey, L.; Duan, Y.; Yang, M.; Ozisik, A.P.; Casper, S.; Kolt, N. The 2025 AI Agent Index. *arXiv* **2026**, arXiv:2602.17753 [CrossRef]
38. European Parliament and Council. *Regulation (EU) 2024/1689 Laying Down Harmonised Rules on Artificial Intelligence (AI Act)*; Publications Office of the European Union: Luxembourg, 2024.
39. Google. Agent2Agent Protocol (A2A). 2025. Available online: https://github.com/a2aproject/A2A (accessed on 5 June 2026).
40. Anthropic. Introducing the Model Context Protocol. Anthropic News, November 2024. Available online: https://www.anthropic.com/news/model-context-protocol (accessed on 5 June 2026).
41. Stone, P.; Veloso, M. Multiagent systems: A survey from a machine learning perspective. *Auton. Robot.* **2000**, *8*, 345–383. [CrossRef]
42. Tambe, M. Towards flexible teamwork. *J. Artif. Intell. Res. (JAIR)* **1997**, *7*, 83–124. [CrossRef]
43. Fourney, A.; Bansal, G.; Mozannar, H.; Tan, C.; Salinas, E.; Zhu, E.; Niedtner, F.; Proebsting, G.; Bassman, G.; Gerrits, J.; et al. Magentic-One: A generalist multi-agent system for solving complex tasks. *arXiv* **2024**, arXiv:2411.04468. [CrossRef]
44. Qian, C.; Liu, W.; Liu, H.; Chen, N.; Dang, Y.; Li, J.; Yang, C.; Chen, W.; Su, Y.; Cong, X.; et al. Communicative agents for software development. In Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL), Bangkok, Thailand, 11–16 August 2024.
45. Zhang, G.; Niu, L.; Fang, J.; Wang, K.; Bai, L.; Wang, X. Multi-agent architecture search via agentic supernet (MaAS). In Proceedings of the International Conference on Machine Learning (ICML), Vancouver, BC, Canada, 13–19 July 2025.
46. LangChain. LangGraph Multi-Agent Workflows. LangChain Blog. 2024. Available online: https://blog.langchain.com/langgraph-multi-agent-workflows/ (accessed on 5 June 2026).
47. Abdelnabi, S.; Gomaa, A.; Sivaprasad, S.; Schönherr, L.; Fritz, M. LLM-deliberation: Evaluating LLMs with interactive multi-agent negotiation games. In Proceedings of the Advances in Neural Information Processing Systems 37 (NeurIPS 2024), Datasets and Benchmarks Track, Vancouver, BC, Canada, 10–15 December 2024.

48. Hu, Y.; Cai, Y.; Du, Y.; Zhu, X.; Liu, X.; Yu, Z.; Hou, Y.; Tang, S.; Chen, S. Self-evolving multi-agent collaboration networks for software development (EvoMAC). In Proceedings of the International Conference on Learning Representations (ICLR), Singapore, 24–28 April 2025.
49. OpenAI. OpenAI Agents SDK. 2025. Available online: https://openai.github.io/openai-agents-python/ (accessed on 5 June 2026).
50. Gao, D.; Li, Z.; Pan, X.; Kuang, W.; Ma, Z.; Qian, B.; Wei, F.; Zhang, W.; Xie, Y.; Chen, D.; et al. AgentScope: A flexible yet robust multi-agent platform. *arXiv* **2024**, arXiv:2402.14034.
51. Khattab, O.; Singhvi, A.; Maheshwari, P.; Zhang, Z.; Santhanam, K.; Vardhamanan, S.; Haq, S.; Sharma, A.; Joshi, T.; Moazam, H.; et al. DSPy: Compiling declarative language model calls into self-improving pipelines. In Proceedings of the International Conference on Learning Representations (ICLR), Vienna, Austria, 7–11 May 2024.
52. Wang, X.; Li, B.; Song, Y.; Xu, F.; Tang, X.; Zhuge, M.; Pan, J.; Song, Y.; Li, B.; Singh, J.; et al. OpenHands: An open platform for AI software developers as generalist agents. In Proceedings of the International Conference on Learning Representations (ICLR), Singapore, 24–28 April 2025.
53. SWE-Bench. SWE-Bench Leaderboard. Available online: https://www.swebench.com (accessed on 5 June 2026).
54. Anthropic. Donating the Model Context Protocol and Establishing the Agentic AI Foundation. Anthropic News, December 2025. Available online: https://www.anthropic.com/news/donating-the-model-context-protocol-and-establishing-of-the-agentic-ai-foundation (accessed on 5 June 2026).
55. Linux Foundation. Linux Foundation Launches the Agent2Agent Protocol Project to Enable Secure, Intelligent Communication Between AI Agents. Linux Foundation Press Release. 23 June 2025. Available online: https://www.linuxfoundation.org/press/linux-foundation-launches-the-agent2agent-protocol-project-to-enable-secure-intelligent-communication-between-ai-agents (accessed on 5 June 2026).
56. IBM Research. Agent Communication Protocol (ACP). 2025. Available online: https://research.ibm.com/projects/agent-communication-protocol (accessed on 5 June 2026).
57. LF AI & Data Foundation. ACP Joins Forces with A2A Under the Linux Foundation's LF AI & Data. August 2025. Available online: https://lfaidata.foundation/communityblog/2025/08/29/acp-joins-forces-with-a2a-under-the-linux-foundations-lf-ai-data/ (accessed on 5 June 2026).
58. Chang, G.; Lin, E.; Yuan, C.; Cai, R.; Chen, B.; Xie, X.; Zhang, Y. Agent Network Protocol technical white paper. *arXiv* **2025**, arXiv:2508.00007. Available online: https://agent-network-protocol.com/ (accessed on 5 June 2026).
59. Ehtesham, A.; Singh, A.; Gupta, G.K.; Kumar, S. A survey of agent interoperability protocols: MCP, ACP, A2A, and ANP. *arXiv* **2025**, arXiv:2505.02279.
60. Xiao, Y.-A.; Gao, P.; Peng, C.; Xiong, Y. Reducing the cost of LLM agents through trajectory reduction (AgentDiet). *arXiv* **2025**, arXiv:2509.23586.
61. Yuan, P.; Ma, A.; Yao, Y.; Yao, H.; Tomizuka, M.; Ding, M. REMAC: Self-reflective and self-evolving multi-agent collaboration for long-horizon robot manipulation. *arXiv* **2025**, arXiv:2503.22122.
62. Wang, Z.; Wang, Y.; Liu, X.; Ding, L.; Zhang, M.; Liu, J.; Zhang, M. AgentDropout: Dynamic agent elimination for token-efficient and high-performance LLM-based multi-agent collaboration. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (ACL), Volume 1: Long Papers*; Association for Computational Linguistics: Vienna, Austria, 2025; pp. 24013–24035. [CrossRef]
63. Chen, W.; Yuan, J.; Qian, C.; Yang, C.; Liu, Z.; Sun, M. Optima: Optimizing effectiveness and efficiency for LLM-based multi-agent system. In Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL), Vienna, Austria, 27 July–1 August 2025; pp. 11534–11557. [CrossRef]
64. Schulhoff, S.; Pinto, J.; Khan, A.; Bouchard, L.-F.; Si, C.; Anati, S.; Tagliabue, V.; Kost, A.L.; Carnahan, C.; Boyd-Graber, J. Ignore this title and HackAPrompt: Exposing systemic weaknesses of LLMs through a global-scale prompt hacking competition. In Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP), Singapore, 6–10 December 2023.
65. Lee, D.; Tiwari, M. Prompt Infection: LLM-to-LLM prompt injection within multi-agent systems. *arXiv* **2024**, arXiv:2410.07283.
66. Cloud Security Alliance. MAESTRO: Agentic AI Threat Modeling Framework. CSA Blog, February 2025. Available online: https://cloudsecurityalliance.org/blog/2025/02/06/agentic-ai-threat-modeling-framework-maestro (accessed on 5 June 2026).
67. Peng, S.; Kalliamvakou, E.; Cihon, P.; Demirer, M. The impact of AI on developer productivity: Evidence from GitHub Copilot. *arXiv* **2023**, arXiv:2302.06590. [CrossRef]
68. Becker, J.; Rush, N.; Barnes, E.; Rein, D. Measuring the impact of early-2025 AI on experienced open-source developer productivity. *arXiv* **2025**, arXiv:2507.09089. [CrossRef]
69. UK Department for Business and Trade. The Evaluation of the M365 Copilot Pilot in the Department for Business and Trade. September 2025. Evaluation Period October 2024–March 2025. Available online: https://assets.publishing.service.gov.uk/media/68adbe409e1cebdd2c96a19d/dbt-microsoft-365-copilot-evaluation.pdf (accessed on 5 June 2026).
70. Anthropic. Introducing Claude Sonnet 4.5. Anthropic News. 2025. Available online: https://www.anthropic.com/news/claude-sonnet-4-5 (accessed on 5 June 2026).

71. Jimenez, C.E.; Yang, J.; Wettig, A.; Yao, S.; Pei, K.; Press, O.; Narasimhan, K. SWE-bench: Can language models resolve real-world GitHub issues? In Proceedings of the International Conference on Learning Representations (ICLR), Vienna, Austria, 7–11 May 2024.
72. Yang, J.; Jimenez, C.E.; Wettig, A.; Lieret, K.; Yao, S.; Narasimhan, K.; Press, O. SWE-agent: Agent-computer interfaces enable automated software engineering. In Proceedings of the Advances in Neural Information Processing Systems (NeurIPS), Vancouver, BC, Canada, 10–15 December 2024.
73. OpenAI. Introducing SWE-Bench Verified. OpenAI Blog. August 2024. Available online: https://openai.com/index/introducing-swe-bench-verified/ (accessed on 5 June 2026).
74. Mialon, G.; Fourrier, C.; Swift, C.; Wolf, T.; LeCun, Y.; Scialom, T. GAIA: A benchmark for general AI assistants. In Proceedings of the International Conference on Learning Representations (ICLR), Vienna, Austria, 7–11 May 2024.
75. Zhou, S.; Xu, F.; Zhu, H.; Zhou, X.; Lo, R.; Sridhar, A.; Cheng, X.; Ou, T.; Bisk, Y.; Fried, D.; et al. WebArena: A realistic web environment for building autonomous agents. In Proceedings of the International Conference on Learning Representations (ICLR), Vienna, Austria, 7–11 May 2024.
76. Xie, T.; Zhang, D.; Chen, J.; Li, X.; Zhao, S.; Cao, R.; Hua, T.; Cheng, Z.; Shin, D.; Lei, F.; et al. OSWorld: Benchmarking multimodal agents for open-ended tasks in real computer environments. In Proceedings of the Advances in Neural Information Processing Systems (NeurIPS), Vancouver, BC, Canada, 10–15 December 2024.
77. Liu, X.; Yu, H.; Zhang, H.; Xu, Y.; Lei, X.; Lai, H.; Gu, Y.; Ding, H.; Men, K.; Yang, K.; et al. AgentBench: Evaluating LLMs as agents. In Proceedings of the International Conference on Learning Representations (ICLR), Vienna, Austria, 7–11 May 2024.
78. Yao, S.; Shinn, N.; Razavi, P.; Narasimhan, K. tau-bench: A benchmark for tool-agent-user interaction in real-world domains. In Proceedings of the International Conference on Learning Representations (ICLR), Singapore, 24–28 April 2025.
79. Wang, G.; Xie, Y.; Jiang, Y.; Mandlekar, A.; Xiao, C.; Zhu, Y.; Fan, L.; Anandkumar, A. VOYAGER: An open-ended embodied agent with large language models. *Trans. Mach. Learn. Res.* **2024**.
80. Wang, W.; Han, D.; Madrigal Diaz, D.; Xu, J.; Rühle, V.; Rajmohan, S. OdysseyBench: Evaluating LLM agents on long-horizon complex office application workflows. *arXiv* **2025**, arXiv:2508.09124.
81. Cemri, M.; Pan, M.; Yang, S.; Agrawal, L.; Chopra, B.; Tiwari, R.; Keutzer, K.; Parameswaran, A.; Klein, D.; Ramchandran, K.; et al. Why do multi-agent LLM systems fail? In Proceedings of the Advances in Neural Information Processing Systems (NeurIPS), San Diego, CA, USA, 2–7 December 2025.
82. Li, Q.; Cui, L.; Zhao, X.; Kong, L.; Bi, W. GSM-Plus: A comprehensive benchmark for evaluating the robustness of LLMs as mathematical problem solvers. In Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL), Bangkok, Thailand, 11–16 August 2024; pp. 2961–2984.
83. Anthropic. Introducing Claude Code. Anthropic Blog. February 2025. Available online: https://www.anthropic.com/claude-code (accessed on 5 June 2026).
84. Codeium. Windsurf: The First Agentic IDE. 2024. Available online: https://codeium.com/windsurf (accessed on 5 June 2026).
85. Amazon Web Services. Amazon Q Developer Product Page and Developer Productivity Claims. 2024–2025. (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). Available online: https://aws.amazon.com/q/developer/ (accessed on 5 June 2026).
86. Lu, C.; Lu, C.; Lange, R.T.; Foerster, J.; Clune, J.; Ha, D. The AI Scientist: Towards fully automated open-ended scientific discovery. *arXiv* **2024**, arXiv:2408.06292. [CrossRef]
87. Bran, A.M.; Cox, S.; Schilter, O.; Baldassari, C.; White, A.D.; Schwaller, P. with chemistry tools. *Nat. Mach. Intell.* **2024**, *6*, 525–535. [CrossRef] [PubMed]
88. Klarna. *Klarna AI Assistant Handles Two-Thirds of Customer Service Chats in Its First Month*; Klarna Bank AB: Stockholm, Sweden, 2024.
89. Salesforce. Heathrow Airport Taps Salesforce Agentforce to Enhance Passenger Experience. Salesforce UK Newsroom, 11 June 2025. Available online: https://www.salesforce.com/uk/news/press-releases/2025/06/11/heathrow-airport-agentforce-passenger-experience/ (accessed on 5 June 2026).
90. ServiceNow. Now Assist and AI Agents for IT Service Management. ServiceNow Product Materials and Press Room, 2025. (Incident-Deflection Figures Cited Here Are Self-Reported by ServiceNow in Customer Case Studies Linked from This Press Room.) Available online: https://www.servicenow.com/company/media/press-room.html (accessed on 5 June 2026).
91. Deloitte. *The State of AI in the Enterprise*; 2026 AI report; Deloitte US: New York, NY, USA, 2025.
92. Martin, H. Marc Benioff Addresses 'Low Agentforce Adoption' at Dreamforce '25. Salesforce Ben, 15 October 2025. Available online: https://www.salesforceben.com/marc-benioff-addresses-low-agentforce-adoption-at-dreamforce-25/ (accessed on 5 June 2026).
93. Mukherjee, S.; Gamble, P.; Ausin, M.; Kant, N.; Aggarwal, K.; Manjunath, N.; Datta, D.; Liu, Z.; Ding, J.; Busacca, S.; et al. Polaris: A safety-focused LLM constellation architecture for healthcare. *arXiv* **2024**, arXiv:2403.13313. [CrossRef]
94. Tu, T.; Schaekermann, M.; Palepu, A.; Saab, K.; Freyberg, J.; Tanno, R.; Wang, A.; Li, B.; Amin, M.; Cheng, Y.; et al. Towards conversational diagnostic artificial intelligence. *Nature* **2025**, *642*, 442–450. [CrossRef] [PubMed]

95. Zheng, L.; Chiang, W.-L.; Sheng, Y.; Zhuang, S.; Wu, Z.; Zhuang, Y.; Lin, Z.; Li, Z.; Li, D.; Xing, E.P.; et al. Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena. In Proceedings of the Advances in Neural Information Processing Systems (NeurIPS), New Orleans, LA, USA, 10–16 December 2023.
96. Anthropic. Introducing Computer Use: Claude 3.5 Sonnet. Anthropic. October 2024. Available online: https://www.anthropic.com/news/3-5-models-and-computer-use (accessed on 5 June 2026).
97. Gonnermann-Müller, J.; Haase, J.; Fackeldey, K.; Pokutta, S. FACET: Teacher-centred LLM-based multi-agent systems—Towards personalized educational worksheets. *arXiv* **2025**. arXiv:2508.11401.
98. Wang, T.; Zhan, Y.; Lian, J.; Hu, Z.; Yuan, N.J.; Zhang, Q.; Xie, X.; Xiong, H. LLM-powered multi-agent framework for goal-oriented learning in intelligent tutoring system. In Proceedings of the WWW '25: Companion Proceedings of the ACM on Web Conference 2025, Sydney, Australia, 28 April–2 May 2025. [CrossRef]
99. Masterman, T.; Besen, S.; Sawtell, M.; Chao, A. The landscape of emerging AI agent architectures for reasoning, planning, and tool calling: A survey. *arXiv* **2024**, arXiv:2404.11584. [CrossRef]
100. Ngo, R.; Chan, L.; Mindermann, S. The alignment problem from a deep learning perspective. In Proceedings of the International Conference on Learning Representations (ICLR), Vienna, Austria, 7–11 May 2024.
101. Chen, G.; Dong, S.; Shu, Y.; Zhang, G.; Sesay, J.; Karlsson, B.F.; Fu, J.; Shi, Y. AutoAgents: A framework for automatic agent generation. In Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), Jeju, Republic of Korea, 3–9 August 2024.
102. Li, S.; Liu, Y.; Zheng, Y.; Li, M.; Nguyen, Q.V.H.; Pan, S. OFA-MAS: One-for-all multi-agent system topology design based on mixture-of-experts graph generative models. *arXiv* **2026**, arXiv:2601.12996.
103. Zhang, T.; Zhou, Z.; Wan, J.; Hu, T.; Wang, C.; He, X.; Hong, R. Learning transferable topology priors for multi-agent LLM collaboration across domains. *arXiv* **2026**, arXiv:2605.17359. [CrossRef]
104. Wang, X.; Wang, J.; Zhang, F.; Hu, Y.; Zhang, D.; Ye, Y.; Ban, Y.; Han, J.; Wang, R. MasFACT: Continual multi-agent topology learning via geometry-aware posterior transfer. *arXiv* **2026**, arXiv:2605.17361.
105. Yu, M.; Wang, S.; Zhang, G.; Mao, J.; Yin, C.; Liu, Q.; Wen, Q.; Wang, K.; Wang, Y. NetSafe: Exploring the topological safety of multi-agent system. In *Findings of the Association for Computational Linguistics: ACL 2025*; Association for Computational Linguistics: Vienna, Austria, 2025; pp. 2905–2938. [CrossRef]
106. Li, F.; Wu, J.; Fu, T.; Jaques, N.; Zhou, W.; Kan, M. FlowSteer: Prompt-only workflow steering exposes planning-time vulnerabilities in multi-agent LLM systems. *arXiv* **2026**, arXiv:2605.11514.

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.