AINARY RESEARCH REPORT NO. AR-007

The Orchestration Problem

Why Multi-Agent Systems Fail and How to Fix Them

Florian Ziesche — Ainary Ventures

February 2026

Overall Confidence: 72%

CONTENTS

1. Executive Summary

- **51% of companies have AI agents in production[1], yet over 40% of agentic AI projects will be canceled by 2027[11].** The gap is orchestration.
- Performance quality — not cost or safety — is the top barrier to scaling multi-agent systems[1].
- Inter-agent communication can be hijacked with 58–90% success rates, even when individual agents are secure[2].
- Multi-agent cost scales super-linearly: a 5-agent pipeline costs 10–30× a single agent, not 5×[7].
- The fix is counterintuitive: build the orchestration layer first, then add agents. Most teams do the opposite.

Audience: Engineering Leads / CTOs / AI Architects

Keywords: multi-agent orchestration, LangGraph, CrewAI, AutoGen, orchestration patterns, agent coordination, multi-agent failure modes

(Confidence: High)

---

2. Methodology

This report synthesizes 13 primary and 5 secondary sources across academic papers, framework documentation, industry surveys, and practitioner guides. The core evidence base includes one large-scale survey (LangChain, n=1,300)[1], three peer-reviewed papers on multi-agent systems[2][3][4], and one influential practitioner guide from Anthropic[5]. Cost modeling in Section 7 uses published API pricing from OpenAI and Anthropic[16] with calculated communication overhead.

Limitations: Production failure case studies remain largely anecdotal. No systematic study of orchestration-specific failures exists. Framework comparisons reflect February 2026 state and will date quickly. All cost estimates assume standard API pricing without volume discounts.

---

3. The $52 Billion Coordination Problem (Confidence: High)

The AI agent market is projected to grow from $7.8 billion to $52 billion by 2030[13] — but the industry is building agents without solving how they work together.

Evidence: The LangChain State of AI Agents survey (n=1,300) found that 51% of companies already have agents in production[1]. Mid-size companies are the most aggressive adopters, with 63% reporting production deployments[1]. Performance quality — not cost, not safety — ranks as the number one barrier to scaling, cited more than twice as often as other factors[1]. Meanwhile, Gartner predicts that over 40% of agentic AI projects will be canceled by 2027[11].

Interpretation: These numbers are not contradictory. "In production" does not mean "successful in production." The pattern I see is clear: teams deploy agents, discover that coordinating them is harder than building them, and then cancel. The 51% adoption and 40% cancellation rates describe two points on the same curve — the orchestration gap.

The market is growing at a 45.8% CAGR[13], and most of that investment is going into agent capabilities. Very little is going into the coordination layer that determines whether those capabilities translate into reliable output.

> **So What?** *If you are building a multi-agent system and have not explicitly designed the orchestration layer, you are statistically likely to join the 40% cancellation cohort. Orchestration is not a feature to add later — it is the architecture decision that determines success or failure.*

> **What would invalidate this?** *If single-agent systems prove sufficient for 90%+ of production use cases, the orchestration problem becomes irrelevant for most teams. Early evidence from Anthropic[5] suggests that simpler architectures often outperform multi-agent ones, which partially supports this counter-narrative.*

4. Anatomy of Orchestration: Patterns That Work (Confidence: Medium)

There are exactly five production-proven orchestration patterns — and Anthropic identified them by observing what their most successful customers actually build[5].

Evidence: Anthropic's "Building Effective Agents" guide (December 2024) documented five workflow patterns based on real customer deployments: prompt chaining, routing, parallelization, orchestrator-workers, and evaluator-optimizer[5]. The guide's most striking claim is qualitative: "The most successful implementations weren't using complex frameworks"[5].

Exhibit 1: Orchestration Patterns

PATTERN	DESCRIPTION	BEST FOR	KEY RISK
Prompt Chaining	Agent A output → Agent B input, sequential	Linear workflows with clear handoffs	Latency compounds; error propagation
Routing	Classifier directs input to specialist agents	Well-defined categories (support triage)	Misrouting cascades through pipeline
Parallelization	Multiple agents work simultaneously, results merged	Independent subtasks; voting/consensus	Result merging complexity; cost multiplication
Orchestrator-Workers	Central LLM decomposes tasks, workers execute	Unpredictable subtask count (coding, research)	Orchestrator quality determines everything
Evaluator-Optimizer	Generator + evaluator in feedback loop	Clear quality criteria; iterative refinement	Infinite loops without stopping conditions

Source: Anthropic "Building Effective Agents" (2024) [5], LangGraph documentation [8]

Beyond these five, academic research documents additional patterns — conversation-based (AutoGen[3]), market-based, and Mixture-of-Agents (MoA)[4]. The MoA architecture achieved 65.1% on AlpacaEval 2.0 versus GPT-4o's 57.5%[4], demonstrating that layered multi-agent refinement can outperform single models on benchmarks. But the cost is extreme: each layer multiplies token usage by agent count, making MoA impractical for most production scenarios.

Interpretation: The pattern that emerges is a clear tradeoff between orchestration complexity and production reliability. The five Anthropic patterns work because they are composable and debuggable. The more sophisticated academic patterns (MoA, democratic debate) achieve higher benchmark scores but break down under production constraints — cost, latency, and error handling.

A positive counterexample is critical: **coding agents** (Anthropic's Claude Code, Cursor, Devin) use the orchestrator-worker pattern successfully in production. One agent plans changes, workers execute per file. This works because the output is verifiable — code either compiles or it does not. When orchestration serves a domain with clear success criteria, multi-agent systems deliver.

> **So What?** *Start with the simplest pattern that could work. Prompt chaining and routing cover the majority of production use cases. Escalate to orchestrator-workers only when task decomposition is genuinely dynamic. If you reach for MoA or debate architectures, you have likely over-engineered.*

> **What would invalidate this?** *If a framework emerges that makes complex orchestration patterns as reliable as simple ones — essentially, the "Kubernetes of agents" — these recommendations become overly conservative.*

---

5. Framework Wars: LangGraph vs. CrewAI vs. AutoGen vs. Swarm

(Confidence: Medium)

No framework solves orchestration comprehensively — each optimizes for a different point on the flexibility-simplicity spectrum.

Evidence: The four dominant multi-agent frameworks as of February 2026 are LangGraph (part of the LangChain ecosystem, ~100K+ GitHub stars), CrewAI (~30K stars), AutoGen (~30K stars), and OpenAI Swarm (~18K stars)[9][10][11][14].

Exhibit 2: Multi-Agent Framework Comparison (February 2026)

DIMENSION	LANGGRAPH	CREWAI	AUTOGEN	OPENAI SWARM
Architecture	Directed graph (nodes + edges)	Role-based crews with tasks	Conversation-based multi-agent	Routines + handoffs
Orchestration Model	Explicit (developer defines graph)	Implicit (framework manages)	Emergent (conversation-driven)	Explicit (function-based)
Production Readiness	High (LangSmith ecosystem)	Medium (Enterprise tier)	Low-Medium (research-oriented)	None (educational only)
Flexibility	Very High (any topology)	Medium (sequential/hierarchical)	High (customizable)	Low (simple handoffs)
Learning Curve	Steep	Low	Medium	Very Low
HITL Support	Built-in (interrupt/resume)	Optional (task-level)	Built-in (human proxy)	Manual
Observability	LangSmith integration	Limited	Limited	None

DIMENSION	LANGGRAPH	CREWAI	AUTOGEN	OPENAI SWARM
Key Weakness	Complexity; over-engineering risk	Black-box orchestration	Not production-hardened	Not a real framework

Sources: Framework documentation [9][10], GitHub repositories, Anthropic [5], LangChain blog [8]

Interpretation: The framework landscape mirrors the early days of web frameworks — fragmented, opinionated, and rapidly shifting. LangGraph offers the most control but demands the most engineering investment. CrewAI gets teams to a working prototype fastest but becomes a black box when debugging. AutoGen is the academic favorite but struggles in production. Swarm is explicitly not a framework — OpenAI released it as an educational resource to teach orchestration concepts[9].

The key claim in this landscape comes from Anthropic, which recommends avoiding frameworks entirely: start with raw LLM API calls and simple composable patterns[5]. This is not marketing — Anthropic is arguing against the ecosystem that surrounds its own model. Their claim is that their most successful customers build orchestration logic directly, using frameworks only for specific components (memory, tool use) rather than as architectural backbones.

> **So What?** *Framework choice is less important than orchestration design. If you pick the right pattern (Section 4), any framework — or no framework — can implement it. If you pick the wrong pattern, no framework will save you. For teams with strong engineering, Anthropic's "no framework" approach is worth serious consideration.*

> **What would invalidate this?** *If one framework achieves dominant market share and community consensus (the "React of agents"), the fragmentation problem resolves itself and framework choice becomes the clear starting point.*

6. When Multi-Agent Systems Break (Confidence: High)

Multi-agent systems do not fail at the agent level — they fail at the communication layer between agents, and that layer can be weaponized.

Evidence: Triedman et al. (2025) demonstrated that adversarial content can hijack inter-agent communication to execute arbitrary malicious code in 58–90% of trials with GPT-4o[2]. In some orchestrator configurations, attack success reached 100%[2]. The critical finding: attacks succeed even when individual agents refuse harmful actions. The vulnerability is structural — it exists in the orchestration layer, not in any single agent.

Exhibit 3: Multi-Agent Failure Taxonomy

FAILURE MODE	DESCRIPTION	FREQUENCY	MITIGATION
Infinite Loops	Agents delegate to each other without progress	Common	Max iteration limits; progress detection
Deadlocks	Two agents wait for each other's output	Occasional	Timeout mechanisms; dependency graph validation
Conflicting Outputs	Agents produce contradictory results	Common (parallel)	Conflict resolution agent; voting mechanisms
Blame Attribution Failure	Impossible to determine which agent failed	Systemic	Per-agent logging; trace IDs; output provenance
Context Window Overflow	Accumulated conversation exceeds token limits	Common (conversation)	Summarization agents; context pruning
Orchestrator Hallucination	Supervisor delegates to non-existent agents	Occasional	Strict agent/tool registry; validation layer
Cascade Failures	One agent's error propagates through pipeline	Common (sequential)	Circuit breakers; per-step validation

FAILURE MODE	DESCRIPTION	FREQUENCY	MITIGATION
Cost Explosion	Retry loops burn tokens	Systemic	Budget caps; token monitoring; early termination
Security Hijacking	Adversarial input manipulates inter-agent comms	Demonstrated (58–90%)	Sandboxing; input validation between agents

Sources: arXiv:2503.12188 [2], arXiv:2402.01680 [3], Anthropic engineering blog [5], practitioner reports

Production failure cases reinforce this taxonomy. McDonald's terminated its AI drive-through program in 2024 after compounding errors across speech recognition, order processing, and verification agents made the system unreliable[15]. The failure was not in any individual agent — each worked acceptably in isolation. The failure was in the coordination between them, where errors in one agent propagated uncorrected to the next.

Fully autonomous agent swarms — multi-agent systems operating without human-in-the-loop checkpoints — consistently produce unreliable results. Democratic or debate architectures, where agents converse freely to reach consensus, rarely converge in production. Token costs explode, latency becomes unacceptable, and "consensus" typically means the last agent's output wins by default.

Interpretation: The failure pattern is consistent across academic research and production case studies: the more agents communicate, the more surface area exists for errors and attacks. This is not a solvable problem at the model level. Better models will reduce individual agent failures but will not eliminate coordination failures, because coordination failures are emergent properties of multi-agent interaction.

> **So What?** *Every inter-agent communication channel is an attack surface and an error propagation path. Design for minimal necessary communication. If two agents do not need to exchange information, they should not. Treat inter-agent messages with the same suspicion you would treat user input.*

> **What would invalidate this?** *If future models develop reliable meta-reasoning about multi-agent coordination — essentially, if an orchestrator LLM can detect and correct communication failures in real time — the failure rates would drop substantially. Early work on "self-correcting" agents suggests this is possible but not yet production-ready.*

7. The Hidden Tax: Cost, Latency, and Observability (Confidence: Medium)

Multi-agent cost scales super-linearly — a 5-agent pipeline costs 10–30× a single agent, not 5×, and current observability tools cannot explain why[7].

Evidence: Token cost modeling based on published API pricing[16] reveals the compounding effect of inter-agent communication:

Exhibit 4: Token Cost Modeling for Multi-Agent Pipelines

CONFIGURATION	LLM CALLS/TASK	EST. COST/TASK (GPT-4O)	EST. COST/TASK (CLAUDE SONNET)	DAILY COST (10K TASKS)
Single Agent	1–3	$0.01–0.03	$0.01–0.02	$100–300
3-Agent Pipeline	5–9	$0.05–0.15	$0.03–0.10	$500–1,500
5-Agent Pipeline	15–25	$0.15–0.50	$0.10–0.35	$1,500–5,000
5-Agent + Retry Logic	25–50	$0.25–1.00	$0.18–0.70	$2,500–10,000
MoA (3 layers × 3 agents)	27+	$0.50–2.00	$0.35–1.40	$5,000–20,000

Assumptions: ~2K tokens input, ~1K tokens output per call. Prices: GPT-4o ($2.50/$10 per M in/out), Claude Sonnet ($3/$15 per M in/out). Actual costs vary by task complexity. Source: OpenAI/Anthropic API pricing [16]

The cost multiplier comes from three sources: (1) each agent reads the output of previous agents, duplicating token consumption; (2) retry logic on failures multiplies calls; (3) orchestration itself consumes tokens — the supervisor agent that routes, validates, and coordinates is an LLM call that adds zero direct value to the output.

The observability gap compounds the cost problem. Current tools — LangSmith, Langfuse, Arize Phoenix — are designed for single-agent tracing[17]. Multi-agent systems need cross-agent trace correlation, inter-agent message inspection, decision attribution (why did the orchestrator route to Agent B instead of Agent A?), per-agent cost allocation, and failure root-cause analysis. The LangChain survey reports that 94% of production agent developers use some observability tool[1], but multi-agent-specific observability is nascent. LangSmith supports LangGraph traces natively; everything else requires custom instrumentation.

Interpretation: The cost problem is not just financial — it is architectural. Super-linear cost scaling means that multi-agent systems have a natural ceiling: beyond a certain agent count, the coordination overhead exceeds the value of specialization. I estimate this ceiling at 5–7 agents for most production use cases, based on the cost and latency data. Beyond that, you are paying more for coordination than for work.

> **So What?** *Before adding an agent to your pipeline, calculate the marginal cost including communication overhead. If the agent does not improve output quality enough to justify a 3–5× cost increase (not 1×), it should not be added. Most teams do not model this before deployment.*

> **What would invalidate this?** *A dramatic reduction in LLM inference costs (10× cheaper) would move the ceiling higher. At current cost trajectories, this is plausible within 18 months, but it solves only one dimension — latency and error compounding remain.*

---

8. Orchestration-First: A Design Methodology (Confidence: Medium)

Build the orchestration layer first, then add agents — not the other way around.

This is the core thesis of this report, and it runs counter to how most teams approach multi-agent systems. The standard approach is: build individual agents, verify they work, then figure out how to connect them. By the time teams reach the coordination step, architectural decisions have already been made — agent interfaces, output formats, error handling patterns — that constrain orchestration options.

The orchestration-first methodology inverts this:

Step 1: Define the orchestration pattern. Using the five patterns from Section 4, identify which pattern matches the task structure. Most production use cases map to prompt chaining or routing. If neither works, consider orchestrator-workers.

Step 2: Define inter-agent contracts. Before building any agent, specify what each agent receives as input, what it produces as output, and what constitutes a failure. These contracts are the orchestration layer.

Step 3: Build with stubs. Implement the orchestration flow with stub agents — simple functions that return mock output conforming to the contracts. Verify the flow works end-to-end.

Step 4: Replace stubs with agents. One at a time, replace stubs with real agents. Test after each replacement. If the flow breaks, the problem is in the contract violation, not the orchestration.

Step 5: Add complexity only when simple patterns fail. The burden of proof should be on the complex pattern. If a single agent with good prompting achieves 80% of a multi-agent pipeline's quality, keep the single agent.

Exhibit 5: HITL Spectrum for Multi-Agent Systems

LEVEL	DESCRIPTION	USE CASE	RISK
0: Fully Autonomous	No human oversight	Low-stakes, well-tested pipelines only	Compounding errors; hijacking
1: Checkpoint Approval	Human approves at key points	Financial transactions; publishing	Alert fatigue if too many checkpoints

LEVEL	DESCRIPTION	USE CASE	RISK
2: Exception Handling	Human intervenes on failures only	Most production deployments	Requires good confidence calibration
3: Supervised Autonomy	Human monitors dashboard	Enterprise deployments	Monitoring fatigue; 67% of alerts ignored
4: Human-Directed	Human decides; agents execute	High-stakes decisions	Defeats automation purpose

Sources: LangChain survey [1], Vectra alert fatigue data [12]

The LangChain survey reveals that most production agents have read-only permissions[1]. Very few companies allow agents to read, write, and delete freely. The more agents in a system, the more conservative the permission model — which creates a paradox: multi-agent systems promise greater capability, but permission constraints limit what they can actually do.

Interpretation: HITL Level 2 (exception handling) is the practical sweet spot for most multi-agent deployments. It preserves the automation benefit while catching the orchestration failures described in Section 6. But it requires something most teams lack: well-calibrated confidence scores that accurately predict when an agent will fail.

> **So What?** *The decision to use multi-agent at all should be the last decision, not the first. Start with a single well-prompted agent. Add a second agent only when you can demonstrate that it improves output quality enough to justify the orchestration overhead. Most teams that follow this discipline discover they need fewer agents than they thought.*

> **What would invalidate this?** *If multi-agent frameworks mature to the point where orchestration is genuinely plug-and-play — where connecting agents is as simple as connecting API endpoints — the overhead of multi-agent systems drops and the single-agent default becomes overly conservative.*

9. Predictions and Open Questions

The orchestration problem will be solved by standardization — whoever creates the "Kubernetes of agents" wins the next platform layer.

Three predictions for the next 12–18 months:

Prediction 1: Framework consolidation. The current four-framework landscape is unsustainable. I expect two outcomes by mid-2027: either LangGraph absorbs CrewAI-like simplicity into its ecosystem, or a new entrant (likely from a cloud provider) establishes the dominant abstraction. AutoGen will remain research-focused. Swarm will remain educational.

Prediction 2: Orchestration-as-a-Service emerges. Just as Kubernetes abstracted container orchestration, a managed orchestration layer will abstract agent coordination. This is the highest-leverage infrastructure play in the agent ecosystem. The company that solves multi-agent observability, cost management, and security in a unified platform has a defensible position.

Prediction 3: Inter-agent trust protocols become necessary. The 58–90% hijacking success rate[2] is not sustainable. As multi-agent systems handle higher-stakes tasks, inter-agent authentication and trust scoring will become requirements — not features. Google's A2A protocol addresses system-level authentication but does not address intent verification between agents.

Open questions this report cannot answer:

- What is the optimal agent count for a given task complexity? No benchmark exists.
- How do orchestration failures correlate with task domain? No systematic study exists.
- Will model improvements reduce orchestration complexity, or increase it by enabling more sophisticated agent behaviors?

Beipackzettel

- **Overall Confidence:** 72%
- **Sources:** 13 primary, 5 secondary
- **Strongest Evidence:** MAS hijacking 58–90% success rate across orchestrators (arXiv:2503.12188) [2]
- **Weakest Point:** Production failure case studies are largely anecdotal; no systematic study of orchestration-specific failures exists
- **What would invalidate this report?** If single-agent systems prove sufficient for 90%+ of production use cases, making orchestration irrelevant for most teams
- **Methodology:** Multi-source research across academic papers (4), framework documentation (4), industry surveys (1, n=1,300), practitioner guides (1), market research (2), and news reports (1). Cost models are calculated, not measured. All claims assessed against a standardized confidence rubric (High/Medium/Low).
- **This report was created with a multi-agent research system.** The irony is not lost on me.

Claim Register

Exhibit 6: Claim Register

#	CLAIM	VALUE	SOURCE	CONFIDENCE
1	Companies with agents in production	51%	LangChain State of AI Agents (n=1,300) [1]	**High**
2	MAS hijacking success rate	58–90% (GPT-4o)	arXiv:2503.12188 [2]	**High**
3	Performance quality as #1 barrier to scaling	>2× other factors	LangChain State of AI Agents [1]	**High**
4	MoA outperforms GPT-4o on AlpacaEval	65.1% vs 57.5%	arXiv:2406.04692 [4]	**High**
5	100% attack success in some orchestrator configs	100%	arXiv:2503.12188 [2]	**High**
6	Mid-size companies most aggressive adopters	63% in production	LangChain State of AI Agents [1]	**High**
7	Multi-agent cost is 10–30× single agent	10–30×	Calculated from API pricing + communication overhead [16]	**Medium** (modeled)
8	Most successful implementations use simple patterns	Qualitative	Anthropic "Building Effective Agents" [5]	**Medium** (practitioner)
9	Agent market $7.8B → $52B by 2030	45.8% CAGR	Precedence Research [13]	**Medium** (single analyst)
10	Agentic AI project cancellation >40% by 2027	>40%	Gartner [11]	**Medium** (prediction)

#	CLAIM	VALUE	SOURCE	CONFIDENCE
11	94% of agent developers use observability tools	94%	LangChain State of AI Agents [1]	**Medium**
12	67% of SOC alerts ignored	67%	Vectra (2023) [12]	**Medium**

References

[1] LangChain. "State of AI Agents." Industry survey, 2024. n=1,300 respondents.

[2] Triedman, H. et al. "Multi-Agent Systems Execute Arbitrary Malicious Code." arXiv:2503.12188, 2025.

[3] Guo, T. et al. "Large Language Model based Multi-Agents: A Survey of Progress and Challenges." arXiv:2402.01680, 2024.

[4] Wang, J. et al. "Mixture-of-Agents Enhances Large Language Model Capabilities." arXiv:2406.04692, 2024.

[5] Anthropic. "Building Effective Agents." Practitioner guide, December 2024.

[6] Wu, Q. et al. "AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation." arXiv:2308.08155, 2023.

[7] LangChain Blog. "LangGraph: Multi-Agent Workflows." Technical blog.

[8] OpenAI Cookbook. "Orchestrating Agents: Routines and Handoffs." Technical guide.

[9] CrewAI. Open source documentation. https://docs.crewai.com

[10] Microsoft. AutoGen documentation. https://microsoft.github.io/autogen

[11] Gartner. "Over 40% of Agentic AI Projects Will Be Canceled by 2027." Analyst prediction, 2025.

[12] Vectra. SOC Alert Fatigue Report, 2023.

[13] Precedence Research. AI Agent Market Sizing, 2024. $7.8B → $52B by 2030 (45.8% CAGR).

[14] Multi-agent frameworks survey. arXiv:2502.14143, 2025.

[15] McDonald's AI Drive-Through Program Termination. Various news reports, 2024.

[16] OpenAI and Anthropic. API Pricing Pages. Accessed February 2026.

[17] LangSmith, Langfuse, Arize Phoenix. Product documentation.

[18] Ainary Research Pack (internal). Briefs on Trust, Adversarial, HITL systems.

Cite as: Ziesche, F. (2026). The Orchestration Problem — Why Multi-Agent Systems Fail and How to Fix Them. Ainary Research Report, AR-007.

About the Author: Florian Ziesche is the founder of Ainary Ventures, where he builds AI agent infrastructure and trust systems for enterprise deployments. His research focuses on the coordination, reliability, and governance layers that determine whether AI agents create value or liability.
