AI SYSTEMS

Multi-Agent LLM Systems: Orchestration Patterns, Failure Modes, and When More Agents Help

A practitioner review of multi-agent LLM orchestration - orchestrator-workers, pipelines, debate - with a failure-mode taxonomy and evidence on when adding agents pays.

Bhaskar Roy Sarkar, cazyweb · July 17, 2026 ·

multi-agent systems · LLM agents · orchestration · agent workflows · AI reliability · agent evaluation

Download PDF

ABSTRACT

Multi-agent LLM systems promise what single agents cannot deliver: parallel breadth, role specialization, and independent verification. The research record is genuinely encouraging - role-played agent societies write working software, debate between model instances improves factuality, and simply sampling more agents lifts accuracy on many tasks - yet a growing empirical literature shows most multi-agent failures are coordination failures, not capability failures. This paper reviews the major orchestration patterns (orchestrator-workers, sequential pipelines, debate and judge panels, and optimizable agent graphs), the frameworks that popularized them, and the evidence on when adding agents helps versus when it multiplies cost and failure surface. We ground the review in a production deployment: a growth-experimentation pipeline in which specialized

agents hand reviewed artifacts to one another under deterministic orchestration with human approval gates. The synthesis is a design position: multi-agent systems succeed when the multiplicity serves parallelism, context isolation, or verification, when orchestration is ordinary code rather than agent negotiation, and when handoffs are concrete artifacts a human or validator can inspect.

1. Introduction

The idea that many cooperating models might outperform one is older than the current agent wave - it descends from ensemble methods and from multi-agent systems research in classical AI. What the LLM era added is agents that coordinate in natural language, play roles, and criticize each other's work. The demonstrations arrived quickly: agent societies that simulate believable social behavior[1], role-played assistant pairs that decompose tasks conversationally[2], and software teams of agents that take a one-line requirement to running code[3][4].

What arrived more slowly is discrimination: under what conditions does adding agents actually help, and what does it cost? Surveys of the field catalogue architectures faster than evidence accumulates for them[5][6], and the first large empirical study of multi-agent failures found that the majority trace to system design and coordination, not to the underlying models.[7] This review takes the practitioner's side of that question. We survey the orchestration patterns, summarize the evidence for and against multiplicity, and ground both in a production deployment we operate: a growth-experimentation pipeline in which specialized agents (design, code, reporting) collaborate through reviewed artifacts under deterministic orchestration.

2. Why Multiple Agents: The Three Legitimate Reasons

Stripped of anthropomorphism, a multi-agent system is a way of partitioning context, capability, and judgment. Each partition has a distinct justification.

Parallel breadth. Some tasks decompose into many independent subtasks that would overflow one context window. Anthropic's account of its production research system is the clearest published case: an orchestrator decomposes a research question, spawns parallel subagents with their own context windows, and synthesizes their findings; the parallel system outperformed a single-agent baseline substantially on breadth-heavy queries, while consuming several times the tokens.[8] Relatedly, Li et al. show that simply sampling many agents and taking a majority vote improves accuracy across a range of tasks, with gains that scale with ensemble size before plateauing - multiplicity as variance reduction, no coordination required.[9]

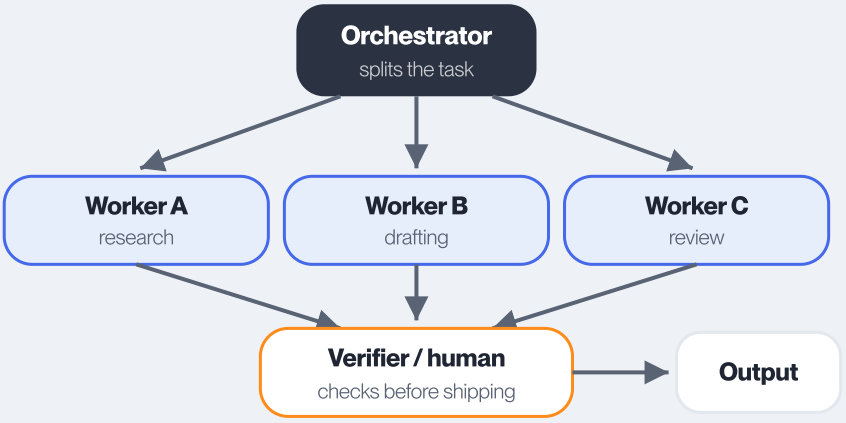
Context isolation and specialization. A design task, a coding task, and a review task want different instructions, different context, and different tools. MetaGPT encodes this as standard operating procedures: agents hold roles (product manager, architect, engineer) and communicate through structured artifacts rather than free chat, which the authors credit for reduced cascading errors.[4] ChatDev reaches the same destination through structured dialogue phases.[3] The lesson practitioners should extract is not the theater of job titles; it is that scoping context per role, and forcing communication through typed artifacts, is what contains error propagation.

Independent verification. A second model instance, prompted adversarially, catches errors the author instance is structurally blind to. Debate between model instances improves factuality and reasoning over single-model baselines[10], encourages divergent thinking that single self-reflection does not produce[11], and multi-agent judge panels align better with human evaluation than single judges on text-quality tasks.[12]

If a proposed multi-agent design does not clearly serve one of these three - parallelism, isolation, verification - the multiplicity is probably decoration.

3. Orchestration Patterns

Orchestrator-workers. A lead agent (or plain code) decomposes the task, delegates to workers, and synthesizes. This is the backbone of the production systems with published post-mortems[8], and its known failure points are instructive: workers duplicating effort when task boundaries are vague, and effort miscalibrated to query complexity - both fixed by more explicit delegation contracts, not by smarter workers.



The orchestrator-workers pattern: a lead agent decomposes the task and delegates to specialist workers that run in parallel; their outputs pass through an independent verification stage before anything is synthesized and shipped.

Sequential pipelines. Stages arranged in a fixed order, each consuming the previous stage's artifact, typically with validation or human review between stages. This is the least glamorous and, in our experience, the most robust pattern; Section 5 describes our deployment of it.

Debate and judge panels. Agents argue over a shared answer[10][11] or evaluate another agent's output collectively[12]. In production these appear less as free-form argument and more as a bounded verification stage: N independent critics, majority rule, applied where a false positive is expensive.

Conversation frameworks and agent graphs. AutoGen generalized the space with programmable multi-agent conversations[13]; GPTSwarm goes further and treats the agent system as a computational graph whose edges -

which agent talks to which - are themselves optimized.[14] These are valuable research instruments. For production, their flexibility is the hazard: every degree of freedom in inter-agent communication is failure surface that deterministic orchestration does not carry.

4. Failure Modes: What Actually Breaks

Cemri et al. assembled the first substantial failure taxonomy from real multi-agent traces across popular frameworks, and the headline finding deserves emphasis: failures concentrate in specification problems (ambiguous roles, underspecified handoffs), inter-agent misalignment (agents proceeding on divergent assumptions, withholding or corrupting context), and weak verification (no stage empowered to reject bad work), rather than in raw model capability.[7] The practical corollaries:

- **Specification is architecture.** If a role or handoff is ambiguous to a careful human reader, agents will diverge on it. The MetaGPT result - structured artifacts beat free chat - is the same finding from the constructive direction.[4]
- **Free-form inter-agent chat is a liability.** Every message an agent composes for another agent is an opportunity to drop a constraint or invent one; typed artifacts and code-mediated handoffs remove the channel.
- **Verification must be empowered and independent.** A verifier that cannot reject, or that shares the author's context and biases, is ceremony. The debate literature works precisely because the critics are independent.[10]
- **Costs compound.** Multi-agent systems consume multiples of single-agent tokens[8], and voting-style gains plateau[9]; past the plateau you are buying latency and bills, not accuracy.

5. A Production Case: Specialist Agents Under Deterministic Orchestration

The experimentation pipeline we operate for growth programs illustrates the sequential pattern with the failure taxonomy designed against from the start. Five stages:

1. **Deterministic context assembly.** A headless-browser scraper (no LLM) captures the client's live page: full-page screenshots at desktop and mobile viewports, extracted design tokens, stylesheets, and a static, script-stripped snapshot of the control page. Reliability work lives in code.
2. **Design agent.** A sandboxed agent - four file tools, scoped to a staged session directory containing the assembled context - produces one artifact: a side-by-side mockup in which the control is the faithfully cloned real page and the variation differs only by the tested change.
3. **Human approval gate.** The mockup is reviewed and versioned; nothing proceeds without approval.
4. **Code agent.** A second sandboxed agent consumes the approved design and emits the variation stylesheet, script, and setup documentation for the testing platform, behind its own approval step.
5. **Report agent.** After the experiment concludes, metrics and significance are computed by code and frozen; a zero-tool agent narrates the numbers it is given and cannot alter them.

Against the taxonomy of Section 4: the agents never address each other in natural language, so the inter-agent misalignment channel does not exist; every handoff is a concrete artifact (a directory of context, an approved HTML file, a frozen metrics object) that a human or validator inspects; verification is independent and empowered at the two expensive boundaries; and orchestration - ordering, versioning, gating - is ordinary application logic, versioned and testable. The system is multi-agent in the sense that matters (specialized contexts, least-privilege capability per role, independent review) while refusing the coordination surface where the measured failures concentrate. We offer this as an architectural observation, not a benchmark claim.

6. Practitioner Guidance

1. **Start with one agent.** Split only when you can name which of the three justifications - parallelism, isolation, verification - the split serves. The single-agent baseline is also your cost floor.[9]
2. **Orchestrate in code.** The workflow's shape (who runs, in what order, what gates exist) should be deterministic application logic. Reserve agent autonomy for within-stage work.[8]
3. **Make handoffs artifacts.** Files, structured objects, drafts in a review queue - anything a human can point at. If you cannot inspect what stage A gave stage B, you cannot debug the system.[4]
4. **Write delegation contracts.** Task boundaries, output formats, and effort budgets per worker, explicit enough that a careful human could not misread them.[7]
5. **Buy verification where errors are expensive.** Independent critics or human gates at external boundaries; audit-after everywhere else.[10][12]
6. **Meter the ensemble.** Token multipliers are real and gains plateau; measure the marginal agent's contribution before keeping it.[8][9]

7. Conclusion

The multi-agent literature supplies both the promise and the warning label. Societies of specialized agents genuinely extend what one model can do - more breadth than one context window, more scrutiny than one perspective - and the measured failures are overwhelmingly failures of coordination design, which is to say, failures a builder controls. The systems that work treat multiplicity as an engineering resource with a price: agents where parallelism, isolation, or verification demands them; code everywhere else; artifacts at every seam; and humans at the edges where mistakes cost money. Multi-agent is not a maturity level past single-agent. It is a specific tool, and like the agents themselves, it performs best inside well-designed constraints.

References

1. Park, J. S., O'Brien, J. C., Cai, C. J., et al. (2023). Generative Agents: Interactive Simulacra of Human Behavior. *UIST 2023*. https://doi.org/10.48550/arXiv.2304.03442

2. Li, G., Hammoud, H. A. A. K., Itani, H., et al. (2023). CAMEL: Communicative Agents for 'Mind' Exploration of Large Language Model Society. *NeurIPS 2023*. https://doi.org/10.48550/arXiv.2303.17760

3. Qian, C., Liu, W., Liu, H., et al. (2024). ChatDev: Communicative Agents for Software Development. *ACL 2024*. https://doi.org/10.48550/arXiv.2307.07924

4. Hong, S., Zhuge, M., Chen, J., et al. (2024). MetaGPT: Meta Programming for A Multi-Agent Collaborative Framework. *ICLR 2024*. https://doi.org/10.48550/arXiv.2308.00352

5. Guo, T., Chen, X., Wang, Y., et al. (2024). Large Language Model based Multi-Agents: A Survey of Progress and Challenges. *IJCAI 2024*. https://doi.org/10.48550/arXiv.2402.01680

6. Xi, Z., Chen, W., Guo, X., et al. (2023). The Rise and Potential of Large Language Model Based Agents: A Survey. *arXiv preprint*. https://doi.org/10.48550/arXiv.2309.07864

7. Cemri, M., Pan, M. Z., Yang, S., et al. (2025). Why Do Multi-Agent LLM Systems Fail?. *arXiv preprint*. https://doi.org/10.48550/arXiv.2503.13657

8. Hadfield, J., Zhang, B., Lien, K., et al. (2025). How we built our multi-agent research system. *Anthropic*. https://www.anthropic.com/engineering/built-multi-agent-research-system

9. Li, J., Zhang, Q., Yu, Y., et al. (2024). More Agents Is All You Need. *Transactions on Machine Learning Research*. https://doi.org/10.48550/arXiv.2402.05120

10. Du, Y., Li, S., Torralba, A., et al. (2024). Improving Factuality and Reasoning in Language Models through Multiagent Debate. *ICML 2024*. https://doi.org/10.48550/arXiv.2305.14325

11. Liang, T., He, Z., Jiao, W., et al. (2024). Encouraging Divergent Thinking in Large Language Models through Multi-Agent Debate. *EMNLP 2024*. https://doi.org/10.48550/arXiv.2305.19118

12. Chan, C.-M., Chen, W., Su, Y., et al. (2024). ChatEval: Towards Better LLM-based Evaluators through Multi-Agent Debate. *ICLR 2024*. https://doi.org/10.48550/arXiv.2308.07201

13. Wu, Q., Bansal, G., Zhang, J., et al. (2023). AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation. *arXiv preprint*. https://doi.org/10.48550/arXiv.2308.08155

14. Zhuge, M., Wang, W., Kirsch, L., et al. (2024). Language Agents as Optimizable Graphs. *ICML 2024*. https://doi.org/10.48550/arXiv.2402.16823

Cite this

APA Copy

```
Bhaskar Roy Sarkar (2026). Multi-Agent LLM Systems: Orchestration
Patterns, Failure Modes, and When More Agents Help. cazyweb Research.
https://cazyweb.com/research/multi-agent-llm-systems-orchestration-
patterns
```

BIBTEX Copy

```
@techreport{multi-agent-llm-systems-orchestration-patterns,
  author = {Bhaskar Roy Sarkar},
  title = {Multi-Agent LLM Systems: Orchestration Patterns, Failure
Modes, and When More Agents Help},
  institution = {cazyweb Research},
  year = {2026},
  url = {https://cazyweb.com/research/multi-agent-llm-systems-
orchestration-patterns}
}
```

RELATED READING

Multi-agent AI workflows for growth teams

How we run our agency on AI agents

Want a team to run experiments like this for you? See how we help

Let's build something that converts.

Start a project with us, or join the bench of specialists and agencies we deliver with.

Start a project ↗

cazyweb

The engineering and growth team behind leading brands and companies across the US and Europe.

CRO and ecommerce insights

Occasional, genuinely useful notes. No fluff.

you@company.com

Subscribe →

Work email only. No spam. Unsubscribe any time. Privacy.

Solutions

- Product Optimization
- Experimentation Program
- cazyweb Platform

For Ecommerce

- CRO Program
- Email & Retention
- Paid Ads
- Landing Pages
- Revenue Calculator FREE
- A/B Test Calculator FREE
- CRO Agency
- Shopify CRO
- Ecommerce Growth
- Website Performance

Company

Explore

CazyWeb IT Solutions
GSTIN: 19JTLPS8476P2ZZ
hello@cazyweb.com

© 2026 cazyweb. All rights reserved.