

Epistemic Topology for AI Agent Systems

Predictive Bounds for Review, Parallelism, and Tool Density

Bogdan Banu
bogdan@banu.be

March 30, 2026

Abstract

AI agent architectures are often chosen heuristically even though coordination topology strongly affects error propagation, handoff overhead, and tool-routing cost. We present a compact topology-centered model of agent systems based on typed wiring diagrams and an epistemic interpretation of what different coordination patterns make visible to workers and coordinators. From that model we derive architecture-level results for reviewer-gated execution, independent versus centralized aggregation, sequential coordination penalty, parallel acceleration under task decomposability, and tool-density scaling. We then compare those qualitative predictions to large-scale external evaluations of multi-agent LLM systems rather than claiming direct benchmark fit. The result is a design calculus for choosing agent topology under reliability and cost constraints.

1 Introduction

AI agent systems increasingly rely on coordination patterns such as planner-worker splits, reviewer gates, specialist swarms, and distributed tool ownership. In practice, these topologies are still chosen mostly heuristically. The central question of this paper is simple: *how should an agent architect choose among common coordination topologies when trading off reliability, communication cost, and decomposition benefit?*

Our starting point is that many familiar coordination pathologies are architectural rather than purely prompt-level failures. A reviewer can suppress some classes of error that a flat worker ensemble cannot. A sequential task can become slower and less reliable when decomposed across multiple agents. A large toolset can become harder to use when split across multiple specialists. These claims are familiar in practice, but they are usually discussed informally and only after systems have already been built.

We use a minimal typed interface model inspired by wiring diagrams and interpret it epistemically: topology determines what each worker or coordinator can observe, and those observation constraints induce predictable costs and failure modes. Biology remains a motivating heuristic in the background, but it is not the main claim of this paper. The motivating intuition is simply that local wiring patterns can stabilize or destabilize distributed information-processing systems, a perspective that has proved useful in the study of biological network motifs [6, 1].

Contributions. This paper makes three contributions:

1. It introduces a minimal typed and epistemic model for agent coordination, with enough structure to reason about what different topologies make visible to workers and coordinators.

2. It derives predictive results for reviewer-gated execution, error amplification, sequential handoffs, parallel decomposition, and tool-density scaling.
3. It compares those qualitative predictions to large-scale external evaluations of multi-agent systems and maps them to an executable coordination substrate in the Operon implementation.

2 Related Work

This paper sits at the intersection of AI agent systems, epistemic reasoning for distributed systems, and lightweight categorical models of typed composition.

2.1 Network Motifs and Coordination Heuristics

Systems biology motivates the idea that local wiring patterns can induce stable system-level behavior. The network-motif literature identifies recurrent subgraphs such as feed-forward loops and signaling bottlenecks as functional building blocks rather than accidental graph fragments [6, 1]. We borrow that design perspective, but not its biochemical mechanistic claims. In this paper, biology serves only as motivation for taking coordination topology seriously in agent systems.

2.2 Applied Category Theory as Interface Language

We use polynomial-style interfaces and wiring diagrams as a compact language for typed composition rather than as the paper’s main novelty. Prior work in applied category theory has shown that polynomial functors and wiring diagrams provide a disciplined language for open, compositional systems [3, 7]. Our use of that machinery is deliberately lightweight: it gives the topology discussion enough precision to support theorem statements and implementation mapping without requiring the paper to be read primarily as a category-theory contribution.

2.3 Epistemic Logic and Multi-Agent Coordination

The epistemic layer draws on standard multi-agent notions of individual, mutual, and distributed knowledge [2, 4]. Our use of epistemic logic is operational rather than foundational: the goal is to connect visibility constraints induced by common AI-agent topologies to concrete tradeoffs in review, handoff, parallelism, and tool use.

2.4 Agent Architecture Evaluation

Much recent work on LLM systems has improved reasoning quality through prompt and inference strategies rather than through explicit architectural analysis [8]. More recent large-scale evaluations of multi-agent LLM systems make clear that coordination topology itself can dominate outcomes on some workloads [5]. This paper is best read as a response to that gap: it proposes a small formal language for topology choice and uses external benchmark results as qualitative consistency checks rather than as direct fit targets.

3 A Minimal Typed Model of Agent Coordination

We model an executable capability as a typed interface consuming observations and producing downstream effects. At the level of notation, one convenient interface language is a polynomial-

style interface:

$$P_A(y) = \sum_{o \in O} y^{I_o}, \quad (1)$$

where O indexes output positions and each I_o is the set of input directions relevant to that output. In this paper the notation is used modestly: it supplies a compact typed vocabulary for composition and visibility, not a full account of agent internals.

Each module m_i has local state S_i , a readout map $r_i : S_i \rightarrow O_i$, and an update map $u_i : S_i \times I_i \rightarrow S_i$. An agent architecture is then a typed wiring diagram $D = (M, W)$ whose nodes are modules and whose edges connect named ports. Each port carries a data type and, when relevant, an integrity level; a wire is valid only if the source can legally flow to the destination or if an attached optic mediates the transfer. This gives a lightweight notion of well-typed composition at the architecture level.

We use three coarse composition patterns:

- **Parallel composition** (\otimes): modules receive independent or shared task inputs and run without direct inter-agent wires.
- **Serial / hub composition** (\circ): one module consumes another module’s output, including the common case of a central coordinator or reviewer.
- **Feedback / trace** (Tr): outputs are fed back into later rounds of execution.

Explicit feedback edges are modeled in the wiring itself, while hidden chain-of-thought or internal reasoning remains part of local state rather than a visible wire. This separation matters because the theorems in Sections 4 and 5 concern what the topology exposes to other modules, not what stays hidden inside a single model invocation. A planner that silently reasons longer is not yet a multi-agent coordination pattern; a planner that emits an intermediate artifact for another module to consume is.

4 What Different Topologies Let Agents Observe

To reason about coordination, we interpret a wiring diagram as constraining who can observe which facts. Let S denote the global execution state of a diagram and let $\text{obs}_i : S \rightarrow \Omega_i$ be the observation function induced by the inputs, outputs, and local readout visible to agent i . This observation map induces an indistinguishability relation:

$$s \sim_i s' \iff \text{obs}_i(s) = \text{obs}_i(s').$$

Agent i knows a proposition φ exactly when φ holds in every global state compatible with its observation.

We use the standard operators:

$$K_i(\varphi), \quad E_G(\varphi) = \bigwedge_{i \in G} K_i(\varphi), \quad C_G(\varphi), \quad D_G(\varphi).$$

Here $K_i(\varphi)$ means agent i knows φ , $E_G(\varphi)$ means every agent in group G individually knows φ , $C_G(\varphi)$ denotes common knowledge, and $D_G(\varphi)$ captures what would be knowable if the group’s observations were pooled.

Why pooled knowledge matters. Many agent architectures fail not because no worker ever observes the right fact, but because the right fact is observed only locally and never made available where a decision is taken. The distinction between local knowledge, mutual knowledge, and pooled knowledge therefore becomes architectural rather than purely logical.

Topology-level intuition. The wiring diagram’s topology directly determines which epistemic operators are easy or expensive to realize:

- **Independent** (\otimes). Each worker observes only its local task state. Distributed knowledge can be rich because workers see different parts of the task, but mutual knowledge is limited to the shared initial input unless extra communication is added.
- **Centralized** (\circ **with hub**). The hub receives all worker outputs, so for propositions expressible in that pooled output tuple, the hub’s partition refines the pooled worker-output partition. Workers, by contrast, typically do not see one another’s outputs unless the hub broadcasts them back.
- **Feedback-rich decentralized coordination** (\otimes with Tr). Peer-to-peer wires create overlapping observation sets. Repeated rounds can refine mutual knowledge toward common knowledge, but each round incurs communication and synchronization cost.

The predictive results in the next section treat these visibility patterns as the main explanatory variable for coordination cost and reliability.

5 Predictive Results for Architecture Choice

Reading guide. Each result below has the same structure. First, the topology fixes what each worker or coordinator can observe. Second, that visibility pattern induces a cost, error, or planning bound through a simple inequality such as a union bound, the data processing inequality, or a critical-path argument. The epistemic notation is bookkeeping for visibility: if an agent cannot observe a fact directly, recovering that fact later requires communication, compression, or review.

Worked examples. Each theorem is paired with a small illustrative agent-architecture example. These examples are design calculations, not benchmark measurements.

5.1 Reviewer Gates and Correlated Failure

A minimal reviewer gate can be modeled as a coherent feed-forward loop: a generator proposes an action, but the action is released only if a reviewer also emits approval.

Theorem 1 (Reviewer-Gated Error Suppression). *Let A_{gen} be a generator agent and A_{ver} be a reviewer. Let E_{gen} denote the event that the generator emits an erroneous candidate, and let M_{ver} denote the event that the reviewer misses that error and still emits approval.*

(a) **Direct link.** *Without a reviewer gate,*

$$P(\text{Fail}_{\text{direct}}) = P(E_{\text{gen}}).$$

(b) **Reviewer gate under independence.** *If E_{gen} and M_{ver} are independent,*

$$P(\text{Fail}_{\text{gate}}) = P(E_{\text{gen}}) P(M_{\text{ver}}).$$

- (c) **Reviewer gate under correlation.** Let $p = P(E_{\text{gen}})$, $q = P(M_{\text{ver}})$, and let ρ be the Bernoulli correlation between the two events. Then

$$P(\text{Fail}_{\text{gate}}) = pq + \rho\sqrt{p(1-p)q(1-q)}.$$

Assumptions. The theorem uses the minimal failure model for a two-stage execution gate: in the direct case an erroneous action is emitted exactly when the generator emits an erroneous candidate; in the gated case an erroneous action is emitted exactly when the generator emits an error and the reviewer still approves it.

Operational takeaway. Adding a reviewer buys multiplicative safety only when the reviewer is not making the same mistakes as the generator. The gate provides the conjunction; diversity in model family, evidence source, or checking mechanism determines how much of that multiplicative gain survives in practice.

Proof sketch. In the direct topology there is only one path from the generator to the action sink, so $\text{Fail}_{\text{direct}} = E_{\text{gen}}$. In the gated topology an erroneous final action can occur only if the generator emits an error and the reviewer misses it while still approving, so $\text{Fail}_{\text{gate}} = E_{\text{gen}} \wedge M_{\text{ver}}$. Independence gives the product pq . For the correlated case, writing $X = \mathbf{1}_{E_{\text{gen}}}$ and $Y = \mathbf{1}_{M_{\text{ver}}}$ yields

$$\mathbb{E}[XY] = \mathbb{E}[X]\mathbb{E}[Y] + \text{Cov}(X, Y) = pq + \rho\sqrt{p(1-p)q(1-q)}.$$

□

Worked Agent Example (Execution Gate). Suppose a code-writing agent proposes a migration and a reviewer checks it before execution. If the generator error rate is $p = 0.1$ and the reviewer miss rate is $q = 0.1$, independence gives $P(\text{Fail}_{\text{gate}}) = 0.01$ instead of 0.1. But if reviewer misses are positively correlated with generator errors, say $\rho = 0.5$, the failure rate rises to 0.055. The gate still helps, but much less than the independent ideal.

5.2 Error Amplification in Independent and Centralized Aggregation

Theorem 2 (Error Amplification Bound). *Consider n agents processing independent subtasks with error probability p , aggregated by a final combiner. The error amplification factor A (aggregate failure probability divided by single-agent failure probability) depends on the coordination topology:*

- (a) **Independent workers.** *If the aggregator observes only final worker outputs,*

$$A_{\otimes} \leq n. \tag{2}$$

- (b) **Centralized coordination.** *If a hub sees all worker outputs before aggregation and detects an error with rate $d = P(\text{hub detects error} \mid \text{error occurred})$, then*

$$A_{\circ} \leq n(1 - d). \tag{3}$$

Assumptions. Worker errors are independent across subtasks, the aggregate fails when at least one erroneous subtask survives to the final output, and in the centralized case the hub has an approximately stationary per-error detection rate d across subtasks.

Operational takeaway. Adding workers without an effective review bottleneck scales failure opportunities roughly with the number of workers. A hub helps exactly to the extent that it can see and suppress cross-worker inconsistencies.

Proof sketch. For independent workers, the aggregate fails with probability $P(\text{failure}_\otimes) = 1 - (1 - p)^n \leq np$ by the union bound, so $A_\otimes = P(\text{failure}_\otimes)/p \leq n$. With a centralized hub, each subtask error reaches the final output only if it occurs and escapes detection, so $P(\text{failure}_\circ) = 1 - (1 - p(1 - d))^n \leq np(1 - d)$, giving $A_\circ \leq n(1 - d)$. In the small- p regime the exact ratio approaches $1/(1 - d)$, recovering the intuition that review quality directly controls the gain from centralization. \square

Worked Agent Example (Code Review Gate). Suppose three code-generation workers each draft part of a deployment change, and a release gate accepts the merged result. If each worker has error rate $p = 0.1$, an ungated independent aggregate fails with probability $1 - (1 - 0.1)^3 = 0.271$, so $A_\otimes = 2.71$. If a central reviewer catches half of all worker errors ($d = 0.5$), then the centralized failure rate becomes $1 - (1 - 0.1(1 - 0.5))^3 \approx 0.143$, so $A_\circ \approx 1.43$.

5.3 Sequential Coordination Penalty

Theorem 3 (Sequential Coordination Penalty). *For a task with k strictly ordered steps decomposed across agents, each inter-agent handoff must at minimum establish $K_{\text{receiver}}(\text{result}_j)$. The overhead is:*

- (a) **Single agent.** $K_i(\text{result}_j)$ is trivial because the agent computed the result. Total cost is $k \cdot c_{\text{step}}$.
- (b) **Multi-agent pipeline.** Each of $h \leq k - 1$ handoffs incurs communication cost c_{comm} and epistemic reconstruction loss

$$\Delta I_j = I(S_{\text{sender}}; r_j) - I(\text{obs}_{\text{receiver}}(S_{\text{sender}}); r_j), \quad (4)$$

which is nonnegative by data processing and strictly positive whenever the handoff is lossy on the task-relevant support.

Assumptions. The task is strictly sequential, later steps depend on earlier results, each handoff transmits only a summary of the sender’s relevant state, and the receiver must reconstruct enough of that state to continue execution.

Operational takeaway. This is the warning against gratuitous decomposition: a sequential pipeline benefits from multiple agents only if a handoff creates new observations or new capabilities. Otherwise the system pays communication cost plus lossy-summary cost.

Proof sketch. A single agent keeps prior results in its own local state, so no handoff is required and total cost is $k \cdot c_{\text{step}}$. When step j is handed from agent a to agent b , the receiver observes only a transformed summary of the sender’s state. By the data processing inequality, $I(\text{obs}_b(S_a); r_j) \leq I(S_a; r_j)$, so the receiver cannot gain information at handoff and typically loses some. Each handoff therefore adds message-construction cost, parsing cost, and reconstruction effort proportional to the lost task-relevant information. \square

Worked Agent Example (Bug-Fix Relay). Consider a three-step bug-fix task: interpret the failing test, locate the root cause, and write the patch. A single coding agent pays $3c_{\text{step}}$. If the task is split across planner, debugger, and patcher with $h = 2$ handoffs, $c_{\text{step}} = 100$ tokens, $c_{\text{comm}} = 25$, and $c_{\text{recon}} = 20$ per handoff, then $C_{\text{single}} = 300$ while $C_{\text{multi}} = 3 \cdot 100 + 2 \cdot (25 + 20) = 390$.

5.4 Parallel Acceleration Under Task Decomposability

The following two results rely on informal notions of epistemic independence and worst-case planner models rather than sharply specified execution semantics. We present them as architecture heuristics that formalize useful design intuitions, not as theorem-level predictive theory.

Definition 1 (Parallel Acceleration under Epistemic Independence). *A task decomposes into m subtasks. Define two subtasks as epistemically independent iff neither solution requires knowledge of the other’s result.*

(a) *Under full epistemic independence with centralized coordination, the speedup is*

$$S = \frac{\sum_{i=1}^m c_{\text{sub}_i}}{\max_i(c_{\text{sub}_i}) + c_{\text{assign}} + c_{\text{agg}}}. \quad (5)$$

(b) *For propositions determined solely by the tuple of worker outputs, the coordinator attains the corresponding pooled-output knowledge at aggregation as an architectural byproduct. Error detection from Theorem 2 therefore comes with little extra communication cost.*

(c) *If subtasks are only partially independent, parallel execution sacrifices quality in proportion to the unresolved mutual information between subtask solutions.*

Assumptions. Subtasks can be assigned independently, coordinator assignment and aggregation are additive overhead terms, and no hidden blocking dependency forces a worker to wait for another worker’s intermediate result.

Operational takeaway. This is the positive case for multi-agent systems: if subtasks can be solved from local observations alone, specialization plus a coordinator can reduce wall-clock cost and often improve quality. If subtasks share unresolved dependencies, parallelism turns into premature decomposition.

Argument. Under epistemic independence, all m subtasks can run concurrently and wall-clock cost is determined by the slowest subtask plus coordinator overhead, giving Eq. (5). Because the coordinator must receive all worker outputs to aggregate them, it also obtains pooled-output visibility as part of normal execution. When subtasks are not independent, some worker decisions depend on facts hidden in another worker’s local state; running them in parallel then replaces needed information sharing with guesswork, reducing solution quality. \square

Worked Agent Example (Due-Diligence Swarm). Suppose an orchestrator assigns three largely independent subtasks for a vendor assessment: legal review, security posture, and cost modeling. If each subtask costs about 8 minutes of agent time and assignment plus aggregation costs 2 minutes total, then

$$S = \frac{8 + 8 + 8}{8 + 2} = 2.4.$$

The coordinator also receives all three outputs at aggregation, so cross-checking across workstreams comes with little extra communication cost.

5.5 Tool Density and the Coordination Tax

Definition 2 (Tool Density Scaling). Consider t tools distributed across n agents, with $|T_i| = t/n$ tools per agent under an approximately balanced partition.

- (a) **Single agent.** Planning cost is $O(t)$ per step because the agent scans one unified tool inventory in context.
- (b) **Multi-agent.** If agent i needs a tool in T_j for $j \neq i$, it must discover the tool, request remote execution, and interpret the result with information loss ΔI from Eq. (4). The coordination overhead per step scales as

$$C_{\text{coord}} = O\left(t \cdot \left(1 - \frac{|T_i \cap T_{\text{needed}}|}{|T_{\text{needed}}|}\right) \cdot c_{\text{comm}}\right). \quad (6)$$

- (c) **Cross-agent planning overhead.** The planner must also reason about remote tool capabilities, yielding worst-case planning cost

$$C_{\text{plan}} = O(tn), \quad (7)$$

which is a multiplicative n -factor over the single-agent baseline and becomes bilinear when both t and n grow.

Assumptions. Tools are partitioned approximately evenly across agents, remote tool use requires explicit discovery and delegation, and the planner may in the worst case need to reason over every tool-agent pairing relevant to the current subgoal.

Operational takeaway. Splitting a toolset across agents changes one local decision into three coordination steps: discover who has the tool, delegate execution, then interpret the returned result without the executor’s full context. That is why tool-heavy tasks often punish over-distribution.

Argument. A single agent scans one tool inventory, so planning cost is linear in t . Under partition, a worker sees only its local tool subset. Remote-tool use therefore introduces discovery, delegation, and result reconstruction costs, producing the coordination term in Eq. (6). Planning also becomes second-order: the agent must reason not just about tool applicability but about which other agent knows how to apply which tool. In the worst case that produces $O(tn)$ candidate comparisons. \square

Worked Agent Example (Tool-Split SWE Agent). Imagine a software-engineering assistant with 18 tools split across 3 agents: one owns search tools, one owns git and test tools, and one owns deployment tools. A planner diagnosing a failing CI job directly holds only 6 tools and must treat the remaining 12 as remote. The planning problem expands from “which of 18 tools should I call next?” to “which agent owns the relevant tool, how do I route the subproblem, and how do I interpret the result without that agent’s full execution context?”

6 Comparison to Large-Scale Agent Evaluations

The empirical role of this section is deliberately narrow. We compare the qualitative predictions of the preceding topology results to architecture-level trends reported in large-scale external evaluations of agent systems [5]. These reported metrics are not literal plug-in values of the simplified theorem parameters. They are used here only as external consistency checks.

Kim et al. [5] report several architecture-level patterns that align with the topology results in Section 5. Independent coordination exhibits much higher error amplification than centralized coordination ($A_e = 17.2$ versus 4.4), consistent with the claim that a validation bottleneck can suppress unchecked worker errors, though these aggregates should not be read as literal estimates of the theorem parameters. PlanCraft degrades under every multi-agent topology, from -39.1% to -70.1% relative to SAS, consistent with the sequential-penalty result that manufactured handoffs are pure overhead when they add no new observations. Finance-Agent shows the opposite regime: centralized MAS improves success from 0.349 to 0.631 ($+80.8\%$), which is consistent with the positive case for decomposable tasks, but still does not directly estimate the speedup variable S . In the tool-heavy Workbench domain, $T = 16$ tools with typical $n = 3$ agent configurations implies substantial remote-tool use, and the reported negative efficiency-tool interaction ($\hat{\beta}_{E_e \times T} = -0.267$, $p < 0.001$) is consistent with the coordination tax from the tool-density bound (Definition 2).

Taken together, these results support the topology-level ordering rather than a numerical fit: reviewer or hub structures help when multiple workers can independently introduce costly errors, strictly sequential tasks should remain consolidated unless a handoff introduces a real new capability, parallel specialist swarms help when subtasks are close to conditionally independent, and tool fragmentation must be treated as an explicit routing problem rather than a free decomposition.

Topology	Epistemic Property	Prediction	Observed
Independent (\otimes)	No inter-agent K_i	Highest error amplification	Architecture-level $A_e = 17.2$
Centralized ($\circ + \text{hub}$)	Hub pools worker outputs	Validation bottleneck lowers amplification	Architecture-level $A_e = 4.4$
Multi-agent + sequential	Requires manufactured K_{receiver}	Strictly sequential tasks degrade under handoffs	PlanCraft: -39.1% to -70.1% vs. SAS
Multi-agent + parallel	Approximate epistemic independence	Large gains on decomposable tasks	Finance-Agent Centralized: $+80.8\%$ vs. SAS
Multi-agent + high $ T $	Knowledge fragmentation	Coordination tax grows with t and n	Workbench $T = 16$; $\hat{\beta}_{E_e \times T} = -0.267$

Table 1: Epistemic predictions versus empirical observations from Kim et al. [5]. The table compares topology-level qualitative predictions to architecture-level metrics. Reported percentages and error-amplification factors are empirical aggregates, not literal plug-in values of the simplified theorem parameters.

This comparison is limited to a single external study [5]. Broader empirical validation across multiple benchmarks and framework configurations remains future work.

7 Executable Coordination Substrate

The submission version only needs enough implementation detail to show that the coordination model is executable. The core pieces retained from the full implementation are the typed wiring substrate, the runtime executor, the wire-level optics relevant to routing, and the finite-trace comparison machinery used to compare diagram behaviors.

Typed wiring. The file `operon_ai/core/wagent.py` provides the static coordination substrate. Its core data structures are:

- `PortType`, a decorated port type carrying a `DataType` and an `IntegrityLabel`;
- `ModuleSpec`, a module declaration with named input and output ports, capability annotations, and optional cost metadata;
- `Wire`, a directed connection between ports, optionally carrying an optic or a denaturation filter;
- `WiringDiagram`, a collection of modules and well-typed wires.

Static connection checks enforce type compatibility and integrity monotonicity unless a wire explicitly attaches an optic that takes responsibility for runtime mediation. For the current paper, the important point is that coordination structure is represented directly and checked before execution.

Runtime execution. A lightweight runtime executor evaluates each module once its required inputs are available. Runtime values retain the type and integrity information used by the static model, and execution records capture module inputs, outputs, and execution order. This makes the wiring semantics inspectable rather than purely schematic.

This execution model is intentionally lightweight. It is sufficient to instantiate the topologies studied in this paper, but it is not presented as a full planner, scheduler, or benchmark harness for production LLM agents.

Wire-level routing. The file `operon_ai/core/optics.py` implements the optics relevant to Paper 1:

- `PrismOptic`, which conditionally routes values by type;
- `TraversalOptic`, which maps a transform over collections;
- `ComposedOptic`, which chains wire-level routing and transformation.

These optics matter for the topology story because they make nontrivial wiring patterns executable: a hub can route different output kinds to different reviewers, and a tool-routing architecture can fan out or transform returned values before aggregation.

Finite-trace comparison. The coalgebraic state-machine layer makes the model executable through `StateMachine` together with parallel and sequential composition operators. A bounded comparison helper checks two machines on a supplied finite input sequence and returns a witness if they diverge.

For Paper 1 this plays a modest role. It does not provide general semantic verification for arbitrary agent systems, but it does support deterministic finite-trace comparisons between alternative coordination layouts under a fixed input script. That is enough to justify the claim that the topology model is implemented rather than only described.

Scope boundary. The larger Operon repository also contains surveillance, healing, multicellular, and other biologically inspired subsystems. Those components are intentionally excluded from this paper unless they are directly needed by a topology experiment, because the contribution here is architecture choice in agent systems, not the full framework inventory.

8 Scope and Limitations

The results in this paper are architecture-level claims, not a substitute for a full benchmark campaign on a production agent stack. Section 6 compares the topology results to external architecture-level aggregates from Kim et al. [5], but that comparison is qualitative. It does not fit the theorem parameters directly, and it is not an end-to-end evaluation of the Operon runtime on the same benchmark suite.

The executable substrate described in Section 7 is likewise narrower than a full semantic verification framework. The finite-trace comparison machinery operates on supplied deterministic input scripts, so it does not establish general behavioral equivalence for nondeterministic, tool-rich, or continuously interactive agent systems. The implementation shows that the topology model can be instantiated and inspected, not that all possible rewritings or coordination patterns are formally certified.

Finally, the topology classes studied here are intentionally coarse. Real deployments often mix reviewer gates, hub coordination, local feedback, and dynamic tool routing inside the same workflow. Model quality, prompt design, and tool reliability can also dominate topology effects in specific domains. The present paper should therefore be read as a compact architectural calculus for agent coordination, with direct benchmarking and adaptive topology selection left to future work.

9 Conclusion

The narrow claim of this paper is that agent coordination topology constrains what workers and coordinators can observe, and that those visibility constraints are strong enough to predict several practically relevant tradeoffs: when review helps, when decomposition hurts, when parallelism pays off, and when tool fragmentation becomes a tax. The formal apparatus is useful to the extent that it sharpens those architecture choices.

The practical implication is that topology should be treated as a first-class design variable in agent engineering. Single agents remain strong on strictly sequential tasks. Reviewer or hub structures help when multiple workers can independently introduce costly errors. Specialist swarms pay off when subtasks are close to epistemically independent and coordinator overhead is small. Tool-heavy workloads punish gratuitous fragmentation unless routing is itself part of the design.

This paper is therefore not a claim that one topology dominates all others, nor a claim that the current theory replaces benchmark evaluation. It is a step toward a more explicit architecture calculus for AI agent systems: one in which coordination choices are justified by visibility, decomposition structure, and cost rather than by orchestration folklore alone.

References

- [1] Uri Alon. Network motifs: theory and experimental approaches. *Nature Reviews Genetics*, 8(6):450–461, 2007.
- [2] Ronald Fagin, Joseph Y Halpern, Yoram Moses, and Moshe Y Vardi. *Reasoning About Knowledge*. MIT Press, 1995.
- [3] Brendan Fong and David I Spivak. *Seven sketches in compositionality: An invitation to applied category theory*. Cambridge University Press, 2019.
- [4] Joseph Y Halpern and Yoram Moses. Knowledge and common knowledge in a distributed environment. *Journal of the ACM*, 37(3):549–587, 1990.
- [5] Yubin Kim, Ken Gu, Chanwoo Park, Chunjong Park, Samuel Schmidgall, A. Ali Heydari, Yao Yan, Zhihan Zhang, Yuchen Zhuang, Yun Liu, Mark Malhotra, Paul Pu Liang, Hae Won Park, Yuzhe Yang, Xuhai Xu, Yilun Du, Shwetak Patel, Tim Althoff, Daniel McDuff, and Xin Liu. Towards a science of scaling agent systems. *arXiv preprint arXiv:2512.08296*, 2025. Google Research, Google DeepMind, and MIT.
- [6] Ron Milo, Shai Shen-Orr, Shalev Itzkovitz, Nadav Kashtan, Dmitri Chklovskii, and Uri Alon. Network motifs: simple building blocks of complex networks. *Science*, 298(5594):824–827, 2002.
- [7] David I Spivak. Learners’ languages. *Compositionality*, 3(4), 2021.
- [8] Jason Wei, Xuezhi Yi, Maarten Zhang, Yiming Liu, Xinyu Li, Xing Wang, Yujia Zhou, Yiming Li, Yuyang Wang, Zhi Li, et al. Chain-of-thought prompting elicits reasoning in large language models. *arXiv preprint arXiv:2201.11903*, 2022. Available at: <https://arxiv.org/abs/2201.11903>.