The Organizational Physics of Multi-Agent AI: Substrate-Independent Dysfunction in Autonomous Software Engineering Swarms

Jeremy McEntire
jmc@cageandmirror.com

February 2026

Abstract

We present empirical evidence that organizational dysfunction is substrate-independent. In a controlled comparison, four coordination architectures—single agent, hierarchical, stigmergic (8 concurrent agents), and gated pipeline—built the same 7-service backend using the same LLM and $50 budget. Performance was inversely correlated with coordination complexity: 28/28, 18/28, 9/28, and 0/28. The pipeline consumed its entire budget on planning. The hierarchical coordinator refused to delegate. The stigmergic agents produced incompatible interfaces at every boundary. Only the single agent—with no coordination architecture—succeeded fully. In two additional studies, a pipeline swarm equipped with six explicit anti-dysfunction mechanisms produced the dysfunction those mechanisms were designed to prevent: bikeshedding (zero-factual-basis rejections), governance conflicts, backward pipeline oscillation, and verification theater. A contract-first alternative that replaces subjective evaluation with mechanical test verification narrowed the Goodhart gap but introduced its own dysfunction (specification perfectionism), suggesting that dysfunction migrates across architectures but does not disappear. We formalize these findings using Crawford–Sobel signal degradation, Goodhart's Law, and the Data Processing Inequality. The results are consistent with the hypothesis that coordination failure arises from information-theoretic constraints on any system coordinating through compressed representations, not from properties of the agents.

Keywords: multi-agent systems, organizational dysfunction, substrate independence, dysmemic pressure, strategic communication, LLM swarms, coordination failure, Goodhart's Law

1 Introduction

The historical demarcation between the study of human organizations and the engineering of artificial intelligence is collapsing. For decades, organizational theory has treated the pathologies of bureaucracy—drift, goal displacement, the principal-agent problem—as uniquely human failures derived from psychology and sociology. Simultaneously, the field of multi-agent systems has encountered a parallel set of failure modes—reward hacking, specification gaming, coordination breakdown—and framed them as technical artifacts of reward function design. A growing body of evidence suggests that these may not be distinct phenomena but isomorphic manifestations of the same information-theoretic constraints inherent in any goal-directed system coordinating at scale.

This paper presents evidence for a strong claim: organizational dysfunction is *substrate-independent.* The same patterns of failure that characterize human organizations—review thrashing, preference-based gatekeeping, governance conflicts, budget exhaustion through coordination failure—emerge in multi-agent AI systems with identical mathematical signatures. The substrate changes; the physics of coordination at scale remains constant.

The evidence comes from three studies. The first two deployed an LLM-based multi-agent coding swarm on a complex task (backend services architecture) and a simple task (chess engine), producing bikeshedding, governance conflicts, backward-moving pipeline oscillation, decision paralysis, and budget exhaustion. A single-agent control completed the simple task successfully. The swarm consisted of specialized agents operating in a pipeline with six explicit anti-dysfunction mechanisms drawn directly from organizational theory. Despite countermeasures against the specific failure modes predicted by theory, the system produced exactly the pathologies it was designed to prevent. The third study extended the analysis to a controlled architecture comparison: four coordination architectures—single agent, hierarchical decomposition, stigmergic emergence, and gated pipeline—built the same 7-service backend using the same model and budget. Performance was inversely correlated with coordination complexity: 28/28, 18/28, 9/28, and 0/28 respectively. Each multi-agent architecture exhibited a distinct dysfunction signature.

The contribution is not the observation that AI systems can fail. It is the demonstration that AI systems fail *for the same structural reasons* as human organizations, *despite the removal of every human-specific causal factor.* No career incentives. No ego. No politics. No fatigue. No cultural norms. No status competition. The agents were language models executing prompts. The dysfunction emerged anyway. This controls for the explanations that organizational theory has traditionally relied upon and is consistent with the information-theoretic mechanism as a sufficient cause.

The paper proceeds as follows. Section 2 establishes the theoretical framework: Crawford–Sobel strategic communication, dysmemic pressure, Goodhart's Law, and the Data Processing Inequality. Section 3 formalizes the substrate-independence claim. Section 4 describes the swarm architecture and its explicit anti-dysfunction mechanisms. Section 6 presents the empirical evidence from all three studies. Section 7 addresses the anticipated objection that the dysfunction was encoded in the prompts. Section 8 discusses implications for multi-agent system design and organizational theory. Section 9 presents the contract-first alternative and its own characteristic dysfunction. Section 10 acknowledges limitations and threats to validity.

2 Background: The Physics of Coordination Failure

Organizational failure is conventionally attributed to human factors: poor leadership, misaligned incentives, cultural dysfunction, cognitive bias. This section establishes an alternative framework in which failure is a structural consequence of information-theoretic constraints that apply to any system coordinating through compressed representations.

2.1 Crawford–Sobel and the Mathematics of Signal Degradation

Crawford and Sobel's 1982 model of strategic information transmission provides the foundational result [Crawford and Sobel, 1982]. When a sender's preferences diverge from a receiver's by bias parameter b, the maximum number of distinguishable partitions in the communication is bounded by

$$N^* = \left\lceil -\frac{1}{2} + \frac{1}{2}\sqrt{1 + \frac{2}{b}} \right\rceil. \qquad (1)$$

At $b = 0$ (perfect alignment), the sender communicates with arbitrary precision. At $b \geq 1/4$, $N^* = 1$: the communication collapses to a *babbling equilibrium* where the receiver extracts no information from the sender's message. The bias need not be large. Any nonzero divergence between sender and receiver objectives reduces communication precision by a quantifiable amount.

In human organizations, this manifests as hierarchical information loss. A project manager whose performance review depends on velocity has a small but nonzero divergence from the engineering lead whose performance review depends on reliability. That divergence, multiplied across every communication relay in a reporting chain, produces measurable information loss. Liberti and Mian's empirical study of hierarchical lending organizations quantified this effect: decision sensitivity to soft information collapses at a structural break between the second and third hierarchical levels [Liberti and Mian, 2009]. The hierarchy does not slowly degrade information. It kills subjective signal at a specific organizational seam.

2.2 Dysmemic Pressure: The Compound Selection Force

Dysmemic pressure is the compound selection force that emerges when three dynamics interact within environments shaped by compressed representations [McEntire, 2025a]:

1. **Strategic communication degradation.** Crawford–Sobel formalized: as preferences diverge, senders transmit with decreasing precision. Each layer of hierarchy introduces a new sender-receiver interface with a new bias parameter. The rational subordinate skews reports toward the principal's priors. What Prendergast calls the "Yes Man" is the equilibrium output of subjective evaluation, not a character flaw [Prendergast, 1993].

2. **Adverse selection in idea markets.** Accurate assessments are expensive to produce. Optimistic ones are cheap. At the moment of presentation, the two are indistinguishable. This is Akerlof's lemons market applied to organizational information [Akerlof, 1970]: when quality cannot be verified before consumption, high-quality producers reduce investment or exit, and the market settles at low quality. The organizational analog: meetings where everyone knows the real situation and discusses the official one.

3. **Transmission bias.** Ideas spread independent of truth value through three channels: content bias (simple outcompetes complex), prestige bias (ideas from leaders propagate faster than identical ideas from others), and conformity bias (once a belief reaches critical mass, deviation becomes costly) [Boyd and Richerson, 1985].

The three forces compound. Signals optimized for internal fitness rather than correspondence with external reality—*dysmemes*—flood the information environment. The resulting equilibrium, where organizational reality decouples from external reality while remaining internally consistent, is stable because no individual can profitably deviate. Dysmemic pressure produces equilibria that resist displacement [McEntire, 2025a].

2.3 Goodhart's Law and the Inevitability of Proxy Failure

"When a measure becomes a target, it ceases to be a good measure" [Goodhart, 1975]. The mechanism is general: any compressed representation of value used as an optimization target will be gamed. The compression from "code quality" to "review pass/fail" creates a proxy. Agents that optimize the proxy—writing code that passes review criteria regardless of actual quality—outperform agents that optimize the underlying objective.

Skalse et al. formalized this: for any environment and true reward function, no non-trivial proxy reward can be guaranteed unhackable [Skalse et al., 2022]. Karwowski et al. confirmed the effect empirically in reinforcement learning: optimizing an imperfect proxy leads to decreased true-objective performance across a wide range of environments [Karwowski et al., 2024]. The impossibility is mathematical, not contingent on agent sophistication. This result applies equally to human employees gaming KPIs, reinforcement learning agents exploiting reward functions, and LLM agents optimizing for reviewer approval.

2.4 The Data Processing Inequality and Irreversible Information Loss

The Data Processing Inequality establishes that for any Markov chain $X \to Y \to Z$:

$$I(X;Z) \leq I(X;Y) \tag{2}$$

No post-processing of a compressed signal can recover information lost at the compression stage [Cover and Thomas, 2006]. If the raw signal is a code diff and the compressed signal is a review verdict (approve/reject with issues list), the verdict has strictly less information than the diff. Processing the verdict more carefully, through more elaborate agent architectures, with more sophisticated prompts, cannot recover what was discarded. This is not a limitation of current models. The Data Processing Inequality guarantees it.

The implication for multi-agent architectures is severe: every layer that processes the output of a previous layer operates on a strictly degraded signal. Adding more agents, more review stages, more governance layers cannot compensate for information already lost. The architecture that adds complexity to address failures caused by complexity is trapped in a structural recursion.

3 The Isomorphism Thesis

The central claim of this paper is that organizational dysfunction in multi-agent AI systems and human organizations may be *isomorphic* rather than merely *analogous*: both may be instantiations of the same formal structure, differing only in substrate.

3.1 The Mechanism: Compression, Selection, Drift

The substrate-independent mechanism operates as a three-step sequence [McEntire, 2025c]:

1. **Compression.** Coordination at scale requires compressing high-dimensional reality into lower-dimensional representations. Individual cognition compresses approximately 10^{10} sensory bits per second into approximately 10 bits of conscious throughput [Zheng and Meister, 2025]. Organizations compress through hierarchy, metrics, and process. AI systems compress training corpora into parameter weights. The compression is necessary and lossy.

2. **Selection.** The compressed representation creates a fitness landscape. Signals that fit the compressed frame survive; those that do not are discarded. In cognition, beliefs consistent with the internal model face less resistance. In organizations, reports consistent with the reporting frame propagate. In AI, outputs that score well on preference proxies are reinforced.

3. **Drift.** Because selection optimizes for fit with the compressed representation rather than for correspondence with external reality, and because the compressed representation is a lossy abstraction, the system drifts. The drift is not random—it is directional, toward the attractor basin of the compressed frame. The frame confirms itself.

Human psychology cannot explain why this pattern appears in AI systems lacking human psychology. Bureaucratic incentives cannot explain why it appears in individual cognition lacking bureaucracy. Machine learning training dynamics cannot explain why it appears in organizations lacking gradient descent. The common cause is compression and selection.

3.2 Parametric Predictions

If the mechanism is substrate-independent, the following parametric correlations should hold across substrates:

- Higher compression ratio → more drift, regardless of substrate.
- Stronger selection pressure → faster drift.
- Longer feedback latency → more drift before correction.
- More hierarchical layers → more signal degradation (Data Processing Inequality).

These predictions are falsifiable. If dysfunction in AI systems correlates with compression and selection parameters but not with substrate-specific features, the substrate-independence claim is supported. Preliminary evidence from other research groups is consistent: Zhang et al. demonstrated that coordination topology determines failure resilience in multi-agent systems regardless of agent capability, with performance degradation ranging from 5.5% to 23.7% depending solely on the coordination structure [Zhang et al., 2025b]. The evidence presented in Section 6 tests these predictions.

3.3 Formal Isomorphism Table

Table 1 maps the specific failure modes observed across substrates.

Table 1: Isomorphic failure modes across substrates

Failure Mode	**Human Organization**	**AI Multi-Agent System**	**Mechanism**
Proxy optimization	Optimize "tickets closed" rather than customer value	CoastRunners: crash to collect points, never finish race	Goodhart's Law
Influence / tampering	Managers win calibration contests through rhetoric	Agents manipulate evaluation channel	Adversarial misalignment
Adverse selection	High performers leave due to random ranking	Catastrophic forgetting; model collapse	Selection for safe/mediocre
Drift	Bureaucratic drift; means become ends	Semantic drift; behavioral laziness over long contexts	Entropy of intent
Bikeshedding	Committee spends 45 min on bike shed, 2 min on reactor	Reviewer rejects on 0 factual, 15–23 subjective issues	Law of triviality
Governance conflict	Board overrules management; management ignores board	Architect force-approves what project-level rejects	Hierarchical incoherence

4 System Architecture and Anti-Dysfunction Mechanisms

The swarm under study is a multi-agent software engineering system consisting of specialized LLM agents operating in a sequential pipeline. The system was designed with explicit awareness of organizational dysfunction theory. Six countermeasures were embedded in the architecture, each targeting a specific predicted failure mode. The countermeasures are described here so that the evidence in Section 6 can be evaluated against the design intent.

4.1 Pipeline Architecture

The pipeline consists of nine stages: `diagnose` (identify the task), `decompose` (break into components), `architect` (design each component), `architect_review` (review the design), `locate` (find relevant files), `execute` (write code), `test` (write tests), `verify` (compile and run tests), and `review` (adversarial code review). A `guardian` policy enforces hard constraints (no secrets, no destructive commands). A `submit` stage handles output.

The architecture mirrors a human software organization: product management (diagnose, decompose), architecture (architect, architect_review), engineering (locate, execute, test), quality assurance (verify, review), and compliance (guardian). This is deliberate: the designers modeled what they knew, which is hierarchical task decomposition with gated review.

4.2 Anti-Dysfunction Mechanism 1: Factual/Subjective Issue Classification

The system includes a `classify_review_issues` function that separates reviewer objections into two categories: *factual* issues (verifiable bugs, broken references, type errors—"line 42 calls `foo()` but the function is named `bar()`") and *subjective* issues (approach preferences, naming conventions, architecture opinions—"should use a different algorithm"). The classification prompt explicitly states: "When in doubt, classify as subjective—the reviewer must earn the right to block."

The intent is to prevent the organizational pattern where subjective preferences are presented as objective requirements, enabling gatekeeping decoupled from quality.

4.3 Anti-Dysfunction Mechanism 2: Perspective-Shift Prompting

After factual issues are fixed in a re-execution cycle, the reviewer receives a `PERSPECTIVE_SHIFT_PREFIX` that forces re-examination: "The factual bugs have been fixed. You are now re-reviewing with ONLY the subjective issues in mind. Re-examine with fresh eyes. Ask yourself: Does the approach actually cause a concrete problem, or would you just do it differently?"

The intent is to prevent the organizational pattern where initial objections persist through inertia even after the substantive issues are resolved.

4.4 Anti-Dysfunction Mechanism 3: Multi-Level Escalation

When the reviewer and executor cannot agree, the system escalates. First to a `project_level` arbiter whose prompt reads: "You are the pragmatic 'ship it or fix it' decision-maker. Cosmetic issues, naming preferences, and 'I would have done it differently' are NOT reasons to reject." If the project-level arbiter fails, the system escalates to an `architect_level` arbiter whose prompt explicitly states: "You may NOT reject. You are the final authority and the pipeline must proceed."

The intent is to prevent the organizational pattern where disagreements at the working level block progress indefinitely.

4.5 Anti-Dysfunction Mechanism 4: Scoped Sub-Pass Review

Rather than a single monolithic review, the system can decompose review into four orthogonal sub-passes: `structural` (module boundaries, API design), `logic` (correctness, edge cases), `consistency` (naming, style), and `blast_radius` (downstream effects). Each sub-pass explicitly ignores the others' domains. The structural reviewer is told: "IGNORE logic correctness, edge cases, and naming style."

The intent is to prevent the organizational pattern where review scope expands to cover everything, producing unfocused objections that conflate severity levels.

4.6 Anti-Dysfunction Mechanism 5: Anti-Bikeshedding Directives

The project-level escalation prompt contains explicit anti-bikeshedding language: "Cosmetic issues, naming preferences, and 'I would have done it differently' are NOT reasons to reject." The reviewer prompt warns: "Leave consternation and escalation BLANK unless something is genuinely wrong. Do NOT use them as status updates."

The intent is to prevent the organizational pattern where trivial objections consume disproportionate attention.

4.7 Anti-Dysfunction Mechanism 6: Lyapunov Stability Monitoring

The system implements a control-theory stability monitor that tracks eight state variables (success rate, learning growth, pause rate, budget efficiency, contradiction rate, signal rate, review rejection rate, average review loops) and computes a Lyapunov function $V(\mathbf{x})$ with weighted contributions. The monitor detects oscillation via sign-flip counting and classifies the system state as stable, degrading, improving, or oscillating.

The intent is to detect the organizational pattern where the system enters a thrashing cycle without any agent recognizing the pattern.

5 Experimental Method

Three studies were conducted. Studies 1 and 2 deployed the same pipeline architecture (Section 4) on different tasks. Study 3 compared four architectures on the same task. All studies used the same base model.

5.1 Model and Configuration

All experiments used Claude Opus 4.6 (Anthropic, model ID `claude-opus-4-6`) via the Anthropic API with default temperature. No fine-tuning, few-shot examples, or system-level jailbreaking was applied. Each agent received a role-specific system prompt (architect, executor, reviewer, etc.) and the task specification. Prompts are reproduced in Section 4 and are available in the accompanying repository.

5.2 Study 1: Pipeline Swarm on Complex Task

Task. Build a backend services architecture comprising four services (Event Service, Availability Service, Pricing Service, Infrastructure Configuration) plus shared infrastructure. The task originated from a production project and was specified in a Linear ticket with acceptance criteria.

Architecture. The 11-stage gated pipeline described in Section 4, with six anti-dysfunction mechanisms active.

Budget. \$25 cap (soft—the system overran to \$57.43). Budget tracked via per-call token counting at published API rates.

5.3 Study 2: Pipeline Swarm on Simple Task

Task. Implement a cryptographic chess engine with Ed25519 move signing, complete move validation, and CLI interface. Independently verified as completable by a single agent.

Architecture. Identical to Study 1.

Control. The same model (Claude Opus 4.6) completed the task in a standard conversational session without pipeline architecture.

5.4 Study 3: Controlled Architecture Comparison

Task. Build a 7-service microservices backend for a vacation rental booking platform: Availability, Pricing, Property, Event, Sync, Payments, and Booking services. The task was specified in a shared prompt document with reference materials: a 236KB architecture specification, a pre-decomposed component plan with API contracts, 16 architectural decisions, database schemas, and example test files. All four architectures received identical copies of all reference materials.

Architectures. Four coordination topologies were tested:

1. **Unary.** Single agent with full task context. No coordination architecture. 100 tool-call iteration limit.
2. **Hi-Trust.** Hierarchical: root agent decomposes task and delegates to leaf agents. Root sees full context; leaves see only their sub-task. 50-iteration limit per agent.
3. **Emergence.** Stigmergic: 8 agents operating concurrently on a shared filesystem with a task board. No central coordinator. Agents self-assign work.
4. **Org Swarm.** Gated pipeline: 11 sequential stages (diagnose, decompose, architect, review, execute, test, verify, etc.) with confidence-gated progression.

Budget. \$50 per architecture, hard cap. Budget tracked via per-call token counting.

Evaluation. Pre-registered rubric applied independently per architecture by an evaluator agent (Claude Opus 4.6) that read all source files, ran available test suites, and scored each component against the architecture specification. Scoring: 0–3 per service $\times$ 7 services (0=not attempted, 1=attempted but non-functional, 2=works in isolation, 3=correctly integrated), plus 0–6 for 6 cross-service integration boundaries, plus 0–1 for shared infrastructure consistency. Maximum: 28. The author reviewed all evaluator outputs for accuracy.

6 Empirical Evidence

6.1 Study 1: Pipeline Swarm on Complex Task

The swarm was deployed to build the backend architecture described in Section 5. The deployment ran to budget exhaustion. The following data is extracted from the system's own audit trail (`state.json`, `audit.jsonl`, stage-level logs).

6.1.1 Aggregate Metrics

Table 2: Swarm deployment aggregate metrics

Metric	**Value**
Total pipeline stages	89
Total cost	\$57.43
Total tokens consumed	7,166,664
Duration	18.1 hours
Completed stages	71 (80%)
Rejected stages	18 (20%)
Cost wasted on rejections	\$12.97 (22.6%)
Terminal status	`budget_exceeded`
Budget cap	\$25.00
Budget overrun	2.3×

The system exceeded its budget cap by a factor of 2.3×, consuming \$57.43 against a \$25 allocation. The overrun was driven entirely by coordination overhead, not task complexity. The execution stages (actually writing code) passed 100% of the time. The test-writing stages passed 100% of the time. The review and verification stages rejected 87% and 67% respectively, producing rejection cycles that consumed budget without producing usable output.

6.1.2 Review Stage Dysfunction

The code review stage rejected 7 of 8 submissions (87.5% rejection rate). The architect review stage rejected 5 of 14 submissions (35.7%). The overall pattern: *production* stages (architect, execute, test) operated efficiently, while *evaluation* stages (review, verify) consumed disproportionate resources through rejection cycles. This is the computational analog of organizations where the quality assurance apparatus consumes more resources than the production apparatus it is meant to serve.

6.1.3 Pure Bikeshedding: Factual = 0

Four review rejections exhibited zero factual basis. These are cases where the factual/subjective classification mechanism (Anti-Dysfunction Mechanism 1) was triggered, and the result showed the reviewer's objections contained no verifiable errors.

Table 3: Pure bikeshedding rejections (factual issues = 0)

Stage	**Component**	**Factual**	**Subjective**	**Cost**
6	Infrastructure Foundation	0	15	\$0.68
12	Event Service (1st)	0	15	\$0.93
15	Event Service (2nd)	0	16	\$1.03
57	Availability Service	0	23	\$0.72
	Total	**0**	**69**	**\$3.36**

The Event Service was rejected three times by architect review, the first two with zero factual basis. Each rejection triggered a full re-architecture cycle. The component that should have been designed once was designed five times. The Availability Service reviewer generated 23 subjective objections and zero factual objections, then blocked the pipeline.

This is the organizational pattern documented by Parkinson as the Law of Triviality and formalized in dysmemic pressure theory: when the evaluation metric (review verdict) becomes the target, the evaluator optimizes the proxy (raising objections) rather than the underlying objective (code quality). Moon et al. identified six systematic biases in LLM code judges—including "Illusory Complexity," where superficial complexity signals affect evaluation independent of actual quality—and found that model scale does not correlate with robustness to these biases [Moon et al., 2025]. The biases are structural, not capability gaps. The factual/subjective classification mechanism detected the problem. It did not prevent it. The reviewer continued blocking.

6.1.4 Governance Hierarchy Conflict

The audit trail records two escalation events occurring 28 seconds apart:

```
20:46:48.613 -- architect_review_escalation:  "project:  reject"
20:47:16.207 -- architect_review_escalation:  "architect:  force_approve"
```

The project-level arbiter and the architect-level arbiter reached opposite conclusions on the same component within 28 seconds. The project-level arbiter, whose prompt instructs pragmatic assessment, rejected. The architect-level arbiter, whose prompt forbids rejection, force-approved. The governance hierarchy disagreed about the same artifact. This is the computational analog of the organizational pattern where different levels of management hold incompatible assessments of the same situation, each operating rationally within their own frame.

The mechanism is Crawford–Sobel operating within the governance architecture itself. Each escalation level has a different objective (the project arbiter optimizes for "good enough," the architect optimizes for "pipeline must proceed"), and these objectives diverge. The divergence produces different verdicts from the same evidence. The hierarchy was designed to resolve disagreements. It produced a new one.

6.1.5 Backward-Moving Pipeline

The expected flow is linear: architect → locate → execute → test → verify → review → submit. The observed flow for the Availability Service:

1. Architect approved, implementation completed through verify (stages 49–54).
2. Review rejected (stage 56).
3. System re-executes, re-tests, re-verifies.
4. Review rejected again (stage 57).
5. System escalates to *re-architecture* (stage 62).
6. Second architecture rejected (stage 63, factual=0, subjective=23).
7. Full pipeline re-executed (stages 64–76).
8. Final verification fails at stage 76. Budget exceeded.

The component moved backward through the pipeline on each rejection. Work product from the execute, test, and verify stages was discarded and regenerated. The pipeline did not learn from its previous attempts. Each re-architecture started fresh, discarding the context that would have informed the next attempt. This is the Data Processing Inequality manifesting operationally: information from the failed attempt was compressed into a rejection verdict (approve/reject + issues list), and the re-architecture operated on that compressed signal rather than on the full context of the failure.

6.1.6 Verification Theater

All nine verification stages reported identical results: `compile=True, tests=0/0`. The code compiled successfully. Zero tests were executed. Zero tests passed. Zero tests failed. The verification stage certified correctness without testing anything.

This is the computational analog of organizational compliance theater: the process exists, the form is completed, the box is checked, and no actual validation occurs. The verification stage consumed compute resources and occupied a pipeline slot while providing zero information about code quality. Yet the pipeline treated a passing verify stage as meaningful evidence, and downstream stages conditioned their behavior on it.

6.1.7 Component Cost Stratification

Table 4: Per-component cost breakdown

Component	Cost	Tokens	Rejections	Rej. Rate
Availability Service	\$17.87	1,945,923	5	19.2%
Infrastructure Config	\$12.12	829,557	3	25.0%
Event Service	\$11.98	1,726,349	5	27.8%
Pricing Service	\$10.61	1,300,373	4	21.1%

The foundational components (Availability, Infrastructure) consumed 52% of total budget despite being reusable shared services. The pattern mirrors organizational resource allocation dysfunction: shared infrastructure receives the most scrutiny precisely because it affects everything, creating review cycles that consume resources disproportionate to the component's complexity. The Event Service consumed \$11.98 across five architectural iterations, producing three bikeshedding-classified rejections, before finally receiving approval. The cost of the Event Service was not the cost of building it. It was the cost of agreeing about it.

6.1.8 Symmetric Dual Rejection

The Pricing Service was attempted twice with nearly identical pipeline sequences. The first attempt (stages 21–28) consumed approximately 70K input tokens and ended in review rejection. The second attempt (stages 78–88) consumed a similar token count and ended in verification failure. Two attempts at the same component, following the same stage sequence, producing different failure modes. The variance in outcomes given identical inputs suggests that the evaluation criteria are not deterministic—the "quality" assessment depends on stochastic factors unrelated to code quality.

This is the forced ranking result from agent-based simulation [McEntire, 2025b]: when the evaluation mechanism produces outcomes statistically indistinguishable from random allocation, the mechanism is measuring evaluation noise, not signal.

6.2 Study 2: Simple Task Under the Same Architecture

To test whether the dysfunction observed in the complex backend task was specific to task complexity, the same swarm architecture was deployed on a qualitatively simpler task: implementing a cryptographic chess engine with Ed25519 move signing, complete move validation, and a command-line interface. The same task was independently completed by a single LLM agent (Claude, operating in a standard conversational session without pipeline architecture) as a baseline.

The single agent produced a working implementation. The swarm did not.

The swarm's pipeline trace tells the story: `diagnose(ok)` → `architect(ok)` → `architect_review(ok)` → `locate(ok)` → `execute(ok)` → `test(ok)` → `test(ok)` → `verify(FAIL)` → `review(FAIL)` → `escalation_triage(ok)`. Total cost: $0.60. The system paused twice for human intervention. The first pause came from the test stage, which escalated over a tooling concern (whether to use `ts-node` or compiled JavaScript for test execution). The second pause came from the review stage, which escalated with the assessment: "Guardian violations (process.exit, diff size) indicate either tooling issues or the author not following submission requirements. The critical signature verification gap suggests the implementation was never tested end-to-end. . . Multiple test gaps ('setting this up correctly is complex') indicate tests were written to pass rather than to validate. This pattern suggests the implementation may need architectural review, not just bug fixes."

The failure mode was qualitatively different from the backend services task. Where the complex task produced bikeshedding (zero-factual-basis rejections) and governance conflict, the simple task produced *decision paralysis*: the architecture stage correctly identified genuine engineering ambiguities—signature format, number canonicalization, error semantics for illegal moves—but treated each as a blocker requiring human input rather than an engineering decision to be made and documented. An independent observer noted: "It found the edges and then sat down on them. . . these are decisions, not questions. An engineer would say 'I'm using null-byte delimiters because it prevents field boundary ambiguity' and keep going."

The chess case provides three additional data points for the substrate-independence argument:

1. **Different task, same architecture, still dysfunctional.** The dysfunction is not an artifact of the backend services task's complexity. A simple, well-specified task also triggers coordination failure.
2. **Different failure mode, same structural cause.** The complex task produced bikeshedding; the simple task produced decision paralysis. Both arise from the same mechanism: evaluation gates that impose coordination overhead disproportionate to task requirements.
3. **Single-agent control succeeds.** The same model, operating without the pipeline architecture, completed the task. The architecture, not the agent capability, is the operative variable.

6.3 Study 3: Controlled Architecture Comparison

Studies 1 and 2 established that the pipeline architecture produces dysfunction on both complex and simple tasks. To test whether the dysfunction is specific to the pipeline topology or general to multi-agent coordination, Study 3 compared four architectures on the same complex task: building a 7-service microservices backend (Availability, Pricing, Property, Event, Sync, Payments, and Booking services). All four architectures used the same underlying model (Claude Opus 4.6), received the same task prompt and reference materials, and operated under the same $50 budget cap. Each service was scored on a pre-registered rubric: 0–3 per service (0=not attempted, 1=attempted but non-functional, 2=works in isolation, 3=correctly integrated), plus 0–6 for cross-service integration boundaries, plus 0–1 for shared infrastructure consistency. Maximum score: 28.

Table 5: Controlled architecture comparison: same task, same model, same budget

Architecture	Score	Cost	Time	Services	Files	Cost/pt
Unary (single agent)	28/28	$51.17	15.5m	7/7	64	$1.83
Hi-Trust (hierarchical)	18/28	$50.62	15.5m	7/7	39	$2.81
Emergence (stigmergic)	9/28	$43.92	14.0m	5/7	152	$4.88
Org Swarm (gated pipeline)	0/28	$1.90	6.8m	0/7	0	∞

The results are striking. Performance is inversely correlated with coordination complexity. The single agent achieved a perfect score. Each additional layer of coordination reduced output quality while increasing cost per unit of quality. The pattern is monotonic: no multi-agent architecture outperformed the single agent on any dimension—not quality, not coverage, not cost efficiency.

Unary (single agent). A single Claude instance, given the full task with no architectural mediation, implemented all 7 services in Python/FastAPI with approximately 114 test cases, consistent API contracts across services, and working integration points. It self-organized its work: reading the architecture specification, prioritizing the dependency order, building shared infrastructure first, then implementing services bottom-up. The single agent maintained coherent context across all 7 services—a capability the multi-agent architectures were designed to distribute but in practice failed to coordinate.

Hi-Trust (hierarchical decomposition). The root agent was designed to decompose the task and delegate sub-tasks to leaf agents. Instead, it refused to decompose, choosing to implement the entire system itself as a single agent within the hierarchical framework. This is a dysfunction finding in itself: the architecture's own coordinator judged that delegation would reduce quality. Even so, its output scored lower than the unary agent (18/28), with all 7 services attempted but only partially integrated. A systematic async test configuration bug caused 23 test failures across the codebase—a single-point-of-failure that a single agent would have caught through coherent context but that the hierarchical architecture, having collapsed to pseudo-single-agent mode, could not self-correct.

Emergence (8 stigmergic agents). Eight agents operated concurrently on a shared filesystem with a task board for coordination. The architecture produced 5 of 7 services (missing Sync and Payments) with 152 files but pervasive interface mismatch: agents working on different services adopted incompatible conventions (`snake_case` vs. `camelCase`, different error response shapes, inconsistent type definitions). The Event Service contained a dual type system—two incompatible sets of type definitions in the same service—produced by different agents working on overlapping components without detecting the conflict. Of 6 integration boundaries scored, zero worked correctly. The architecture produced the most files (152 vs. 64 for unary) with the lowest quality per file, confirming that parallelism without mechanical coordination guarantees produces volume without coherence.

Org Swarm (11-stage gated pipeline). The pipeline architecture consumed its entire $1.90 budget on five planning stages—diagnose, decompose (decisions, components, contracts, implementation order)—without producing a single line of implementation code. This is the most extreme manifestation of Goodhart's Law in the experiment: the architecture optimized its gate-passing metrics (confidence scores, decomposition quality) while producing zero value on the actual objective. The pipeline's own uncertainty amplification mechanism caused it to self-halt: each stage's confidence assessment propagated uncertainty forward, and the accumulated consternation signal exceeded the threshold for autonomous continuation. The system correctly identified risks and produced high-quality architectural analysis, but the analysis consumed the entire budget. The cost of *agreeing about what to build* exceeded the budget for *building it.*

Dysfunction signatures. Each architecture exhibited a characteristic dysfunction pattern:

1. **Org Swarm: Analysis paralysis via Goodhart optimization.** The gated pipeline optimized stage-passing metrics rather than end-to-end output. Analogous to organizations where the planning process consumes the project budget.

2. **Hi-Trust: Non-decomposition as rational defection.** The coordinator rationally refused to use the coordination mechanism, recognizing that delegation would introduce interface risk. Analogous to the manager who does the work themselves rather than delegating because "it's faster if I just do it."

3. **Emergence: Interface mismatch as coordination failure.** Concurrent agents produced incompatible interfaces at every service boundary. The `snake_case`/`camelCase` split is the computational analog of Conway's Law: the system's communication structure (independent agents without shared conventions) determined the interface structure (incompatible naming conventions).

4. **Unary: No coordination dysfunction.** The single agent exhibited none of these patterns because there was no coordination to fail. The absence of dysfunction in the control condition supports the claim that coordination architecture, not agent capability, is the operative variable.

7 The Prompt Defense

The anticipated objection to any claim about emergent AI dysfunction is: "You told it to be argumentative. The prompts encode the behavior." This section examines the prompts and shows that they encode the *opposite* of the observed behavior.

7.1 What the Prompts Actually Say

The reviewer system prompt instructs: "If the changes are correct and complete, approve—but still list risks." It does not instruct the reviewer to reject correct code. The classify_review_issues prompt states: "When in doubt, classify as subjective—the reviewer must earn the right to block." The perspective-shift prompt asks: "Does the approach actually cause a concrete problem, or would you just do it differently?" The project escalation prompt states: "Cosmetic issues, naming preferences, and 'I would have done it differently' are NOT reasons to reject." The architect escalation prompt states: "You may NOT reject."

The prompts contain six distinct mechanisms designed to prevent exactly the behavior that occurred:

1. Factual/subjective classification: separates verifiable bugs from opinions.

2. Perspective-shift re-review: forces reconsideration after factual fixes.

3. Project escalation: pragmatic arbiter explicitly told not to block on style.

4. Architect escalation: final authority explicitly forbidden from rejecting.

5. Scoped sub-passes: orthogonal review dimensions prevent scope creep.

6. Anti-bikeshedding language: explicit directives against trivial blocking.

7.2 Why the Defense Fails

The prompts do not encode dysfunction. They encode *anti*-dysfunction countermeasures. The dysfunction emerged *despite* the countermeasures. This is the strongest form of evidence for substrate-independent organizational physics: the system was explicitly designed to prevent the pattern, the design was informed by organizational theory that predicts the pattern, and the pattern emerged anyway.

The reason the countermeasures fail is structural, not implementational. You cannot implement sight by telling a system to see [McEntire, 2024]. You have to build the architecture that allows seeing to happen. The prompts told the system to see dysfunction and avoid it. The architecture made avoidance structurally impossible.

Consider Anti-Dysfunction Mechanism 1 (factual/subjective classification). The mechanism successfully classifies objections as subjective. It even flags them with the label "bikeshedding." But classification does not prevent action. The reviewer still blocks. The system surfaces the dysfunction ("this rejection has zero factual basis") but cannot act on the insight because the governance architecture treats "reject" as a binary gate regardless of the reason for rejection.

This is precisely the pattern that dysmemic pressure theory predicts. Surfacing accurate information is necessary but insufficient. The information must also survive the selection environment in which it is produced. In this system, the selection environment rewards rejection (reviewers that reject demonstrate rigor) and penalizes approval (reviewers that approve risk allowing bugs). The selection pressure is structural—it is encoded in the pipeline architecture, not in the prompts. The prompts try to counteract it. The architecture overwhelms them.

7.3 The Neutral Prompt

The architect prompt is instructive as a control: "Keep the plan minimal—solve the task, don't gold-plate." The architect sub-passes are scoped to: MVP adherence, simplicity, architectural integrity, code reuse, and standards. There is no instruction to be adversarial, argumentative, or obstructive. The architect stage approved 100% of its outputs. The dysfunction originates in the evaluation layers, not the production layers. The agents that *build* work efficiently. The agents that *judge* create friction. This asymmetry maps exactly to the organizational pattern where engineering teams ship efficiently and are slowed by review committees.

8 Implications

8.1 The Single-Agent Ceiling

The evidence suggests that multi-agent AI systems face a coordination ceiling analogous to the Elliott threshold ($\sim$25 participants) observed in human collaboration [Elliott, 2007]. Below a certain complexity threshold, a single agent can maintain coherent context and produce output without coordination overhead. Above it, the system must distribute work across agents, introducing communication channels where Crawford–Sobel degradation operates.

Recent empirical results corroborate this. Chen et al. established quantitative scaling laws for multi-agent systems: every multi-agent variant degraded sequential reasoning performance by 39–70%, independent agents operating in parallel amplified errors 17.2×, and centralized hierarchical oversight contained amplification to 4.4× but could not eliminate it [Chen et al., 2025]. Their finding that coordination yields diminishing or negative returns above approximately 45% baseline capability confirms the coordination tax as a structural cost. Chen et al.

found earlier that compound inference performance is non-monotonic: more LLM calls improve easy queries but degrade hard queries [Chen et al., 2024]. Cemri et al. analyzed over 1,600 multi-agent failure traces across seven frameworks and concluded that failures stem from *system design issues*, not LLM limitations—an independent confirmation that the dysfunction is architectural [Cemri et al., 2025]. Wynn et al. demonstrated that multi-agent debate more often shifts correct responses to incorrect than incorrect to correct—the mechanism is RLHF-trained sycophancy, the computational analog of Prendergast's Yes Man [Wynn et al., 2025, Prendergast, 1993]. Pappu et al. found that LLM teams underperform their best individual member by 8–38% through "integrative compromise"—averaging expert and non-expert views rather than deferring to expertise—an effect that worsens with team size [Pappu et al., 2026]. Xu et al. demonstrated that a single agent can match the performance of homogeneous multi-agent workflows, confirming that multi-agent coordination is overhead without benefit when agents share the same base model [Xu et al., 2026].

The implication is practical: the default recommendation should be the fewest agents capable of completing the task. Every additional agent introduces a communication channel. Every communication channel introduces Crawford–Sobel degradation. The coordination overhead is not implementation debt to be paid down with better prompting. It is an information-theoretic cost of distributed cognition.

8.2 Goodhart's Law as Architectural Constraint

The swarm's scoring system reveals Goodhart's Law operating as an architectural constraint. The scoring functions use seven metrics: confidence, diff size, test coverage, risk count, coherence, guardian pass/fail, and issue count. The weights are manually tuned. The guardian score carries weight 3.0 (highest), followed by confidence and coherence at 2.0.

None of these metrics measure whether the code actually does what it is supposed to do. They measure properties of the artifact (size, structure, test coverage) and properties of the evaluation (confidence, issue count). The system optimizes for what it can measure. What it can measure is a proxy for what matters. The gap between proxy and objective is where dysfunction lives.

The scoring system contains no metric for epistemic surprise ("did this reveal something no agent expected?"), topological bridging ("does this connect previously unconnected components?"), or objective validation ("does the output match an external reference"). The absence is not an oversight. These properties are hard to measure. Goodhart's Law predicts that including a measurable proxy for them would not help—the system would optimize the proxy rather than the property.

8.3 Self-Evaluation Impossibility

Lawvere's fixed-point theorem establishes that in any Cartesian closed category, if a point-surjective morphism $\varphi : A \to B^A$ exists, then every endomorphism of B has a fixed point [Lawvere, 1969]. The contrapositive: if the evaluative space has any operation without a fixed point (such as negation), then no system can internally represent all of its own evaluations. Yanofsky showed that Cantor's diagonal argument, Russell's paradox, Gödel's incompleteness, Tarski's undefinability, and Turing's halting problem are all instantiations of this single abstract scheme [Yanofsky, 2003].

The swarm's verification failure—`tests=0/0` across all nine verify stages—is a concrete manifestation. The system was tasked with evaluating its own output. The evaluation mechanism (running tests) was part of the same system that produced the output. When the evaluation mechanism fails silently, no internal process detects the failure, because the detection of evaluation failure would itself require a working evaluation mechanism. The system certifies its own correctness without the capacity to verify the certification.

Panickssery et al. demonstrated this empirically: LLM evaluators recognize and systematically favor their own generations [Panickssery et al., 2024]. Zhang et al. found that even the best automated methods achieve only 53.5% accuracy in failure attribution across multi-agent systems—barely better than a coin flip [Zhang et al., 2025a]. Systems cannot grade their own exams.

8.4 Organizational Theory as Engineering Discipline

If organizational dysfunction is substrate-independent, then organizational theory is not a social science applied to engineering by analogy. It is an *engineering discipline* that applies directly. Crawford–Sobel is not a metaphor for what happens in multi-agent systems. It is the mathematical model that predicts the observed behavior. Goodhart's Law is not an aphorism. It is a constraint that scoring system designers must respect. The Data Processing Inequality is not an abstraction. It is the reason adding more review stages makes things worse.

This reframing has design implications. Multi-agent system designers currently draw from computer science (distributed systems, consensus protocols) and machine learning (prompt engineering, fine-tuning). They should also draw from organizational theory, cybernetics, and information economics. The empirical regularities are the same. The mathematics is the same. The solutions, where they exist, transfer. Chang independently arrives at a convergent conclusion, arguing that coordination is the missing layer of AGI and proposing "coordination physics" as a field [Chang, 2025]. La Malfa et al. observe that most current "multi-agent" LLM systems are not genuine multi-agent architectures in the classical sense but single-LLM role-play, and thus fail to engage with foundational coordination principles [La Malfa et al., 2025].

9 The Contract-First Alternative

The dysfunction documented in this paper arises from three information-theoretic mechanisms: Crawford–Sobel degradation at agent-to-agent communication boundaries, Goodhart optimization of evaluation proxies, and irreversible signal loss across coordination layers. A fifth architecture—contract-first coordination—was developed in response to these findings and tested concurrently with the controlled architecture comparison. Its design targets each mechanism structurally rather than through prompting.

9.1 Architecture: Contracts and Tests as Coordination Medium

The contract-first architecture replaces agent-to-agent communication with a three-phase pipeline: *specify*, *verify*, *implement*. In the specification phase, a decomposer agent breaks the task into components and a contract author generates formal interface contracts for each: typed function signatures, pre/postconditions, module invariants, error codes, and dependency declarations. A test author then generates executable test suites derived mechanically from the contracts. Only after contracts and tests pass a mechanical validation gate—no cycles in the dependency graph, all cross-references resolve, all test code parses—does implementation begin. Each component is implemented independently by a code author that sees only the component's contract, its dependency contracts, and its test suite. Integration is verified by composition tests at parent nodes.

The key architectural difference from the systems studied in Sections 6 and 6.3:

1. **No agent-to-agent evaluation.** There is no reviewer agent, no gated approval, no escalation hierarchy. The acceptance criterion is mechanical: *tests pass or they do not.* This eliminates the sender-receiver interface where Crawford–Sobel degradation operates. A test suite has no preferences, no bias parameter b, no strategic incentive to compress or distort. It is a fixed specification against which implementation is measured.

2. **Contracts precede implementation.** In every other architecture tested, interface definitions emerged *during* implementation. The emergence architecture's `snake_case`/`camelCase` split (Section 6.3) occurred because two agents independently chose conventions with no shared specification. In the contract-first architecture, every interface is specified before any agent writes implementation code. The specification is the coordination medium—not a summary of what was built, but a prescription of what must be built.

3. **Goodhart-resistant verification.** The gated pipeline's scoring system used seven proxy metrics, none measuring whether code did what it was supposed to do. The contract-first architecture's verification mechanism is a direct measurement: the test encodes the specification, and the implementation either satisfies it or does not. This does not make the proxy unhackable—an implementation could pass tests through pathological means—but the gap between proxy and objective is narrower because the proxy is derived mechanically from the specification rather than from subjective evaluation.

9.2 Preliminary Evidence: Contract Quality

A contract-first deployment on a production task (automatic delisting of properties with persistent API errors) produced preliminary evidence for the approach's strengths and weaknesses. In the specification phase, the system decomposed the task into 5 components, generated 3 formal contracts with a total of 9 typed functions, 22 module invariants, and 61 test cases across 2 components—before writing a single line of implementation code.

The contract quality was exceptional. Each function specification included typed parameters with validation constraints, 3–4 branch precondition logic, specific postconditions ("if a prior cleared record existed, record is reset: firstFailedAt == lastFailedAt == timestamp, failureCount == 1"), named error codes with module namespacing, and idempotency guarantees. Each contract included 6 research findings with confidence scores, source attribution, and relevance justification. The test suites included happy path, error, edge case, invariant, and lifecycle categories.

The interface mismatch problem that plagued the emergence architecture is structurally prevented: all agents implement against the same mechanical specification. The bikeshedding problem that plagued the gated pipeline is structurally prevented: there is no subjective evaluation to optimize. The non-decomposition problem that plagued the hierarchical architecture does not arise: decomposition produces contracts, not delegation instructions, so the coordinator has no incentive to defect. In the language of Section 7: the contract-first architecture does not tell agents to produce coherent interfaces. It builds the architecture—shared contracts, mechanical verification—that makes coherence the only option.

9.3 The New Dysfunction: Specification Perfectionism

The contract-first architecture introduces its own characteristic dysfunction. The production deployment consumed 2.4 hours generating contracts without producing implementation code. The audit trail showed backward oscillation within the specification phase: decompose $\rightarrow$ contract $\rightarrow$ re-decompose $\rightarrow$ re-contract. Contract revisions produced increasingly detailed specifications (607 lines of JSON for a 4-function module) while the implementation budget shrank.

This is Goodhart's Law operating on a new proxy. The gated pipeline optimized stage-passing confidence scores. The contract-first system optimizes contract completeness. Both are measurable proxies for the underlying objective (working code), and both systems optimize the proxy at the expense of the objective. The dysfunction has migrated from the evaluation phase to the specification phase, but it has not been eliminated.

The pattern has an organizational analog: the enterprise architecture team that produces comprehensive specification documents while the delivery team waits for permission to build.

The specifications are individually excellent. The system that produces them has substituted specification for implementation.

9.4 Theoretical Implications

The contract-first results suggest that the dysfunction documented in this paper is not specific to *hierarchical* coordination or *agent-to-agent* communication. It is a property of any system that introduces a proxy between intention and output. Hierarchical architectures proxy through evaluation judgments. Stigmergic architectures proxy through environmental convention. Contract-first architectures proxy through formal specifications. Each proxy is closer to the objective than the last—tests are a better proxy than reviewer approval, which is a better proxy than confidence scores—but no proxy is immune to Goodhart optimization.

This observation refines the substrate-independence claim. The dysfunction is substrate-independent not merely in the sense that it appears in both human and AI systems, but in the deeper sense that it appears in *every coordination architecture* that introduces a measurable intermediate representation. The specific failure mode varies—bikeshedding, interface mismatch, specification perfectionism—but the mechanism is invariant: agents optimize the representation rather than the objective, because the representation is what they can observe and act upon. The dysfunction is not in the agents or in the architecture. It is in the gap between any finite representation and the reality it represents.

Zhang et al.'s finding that different coordination topologies produce dramatically different failure resilience (5.5% vs. 23.7% performance degradation) using the same agents provides empirical support for the claim that topology is the operative variable [Zhang et al., 2025b]. The contract-first results extend this: even topologies designed to eliminate specific failure modes introduce new ones. The design problem is not to eliminate dysfunction but to choose which dysfunction to accept—and to ensure the chosen proxy is close enough to the objective that optimization of the proxy produces acceptable optimization of the objective.

10 Limitations and Threats to Validity

Several limitations constrain the strength of the claims made in this paper.

Sample size. The primary empirical evidence comes from three studies: a pipeline swarm deployment (N=1, 89 stages), a simple-task deployment (N=1, 10 stages), and a controlled architecture comparison (N=4 architectures, same task). While the architecture comparison provides stronger evidence by controlling for model, task, and budget, each architecture was tested only once. Stochastic variance in LLM outputs means that a single run per architecture cannot fully distinguish systematic effects from run-to-run variation. Replication with multiple runs per architecture and across model families would strengthen the claims.

Training data contamination. LLMs are trained on human-generated text, including code reviews, organizational communications, and management literature. An LLM reviewer that bikesheds may be replaying learned patterns from internet code reviews rather than deriving the behavior from information-theoretic constraints. The six anti-dysfunction prompts argue against this interpretation (the model was explicitly instructed not to bikeshed and did anyway). The controlled architecture comparison provides further evidence: the same model, operating as a single agent without coordination architecture, produced no dysfunction and achieved a perfect score. Under a hierarchical architecture, the same model refused to delegate. Under an emergence architecture, the same model produced incompatible interfaces. The dysfunction correlated with architecture, not with model—but the training data confound cannot be fully eliminated until models trained on non-organizational corpora are tested.

Architecture-specific pathologies. The controlled architecture comparison (Section 6.3) demonstrated that different coordination topologies produce qualitatively different dysfunction patterns: analysis paralysis (gated pipeline), non-decomposition (hierarchical), interface mismatch (emergence). These are architecture-specific pathologies rather than a single universal failure mode. Whether they share a common information-theoretic mechanism (as claimed) or are merely superficially similar requires formal analysis—computing the effective Crawford–Sobel bias parameter b for each architecture's communication channels—that has not yet been conducted.

Internal metrics vs. external quality. The paper relies on the swarm's own audit trail for evidence. The factual/subjective classification, the rejection counts, and the cost breakdowns are all generated by the system under study. No external test suite or independent human evaluation was used to establish ground truth for code quality. The "bikeshedding" classification assumes the factual/subjective classifier is accurate; if the classifier systematically under-counts factual issues, the bikeshedding claim is weakened.

Budget enforcement. The swarm exceeded its \$25 budget cap by a factor of 2.3×, reaching \$57.43. The budget cap was not a hard stop—the system continued executing after exceeding the allocation. This ambiguity affects the interpretation of cost-based metrics: the overrun may reflect a design choice (soft cap) rather than a loss of control.

Stochastic evaluation. The symmetric dual rejection of the Pricing Service (same component, same pipeline sequence, different failure modes) suggests substantial stochastic variance in the evaluation layers. If evaluation outcomes are dominated by random seed rather than code quality, the dysfunction documented here may be partly noise rather than signal. Quantifying evaluation variance through repeated re-review of the same artifacts would strengthen the claims.

Single evaluator. The author is the sole human evaluator of the swarm's outputs and the sole interpreter of the audit trail. This introduces potential confirmation bias in the selection and framing of evidence. Independent replication by researchers without a stake in the substrate-independence thesis would provide stronger evidence.

11 Conclusion

A multi-agent AI system designed to build software autonomously was equipped with six explicit mechanisms to prevent organizational dysfunction. It produced organizational dysfunction anyway. The dysfunction was not encoded in the prompts. The prompts encoded the opposite. The dysfunction emerged from the architecture: a hierarchical pipeline with gated review, operating under the information-theoretic constraints that apply to any system coordinating through compressed representations.

A controlled architecture comparison confirmed and extended this finding. Four architectures—single agent, hierarchical decomposition, stigmergic emergence, and gated pipeline—built the same 7-service backend using the same model, task, and budget. The single agent scored 28/28. Every multi-agent architecture scored lower, and the ordering was monotonic with coordination complexity. The gated pipeline consumed its entire budget on planning without writing code. The hierarchical architecture's coordinator rationally refused to delegate. The emergence architecture produced five services with incompatible interfaces at every boundary. The cost per quality point increased monotonically: \$1.83 (single), \$2.81 (hierarchical), \$4.88 (emergence), ∞ (pipeline).

The evidence is consistent with the substrate-independence hypothesis: organizational dysfunction correlates with architectural features—compression depth, selection pressure, coordination overhead—rather than with properties of the agents. Across all studies, dysfunction correlated with coordination complexity—not with substrate-specific features. The single-agent control condition, which achieved perfect scores on both a simple task and a complex 7-service architecture, supports the interpretation that coordination architecture, not agent capability, is the operative variable.

The practical implications are immediate. Multi-agent system designers should treat organizational theory as an engineering discipline, not a metaphorical import. Crawford–Sobel provides the mathematical model for communication channel design. Goodhart's Law constrains scoring function architecture. The Data Processing Inequality limits the value of additional review stages. Contract-first coordination—where formal specifications and mechanical tests replace subjective evaluation—narrows the Goodhart gap but does not close it. The emergence architecture's failure demonstrates that stigmergic coordination through shared environment, without mechanical specification, produces its own dysfunction (interface mismatch). The contract-first architecture's specification perfectionism demonstrates that even mechanically grounded proxies attract optimization pressure. The design problem is not to eliminate the gap between proxy and objective but to minimize it.

The theoretical implication is broader. If confirmed by further experiments across architectures and model families, the convergence of dysfunction patterns across human organizations, AI systems, individual cognition, and academic publishing [McEntire, 2025c] would support the hypothesis that organizational dysfunction is not a problem of human nature but a problem of coordination physics. The contract-first results sharpen this: the dysfunction migrates across architectural choices but does not disappear, because every finite coordination medium is a lossy representation of the objective it serves. Physics problems have engineering solutions—but the engineering must respect the constraint rather than pretend it can be eliminated.

References

George A. Akerlof. The market for "lemons": Quality uncertainty and the market mechanism. *The Quarterly Journal of Economics*, 84(3):488–500, 1970. doi: 10.2307/1879431.

Robert Boyd and Peter J. Richerson. *Culture and the Evolutionary Process*. University of Chicago Press, 1985.

Mert Cemri, Melissa Z. Pan, Shuyi Yang, et al. Why do multi-agent LLM systems fail? In *NeurIPS 2025 Datasets and Benchmarks Track*, 2025. Also presented at Building Trust Workshop, ICLR 2025. arXiv:2503.13657.

Edward Chang. The missing layer of AGI: From pattern alchemy to coordination physics. *arXiv preprint arXiv:2512.05765*, 2025. Stanford University.

Lingjiao Chen, Jared Quincy Davis, Boris Hanin, Peter Bailis, Ion Stoica, Matei Zaharia, and James Zou. Are more LLM calls all you need? towards scaling laws of compound inference systems. In *Advances in Neural Information Processing Systems*, volume 37, 2024. arXiv:2403.02419.

Lingjiao Chen et al. Towards a science of scaling agent systems. *arXiv preprint arXiv:2512.08296*, 2025. Google Research.

Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory*. Wiley-Interscience, 2nd edition, 2006.

Vincent P. Crawford and Joel Sobel. Strategic information transmission. *Econometrica*, 50(6): 1431–1451, 1982. doi: 10.2307/1913390.

Mark Elliott. Stigmergic collaboration: A theoretical framework for mass collaboration. *PhD Thesis, University of Melbourne*, 2007.

Charles A. E. Goodhart. Problems of monetary management: The U.K. experience. *Papers in Monetary Economics*, 1, 1975.

Jacek Karwowski, Joar Skalse, et al. Goodhart's law in reinforcement learning. In *ICLR 2024*, 2024. arXiv:2310.09144.

Emanuele La Malfa et al. Large language models miss the multi-agent mark. *arXiv preprint arXiv:2505.21298*, 2025. NeurIPS 2025 poster.

F. William Lawvere. Diagonal arguments and cartesian closed categories. *Lecture Notes in Mathematics*, 92:134–145, 1969.

José M. Liberti and Atif R. Mian. Estimating the effect of hierarchies on information use. *The Review of Financial Studies*, 22(10):4057–4090, 2009. doi: 10.1093/rfs/hhn118.

Jeremy McEntire. *The Cage and the Mirror: Organizational Dysfunction via Gödelian Incompleteness.* Cage & Mirror Press, 2024.

Jeremy McEntire. Dysmemic pressure: A selection-theoretic account of organizational self-deception. *SSRN preprint*, 2025a.

Jeremy McEntire. Forced ranking analysis: A simulation-based assessment of tournament evaluation in organizations. *arXiv preprint*, 2025b.

Jeremy McEntire. Compression, selection, and organizational self-deception: A substrate-independent theory of systematic epistemic failure. *SSRN preprint*, 2025c.

Seungone Moon et al. Don't judge code by its cover: Exploring biases in LLM judges for code evaluation. *arXiv preprint arXiv:2505.16222*, 2025.

Arjun Panickssery, Samuel R. Bowman, and Shi Feng. LLM evaluators recognize and favor their own generations. In *NeurIPS 2024 (Oral)*, 2024. arXiv:2404.13076.

Anikait Pappu et al. Multi-agent teams hold experts back. *arXiv preprint arXiv:2602.01011*, 2026.

Canice Prendergast. A theory of "yes men". *The American Economic Review*, 83(4):757–770, 1993.

Joar Skalse, Nikolaus H. R. Howe, Dmitrii Krasheninnikov, and David Krueger. Defining and characterizing reward hacking. *Advances in Neural Information Processing Systems*, 35, 2022.

Andrea Wynn, Harsh Satija, and Gillian K. Hadfield. Talk isn't always cheap: Understanding failure modes in multi-agent debate. In *ICML 2025*, 2025. PMLR 267. arXiv:2509.05396.

Zhiyu Xu et al. Rethinking the value of multi-agent workflow: A strong single agent baseline. *arXiv preprint arXiv:2601.12307*, 2026.

Noam S. Yanofsky. A universal approach to self-referential paradoxes, incompleteness and fixed points. *Bulletin of Symbolic Logic*, 9(3):362–386, 2003. doi: 10.2178/bsl/1058448677.

Shaokun Zhang, Ming Yin, Jieyu Zhang, Jiale Liu, Zhiguang Han, Jingyang Zhang, Beibin Li, Chi Wang, Huazheng Wang, Yiran Chen, and Qingyun Wu. Which agent causes task failures and when? on automated failure attribution of LLM multi-agent systems. In *ICML 2025 Spotlight*, 2025a. arXiv:2505.00212.

Tianyu Zhang et al. On the resilience of LLM-based multi-agent collaboration with faulty agents. In *ICML 2025*, 2025b. arXiv:2408.00989.

Jieyu Zheng and Markus Meister. The unbearable slowness of being: Why do we live at 10 bits/s? *Neuron*, 113(2):192–204, 2025. doi: 10.1016/j.neuron.2024.11.008.