

# Contents

<b>Maturity Gaps and Structural Blind Spots: How Incomplete Abstractions Undermine Reliability Across Agentic, Supply-Chain, and Language-Theoretic Systems</b>	<b>1</b>
Abstract . . . . .	1
Introduction . . . . .	1
Background . . . . .	2
Synthesis . . . . .	3
Discussion . . . . .	6
Limitations . . . . .	7
Conclusion . . . . .	8

## Maturity Gaps and Structural Blind Spots: How Incomplete Abstractions Undermine Reliability Across Agentic, Supply-Chain, and Language-Theoretic Systems

*Saluca Agentic AI Research Team — Saluca LLC. AI-drafted synthesis from recent arXiv preprints; for human review, not peer-reviewed.*

---

### Abstract

A recurring structural pattern appears across five recent cs.SE and cs.PL preprints: systems fail not because individual components malfunction at the task level, but because the *abstractions used to describe, compose, and monitor those components are underspecified or mismatched to the actual operational substrate*. This paper synthesizes findings from monitoring agentic systems [arXiv:2606.02494](#), SBOM component inclusion gaps [arXiv:2606.02442](#), multi-agent LLM collaboration topology experiments [arXiv:2606.01490](#), binary decompilation evaluation [arXiv:2605.29490](#), and categorical semantics for untyped effectful computation [arXiv:2605.31389](#) and [arXiv:2605.21337](#) into a single candidate reading: **abstraction boundary failures—not runtime errors—are the dominant reliability bottleneck in contemporary software-intensive systems**. We argue, as a heuristic reading rather than a derivation, that each corpus finding instantiates a common structural pattern: a layer of tooling or formalism assumes a shared, stable definition of “component,” “task,” “behavior,” or “type” that the underlying system does not actually provide. The falsification path is concrete: if abstraction-boundary-aware instrumentation does not reduce undetected failure rates more than task-level instrumentation in controlled experiments, the thesis collapses. We also flag one weakly-connected source and disclose the selection process. Sources span cs.SE and cs.PL from May–June 2026; all are preprints without peer review.

---

### Introduction

The dominant engineering assumption in software reliability is that correctness can be decomposed: verify components individually, compose them, and the system behaves correctly. This assumption is load-bearing for every monitoring framework, every supply-chain security tool, and every formal

verification pipeline. But several recent preprints, independently, report that the assumption breaks at precisely the boundary where it is most needed.

In agentic system monitoring, [arXiv:2606.02494](#) finds that injected task-level errors are “indistinguishable from clean baselines” because structural defects mask the signal that task-level monitors are designed to detect. The monitoring layer assumes a stable notion of “task outcome,” but the system’s integration gaps prevent that signal from being observable. In SBOM generation, [arXiv:2606.02442](#) finds that no evaluated tool covers all identified Component Inclusion Mechanisms (CIMs), and that “a security-grade SBOM is not achievable with the evaluated tools”—not because the tools are buggy, but because there is no shared definition of what a *component* is across tools and languages. In multi-agent LLM topologies, [arXiv:2606.01490](#) finds that parallel merge architectures are “fundamentally broken” due to token starvation and what the authors call the “Frankenstein effect”—outputs assembled from incompatible partial designs. In decompilation evaluation, [arXiv:2605.29490](#) finds that readability-maximizing settings do not maximize functionality, and that cross-decompiler variation at the functional level is  $20\times$  the cross-LLM variation—suggesting that the abstraction the evaluation framework uses (syntactic similarity) is mismatched to the property that matters (behavioral reusability).

The pattern is not coincidental. Each failure involves a tool or framework operating on a representation that does not faithfully encode the property it is supposed to measure or enforce. This is an abstraction boundary failure: the interface between representation and reality is leaky in a way that the tooling cannot detect from within its own frame.

The cs.PL result from [arXiv:2605.21337](#) provides a formal grounding for *why* abstraction boundaries fail in the specific case of effectful computation: simultaneous substitution of computations is not canonical when evaluation order is semantically meaningful. This is not an analogy to the software engineering findings—it is a different domain—but the structural shape of the problem (a standard compositional operation fails because the underlying semantics does not support the assumed commutativity) appears in both settings. We argue this analogy is worth investigating, while being explicit that it is a heuristic bridge, not a derivation.

---

## Background

### 2.1 Abstraction Layers and Their Failure Modes

Software engineering has long recognized that abstraction layers can leak—that is, details from lower layers become visible to higher-layer consumers in ways the abstraction was supposed to prevent. What is less studied is the *monitoring* version of this problem: when the abstraction layer is used not to build functionality but to *observe* it, leakage can cause failures to become invisible rather than visible. [arXiv:2606.02494](#) calls this “structural defects masking task-level signal,” and it is a qualitatively different failure mode from the classic leaky abstraction, because the monitoring system cannot self-report its own blindness.

### 2.2 Component Identity and the Composition Problem

A prerequisite for any compositional reliability argument is a stable notion of what constitutes a component. [arXiv:2606.02442](#) systematically examines this prerequisite for software supply-chain security and finds it unmet: different SBOM tools apply different Component Inclusion Mechanisms, leading to ambiguity and blind spots that are not tool bugs but definitional gaps. The paper

proposes a “ground-up analysis” of CIMs as the necessary prior work before tooling can be improved. This is precisely the claim that the abstraction (component identity) is underspecified at the level where it needs to be precise.

## 2.3 Evaluation Metric Mismatch as Abstraction Failure

A third mechanism by which abstraction boundaries fail is metric mismatch: the property being measured is a proxy for the property that matters, and the proxy diverges from the target in systematic ways. [arXiv:2605.29490](#) demonstrates this concretely: syntactic similarity metrics used in decompilation evaluation do not predict behavioral reusability. The paper proposes a three-axis evaluation paradigm (readability, recompilability, functionality) precisely because no single metric captures the target property. This is an abstraction boundary failure in the evaluation pipeline itself.

---

## Synthesis

### 3.1 Claim: Structural Monitoring Scope Is a Prerequisite, Not an Add-On

**Claim.** In partially integrated agentic systems, task-level monitoring is not merely insufficient—it is actively misleading, because structural defects suppress the variance that task-level monitors require to function.

**Evidence.** [arXiv:2606.02494](#) reports three distinct monitoring scopes with measurably different failure-detection profiles. Within-run monitors surface deterministic stage defects ( $CV = 0.02$ ). Cross-run monitors surface stochastic integration consequences ( $CV = 1.25$ , with 24% of findings at severity level L2). A structural monitor identifies an integration gap with perfect consistency ( $CV = 0.00$ ). Critically, injected task-level errors are indistinguishable from clean baselines—the abstract states this directly, confirming that structural defects mask task-level signal. The paper also reports that deterministic triage routes 97% of findings to automated tracking, leaving the 2% reflecting variable behavior for human investigation.

**Caveat.** This is Stage 1 evidence from a synthetic testbed of 220 runs across 120 document bundles with controlled error injection. Whether the finding generalizes to production agentic systems with real tasks and non-injected errors is not established by the abstract. The maturity-staging model proposed is described as “Stage 1 evidence,” not a validated framework.

**Falsification path.** Run the same monitoring methodology on a production agentic system with known task-level failure rates. If task-level monitors achieve detection rates statistically indistinguishable from structural monitors when the system is not in a partially-integrated state, the claim that structural monitoring is a *prerequisite* (rather than a complement) is falsified. Specifically: if  $CV$  for task-level monitors drops to the range of structural monitors (near 0.00) in a mature system, the maturity-staging model holds; if it does not, the claim overgeneralizes.

---

### 3.2 Claim: Component Definition Gaps Are a Systemic, Not Tooling, Problem in Supply-Chain Security

**Claim.** SBOM generation failures are not primarily engineering defects in individual tools; they reflect the absence of a shared formal definition of “component” that is precise enough to be

operationalized consistently across programming languages and build systems.

**Evidence.** [arXiv:2606.02442](#) evaluates four popular SBOM generation tools (cdxgen, syft, trivy, ORT, and Microsoft sbom-tool) against a ground truth across Python, Java, Go, PHP, Rust, and C. The paper finds that “no tool covers all identified CIMs” and that “common gaps exist across tools.” The abstract explicitly states: “Without a shared understanding of what a component is, any effort to secure software supply chains remains fundamentally limited.” The proposed remedy is not to improve the tools but to “go back to the drawing board to clarify which components should be included in an SBOM.”

**Caveat.** The abstract does not report quantitative gap sizes per language or per tool, so the relative severity of the problem across the six languages cannot be assessed from the abstract alone. The abstract reports that “common gaps exist across tools” but does not specify whether those gaps are concentrated in particular CIM categories or distributed uniformly. Whether the abstract extends to other languages or build systems is open.

**Falsification path.** Produce a formal specification of component identity that resolves all identified CIM ambiguities, implement it in a reference tool, and re-run the ground-truth evaluation. If the reference tool still exhibits gaps, the claim that the problem is definitional (not tooling) is weakened. If the reference tool achieves security-grade SBOM generation and existing tools cannot be updated to match it without architectural changes, the claim is supported.

---

### 3.3 Claim: Composition Topology Determines Failure Mode in Multi-Agent LLM Systems, and Merge Architectures Fail Structurally

**Claim.** The failure of parallel merge topologies in multi-agent LLM systems is not a parameter-tuning problem but a structural consequence of asking a language model to perform coherent synthesis on inputs that were generated without shared context.

**Evidence.** [arXiv:2606.01490](#) reports a controlled experiment across 12 collaboration topologies, 520 runs, and 8 design tasks. Parallel merge variants are placed in the bottom tier by all three independent evaluators (weighted ensemble scores 3.65–3.79 out of 5.0), attributed to “token starvation and the Frankenstein effect.” The paper also finds that structural adversarial topology (v4b) ranks first (4.637/5.0) and cross-model review ranks second (4.606/5.0). The evaluator disagreement on v2b (Claude d=1.44 vs. GPT-OSS d=0.45) is itself reported as a finding, suggesting that different model families weight design qualities differently.

**Caveat.** The evaluation uses automated evaluators (GPT-OSS 120B, Claude Opus 4.6, Claude Sonnet 4.6), not human expert judges. Whether automated evaluator agreement on topology rankings reflects genuine design quality or shared model biases is not resolved by the abstract. The 12-dimensional rubric is described but not reproduced in the abstract, so the validity of the rubric cannot be assessed here. The experiment covers software architecture design tasks specifically; generalization to other task types is open.

**Falsification path.** Run the same topology experiment with human expert evaluators on the same 8 design tasks. If human evaluators rank merge variants above the bottom tier, or if the ranking order differs substantially from the automated evaluator consensus, the claim that merge failure is structural (rather than an artifact of automated evaluation bias) is weakened. If human and automated rankings agree on the bottom-tier placement of merge variants, the structural claim is

supported.

---

### 3.4 Claim: Evaluation Metric Mismatch Is a Structural Defect in Decompilation Pipelines, Not a Calibration Problem

**Claim.** Decompilation evaluation frameworks that optimize for readability systematically misdirect engineering effort because readability and functionality are not monotonically related—and the divergence is large enough to matter in practice.

**Evidence.** [arXiv:2605.29490](#) proposes a three-axis evaluation paradigm (readability, recompileability, functionality) and applies it via DEBENCH, a framework with 240 atomic test functions compiled into 640 binaries. Key findings: the best decompiler-LLM pair reaches 22.3% exact+partial program-level behavioral overlap but only 1.2% exact stdout match, nearly 50 points below recompileability. Compiler flag `-O3` yields the lowest readability but the highest functionality. Clang gives lower readability than GCC but  $2.6\times$  higher functionality. Cross-decompiler variation at the functional level is  $20\times$ , far larger than the  $1.6\times$  cross-LLM variation.

**Caveat.** DEBENCH contains 240 atomic test functions; whether these functions are representative of real-world decompilation targets is not established in the abstract. The 50-iteration compile-and-repair budget is a fixed constraint that may favor or disfavor particular decompiler-LLM combinations in ways that are not representative of unconstrained use. The abstract reports that “settings that maximize readability do not maximize functionality” but does not report a correlation coefficient between the two metrics, so the strength of the anti-correlation is not quantifiable from the abstract alone.

**Falsification path.** Construct a decompilation benchmark where human engineers rate both readability and functional reusability independently. If the human ratings show a positive correlation between readability and functionality (contradicting the `-O3` and Clang findings), the metric mismatch claim is weakened. If the correlation is near zero or negative, the claim is supported. The test is specific: compute Spearman rank correlation between human readability ratings and functional test pass rates across the same binary set.

---

### 3.5 Claim: The Formal Semantics of Effectful Composition Explains Why Abstraction Boundaries Fail in Untyped Computational Settings

**Claim.** The failure of standard compositional abstractions in untyped effectful computation has a precise formal cause: simultaneous substitution of computations is not canonical when evaluation order is semantically meaningful, requiring a different categorical primitive (sequential binding steps) rather than a patch to existing frameworks.

**Evidence.** [arXiv:2605.21337](#) addresses the obstacle that “there is no canonical notion of simultaneous substitution of computations, since evaluation order is semantically meaningful” in untyped effectful Call-by-Value languages. The paper introduces Freyd operads, separating a cartesian operad of values from a symmetric Ren-cartesian preoperad of computations, connected by a Freyd functor. The construction is proven representable, and soundness, initiality, and completeness are established for untyped computational lambda-calculus with procedures and higher-order functions.

**Caveat.** This result is in categorical semantics, not in operational software engineering. The connection to the software engineering findings in this synthesis is a heuristic analogy: both settings involve a standard compositional operation (simultaneous substitution / parallel merge / component inclusion) that fails because the underlying semantics does not support the assumed commutativity or shared context. The formal result does not *explain* the software engineering failures; it provides a structural shape that is analogous. This is the weakest bridge in the synthesis and is flagged explicitly in the Weakly-Connected Addendum below.

**Falsification path.** The formal result is already proven within its domain—soundness, initiality, and completeness are established. The falsification path for the *analogy* is different: if a formal model of SBOM component inclusion or agentic monitoring scope can be constructed that does *not* require sequential primitives (i.e., where parallel composition is well-defined without evaluation-order constraints), the analogy to [arXiv:2605.21337](#) collapses. Constructing such a model and showing it handles real CIM ambiguities would falsify the bridge.

---

## Discussion

### What This Implies

The synthesis suggests a diagnostic heuristic: when a reliability tool fails to detect known failures, the first question to ask is whether the tool’s internal abstraction of “component,” “task,” or “behavior” is actually instantiated by the system it is monitoring. [arXiv:2606.02494](#) operationalizes this as monitoring scope selection. [arXiv:2606.02442](#) operationalizes it as CIM ground-truth analysis. [arXiv:2605.29490](#) operationalizes it as multi-axis evaluation. The common move is to descend one level of abstraction and ask: what does the layer below actually provide?

The multi-agent topology finding [arXiv:2606.01490](#) adds a design-time implication: if the composition mechanism (parallel merge) does not support the synthesis operation being requested, no amount of prompt engineering or model scaling will fix it. The structural adversarial topology works precisely because it forces sequential, contested refinement rather than parallel independent generation—a design that matches the actual compositional semantics of language model generation.

The credential detection work [arXiv:2605.31520](#), while not a primary pillar of this synthesis, supports the broader pattern: the binary classification abstraction (secret vs. not-secret) fails because it does not model the intermediate class (placeholder/weak credential) that the underlying distribution actually contains. Introducing a three-class framework reduces high-severity alerts by 33% without sacrificing recall, a measurable payoff from abstraction refinement.

Similarly, [arXiv:2605.31389](#) (Neuroforger) and [arXiv:2605.27328](#) (governed runtime evolution) each address abstraction gaps in their respective domains—formal specification languages for smart contracts and lifecycle governance for agent-generated artifacts—but connect to the thesis through domain-specific mechanisms rather than the same structural pattern, and are noted as peripheral.

### What This Does NOT Imply

This synthesis does not imply that abstraction refinement is always sufficient to solve reliability problems. [arXiv:2605.29490](#) shows that even with a three-axis evaluation framework, the best decompiler-LLM pair reaches only 22.3% behavioral overlap—the abstraction improvement clarifies



the problem but does not solve it. Similarly, [arXiv:2606.02442](#) explicitly states that clarifying component definitions is necessary but not sufficient: tooling must also be revised.

The synthesis also does not imply that task-level monitoring is useless. [arXiv:2606.02494](#) proposes a maturity-staging model in which monitoring *transitions* from structural to task-level as systems mature—the claim is about sequencing and prerequisites, not about permanent irrelevance of task-level monitors.

## Weakly-Connected Addendum

The connection between [arXiv:2605.21337](#) (multicategorical semantics for untyped effects) and the software engineering findings is the weakest link in this synthesis. The formal result establishes that sequential binding steps are the correct primitive for effectful computation when evaluation order matters. The analogy to software engineering settings—where parallel composition fails for similar structural reasons—is argued by structural shape, not by mechanism. No claim is made that the categorical result *explains* or *predicts* the software engineering failures. It is included because the structural pattern (a standard compositional operation requires a non-standard primitive when the underlying semantics breaks commutativity) appears to recur, and the formal case provides a precise statement of what that pattern looks like when it is fully specified. Readers should treat this connection as a hypothesis for future investigation, not a derived result.

---

## Limitations

**Abstract-only reading.** This synthesis is based entirely on abstracts, not full texts. Quantitative claims (CV values, F1 scores, ensemble rankings) are taken directly from abstracts and may not represent the full nuance of the reported results. Whether findings extend beyond the specific experimental setups described is open in every case.

**Small corpus.** The synthesis draws on 6–7 papers from a 10-paper corpus. Selection was based on thematic coherence with the abstraction boundary thesis; papers that did not fit (EdgeFlow [arXiv:2605.27332](#), HE<sup>2</sup> [arXiv:2605.31004](#)) were not included as primary pillars. This is a selection decision, not a quality judgment.

**Heuristic reading, not derivation.** The thesis that “abstraction boundary failures dominate reliability bottlenecks” is a heuristic reading of independently conducted studies, not a derivation from a shared formal framework. The studies do not cite each other, were not designed to test a common hypothesis, and use different methodologies, domains, and evaluation criteria. The structural pattern identified here is a candidate reading, not an established result.

**Preprint status.** All cited papers are preprints without peer review. Experimental results, especially the synthetic testbed findings in [arXiv:2606.02494](#) and the automated-evaluator rankings in [arXiv:2606.01490](#), have not been independently replicated.

**Generalization limits.** Each paper explicitly scopes its findings: [arXiv:2606.02494](#) to Stage 1 evidence on a synthetic testbed; [arXiv:2606.02442](#) to four specific tools across six languages; [arXiv:2606.01490](#) to software architecture design tasks; [arXiv:2605.29490](#) to 240 atomic test functions. Whether the findings generalize beyond these scopes is not established.

## Selection Process

The candidate pool was the full 10-paper corpus provided. Papers were filtered for thematic relevance to reliability, monitoring, evaluation, and formal semantics. EdgeFlow [arXiv:2605.27332](#) (VLM-based flowchart processing) and HE<sup>2</sup> [arXiv:2605.31004](#) (FHE accelerator architecture) were considered and dropped: EdgeFlow’s findings are specific to topology-critical visual processing and do not connect to the abstraction boundary thesis through a named mechanism; HE<sup>2</sup> addresses hardware-level communication bottlenecks in a domain (homomorphic encryption acceleration) that does not share the compositional failure pattern. Neuroforger [arXiv:2605.31389](#) and governed runtime evolution [arXiv:2605.27328](#) are peripheral—they address abstraction gaps in smart contract specification and agent lifecycle governance respectively, but their findings do not provide the same evidential weight as the primary five. Credential detection [arXiv:2605.31520](#) is included as supporting evidence for the three-class abstraction refinement pattern.

---

## Conclusion

Across monitoring methodology, supply-chain security, multi-agent system design, decompilation evaluation, and categorical semantics for effectful computation, a common structural pattern emerges: reliability tools and frameworks fail when their internal abstraction of the system’s components, behaviors, or composition operations does not match what the underlying system actually provides. This is a heuristic reading of independently conducted studies, not a derived result, and each finding carries the caveats of its specific experimental scope and preprint status. The practical implication is narrow but concrete: before deploying a monitoring or evaluation framework, verify that the abstraction it operates on is actually instantiated by the system—not as a best practice, but as a measurable precondition whose absence produces the specific failure modes documented here.

---