Article

Not peer-reviewed version

SORT-AI: Agentic System Stability in Large-Scale AI Systems Structural Causes of Cost, Instability, and Non-Determinism in Multi-Agent and Tool-Using Workflows

Gregor Herbert Wegener *

Posted Date: 22 January 2026

doi: 10.20944/preprints202601.1741.v1

Keywords: agentic systems; multi-agent coordination; tool-calling pipelines; agentic stability; structural analysis; semantic coupling; intent coherence; non-determinism; operator-based diagnostics; emergent system behavior; planning drift; autonomous AI systems; ghost costs; auditability

 doi:10.20944/preprints202601.1741.v1

Article

SORT-AI: Agentic System Stability in Large-Scale AI Systems Structural Causes of Cost, Instability, and Non-Determinism in Multi-Agent and Tool-Using Workflows

Gregor Herbert Wegener

Friedrichstrasse 4, 10969 Berlin, Germany; gregor.wegener@gmail.com; Tel.: +49 179 2544522

Abstract

The deployment of agentic AI systems—multi-agent orchestrations, tool-calling pipelines, and autonomous planning architectures—introduces operational instabilities that cannot be attributed to interconnect limitations or runtime control conflicts alone. Even in systems with adequate infrastructure and coherent control planes, cost escalation, non-deterministic behavior, and soft degradation persist, pointing toward semantic coupling as a distinct failure domain. This article argues that agentic system stability—the degree to which autonomous agent decisions across planning, tool selection, execution, and verification layers remain mutually consistent with respect to shared intent—constitutes a structural property whose loss gives rise to economically significant inefficiencies. Classical metrics fail to capture stability loss because conflicts between agentic layers are semantically distributed, emergent, and do not manifest as discrete faults. The contribution of this work is a structural problem analysis that positions agentic incoherence as a first-order economic and operational variable, complementing prior analyses of interconnect-induced instability and control plane incoherence. The methodology is deliberately conceptual, avoiding implementation details, framework evaluations, or prescriptive solutions.

Keywords: agentic systems; multi-agent coordination; tool-calling pipelines; agentic stability; structural analysis; semantic coupling; intent coherence; non-determinism; operator-based diagnostics; emergent system behavior; planning drift; autonomous AI systems; ghost costs; auditability

1. Introduction

1.1. The Stability Problem in Agentic AI Systems

The deployment paradigm of contemporary AI systems is shifting from isolated, model-centric inference toward agentic execution environments in which large language models operate as autonomous decision-making components [18,19]. In these settings, models are embedded within extended workflows that encompass goal decomposition, iterative planning, tool invocation, verification, and, increasingly, coordination across multiple agents. System behavior is therefore determined not by a single inference step, but by the cumulative interaction of many locally rational decisions whose effects persist across execution boundaries.

Within this regime, performance degradation and cost escalation can no longer be attributed primarily to model quality or infrastructure limitations. Classical evaluation metrics such as accuracy, latency, or perplexity capture properties of individual inference steps, yet remain insensitive to structural inefficiencies that arise from chained autonomous decisions [74]. Empirical studies of agentic architectures indicate that recursive planning loops, adaptive tool selection, and inter-agent interaction can generate emergent inefficiencies even when each component operates nominally in isolation [12,13]. These effects typically manifest as soft degradation patterns, including increased

 doi:10.20944/preprints202601.1741.v1

variance, non-deterministic execution paths, and disproportionate resource consumption without corresponding gains in task completion.

Multi-agent systems have long been recognized as a distinct system class with characteristic coordination challenges [6,7,9]. The incorporation of large language models as adaptive planners and controllers, however, introduces a qualitatively new instability surface [17]. Unlike traditional distributed systems, where instability is commonly rooted in physical coupling or control-plane conflicts, agentic AI systems exhibit failures driven by semantic coupling at the level of intent, planning, and action selection. This work argues that instability emerging from chained autonomous decisions constitutes a structurally distinct failure mode, complementary to interconnect-induced instability [4] and runtime control incoherence [5], yet irreducible to either.

1.2. Scope, Limitations, and Agentic Boundary Definition

The scope of this work is deliberately constrained to a structural analysis of instability in agentic AI systems. The objective is to identify and formalize the mechanisms through which economically and operationally significant failure modes arise, rather than to propose algorithms, architectural prescriptions, or implementation-level remedies. In line with the broader SORT framework, the analysis is diagnostic and risk-oriented, aiming to clarify system-level properties that are otherwise obscured by component-centric evaluation.

The agentic boundary adopted here separates structural phenomena from model-level and implementation-specific concerns. Within this boundary, the analysis focuses on how intent is generated, propagated, and transformed across planning steps, tool invocations, and agent interactions over extended execution horizons. Outside this boundary lie questions of model internals, prompt construction, and alignment methodologies, which, while important in their own right, address different layers of the system stack [70]. Maintaining this separation avoids conceptual conflation and ensures that agentic system stability is treated as a problem of semantic coupling rather than of model optimization or behavioral tuning [8]. This analysis concerns structural system stability rather than ethical alignment, value alignment, reward hacking, or AGI safety research.

1.3. System Class and Failure Envelope

The analysis applies to a broad class of agentic AI systems in which execution behavior is governed by autonomous planning and action selection rather than by fixed inference pipelines. This includes multi-agent orchestration frameworks [6,10,14], tool-calling inference pipelines [15,20], autonomous planning and execution systems [16,27], retrieval-augmented generation workflows with agentic control logic [3,23], as well as hybrid architectures that incorporate human-in-the-loop supervision [60].

Across these system classes, instability manifests within a characteristic failure envelope that is poorly captured by conventional fault models. Typical patterns include retry cascades that do not surface as explicit errors, gradual planning drift across iterative decision cycles, amplification of tool invocations whose outputs are only partially utilized, divergence of intent across cooperating agents, and progressive saturation of shared contextual resources [28]. Importantly, these effects often remain operationally invisible: systems continue to produce outputs and satisfy local success criteria even as cost, variance, and irreproducibility increase. The identification of this failure envelope provides the foundation for treating agentic system stability as a first-order structural and economic concern rather than as a secondary implementation issue.

2. The Shift from Control-Dominated Failures to Intent-Dominated Failures

2.1. Traditional Assumptions in Agent Design

Early agent-based systems and contemporary large language model integrations share a set of implicit design assumptions inherited from classical software and distributed systems engineering [11, 63]. Individual tool calls are typically treated as atomic and reliable operations whose effects are

 doi:10.20944/preprints202601.1741.v1

confined to their immediate outputs. Planning steps are assumed to be locally optimal and largely independent, such that errors or inefficiencies do not propagate beyond their immediate context. Finally, retry logic is commonly regarded as a sufficient mechanism for handling transient failures, under the assumption that repeated execution converges toward a correct or stable outcome [7,12].

These assumptions hold reasonably well in settings where agents execute short, stateless tasks with limited side effects. However, as agentic systems evolve toward longer execution horizons, richer tool ecosystems, and tighter coupling between planning and execution, their limitations become increasingly apparent [21]. Tool calls may alter shared state, consume irreversible resources, or influence the informational context available to subsequent planning steps. Planning steps are no longer independent, but form chains in which early decisions constrain or bias later reasoning. Under these conditions, retry mechanisms can amplify rather than resolve inefficiencies, leading to cumulative resource consumption without proportional gains in task progress. The failure of these traditional assumptions marks the transition from control-dominated to intent-dominated instability.

2.2. Agentic Execution as an Active System Component

In agentic AI systems, execution cannot be understood as passive model inference embedded within an otherwise stable control structure. Instead, agentic execution constitutes an active system component whose decisions shape global system behavior based on incomplete and locally scoped information [13,22]. Agents act as autonomous decision-makers that select actions, invoke tools, and revise plans in response to intermediate outcomes, thereby introducing feedback loops that extend beyond the boundaries of individual inference steps.

Within this paradigm, tool calls function as actions with side effects rather than as pure queries [24]. Each invocation may modify external state, alter resource availability, or influence the informational context available to subsequent planning steps. Planning itself becomes an iterative and adaptive process, in which feedback from execution informs future decisions and can either dampen or amplify initial conditions [30]. As a result, instability emerges not from isolated errors, but from the dynamic interaction between planning, execution, and feedback. This shift renders many classical control-oriented diagnostics insufficient, as they fail to account for the semantic consequences of agentic decision-making.

2.3. The Agentic Stack in Practice

In practice, contemporary agentic systems are organized as layered stacks that mediate between high-level intent and concrete execution [79]. At the top of this stack resides the planning layer, responsible for goal decomposition and task sequencing. Beneath it, the tool selection layer maps abstract intentions to concrete capabilities through routing and matching mechanisms. The execution layer carries out selected actions by invoking tools and parsing their results, while the verification layer evaluates outputs and determines whether additional action is required. Finally, the supervision layer introduces policy enforcement and, in some cases, human-in-the-loop intervention.

Each layer operates under its own local objectives and constraints, yet their interaction surfaces introduce nontrivial coupling [48]. Decisions made at the planning layer influence tool selection and execution paths, while verification outcomes feed back into subsequent planning cycles. Supervision mechanisms may override or redirect execution in ways that were not anticipated by earlier decisions. These interaction surfaces represent potential conflict zones in which intent can drift, retries can accumulate, and coordination can break down [14]. Importantly, such conflicts often do not manifest as explicit failures within any single layer, but instead emerge from their combined dynamics.

Figure 1 situates agentic system stability within the broader series of structural instability domains addressed by the SORT framework. While physical coupling failures arise from synchronization constraints and logical coupling failures from control-plane incoherence, agentic failures originate at the semantic level, where intent is generated, transformed, and propagated across autonomous decision-making components. This distinction motivates a shift in analytical focus from control correctness to intent coherence, which forms the basis for the subsequent sections.

 doi:10.20944/preprints202601.1741.v1

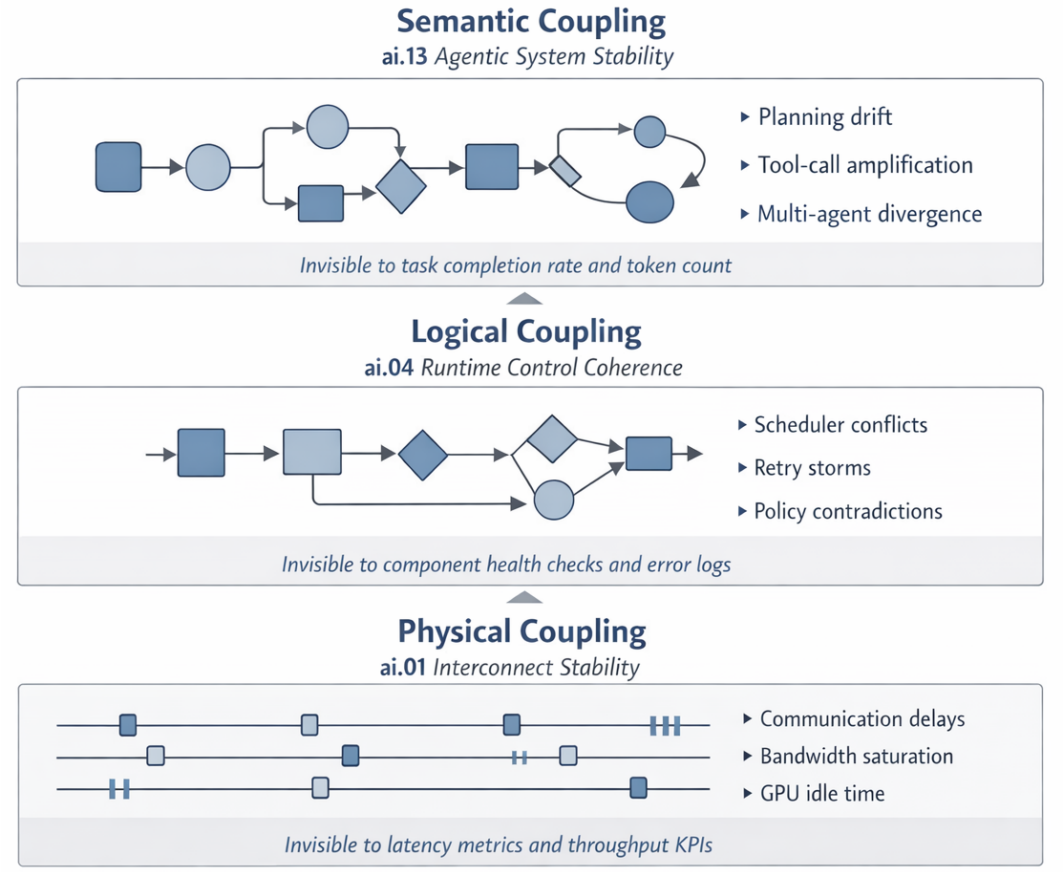

Figure 1. Structural failure domains across physical, logical, and semantic coupling layers. Each layer represents a distinct instability domain that persists even when adjacent layers are adequately addressed. The progression from ai.01 (Interconnect Stability) through ai.04 (Runtime Control Coherence) to ai.13 (Agentic System Stability) reflects increasing abstraction from hardware synchronization to coordination logic and, finally, to intent propagation. Each domain exhibits characteristic failure patterns that remain largely invisible to the metrics traditionally used to monitor the other layers.

3. Why Classical Metrics Fail

3.1. The Measurement Gap in Agentic Systems

Conventional performance metrics employed in the evaluation of AI systems are primarily designed to assess localized properties of computation, such as throughput, latency, or resource consumption at the level of individual components [75]. In agentic systems, these metrics are commonly instantiated as task completion rates, token counts, and end-to-end latency measurements. While such indicators are informative for characterizing isolated inference behavior, they are structurally incapable of capturing inefficiencies that arise from the interaction of autonomous decision-making processes [47].

Task completion rate measures whether a system eventually produces an output deemed acceptable under predefined criteria, but it provides no insight into the internal efficiency of the execution path that led to this outcome [73]. Token count quantifies linguistic activity rather than semantic progress, and may increase monotonically even as effective advancement toward the original goal stagnates. Latency metrics, similarly, aggregate execution time without distinguishing between productive computation and overhead introduced by retries, replanning, or redundant tool invocations. From an end-to-end systems perspective [44], these metrics are inherently local and therefore blind to globally emergent inefficiencies that unfold across extended decision chains.

 doi:10.20944/preprints202601.1741.v1

3.2. The Invisibility of Agentic Incoherence

A defining characteristic of agentic system instability is that it generates economic and operational cost without producing explicit error states. Unlike classical failure modes, in which faults manifest as crashes, exceptions, or violated invariants, agentic incoherence often unfolds under nominal execution conditions [49]. Tool calls may succeed syntactically and semantically, agents may continue to produce outputs, and supervisory checks may report healthy system status, even as overall efficiency degrades.

This invisibility arises because incoherence is not tied to the correctness of individual actions, but to their collective interaction over time [53]. Successful tool invocations can contribute to redundant or contradictory state changes, multi-agent conflicts can be resolved implicitly through overwriting or compensation rather than explicit failure, and planning drift can accumulate gradually across iterations. As a result, the system expends increasing resources to maintain apparent progress, while traditional monitoring frameworks register no anomalies. The phenomenon is consistent with broader observations on the inherent complexity of software systems, where interactions between components give rise to behaviors that cannot be inferred from local correctness alone [45].

3.3. Why Intent Consistency Cannot Be Measured Locally

Intent consistency in agentic systems constitutes a fundamentally non-local property. It reflects the degree to which autonomous decisions, taken at different times and by different agents, remain aligned with an evolving global objective. By construction, such alignment cannot be verified through local inspection of individual actions or decisions [33]. An agent may act correctly with respect to its immediate inputs and policies, yet still contribute to a trajectory that diverges from the original intent when considered at the system level.

This distinguishes agentic incoherence from classical distributed systems consistency problems, which focus on the synchronization of data state across replicas or processes [46,65]. In those settings, consistency can often be formalized and monitored through well-defined invariants. By contrast, intent is a semantic construct that is transformed through planning, interpretation, and execution. Its coherence depends on the cumulative effect of decisions rather than on the correctness of any single step. Consequently, local metrics and checks are structurally incapable of assessing whether intent remains coherent across an agentic execution horizon, rendering classical observability paradigms insufficient for diagnosing instability in such systems [34].

4. Agentic Incoherence as a Structural Phenomenon

4.1. Coherence Versus Incoherence in Agentic Systems

Agentic coherence denotes the degree to which autonomous decisions made by one or more agents remain mutually consistent with a shared or evolving intent across planning, execution, and verification boundaries. Coherence is therefore not a property of individual actions, but of their composition over time and across interacting components [39]. In coherent agentic systems, locally generated plans, selected tools, and verification outcomes collectively reinforce a stable interpretation of intent, even as the system adapts to new information or intermediate results [1,2].

Agentic incoherence arises when this consistency breaks down despite the local correctness of individual decisions [41]. A critical distinction must be drawn between intent alignment and action alignment. Intent alignment refers to the consistency of goals or objectives as represented within and across agents, while action alignment concerns the compatibility of concrete behaviors produced during execution. Systems may exhibit apparent intent alignment at a high level while suffering from action-level divergence, or conversely may execute aligned actions that gradually distort or dilute the original intent. In both cases, incoherence emerges as a structural incompatibility between decision layers rather than as a defect attributable to faulty logic, incorrect implementations, or erroneous model outputs.

Importantly, agentic incoherence should not be conflated with bugs or implementation errors. Bugs represent violations of expected behavior within a specified component, whereas incoherence

 doi:10.20944/preprints202601.1741.v1

reflects a mismatch between interacting components whose individual behaviors remain within specification. This distinction underlines the need to treat incoherence as a system-level phenomenon rooted in semantic coupling, rather than as a collection of localized faults.

4.2. Soft Degradation Versus Hard Failure

Classical fault models in computing systems emphasize hard failures, in which errors manifest through crashes, exceptions, or violated invariants that trigger explicit recovery mechanisms [49]. Agentic systems, by contrast, are particularly susceptible to soft degradation modes that erode efficiency and stability without crossing discrete failure thresholds [52]. These modes are characteristic of adaptive and self-organizing systems, where global behavior emerges from local interactions without centralized control [29].

In the context of agentic incoherence, soft degradation manifests in several recurring patterns. Retry amplification can occur when agents repeatedly invoke tools or replanning routines that succeed locally but fail to advance the overall objective [51]. Planning loops may persist indefinitely without triggering termination conditions, as each iteration appears locally justified. Token consumption may increase steadily without corresponding progress toward task completion, and shared context resources may become saturated through incremental accretion rather than explicit overflow. None of these phenomena necessarily produce error signals, yet each contributes to rising cost, increased variance, and diminished reproducibility.

These degradation modes are particularly insidious because they coexist with nominal system operation. From the perspective of traditional monitoring and health checks, the system remains functional, responsive, and productive. The absence of hard failure signals masks the underlying structural inefficiency, allowing incoherence to persist and accumulate over extended execution horizons.

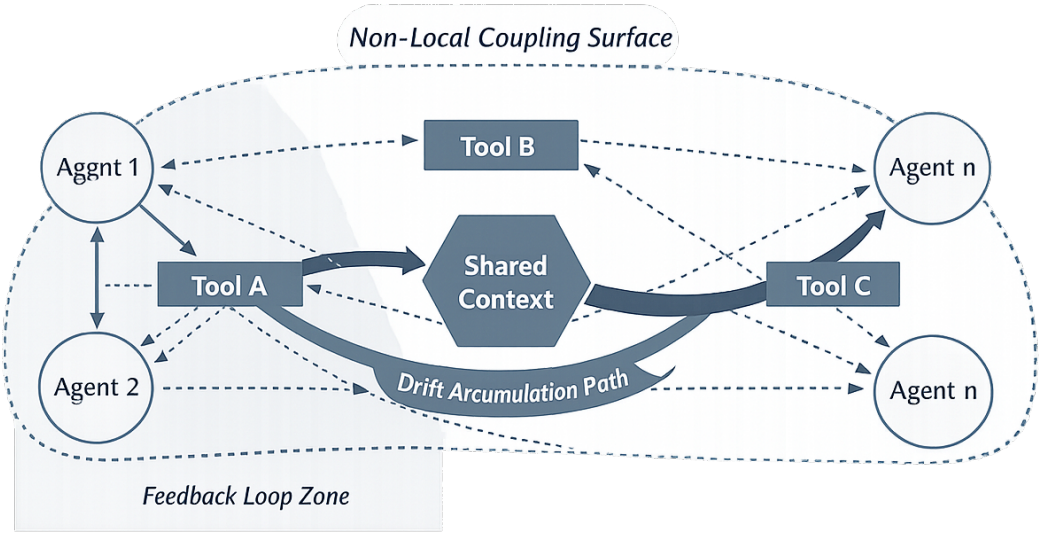

Figure 2. Abstract representation of agentic incoherence as a structural phenomenon. The diagram illustrates agent–tool interactions without reference to specific frameworks or implementations. Key structural features include feedback loops between planning and execution that can amplify initial conditions, non-local coupling surfaces where decisions by one agent affect the action space of others, the shared context as a contested resource with competing claims, and drift accumulation paths in which iterative planning progressively diverges from the original intent without triggering explicit failure conditions.

4.3. Non-Local Coupling and Emergent Effects

Agentic incoherence is fundamentally driven by non-local coupling effects, in which decisions made at one point in the system propagate across agents, tools, and shared resources in ways that cannot be localized to a single component [38]. Such coupling arises from the interaction of planning

 doi:10.20944/preprints202601.1741.v1

dependencies, tool-call side effects, and shared contextual state, producing feedback structures that span multiple layers of the agentic stack [30,37].

Within these structures, drift represents an emergent property rather than an isolated anomaly [42]. As agents adapt their plans based on intermediate outcomes, small deviations in interpretation or execution can accumulate across iterations, gradually shifting the effective intent of the system [31]. Tool-call dependencies that cross agent boundaries may introduce indirect constraints or incentives that were not anticipated during planning. Shared-state conflicts can arise when multiple agents compete for or overwrite contextual information, leading to implicit coordination failures. Feedback loops between planning and execution further entrench these effects, as each iteration reinforces the consequences of prior decisions.

The context window, whether represented as an explicit shared memory or as distributed state across agents, functions as a contested resource within this non-local coupling surface [43]. Competing claims from multiple planning cycles or agents can distort the informational basis for subsequent decisions, exacerbating incoherence without producing explicit signals of failure [35]. Taken together, these emergent effects underscore that agentic incoherence is not reducible to local misbehavior, but arises from the global structure of interactions that define agentic systems.

5. The Economic Dimension: Where Real Costs Arise

5.1. Direct Cost Effects

Agentic incoherence manifests economically long before it appears as technical failure [57]. In contrast to classical software systems, where resource consumption is tightly coupled to productive output, agentic systems exhibit a systematic decoupling between activity and value creation. As established in the economics of information technology [54], increased computational activity does not necessarily translate into proportional gains in productivity.

In agentic systems, direct cost effects arise primarily from structural inefficiencies rather than from explicit faults [58]. These include token consumption without proportional contribution to final outputs, redundant or contradictory API invocations across planning iterations, compute resources consumed by aborted execution paths, and latency amplification caused by unnecessary verification and retry loops. Importantly, these costs are incurred even when individual tasks terminate successfully, making them difficult to attribute using conventional cost models.

The economic signature of such systems is therefore not characterized by failure spikes, but by persistent baseline inflation of operational expenditure [56]. This inflation grows with agentic complexity, as additional planning depth, tool diversity, and coordination surfaces multiply the number of possible non-productive execution paths.

5.2. Hidden and Systemic Costs: Ghost Work in Agentic Systems

Beyond direct costs, agentic systems introduce a class of hidden and systemic expenditures that remain largely invisible to standard monitoring and accounting practices. We refer to these as *ghost costs*, borrowing from established notions of hidden infrastructure and operational overhead in large-scale computing systems [55].

Ghost costs comprise several structurally distinct components. *Ghost Tokens* denote consumed tokens that do not contribute to the final output, typically arising from discarded planning branches or overwritten intermediate states. *Ghost Planning* captures planning iterations that are executed, evaluated, and subsequently abandoned without external effect. *Ghost Tool-Calls* refer to API invocations whose results are either unused, contradicted by later decisions, or rendered obsolete by subsequent planning cycles. Finally, engineering effort expended on diagnosing and stabilizing non-reproducible agent behavior constitutes a substantial but often uncategorized cost component [59].

Figure 3 illustrates this phenomenon by contrasting visible system outputs with invisible resource consumption over time. While reported metrics such as task completion rate, latency, and token usage suggest stable operation, the underlying resource expenditure is dominated by ghost activity. The key

 doi:10.20944/preprints202601.1741.v1

insight is not merely that these costs exist, but that they scale superlinearly with agentic complexity, as feedback loops and non-local coupling surfaces multiply opportunities for non-productive execution.

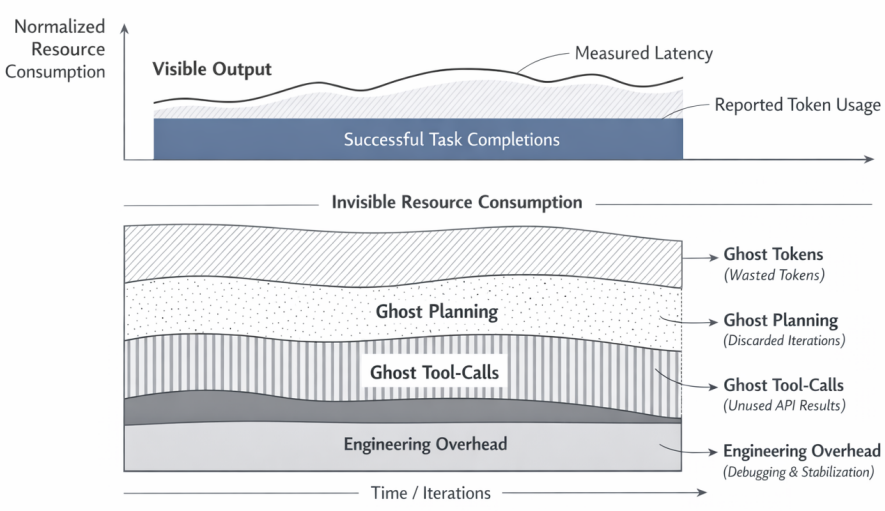

Figure 3. Layered view of visible output versus invisible resource consumption in agentic systems. The upper layer reflects conventionally measured outputs such as task completions, reported token usage, and latency. The lower layer exposes ghost costs, including Ghost Tokens, Ghost Planning, Ghost Tool-Calls, and engineering overhead. Although invisible to standard monitoring, the lower layer constitutes a substantial fraction of total system cost and grows with agentic complexity.

5.3. Measured Metrics Versus Actual System Cost

A central reason why ghost costs persist is the structural mismatch between what is measured and where costs actually arise [62]. Classical observability frameworks focus on metrics that are local, component-centric, and output-oriented. These metrics are well-suited for detecting hard failures, but poorly aligned with the distributed and iterative nature of agentic execution.

Figure 4 makes this discrepancy explicit by juxtaposing measured metrics with actual system cost. On the measured side, latency trends, reported token usage, and successful task completions suggest incremental and manageable growth. On the cost side, however, ghost activity accumulates across iterations, accompanied by rising engineering overhead required to stabilize, debug, and audit system behavior.

Crucially, the dashed boundary between these views represents not a tooling gap, but a conceptual blind spot. The costs incurred by ghost planning loops and non-local tool interactions are not missing data points; they are structurally unobservable within metric frameworks that assume atomic actions and local causality.

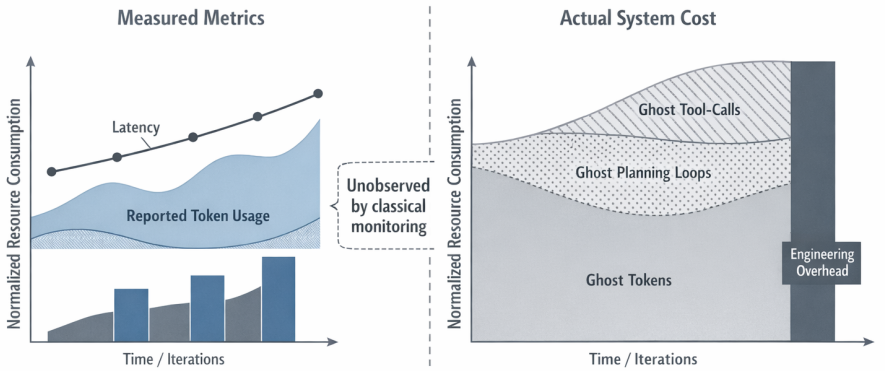

Figure 4. Divergence between measured metrics and actual system cost in agentic systems. While classical monitoring captures latency, reported token usage, and task success, substantial resource consumption remains unobserved. Ghost Tokens, Ghost Planning Loops, Ghost Tool-Calls, and engineering overhead dominate total cost over time, explaining persistent cost escalation despite apparently stable performance metrics.

 doi:10.20944/preprints202601.1741.v1

5.4. Irreproducibility and Audit Risk

In addition to direct and hidden costs, agentic incoherence introduces significant economic risk through irreproducibility [71]. Non-deterministic execution paths, context-dependent planning decisions, and feedback-driven drift make it difficult or impossible to reconstruct decision chains post hoc. This poses substantial challenges for compliance, accountability, and regulatory auditability [66,67].

From an economic perspective, irreproducibility functions as a risk multiplier [69]. Engineering time must be allocated not only to system operation, but to forensic reconstruction of agent behavior, often without reliable logs or stable execution traces. In regulated environments, the inability to demonstrate why a particular autonomous decision was taken translates directly into financial, legal, and reputational exposure.

Taken together, these factors establish agentic incoherence as an economic phenomenon rather than a purely technical one. The primary cost drivers are not model errors or infrastructure failures, but structurally invisible activity, non-local coupling, and the erosion of reproducibility at scale.

6. Why Scaling Is Not a Solution

6.1. More Agents, More Incoherence

A common response to performance and reliability limitations in agentic systems is to increase the number of agents under the assumption that parallelism improves capacity and robustness [72]. While this assumption holds for certain classes of distributed systems, it fails in the presence of semantic coupling and shared intent constraints.

Adding agents increases not only computational capacity but also the coordination surface of the system [40]. Each additional agent introduces new interaction paths, shared-state dependencies, and potential conflicts over goals, tools, and context [34]. In systems where coherence depends on the alignment of intent rather than on independent task execution, these additional interactions scale combinatorially rather than linearly [33].

As a result, increasing the agent count amplifies structural incoherence instead of mitigating it. Conflicting planning decisions propagate across agent boundaries, feedback loops become denser, and non-local coupling effects intensify. Empirically, this manifests as increased retry cascades, duplicated work, and divergent action trajectories despite unchanged or even improved local agent performance. These dynamics are characteristic of complex adaptive systems, where additional components increase system fragility unless higher-order coordination mechanisms are introduced [31].

6.2. Larger Context Windows, Deeper Drift

A second scaling strategy frequently proposed to address agentic instability is the expansion of context windows [78]. Larger contexts are assumed to improve coherence by preserving more historical information and reducing the need for repeated planning or tool invocation.

However, in agentic systems affected by structural incoherence, larger context windows primarily function as buffers for unresolved problems rather than as solutions. Extended context capacity allows planning drift, contradictory decisions, and redundant information to accumulate over longer horizons without triggering explicit failure signals. Instead of enforcing resolution, the system carries forward inconsistencies, embedding them deeper into subsequent decision cycles.

This effect parallels observations in large-scale systems where increased resource availability masks inefficiencies rather than eliminating them [55]. In agentic settings, the accumulation of unresolved intent conflicts within extended context windows increases the difficulty of later reconciliation. The longer the drift persists, the more entangled planning states become, and the higher the cost of restoring coherence.

From a structural perspective, increasing context length expands the state space in which incoherence can persist. The relationship between context capacity and drift accumulation resembles hidden order in adaptive systems [30], where additional degrees of freedom enable complex internal dynamics that remain invisible to external observation.

 doi:10.20944/preprints202601.1741.v1

6.3. More Tool Access, More Amplification Paths

A third scaling approach involves expanding the set of tools available to agents, under the assumption that broader capabilities improve problem-solving effectiveness [26]. While increased tool diversity can enhance expressiveness, it simultaneously introduces new amplification paths for instability.

Each additional tool represents a new branch in the agentic decision graph, increasing the number of possible execution paths, retries, and feedback loops [25]. As tool sets grow, the combinatorial complexity of possible interactions expands rapidly, outpacing the system's ability to maintain coherent intent across planning and execution layers [50].

In practice, this manifests as redundant tool invocations, contradictory actions across agents, and increased ghost activity as agents explore, abandon, and revisit alternative tool-based strategies. Crucially, the expansion of the tool space increases the space of potential incoherence faster than it increases the space of successful outcomes. Without structural mechanisms to constrain and diagnose these interactions, more tools amplify instability rather than resolve it.

Taken together, these observations demonstrate that scaling agentic systems by adding agents, extending context windows, or expanding tool access does not address the root causes of agentic incoherence. Instead, scaling magnifies structural deficiencies, leading to higher costs, deeper drift, and increased irreproducibility without commensurate gains in effective system capability.

7. Toward a Structural Perspective

7.1. The Need for System-Level Analysis

This section motivates the need for system-level analysis of agentic stability, arguing that agent decisions must be understood as structural relations within an operator graph rather than as isolated events [32,36]. Local agent optimization is insufficient to address stability problems that arise from cross-agent dependencies, feedback loops, and non-local coordination effects [64]. Consistent with end-to-end design principles [44], stability must be analyzed at the level where global behavior emerges, not at the level of individual components.

7.2. Operator-Based Approaches to Agentic Stability

Operator-based approaches provide a conceptual framework for reasoning about agentic stability in structurally complex systems [1,2]. Within this framework, agent decisions are modeled as projections onto subspaces of possible actions, tool calls as irreversible operators with side effects, and planning steps as state transitions within an evolving intent manifold. Representing agentic interactions as an operator graph enables explicit reasoning about coordination surfaces, intent drift, and stability conditions that remain invisible in purely event-based analyses.

7.2.1. Auditability and Traceability of Agentic Decisions

A central consequence of the operator-graph perspective is its relevance for auditability and traceability in agentic systems [3,66]. Conventional logging mechanisms capture only sparse, temporally ordered events, whereas the actual agentic decision process unfolds as a dense causal structure spanning multiple agents, tools, and planning alternatives.

Figure 5 illustrates this discrepancy. The left side depicts the full agentic decision chain, including branching alternatives, intent propagation across agent boundaries, and tool invocations with causal dependencies. The right side shows the observable log entries typically available to operators or auditors: isolated timestamps without semantic context or reconstructable causality. The region between these representations constitutes an *auditability gap*, representing structurally unrecoverable information such as unrecorded decision alternatives, invisible intent drift, and missing cross-agent dependencies.

This gap implies that accountability failures in agentic systems are not merely the result of insufficient logging but arise from a structural mismatch between how decisions are made and how

 doi:10.20944/preprints202601.1741.v1

evidence is recorded [68]. Operator-graph representations provide a foundation for closing this gap by enabling reconstruction of decision chains across agent boundaries and supporting governance, compliance, and accountability requirements for autonomous AI systems [67]. Without structural auditability, instability in agentic systems cannot be observed, attributed, or diagnosed—rendering it a prerequisite for governance and risk assessment rather than a control mechanism.

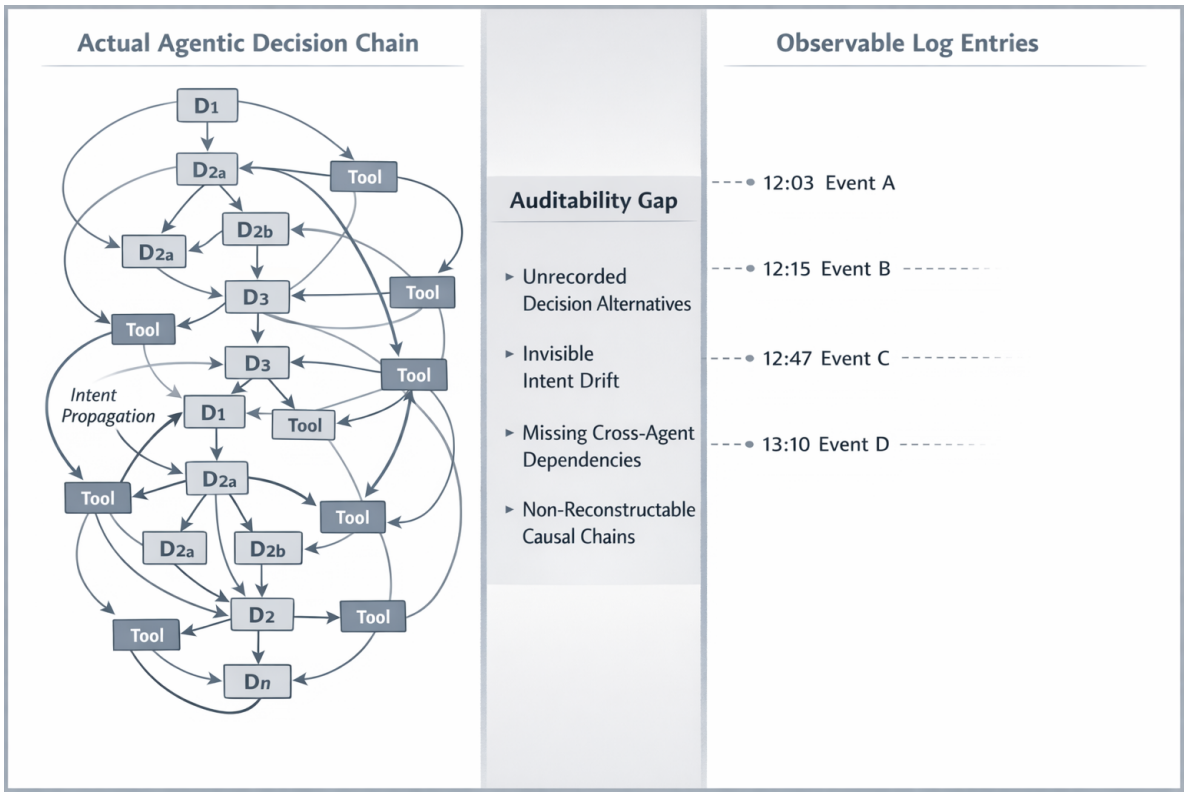

Figure 5. The auditability gap between agentic decision chains and observable log entries. The left side illustrates the actual causal structure of agentic decisions, including branching alternatives, tool invocations, and cross-agent dependencies. The right side shows what conventional logging captures: sparse, temporally ordered events without semantic context or causal links. The central gap represents structurally unrecoverable information—the decision rationale, considered alternatives, and intent propagation that cannot be reconstructed from logs alone. This gap has direct implications for compliance, accountability, and governance of autonomous AI systems.

8. Discussion

8.1. Implications for Architecture and Deployment Decisions

The analysis presented in this work has direct implications for architectural reasoning and deployment decisions involving agentic systems [61]. In particular, it reframes several commonly held assumptions about scalability, modularity, and observability in multi-agent architectures. The results suggest that architectural risk does not scale linearly with agent count or tool diversity but instead emerges from the structure of semantic coupling between agents, tools, and shared context.

From a design perspective, this implies that the decision to deploy multi-agent architectures cannot be evaluated solely on the basis of task decomposition, throughput gains, or functional separation. Instead, teams must consider whether the coordination surface introduced by additional agents exceeds the system's capacity to maintain coherent intent propagation. Similarly, tool integration patterns that appear benign in isolation may introduce hidden amplification paths when embedded in iterative planning loops or cross-agent delegation chains.

Importantly, these implications are not framed as prescriptive design rules but as diagnostic considerations. The structural perspective highlights that certain deployment choices inherently increase the space of potential incoherence, even when all local components operate correctly and no explicit error conditions are observed.

 doi:10.20944/preprints202601.1741.v1

8.2. Open Research Questions

The structural analysis of agentic stability raises several open research questions that are not addressed by existing metrics or evaluation frameworks [76]. A central question concerns the measurability of agentic coherence as a system-level property. Unlike task success or latency, coherence is not directly observable and may only be inferred through indirect structural signals.

A second open question relates to minimal instrumentation. What is the smallest set of observables required to detect intent drift or coordination breakdown without access to model internals or proprietary agent logic? Closely related is the question of how stability dynamics differ between single-agent systems with complex tool use and genuinely multi-agent configurations with shared or partially overlapping objectives.

Finally, the relationship between agentic stability and non-determinism remains insufficiently understood. To what extent does stochasticity exacerbate structural incoherence, and under which conditions can non-deterministic behavior be distinguished from genuine intent divergence? These questions point toward a research agenda focused on structural attribution rather than behavioral optimization.

8.3. Outlook on Empirical Validation

Empirical validation of agentic stability requires experimental designs that differ fundamentally from conventional benchmarking approaches [77]. Rather than measuring performance improvements, suitable testbeds must be capable of exposing divergence between visible outputs and underlying decision structure. This includes controlled variation of agent count, tool access, and planning depth while holding task objectives constant.

Crucially, such validation must remain feasible without access to model internals or proprietary implementations. The structural perspective adopted in this series therefore emphasizes black-box observables, reproducibility, and attribution at the level of interaction patterns rather than parameter inspection. Reproducible validation environments can serve as diagnostic instruments, enabling comparative analysis of structural stability across agentic configurations without conflating stability with task proficiency.

9. Conclusions

This work has argued that cost escalation and operational instability in agentic AI systems cannot be fully explained by infrastructure limitations or control-plane inefficiencies alone. Instead, a significant class of failures arises from loss of coherence at the semantic level, where autonomous agent decisions diverge from shared intent despite local correctness and apparent task success.

Within the SORT series, agentic system stability constitutes the third dimension of structural instability analysis in large-scale AI systems:

- **ai.01 Interconnect Stability**: Physical coupling, including hardware synchronization, communication fabric behavior, and data movement constraints.
- **ai.04 Runtime Control Coherence**: Logical coupling, encompassing schedulers, orchestrators, runtime engines, and policy enforcement mechanisms.
- **ai.13 Agentic System Stability**: Semantic coupling, involving intent propagation, planning dynamics, tool use, and multi-agent coordination.

Each dimension addresses a distinct failure domain that persists even when the others are adequately managed. Improvements in hardware efficiency or runtime orchestration do not eliminate semantic incoherence, just as robust semantic alignment cannot compensate for physical or control-plane instability.

The primary contribution of this analysis is therefore diagnostic rather than prescriptive. It provides a structural vocabulary for identifying, classifying, and reasoning about agentic instability without proposing mitigations, optimization techniques, or architectural remedies. By distinguishing

 doi:10.20944/preprints202601.1741.v1

semantic coupling as an independent source of instability, this work aims to support risk assessment, governance analysis, and system-level understanding of agentic AI deployments.

Data Availability Statement: The complete SORT v6 framework, including operator definitions, validation protocols, and reproducibility manifests, is archived on Zenodo under DOI 10.5281/zenodo.18094128. The MOCK v4 validation environment is archived at 10.5281/zenodo.18050207. Source code and supplementary materials are maintained at GitHub.

Acknowledgments: The author acknowledges the foundational architectural work developed in earlier SORT framework versions, which enabled the completion of the present architecture.

Conflicts of Interest: The author declares no conflicts of interest.

Use of Artificial Intelligence: The author confirms that no artificial intelligence tools were used in the generation of scientific concepts, theoretical frameworks, or results. Automated tools were used exclusively for editorial language refinement and LaTeX formatting assistance.

1. Wegener, G. H. (2025). SORT-AI: A Projection-Based Structural Framework for AI Safety—Alignment Stability, Drift Detection, and Scalable Oversight. *Preprints* **2024121334**. DOI:10.20944/preprints202512.1334.v2
2. Wegener, G. H. (2025). SORT-CX: A Projection-Based Structural Framework for Complex Systems—Operator Geometry, Non-Local Kernels, Drift Diagnostics, and Emergent Stability. *Preprints* **2024121431**. DOI:10.20944/preprints202512.1431.v1
3. Wegener, G. H. (2025). SORT-AI: A Structural Safety and Reliability Framework for Advanced AI Systems with Retrieval-Augmented Generation as a Diagnostic Testbed. *Preprints* **2024121345**. DOI:10.20944/preprints202512.1345.v1
4. Wegener, G. H. (2025). SORT-AI: Interconnect Stability and Cost per Performance in Large-Scale AI Infrastructure—A Structural Analysis of Runtime Instability in Distributed Systems. *Preprints* **2026010161**. DOI:10.20944/preprints202601.0161.v1
5. Wegener, G. H. (2025). SORT-AI: Runtime Control Coherence in Large-Scale AI Systems—Structural Causes of Cost, Instability, and Non-Determinism Beyond Interconnect Failures. *Preprints* **2026010298**. DOI:10.20944/preprints202601.0298.v1
6. Wooldridge, M. (2009). *An Introduction to MultiAgent Systems* (2nd ed.). John Wiley & Sons. ISBN 978-0-470-51946-2.
7. Russell, S., & Norvig, P. (2021). *Artificial Intelligence: A Modern Approach* (4th ed.). Pearson. ISBN 978-0-13-461099-3.
8. Stone, P., et al. (2016). Artificial Intelligence and Life in 2030. *One Hundred Year Study on Artificial Intelligence: Report of the 2015–2016 Study Panel*. Stanford University. ai100.stanford.edu/2016-report
9. Shoham, Y., & Leyton-Brown, K. (2009). *Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations*. Cambridge University Press. ISBN 978-0-521-89943-7.
10. Dorri, A., Kanhere, S. S., & Jurdak, R. (2018). Multi-Agent Systems: A Survey. *IEEE Access* **6**, 28573–28593. DOI:10.1109/ACCESS.2018.2831228
11. Jennings, N. R. (2000). On Agent-Based Software Engineering. *Artificial Intelligence* **117**(2), 277–296. DOI:10.1016/S0004-3702(99)00107-1
12. Yao, S., et al. (2023). ReAct: Synergizing Reasoning and Acting in Language Models. *Proceedings of ICLR 2023*.
13. Shinn, N., et al. (2023). Reflexion: Language Agents with Verbal Reinforcement Learning. *Proceedings of NeurIPS 2023*.
14. Wu, Q., et al. (2023). AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation. *arXiv preprint arXiv:2308.08155*.
15. Schick, T., et al. (2023). Toolformer: Language Models Can Teach Themselves to Use Tools. *Proceedings of NeurIPS 2023*.
16. Wei, J., et al. (2022). Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. *Proceedings of NeurIPS 2022*.
17. Xi, Z., et al. (2023). The Rise and Potential of Large Language Model Based Agents: A Survey. *arXiv preprint arXiv:2309.07864*.

 doi:10.20944/preprints202601.1741.v1

18. Wang, L., et al. (2024). A Survey on Large Language Model Based Autonomous Agents. *Frontiers of Computer Science* **18**(6), 186345. DOI:10.1007/s11704-024-40231-1
19. Mialon, G., et al. (2023). Augmented Language Models: A Survey. *Transactions on Machine Learning Research*.
20. Qin, Y., et al. (2023). Tool Learning with Foundation Models. *arXiv preprint arXiv:2304.08354*.
21. Shen, Y., et al. (2024). HuggingGPT: Solving AI Tasks with ChatGPT and its Friends in Hugging Face. *Proceedings of NeurIPS 2023*.
22. Park, J. S., et al. (2023). Generative Agents: Interactive Simulacra of Human Behavior. *Proceedings of UIST 2023*, 1–22. DOI:10.1145/3586183.3606763
23. Lewis, P., et al. (2020). Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. *Proceedings of NeurIPS 2020*.
24. Patil, S. G., et al. (2023). Gorilla: Large Language Model Connected with Massive APIs. *arXiv preprint arXiv:2305.15334*.
25. Hao, S., et al. (2024). ToolkenGPT: Augmenting Frozen Language Models with Massive Tools via Tool Embeddings. *Proceedings of NeurIPS 2023*.
26. Ruan, J., et al. (2024). TPTU: Large Language Model-Based AI Agents for Task Planning and Tool Usage. *arXiv preprint arXiv:2308.03427*.
27. Huang, W., et al. (2022). Language Models as Zero-Shot Planners: Extracting Actionable Knowledge for Embodied Agents. *Proceedings of ICML 2022*.
28. Gao, L., et al. (2023). PAL: Program-Aided Language Models. *Proceedings of ICML 2023*.
29. De Wolf, T., & Holvoet, T. (2005). Emergence Versus Self-Organisation: Different Concepts but Promising When Combined. In: *Engineering Self-Organising Systems*, pp. 1–15. Springer. DOI:10.1007/11494676_1
30. Holland, J. H. (1995). *Hidden Order: How Adaptation Builds Complexity*. Addison-Wesley. ISBN 978-0-201-44230-4.
31. Levin, S. A. (2003). Complex Adaptive Systems: Exploring the Known, the Unknown and the Unknowable. *Bulletin of the American Mathematical Society* **40**(1), 3–19. DOI:10.1090/S0273-0979-02-00965-5
32. Mitchell, M. (2009). *Complexity: A Guided Tour*. Oxford University Press. ISBN 978-0-19-512441-5.
33. Bar-Yam, Y. (1997). *Dynamics of Complex Systems*. Addison-Wesley. ISBN 978-0-201-55748-1.
34. Newman, M. (2010). *Networks: An Introduction*. Oxford University Press. ISBN 978-0-19-920665-0.
35. Albert, R., & Barabási, A.-L. (2002). Statistical Mechanics of Complex Networks. *Rev. Mod. Phys.* **74**, 47–97. DOI:10.1103/RevModPhys.74.47
36. Scheffer, M., et al. (2009). Early-Warning Signals for Critical Transitions. *Nature* **461**, 53–59. DOI:10.1038/nature08227
37. Strogatz, S. H. (2001). Exploring Complex Networks. *Nature* **410**, 268–276. DOI:10.1038/35065725
38. Watts, D. J., & Strogatz, S. H. (1998). Collective Dynamics of 'Small-World' Networks. *Nature* **393**, 440–442. DOI:10.1038/30918
39. Panait, L., & Luke, S. (2005). Cooperative Multi-Agent Learning: The State of the Art. *Autonomous Agents and Multi-Agent Systems* **11**(3), 387–434. DOI:10.1007/s10458-005-2631-2
40. Stone, P., & Veloso, M. (2000). Multiagent Systems: A Survey from a Machine Learning Perspective. *Autonomous Robots* **8**(3), 345–383. DOI:10.1023/A:1008942012299
41. Durfee, E. H. (1999). Distributed Problem Solving and Planning. In: *Multi-Agent Systems and Applications*, pp. 118–149. Springer.
42. Tumer, K., & Agogino, A. K. (2007). Distributed Agent-Based Air Traffic Flow Management. *Proceedings of AAMAS 2007*, 1–8. DOI:10.1145/1329125.1329434
43. Busoniu, L., Babuska, R., & De Schutter, B. (2008). A Comprehensive Survey of Multiagent Reinforcement Learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part C* **38**(2), 156–172. DOI:10.1109/TSMCC.2007.913919
44. Saltzer, J. H., Reed, D. P., & Clark, D. D. (1984). End-to-End Arguments in System Design. *ACM Transactions on Computer Systems* **2**(4), 277–288. DOI:10.1145/357401.357402
45. Ousterhout, J. (2018). *A Philosophy of Software Design*. Yaknyam Press. ISBN 978-1-7321022-0-0.
46. Lamport, L. (1978). Time, Clocks, and the Ordering of Events in a Distributed System. *Communications of the ACM* **21**(7), 558–565. DOI:10.1145/359545.359563
47. Jain, R. (1991). *The Art of Computer Systems Performance Analysis*. John Wiley & Sons. ISBN 978-0-471-50336-1.
48. Hellerstein, J. L., Diao, Y., Parekh, S., & Tilbury, D. M. (2004). *Feedback Control of Computing Systems*. John Wiley & Sons. ISBN 978-0-471-26637-2.

 doi:10.20944/preprints202601.1741.v1

49. Avizienis, A., et al. (2004). Basic Concepts and Taxonomy of Dependable and Secure Computing. *IEEE Transactions on Dependable and Secure Computing* **1**(1), 11–33. DOI:10.1109/TDSC.2004.2
50. Ananthanarayanan, G., et al. (2013). Effective Straggler Mitigation: Attack of the Clones. *Proceedings of NSDI'13*, 185–198.
51. Dean, J., & Barroso, L. A. (2013). The Tail at Scale. *Communications of the ACM* **56**(2), 74–80. DOI:10.1145/2408776.2408794
52. Gunawi, H. S., et al. (2014). What Bugs Live in the Cloud? A Study of 3000+ Issues in Cloud Systems. *Proceedings of SoCC'14*, 1–14. DOI:10.1145/2670979.2670986
53. Perrow, C. (1984). *Normal Accidents: Living with High-Risk Technologies*. Basic Books. ISBN 978-0-465-05142-9.
54. Brynjolfsson, E., & Hitt, L. M. (2000). Beyond Computation: Information Technology, Organizational Transformation and Business Performance. *Journal of Economic Perspectives* **14**(4), 23–48. DOI:10.1257/jep.14.4.23
55. Barroso, L. A., Clidaras, J., & Hölzle, U. (2013). The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines (2nd ed.). *Synthesis Lectures on Computer Architecture* **8**(3), 1–154. DOI:10.2200/S00516ED2V01Y201306CAC024
56. Barroso, L. A., Hölzle, U., & Ranganathan, P. (2019). The Datacenter as a Computer: Designing Warehouse-Scale Machines (3rd ed.). *Synthesis Lectures on Computer Architecture* **13**(3), 1–189. DOI:10.2200/S00874ED3V01Y201809CAC046
57. Patterson, D. A., & Hennessy, J. L. (2016). *Computer Organization and Design: The Hardware/Software Interface* (5th ed.). Morgan Kaufmann. ISBN 978-0-12-407726-3.
58. Kannan, R. S., et al. (2019). GrandSLAm: Guaranteeing SLAs for Jobs in Microservices Execution Frameworks. *Proceedings of EuroSys'19*, 1–16. DOI:10.1145/3302424.3303958
59. Sculley, D., et al. (2015). Hidden Technical Debt in Machine Learning Systems. *Proceedings of NeurIPS 2015*, 2503–2511.
60. Amershi, S., et al. (2019). Software Engineering for Machine Learning: A Case Study. *Proceedings of ICSE-SEIP'19*, 291–300. DOI:10.1109/ICSE-SEIP.2019.00042
61. Zaharia, M., et al. (2018). Accelerating the Machine Learning Lifecycle with MLflow. *IEEE Data Engineering Bulletin* **41**(4), 39–45.
62. Delimitrou, C., & Kozyrakis, C. (2014). Quasar: Resource-Efficient and QoS-Aware Cluster Management. *Proceedings of ASPLOS'14*, 127–144. DOI:10.1145/2541940.2541941
63. Kephart, J. O., & Chess, D. M. (2003). The Vision of Autonomic Computing. *IEEE Computer* **36**(1), 41–50. DOI:10.1109/MC.2003.1160055
64. Alon, U. (2007). Network Motifs: Theory and Experimental Approaches. *Nature Reviews Genetics* **8**, 450–461. DOI:10.1038/nrg2102
65. Gilbert, S., & Lynch, N. (2002). Brewer's Conjecture and the Feasibility of Consistent, Available, Partition-Tolerant Web Services. *ACM SIGACT News* **33**(2), 51–59. DOI:10.1145/564585.564601
66. Kroll, J. A., et al. (2017). Accountable Algorithms. *University of Pennsylvania Law Review* **165**(3), 633–705.
67. Wachter, S., Mittelstadt, B., & Floridi, L. (2017). Why a Right to Explanation of Automated Decision-Making Does Not Exist in the General Data Protection Regulation. *International Data Privacy Law* **7**(2), 76–99. DOI:10.1093/idpl/ipx005
68. Doshi-Velez, F., & Kim, B. (2017). Towards A Rigorous Science of Interpretable Machine Learning. *arXiv preprint arXiv:1702.08608*.
69. Brundage, M., et al. (2020). Toward Trustworthy AI Development: Mechanisms for Supporting Verifiable Claims. *arXiv preprint arXiv:2004.07213*.
70. Bommasani, R., et al. (2021). On the Opportunities and Risks of Foundation Models. *arXiv preprint arXiv:2108.07258*.
71. Pineau, J., et al. (2021). Improving Reproducibility in Machine Learning Research. *Journal of Machine Learning Research* **22**(164), 1–20.
72. Rabinovich, E., et al. (2023). Scaling Distributed Machine Learning with In-Network Aggregation. *Proceedings of NSDI'23*, 1–19.
73. Liu, X., et al. (2023). AgentBench: Evaluating LLMs as Agents. *Proceedings of ICLR 2024*.
74. Chang, Y., et al. (2024). A Survey on Evaluation of Large Language Models. *ACM Transactions on Intelligent Systems and Technology* **15**(3), 1–45. DOI:10.1145/3641289
75. Lilja, D. J. (2000). *Measuring Computer Performance: A Practitioner's Guide*. Cambridge University Press. ISBN 978-0-521-64105-4.

 doi:10.20944/preprints202601.1741.v1

76. Kapoor, S., & Narayanan, A. (2024). Leakage and the Reproducibility Crisis in ML-Based Science. *Patterns* **5**(4), 100804. DOI:10.1016/j.patter.2023.100804
77. Liang, P., et al. (2023). Holistic Evaluation of Language Models. *Transactions on Machine Learning Research*.
78. Liu, N. F., et al. (2024). Lost in the Middle: How Language Models Use Long Contexts. *Transactions on Machine Learning Research*.
79. Sumers, T. R., et al. (2024). Cognitive Architectures for Language Agents. *Transactions on Machine Learning Research*.