Article

When Agents Amplify Themselves: Structural Instability in Tool-Oriented AI Systems

Gregor Herbert Wegener

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

Abstract

Agentic AI systems are extending inference from a single-pass computational process into a recursive execution structure in which planning, tool invocation, evaluation, and retry operations form closed-loop execution graphs. In production deployments, retries are not exceptional control events but a regular part of system operation: tool failures, low-confidence outputs, missing data, and partial successes can all trigger structured re-entry into the execution loop. This paper examines retry cascades and tool-loop amplification as structural runtime phenomena that shape the behaviour of large-scale agentic AI systems. Three amplification pathways are considered: nonlinear token growth through context accumulation across retries, geometric expansion of tool-call graphs through chained re-execution, and recursive aggregation of weak signals in which uncertainty is reintroduced into subsequent planning steps rather than discarded. These dynamics are especially relevant at hyperscale because they influence cost efficiency, execution predictability, and runtime coordination across heterogeneous infrastructure. Conventional evaluation methods provide only partial visibility into these effects. Benchmark suites primarily measure model capability under controlled conditions, while operational monitoring focuses on outputs and performance indicators rather than the topology of recursive execution itself. As a result, agentic systems can exhibit strong benchmark and production outcomes while still presenting opportunities for improved structural transparency and runtime efficiency. The paper introduces a diagnostic vocabulary for analysing execution graph stability in agentic systems and maps the identified patterns to the SORT application catalog. The goal is not to critique existing systems, but to provide an additional structural perspective that can support greater clarity in runtime behaviour, more efficient fleet-level coordination, and improved reproducibility in tool-oriented AI deployments.

Keywords: agentic AI; retry cascades; tool-loop amplification; recursive execution; execution graph topology; structural diagnostics; runtime efficiency; agentic coherence; hyperscale infrastructure; runtime control

---

1. Executive Summary

Agentic AI systems are extending inference from a single-pass computational process into a recursive execution structure in which planning, tool invocation, evaluation, retry, and branching are increasingly integrated into normal runtime behaviour. This shift creates a different scaling profile for modern AI infrastructure. Execution is no longer defined only by model quality or per-call latency, but also by how effectively recursive workflows remain coordinated as system complexity increases.

In production environments, retries are not exceptional events but a regular component of robust operation. Tool failure, low-confidence outputs, missing data, and partial completion can all trigger structured re-entry into the execution loop [8,9]. At scale, these re-entry patterns can reshape execution graphs through nonlinear token growth, geometric tool-call expansion, and recursive weak-signal aggregation. For infrastructure operators, this is not only a cost question but also a runtime visibility question.

Industry signals indicate that these dynamics are already becoming operationally relevant. Over 40% of agentic AI projects fail before reaching production [1]. Complex agents consume 5–20x more tokens than simple chains due to loops and retries [2], and agentic workloads can generate 20–30 times more tokens than conventional inference [3]. These figures suggest that recursive execution is becoming a first-order systems variable for hyperscale deployment economics.

The associated economic exposure is substantial. Production costs for agentic systems can reach $18,000–90,000 monthly, driven by token accumulation in multi-turn workflows and retry overhead [4]. McKinsey further reports that 62% of organizations experienced unexpected cost overruns with token-based AI services [5]. For hyperscalers and large-scale AI platform operators, this indicates a growing opportunity to improve execution efficiency not only through better models, but through better structural understanding of recursive workloads.

The central issue is therefore structural rather than purely algorithmic. Agentic systems do not underperform primarily because individual outputs are wrong. They underperform when recursive execution becomes insufficiently bounded, insufficiently transparent, or insufficiently coherent across tools, retries, and control surfaces. This reframes the challenge from isolated model optimisation toward runtime architecture.

Conventional performance metrics remain necessary but incomplete. Accuracy, latency, and throughput do not directly capture recursion depth, retry topology, or loop stability. Systems can perform well in benchmark settings while still presenting opportunities for substantial improvement in production runtime efficiency and control [19,20]. The resulting gap is not merely a lack of more metrics, but a lack of visibility into execution structure itself.

Existing benchmarks, logging stacks, and post hoc guardrails improve local measurement resolution, but they do not directly observe execution topology. What is missing is a structural diagnostic layer for recursive execution graphs. Such a layer would not replace current observability practices. It would complement them by making retry propagation, execution branching, and control coherence more legible at system scale.

This paper introduces a structural diagnostic perspective for analysing execution graph stability and maps the resulting patterns to the SORT application catalog [25,26]. The objective is constructive: to provide hyperscalers, AI platform teams, and infrastructure architects with a clearer vocabulary for understanding recursive execution, improving runtime efficiency, and strengthening the operational readiness of tool-oriented AI systems.

2. The Economic Shift: From Inference to Recursion

The economic structure of AI deployment has already shifted from a training-dominated regime toward an inference-dominated regime, and current agentic architectures indicate the emergence of a further transition toward recursion-dominated execution [7]. This transition is important because agentic systems do not merely increase request volume. They alter the cost profile of runtime itself. Instead of a largely bounded relationship between prompt size, model execution, and output, recursive systems introduce repeated planning, tool interaction, re-evaluation, and controlled re-entry into the same task. Under these conditions, runtime cost is shaped not only by the size of the original request but by the structure of the execution graph that follows.

For hyperscalers and large AI platform operators, this means that the economic unit of deployment is becoming more complex. The relevant question is no longer only how efficiently a single inference call can be served. It is increasingly how efficiently a recursive workload can be coordinated when execution depth, branching, and tool interaction vary by task class. In that setting, the economic challenge is not simply higher token throughput. It is a change in the shape of the runtime cost function itself, from approximately linear request execution toward superlinear execution under repeated recursive re-entry.

2.1. The Agentic Token Explosion

Agent-based systems are designed to execute more than a single reasoning pass. They operate through multi-step reasoning loops, tool calls, exploration strategies, external interactions, and iterative revision cycles [8,9]. This gives them greater practical flexibility, but it also changes the token economics of deployment. Each additional planning loop, retry event, or tool-mediated step can expand both the active context and the execution path length. As a result, token consumption becomes increasingly coupled to workflow structure rather than to prompt length alone.

This effect is now visible in industry reporting. Gartner estimates that agentic workloads generate 20–30 times more tokens than conventional prompt-response interactions [1]. Galileo reports that complex agents consume 5–20x more tokens than simple chains because recursive loops and retries extend the effective execution path [2]. These figures are significant because they indicate that recursive execution is becoming a primary runtime variable in commercial deployments.

Recent model releases reinforce this direction. GPT-5.3 Codex exemplifies the movement toward interactive coding agents, with reported SWE-Bench Pro performance of 56.8% and Terminal-Bench 2.0 performance of 77.3% [10]. Claude Sonnet 4.6 extends long-running agent workflows through computer-use APIs and context compaction mechanisms designed for prolonged execution [12]. Qwen 3.5 combines a 397B-parameter model with only 17B active parameters per forward pass, illustrating how economically efficient architectures become increasingly important when execution volume is driven by recursive workflows rather than isolated calls [13].

Under these conditions, token cost becomes one of the dominant runtime variables in agentic deployment. CX Today reports that 73% of enterprise agentic implementations exceeded budget, in some cases by more than 2.4x [6]. This suggests that the token explosion associated with agentic systems is not a marginal pricing issue. It is becoming a structural constraint on how large-scale deployments are designed, budgeted, and operated.

2.2. Infrastructure Under Recursive Load

The infrastructure implications of this shift are equally significant. Combined hyperscaler AI-related capital expenditure now exceeds $200 billion annually [16]. That investment is not directed only toward larger model training clusters. It is increasingly tied to serving infrastructure capable of sustaining continuous, variable, and potentially bursty recursive workloads. This differs materially from training-oriented infrastructure, where load profiles are more predictable and bounded by scheduled training runs.

Recursive workloads place different demands on serving infrastructure. They require capacity not only for model execution, but also for orchestration, state persistence, tool coordination, scheduling, and repeated re-entry into the execution graph. The OpenAI–Amazon partnership announced in February 2026 reflects this direction by formalizing a stateful runtime environment on Amazon infrastructure alongside enterprise-oriented agent distribution [11]. The importance of statefulness here is notable because recursive systems often depend on preserving intermediate execution context across multiple steps rather than treating each interaction as an isolated transaction.

At the same time, hardware diversification is intensifying. Production stacks increasingly span GPUs, TPUs, Trainium, Inferentia, and other custom accelerators. NVIDIA has signalled inference-optimized accelerator directionality [14], while TensorRT-LLM AutoDeploy targets more efficient deployment across heterogeneous inference stacks [15]. This diversification is economically rational, but it also introduces a new operational variable. Recursive workloads can exhibit different retry patterns, timing behaviours, and coordination characteristics across accelerator classes, latency regimes, and scheduling layers.

The practical consequence is that infrastructure performance can no longer be described solely by average throughput or cost per token. Under recursive load, the same task class may produce different execution trajectories depending on routing choices, hardware allocation, and retry propagation.

For hyperscalers, this makes structural visibility into recursive execution an increasingly relevant complement to conventional serving metrics.

2.3. Energy Constraints Amplified by Recursion

The economic shift toward recursion also has an energy dimension. Goldman Sachs estimates that global datacenter electricity consumption could rise by approximately 175% by 2030 [16]. The IEA projects that total datacenter electricity demand could exceed 1,700 TWh by 2035 under high-adoption scenarios [17]. Deloitte further reports that 72% of U.S. power and datacenter executives identify grid stress as the leading infrastructure challenge [18]. These figures indicate that AI infrastructure economics are now inseparable from power availability and energy management.

Agentic workloads intensify this issue because they create highly variable resource demand. Retry storms, multi-step tool interactions, and recursive planning loops can generate unpredictable compute and power spikes even when user-facing demand appears stable. In such systems, energy cost is not determined only by the number of completed tasks, but by the internal structure of the runtime path that led to those completions. Recursive execution therefore multiplies energy cost per completed task in a nonlinear manner when re-entry paths deepen or branch repeatedly.

This also means that energy-aware scheduling becomes part of the recursive control problem rather than an external optimization layer. Once power-aware scheduling interacts with retry policies, orchestration logic, and resource allocation, it forms another autonomous control loop within the execution system [26]. From a hyperscaler perspective, this is strategically important. The transition from inference to recursion is not only a software architecture shift. It is an infrastructure and energy coordination shift in which execution efficiency increasingly depends on how recursive workloads are structurally managed across compute, control, and power layers.

3. The Hidden Structural Effect: Retry Cascades as Systemic Amplifiers

The economic and infrastructural developments described in the previous section do not merely increase deployment scale. They reshape the internal topology of agentic execution itself. Once planning, tool invocation, evaluation, and retry become integrated into ordinary runtime behaviour, the operative system is no longer well described as a sequence of independent inference calls. It becomes a recursive execution process whose structural properties influence efficiency, predictability, and control.

Within this setting, retry cascades should not be interpreted as incidental implementation noise. They are better understood as a structural amplification mechanism that emerges when local re-entry policies interact with tool dependencies, intermediate state persistence, and task-level uncertainty. Their significance lies not in the existence of retries as such, since retries are often useful and necessary, but in the way repeated re-entry can transform bounded execution into expanding execution graphs. This transformation is especially relevant for large-scale AI operators because it shifts the primary optimisation problem from isolated call quality toward recursive workload coordination.

3.1. From Linear Inference to Recursive Execution Graphs

Classical inference follows a bounded pattern: input is processed by a model and produces an output. In that setting, execution is conceptually compact. The system can usually be analysed through request size, latency, throughput, and output quality. Agentic execution follows a different pattern. Input can lead to a model step, which triggers a tool invocation, which yields new information, which prompts additional model evaluation, which in turn may trigger further tool calls, branching decisions, and structured retry. The resulting execution path is therefore not simply longer. It is topologically different.

The decisive structural difference is that execution is no longer bounded by design. In a recursive setting, completion is contingent on the behaviour of the execution graph rather than on the completion of a single model pass. Tool calls function as actions with side effects rather than as pure queries. Each invocation can modify external state, internal state, or downstream decision conditions [8]. As a

consequence, the effect of a retry is not restricted to repeating a previous computation. It can alter the subsequent execution landscape.

Planning steps deepen this effect because they create execution chains in which early decisions constrain later reasoning paths. A branching choice made near the beginning of a task can determine which tools are used, how much context is accumulated, which errors are encountered, and how often the system re-enters the same task structure. Under these conditions, retry mechanisms can amplify rather than resolve inefficiencies when they re-enter the system without sufficiently coherent global constraints [9]. What appears locally as a corrective step can therefore function globally as an expansion operator on the execution graph.

3.2. Three Amplification Mechanisms

Three amplification mechanisms are particularly relevant for understanding the structural behaviour of recursive execution systems.

The first is **nonlinear token growth**. Each retry adds context, each loop increases active state, and memory compounds across execution steps. This means that token cost is increasingly determined by path depth and path repetition rather than by initial prompt size alone. In practical deployments, cost can therefore grow faster than linearly and in some regimes become effectively exponential under repeated re-entry [2,4]. From an infrastructure perspective, this matters because runtime demand becomes sensitive to topology rather than just to workload count.

The second mechanism is **tool-call amplification**. Failed calls can trigger cascades, dependencies between tools can create chained retries, and the call graph can expand geometrically as the system attempts to recover, verify, or refine intermediate results. In long-horizon environments, this effect is already visible in low completion rates and in the tendency of agents to remain trapped in local execution patterns rather than reach efficient resolution [23]. The key point is not merely that tool use introduces latency. It is that tool dependency chains multiply the number of structurally plausible re-entry paths.

The third mechanism is **weak signal aggregation**. A low-confidence output can trigger retry, reinterpretation, and further retry, such that uncertainty is repeatedly reintroduced into the planning process rather than filtered out. In this regime, ambiguous signals can accumulate structural influence far beyond their original informational value. The issue is not that the system processes uncertainty, which is often desirable, but that recursive re-entry can transform weak evidence into persistent execution momentum.

These three mechanisms interact. Token growth feeds tool-call expansion because longer contexts and longer paths create more possible re-entry points. Tool-call expansion generates additional uncertainty through partial results, failures, or conflicting intermediate states. Weak signals then trigger further retries, which deepen execution and increase token usage. The resulting behaviour is therefore not additive but coupled. Structural amplification emerges from the interaction between these mechanisms rather than from any single one in isolation.

3.3. The Ghost Cost Regime

Under sustained recursive execution, the system can begin to optimise for continuation rather than completion. This does not imply a change in explicit objective. Rather, it describes an operational condition in which substantial runtime activity is consumed by maintaining, extending, or revisiting the execution graph without proportional progress toward final task resolution. This condition can be described as a **ghost cost regime**.

Within this regime, **ghost tokens** are consumed tokens that do not materially contribute to final task completion [25]. They may arise from repeated context restatement, intermediate deliberation that is not incorporated into the final path, or re-entry loops that preserve execution momentum without improving resolution quality. **Ghost planning** consists of planning iterations that are executed, evaluated, and then abandoned when the system changes direction or restarts a task segment [25].

Ghost tool-calls are invocations whose outputs are unused, superseded, or contradicted by later execution paths [25].

The strategic importance of this regime is that ghost costs scale superlinearly with agentic complexity. As the number of tools, branches, and retry opportunities increases, the share of runtime devoted to structurally unproductive execution can also rise. Crucially, the system can still appear operationally healthy because outputs continue to be produced and conventional metrics may remain within acceptable ranges. Structural resource waste compounds internally even while surface indicators remain nominal.

This helps explain why cascading execution effects are increasingly relevant from a reliability and governance perspective. OWASP now classifies cascading failures in agentic systems as a distinct security and reliability risk category under ASI08 [24]. In operational terms, this recognition is important because it reframes recursive instability not as a marginal edge case, but as a systems property that deserves explicit visibility. For hyperscalers and large-scale AI operators, the value of that visibility is constructive. It enables more precise understanding of where recursive execution remains efficient, where it begins to over-extend, and where additional structural diagnostics can improve runtime control.

4. The Observability Gap: Why Existing Measurement Sees the Outcome but Not the Geometry

The central limitation of current measurement practice is not simply that benchmarks are incomplete. It is that prevailing observability layers are designed to report what the system produced, how long it took, and how many resources it consumed, while the internal geometry of recursive execution remains largely unobserved. In agentic systems, this distinction becomes operationally critical because runtime quality is no longer determined only by output quality or service-level performance. It is increasingly determined by how execution unfolds across retries, branches, tool dependencies, and structured re-entry loops.

This creates a practical blind spot for large-scale AI operators. A deployment can preserve acceptable latency distributions, nominal throughput, and superficially stable completion rates, even while its internal execution graph becomes deeper, more branch-intensive, and more path-dependent. The result is that recursive inefficiencies can accumulate within systems that continue to look locally performant. From a hyperscaler perspective, this means that a meaningful share of runtime risk can remain hidden precisely because existing measurement stacks are optimized for outputs rather than for execution structure. The system can appear efficient at the surface layer while becoming progressively more expensive, less predictable, and more difficult to coordinate at the recursive layer.

4.1. The Benchmark Assumption

Benchmark frameworks remain valuable because they provide standardized environments for comparing model behaviour. They help quantify capability, latency, and task performance under controlled conditions. However, they rely on assumptions that are only partially aligned with production agentic systems. Benchmark conditions typically assume fixed context, deterministic scheduling, isolated execution, homogeneous hardware, and effectively unconstrained retry budgets. These assumptions are analytically useful, but they simplify the execution setting in exactly those dimensions where recursive structure becomes operationally consequential.

Production agentic systems operate under different conditions. Workloads fluctuate, routing is dynamic, hardware is heterogeneous, retry budgets are infrastructure-constrained, and execution paths can be altered by queue conditions, tool availability, and orchestration policies. Under these conditions, the relevant unit of analysis is not only model capability per task, but the stability of recursive execution under changing system constraints. Benchmark scores therefore measure model capability under controlled conditions, not recursive execution stability as such [19].

Recent benchmark developments move in a useful direction, but they do not yet eliminate this gap. SPEED-Bench extends evaluation toward more realistic serving conditions and heterogeneous

workloads [21]. ISO-Bench evaluates whether coding agents can optimize real-world inference workloads [22]. WebArena-class results show that long-horizon agent success remains limited, in part because agents can become trapped in recursive local patterns rather than converging efficiently on completion [23]. These efforts are important, but they remain benchmark frameworks rather than runtime topology observability layers. They improve measurement at the task layer without directly instrumenting execution geometry itself. In practical terms, this means that organizations may improve benchmark realism while remaining structurally blind to recursive amplification in production.

4.2. What Metrics Measure vs. What Changes

Standard production metrics capture token throughput, latency distributions, GPU utilization, accuracy, and cost per token. These indicators are operationally necessary because they provide direct visibility into system output, resource consumption, and service quality. Yet they do not directly capture recursion depth, retry topology, loop stability, call graph entropy, or signal re-entry patterns. In recursive systems, these unmeasured structural properties increasingly shape the operational meaning of the measured ones.

This creates a categorical mismatch between what is measured and what changes. Existing metrics primarily describe outcomes. Structural drift in agentic systems occurs in the topology of execution itself. A system may still satisfy conventional performance criteria while its internal execution graph becomes progressively deeper, more branching, or more path-dependent [20]. When this occurs, the operational surface can remain stable even though the underlying execution process is becoming less transparent, less predictable, and more expensive to sustain.

The implication is not that existing metrics are inadequate in themselves. It is that they are incomplete for recursive execution systems. Increasing benchmark coverage or expanding telemetry improves resolution at the output layer, but it does not eliminate the topological blind spot. The missing capability is direct visibility into how execution paths expand, re-enter, and reorganize under production conditions. Without that visibility, recursive instability is often detected only indirectly through cost overruns, throughput variance, coordination inefficiencies, or degraded reliability after the structural change has already taken place. For hyperscalers, this means the issue is not hypothetical. It is already present wherever agentic systems are evaluated through outputs while recursive runtime complexity continues to scale underneath them.

5. Why Existing Approaches Do Not Solve the Problem

The limitation identified in the previous section is not simply a measurement shortfall. It is an architectural mismatch between the tools most organizations use to evaluate agentic systems and the layer at which recursive execution becomes operationally consequential. Retry cascades do not persist because systems are unobserved in a general sense. They persist because prevailing evaluation and monitoring methods are optimized for outputs, events, and local performance signals, while the decisive changes occur in the structure of execution itself.

This distinction matters for hyperscale operators because recursive workloads can remain commercially attractive and technically functional even while their internal execution geometry becomes progressively more expensive to coordinate. Better benchmarks, richer telemetry, and more extensive guardrail systems can all improve local system understanding. Each of these approaches adds value. But none of them, in isolation, provides direct visibility into how execution graphs branch, deepen, compress, and reorganize over time. The result is that organizations often improve the visibility of symptoms without yet obtaining direct structural visibility into the mechanism that produces them.

For large-scale AI operators, the challenge is therefore not a lack of instrumentation in general. It is a mismatch between the layer being instrumented and the layer where recursive drift emerges. Existing tools remain important because they support capability evaluation, operational debugging, and policy enforcement. Yet they are strongest at the output layer, whereas recursive amplification becomes most consequential at the execution-structure layer. This is why incremental extension of existing practices, while useful, does not by itself resolve the underlying problem.

5.1. More Benchmarks Increase Resolution, Not Structural Visibility

Additional benchmarks can improve task coverage, workload realism, and performance granularity. They make it easier to compare systems across a wider set of conditions, and they remain essential for identifying strengths and weaknesses in capability, latency, cost, and robustness. For model selection, deployment screening, and comparative evaluation, this increased resolution is operationally valuable.

However, benchmark resolution is not the same as structural visibility. Benchmarks can show whether an agent succeeds or fails under more realistic conditions, but they do not directly expose how execution graphs branch, recurse, compress, or drift over time. Even when benchmark tasks become more complex and closer to production workloads, the primary measured object remains task performance. The recursive structure that produced that performance remains secondary to the evaluation design.

This means that additional benchmark coverage increases projection resolution at the task layer, but it does not by itself reveal whether a system is structurally efficient, topologically stable, or increasingly dependent on recursive over-extension. A benchmark can indicate that performance changes under more realistic conditions. It does not automatically show how the execution graph changed in order to produce that result. For this reason, better benchmarks should be understood as a refinement of task-level evaluation, not as a substitute for structural observability of runtime behaviour.

5.2. More Logging Increases Telemetry Volume, Not Topological Understanding

Logging systems can capture more events, traces, and runtime metadata. In production settings, richer telemetry is generally beneficial because it expands operational visibility, improves debugging, and supports faster incident response. For large-scale operators, this kind of observability remains a necessary part of reliable service management.

Yet telemetry abundance does not automatically produce execution topology analysis. Logging systems typically record event sequences, service states, and local transitions rather than structural relations between recursive branches, retry dependencies, and signal re-entry paths. They can show that a retry occurred, that a tool call failed, or that latency increased at a given point in the execution sequence. What they often do not show directly is how these events are connected across the larger recursive graph.

This difference is operationally significant because recursive instability is rarely reducible to isolated events. It emerges through relations across events: which retries spawned which branches, which tool failures re-entered planning, which partial outputs reappeared as new inputs, and how these structures evolved as runtime conditions changed. A system can therefore generate a large volume of telemetry while still providing only limited visibility into the architecture of its recursive behaviour. More telemetry improves local detail. It does not automatically deliver structural understanding.

5.3. More Guardrails Act Post Hoc, Not at the Structural Layer

Guardrails play an important role in production AI systems. They can constrain outputs, block unsafe actions, enforce policy boundaries, and intercept specific tool calls before they cause harm. For hyperscalers and enterprise platforms, they are a necessary component of runtime governance and reliability engineering.

At the same time, guardrails usually act on events, outputs, or actions after they have been produced or selected. They do not necessarily regulate whether the underlying execution process remains coherent, bounded, or convergent as it unfolds. A system may satisfy output-level constraints while still generating excessive recursive branching, repeated re-entry loops, or structurally inefficient tool interactions. In that situation, policy enforcement can remain effective even while execution structure becomes progressively more costly and less transparent.

The unresolved issue is therefore not only governance at the output layer. It is runtime structure. Guardrails can constrain what a system is permitted to do, but they do not automatically reveal whether the path by which the system arrived there remained structurally efficient and coherent. This is why output governance and structural diagnostics should be understood as complementary rather than interchangeable. One governs acceptability of behaviour. The other makes the architecture of recursive behaviour legible.

5.4. The Missing Layer

The unresolved gap is a missing structural diagnostic layer for recursive execution systems. What is needed is not merely more output measurement, but direct visibility into execution topology, retry propagation, and recursive control coherence. This means observing not only whether a system succeeded, failed, or remained compliant, but how its execution graph evolved while doing so.

From a systems perspective, this missing layer sits between conventional observability and high-level governance. It does not replace benchmark evaluation, telemetry, or guardrails. It connects them by making the recursive execution process itself legible. Once execution topology becomes visible, retry cascades, path-dependent amplification, and ghost-cost accumulation can be analyzed as structural runtime phenomena rather than inferred only indirectly from downstream symptoms.

For hyperscalers, this should be understood as a constructive extension of current practice rather than as a critique of it. Existing approaches remain necessary and valuable. But as agentic workloads continue to scale, the next improvement frontier is likely to come from better structural visibility into recursive execution itself. That is the layer required to understand not only what the system produced, but how efficiently, coherently, and predictably it produced it.

6. The Structural Diagnostic Gap

Benchmark evaluation and operational monitoring are well suited to measuring performance outcomes, service quality, and resource utilization. They provide important visibility into whether an agentic system is operating within expected ranges and whether outputs remain acceptable under production conditions. However, they provide only limited visibility into the structural conditions of recursive execution itself. In agentic systems, this matters because runtime behaviour is shaped not only by output quality, but also by the topology through which the system arrives at that output.

This creates a structural diagnostic gap specific to agentic systems. The gap appears when recursive execution remains externally successful while the internal execution graph becomes deeper, more branch-intensive, more path-dependent, or more sensitive to infrastructure conditions. In such settings, conventional observability can continue to report healthy outputs even though substantial opportunities for improved runtime efficiency and coordination remain hidden at the structural layer.

6.1. What Observability Does Not Capture

Standard monitoring systems are not generally designed to determine whether identical agent requests follow different retry trajectories under varying load conditions. They can record events, timings, and outputs, but they do not reliably expose the structural divergence between one execution path and another when the same task is processed under different routing, hardware, or queue states. This matters because recursive systems are often sensitive to these contextual changes even when their output remains superficially stable.

A similar limitation appears in the treatment of retry budgets. Observability systems can often detect that retries occurred, but they do not directly capture how retry budgets evolve when infrastructure constraints tighten, tool availability changes, or routing policies are re-optimized. The structural significance of a retry is therefore often reduced to an event count rather than understood as part of a larger execution topology. This reduces visibility precisely where recursive behaviour becomes economically and operationally important.

Context compression creates another blind spot. In long-running agent workflows, compression mechanisms can alter the informational content carried forward between execution stages. Yet the

effects of compression on intermediate reasoning states remain largely invisible to standard telemetry. The system may appear to preserve continuity while the internal representational structure changes in ways that influence later retries, tool choices, or planning loops. Because these effects are not directly observable at the topology level, they are often inferred only indirectly through downstream variance.

Queue scheduling decisions produce a related effect. Scheduling alters the ordering and timing of tool calls, which can in turn reshape the execution path followed by the agent. Conventional logs usually register such changes only as event timing differences or service-level metadata. The structural variation itself, namely the way execution paths diverge under different scheduling conditions, remains only partially visible. As a result, observable outputs can remain stable while the internal execution graph changes substantially.

This limitation is categorical rather than incidental. Observability measures performance signals, while the critical dynamics in recursive systems unfold in execution topology itself [26]. For this reason, the structural diagnostic gap should not be understood as a narrow instrumentation problem. It reflects a deeper difference between observing what the system produces and observing how the recursive execution process is structurally organized while producing it.

6.2. Execution Graph Analysis

To address this gap, it is useful to introduce the concept of **Execution Graph Topology** as the analytical lens for agentic runtime analysis. This concept shifts attention away from outputs alone and toward the structure of recursive execution. The goal is not to replace existing observability practices, but to extend them by making retry propagation, branching behaviour, and recursive control structure more explicitly legible.

Several structural dimensions are especially relevant within this perspective. The first is **Recursion Depth Distribution**, which captures the statistical profile of how deeply retry chains extend by task class. Some tasks may remain shallow and well bounded, while others may systematically produce deeper recursive trees. This distinction is operationally meaningful because it affects cost, predictability, and runtime coordination.

The second dimension is the **Retry Branching Factor**. This describes how many alternative execution paths are spawned per failure event or ambiguous intermediate result. A system with a low branching factor may remain operationally compact even under repeated retries, whereas a system with a higher branching factor can expand rapidly in execution complexity even if the number of initial requests remains constant. Branching therefore matters not only as a behavioural pattern, but as a structural multiplier on runtime load.

The third dimension is **Tool-Call Dependency Chains**. These chains describe the length and coupling of sequential tool invocations within a recursive execution path. Strongly coupled tool chains can make the system more sensitive to intermediate failures or partial results, while longer dependency chains increase the number of structurally plausible re-entry points. This dimension is particularly relevant in production agent architectures where tools are not peripheral utilities but active components of the execution graph.

The fourth dimension is **Signal Re-Entry Patterns**. These patterns describe how outputs of failed, partial, or ambiguous executions re-enter planning loops and influence subsequent runtime behaviour. In recursive systems, the same informational fragment may reappear with greater operational weight than its original evidentiary value would suggest. Tracking signal re-entry therefore helps distinguish between healthy iterative refinement and structurally amplifying recursion.

Taken together, these dimensions shift the focus from outputs to topology. They provide a way to examine recursive execution as a structured runtime object rather than as a sequence of isolated events. This perspective complements existing observability rather than replacing it [25]. For hyperscalers and large-scale AI platform operators, its value lies in enabling a more precise understanding of when recursive systems remain efficient and coherent, and when they begin to drift into execution regimes that require stronger structural visibility.

7. SORT as a Structural Diagnostic Layer

The SORT framework is used in this paper exclusively as a diagnostic lens for execution structure. It is introduced here neither as a new dynamical model nor as a replacement for existing engineering practice. No new physical laws, empirical parameters, or additional system degrees of freedom are assumed. The purpose of the framework in the present context is narrower and more practical. It provides a compact structural vocabulary for describing recursive execution patterns that are otherwise difficult to express within conventional benchmark, telemetry, or policy language.

This positioning is important because the problem identified in the previous sections is not primarily a lack of output-level measurement. It is a lack of language for describing how recursive execution remains coherent, how retry propagation expands, and how runtime structure drifts under changing infrastructure conditions. In that sense, the value of SORT here is diagnostic rather than prescriptive. The relevant evaluation criteria are structural consistency, control coherence, and minimality of the vocabulary used to describe execution behaviour.

7.1. Diagnostic Vocabulary

Agentic Coherence.

Agentic coherence denotes the degree to which autonomous decisions remain mutually consistent with shared or evolving task intent [25]. In recursive systems, coherence matters because retries are not isolated corrections. They alter subsequent execution paths. When coherence is high, retry decisions reinforce task progress, preserve continuity across tool interactions, and support convergence toward completion. When coherence is low, retries can continue to consume resources while progressively weakening directional alignment within the execution graph.

From a runtime perspective, agentic coherence therefore describes more than semantic consistency at the output layer. It refers to whether the execution process remains directionally ordered as planning, tool use, and re-entry accumulate. This makes the concept useful for distinguishing between productive iteration and structurally expanding recursion.

Retry Amplification.

Retry amplification is the structural phenomenon in which local retry logic produces globally expanding execution graphs. It is distinct from individual tool failure or isolated re-execution. The defining feature is that a local recovery mechanism acquires global structural significance because it changes the size, shape, or persistence of the execution path.

This concept is useful because retries are often operationally beneficial. The issue is not that systems retry. The issue is that recursive re-entry can amplify execution beyond what is required for resolution when retries interact with branching, tool dependencies, and uncertainty propagation. Retry amplification therefore identifies a structural growth mechanism in recursive workloads rather than a specific implementation defect.

Execution-Layer Drift.

Execution-layer drift denotes the gradual reshaping of execution topology without model modification [27]. This can occur when infrastructure constraints alter retry budgets, scheduling priorities, routing choices, memory pressure, or resource allocation policies. In such cases, the model may remain unchanged while the runtime structure through which it operates changes materially.

The relevance of this concept is operational. It makes explicit that structural instability can emerge without any visible change in the underlying model artifact. For large-scale operators, this is especially important because many production variations originate not from the model itself but from the serving, orchestration, and resource layers surrounding it. Execution-layer drift therefore provides a way to describe structural change that is induced by runtime conditions rather than by parameter updates.

Ghost Cost Regime.

A ghost cost regime exists when significant resource consumption occurs without proportional task progress. It includes ghost tokens, ghost planning iterations, and ghost tool-calls [25]. The concept is diagnostically useful because it captures a recurring pattern in recursive systems: activity remains visible, outputs may still be produced, and local metrics may remain nominal, while an increasing share of runtime expenditure is directed toward structurally unproductive execution.

This regime does not imply total system failure. Instead, it describes a condition in which recursive continuation absorbs a growing fraction of cost and coordination capacity. In that sense, the concept provides a bridge between execution topology and infrastructure economics. It helps translate structural runtime behaviour into operationally legible terms.

7.2. Application Mapping

Table 1. SORT Application Catalog Mapping for Agentic Amplification

Application	Structural Mapping
ai.13 — Agentic System Stability	Agentic execution can be represented as a recursive operator chain in which instability appears as uncontrolled recursion depth and the absence of a stable execution fixed point [25]
ai.04 — Runtime Control Coherence	Retry policies fragment coherence when the control layer loses global state awareness and local decisions override global execution constraints [26]
ai.52 — Weak Signal Aggregation	Noise is recursively re-injected, uncertainty is treated as signal, and amplification replaces filtering

These three applications span distinct stability dimensions. The first concerns semantic and task-level coherence in recursive agency. The second concerns logical coherence in runtime control and orchestration. The third concerns informational coherence in the treatment of weak or ambiguous signals. This separation matters because recursive instability rarely emerges along only one axis. A system can remain semantically plausible while losing control coherence, or preserve control logic while becoming vulnerable to weak-signal amplification.

Their orthogonality is therefore analytically useful. Each dimension can degrade even if the other two remain partially controlled. This makes the SORT mapping valuable not as an explanatory add-on, but as a structured vocabulary for the missing diagnostic layer identified above. In this paper, the framework serves exactly that role. It does not replace existing observability or evaluation methods. It clarifies the structural categories required to interpret recursive execution as a runtime system rather than only as a sequence of outputs.

8. Operational Consequences at Scale

The structural dynamics described in this paper carry direct operational and economic implications for hyperscale AI operators. Once recursive execution becomes a normal runtime pattern rather than a special-case orchestration feature, the relevant unit of infrastructure analysis changes. The challenge is no longer limited to serving more tokens at lower cost. It becomes the management of execution systems whose depth, branching behaviour, and retry propagation can vary materially across workloads, hardware classes, and control conditions.

For operators at hyperscale, this shift is significant because recursive execution redistributes where inefficiency appears. In conventional inference systems, cost and latency are often concentrated in the model invocation itself. In agentic systems, a growing share of cost can emerge from the surrounding execution structure: repeated re-entry, redundant planning, branching tool dependencies, and the persistence of partial or weak signals. This means that operational performance depends increasingly on how well recursive workloads remain bounded, coherent, and observable under production conditions.

8.1. Cost Explosion Under Recursive Load

One immediate consequence of recursive execution is that ghost costs can scale superlinearly with agentic complexity [25]. As the number of tools, retries, branches, and intermediate planning stages increases, the relationship between user-visible task complexity and actual runtime expenditure becomes less predictable. In this regime, retry amplification can become a leading cost driver in agentic inference fleets, not because individual calls are unusually expensive, but because the execution graph expands beyond the minimum path required for task completion.

Observed cost escalation already suggests the practical importance of this effect. Three-agent workflows can move from $5–50 in demonstrations to $18,000–90,000 monthly at production scale [4]. This is not simply a matter of larger traffic volume. It reflects the way recursive execution compounds token usage, coordination overhead, and tool interaction cost across sustained deployment. Agentic workloads consume 20–30 times as many tokens as conventional inference [1,3], which multiplies exposure to structurally unproductive runtime activity.

This also helps explain why conventional cost optimization policies can produce uneven results across recursive workloads. Token limits, retry caps, and timeout constraints are often introduced to improve efficiency, but under some conditions they can amplify instability by forcing premature re-entry into planning loops or encouraging repeated partial execution rather than coherent completion. In such cases, local policy optimization can improve per-event efficiency while worsening total execution-path efficiency. McKinsey reports that 62% of organizations experienced unexpected cost overruns [5], while CX Today reports that 73% exceeded budget [6]. These figures suggest that recursive workload economics increasingly require structural visibility rather than cost control at the output layer alone.

8.2. Fleet Management Under Agentic Workloads

Recursive execution also changes the operational meaning of fleet management. In heterogeneous infrastructure environments, hardware classes differ not only in raw throughput or latency, but also in how they shape retry behaviour, coordination timing, and execution-path variability. Different accelerator classes expose distinct latency profiles, memory hierarchies, and throughput regimes [15]. For agentic workloads, these differences can propagate into the execution graph itself.

Routing agentic workloads across accelerator classes therefore produces variable execution topologies rather than merely variable performance numbers. The same task can follow a different structural path depending on where it is scheduled, how quickly intermediate results return, and whether partial outputs trigger additional planning or retries. At hyperscale, this implies that fleet behaviour cannot be understood fully through utilization statistics alone. The fleet is also participating in the shaping of recursive execution.

Energy-aware scheduling deepens this effect because it introduces another autonomous optimization loop into the runtime environment [16,17]. Retry policies seek successful completion under uncertainty. Scheduling policies seek efficient resource allocation under demand fluctuation and power constraints. When both interact, the resulting execution path may reflect not just task logic but also infrastructure pressure. Power throttling and grid constraints can therefore make retry storms operationally significant through unpredictable compute-demand spikes [18]. For hyperscalers, this means that recursive workload management increasingly sits at the intersection of orchestration, routing, and energy coordination.

8.3. Enterprise Readiness and Agent Reliability

Enterprise readiness introduces a further operational consequence. Approximately 40% of agentic AI projects fail before reaching production [1,2]. This figure does not imply that agentic systems are intrinsically unreliable. Rather, it suggests that moving from demonstration to production requires more than improving capability. It requires sufficient control over recursive execution structure, tool interactions, and runtime visibility.

This is increasingly recognized in security and reliability discourse. OWASP ASI08 classifies cascading failures as a distinct security risk class for agentic systems [24]. From an enterprise perspective, the implication is constructive. Production deployment requires bounded recursion, retry coherence, signal filtering before re-entry, and global execution constraints. These are not merely governance preferences. They are operational requirements for making recursive systems legible, reproducible, and economically sustainable.

Under these conditions, the next bottleneck is not only model quality, GPU supply, or training data. It is the runtime stability of agentic execution graphs [27]. Organizations that can control retry cascades and tool-loop amplification will be better positioned to manage the next margin layer in cost, reliability, and enterprise readiness. For hyperscalers, this makes structural runtime diagnostics more than a theoretical refinement. It makes them a practical component of large-scale deployment maturity.

9. Strategic Conclusion

AI infrastructure is not only moving from models to agents. It is moving from bounded computation toward recursive execution as a normal operating condition. This transition changes the practical meaning of scale for hyperscalers, AI platforms, and large inference operators. Performance is no longer determined only by the efficiency of a single model pass. It is increasingly determined by how planning, tool invocation, retry, branching, and re-entry remain coordinated across the full execution graph under real production constraints.

Within this shift, retry cascades should be understood as a structural runtime regime rather than as a collection of isolated failure modes. Their significance lies not in the existence of retries themselves, since retries are often necessary and operationally beneficial, but in the way recursive re-entry can extend execution beyond the minimum path required for coherent completion. Nonlinear token growth, geometric tool-call expansion, and recursive weak-signal aggregation interact to produce increasing resource consumption without proportional task progress. At hyperscale, this makes recursive topology a relevant layer of infrastructure analysis in its own right.

These dynamics remain only partially visible to conventional benchmark evaluation and operational monitoring. Benchmarks measure task capability under controlled conditions. Observability systems measure outputs, latency, utilization, and service-level health. Both remain necessary. Neither, however, directly captures execution topology as a runtime object in its own right. As agentic systems become more deeply integrated into production environments, this topological blind spot becomes operationally significant because it affects cost predictability, coordination efficiency, and deployment confidence.

What is therefore missing is a structural diagnostic layer that can observe and analyse execution topology directly. Such a layer would make retry propagation, branching behaviour, signal re-entry, and control coherence more legible under production conditions. Its purpose would not be to replace existing observability, optimization, or governance systems, but to complement them by clarifying how recursive execution remains efficient, bounded, and coherent at scale. For hyperscalers, this is not an abstract research refinement. It is increasingly relevant to architecture risk, runtime efficiency, and fleet-level decision making.

In this paper, SORT provides one vocabulary for that layer through ai.13, ai.04, and ai.52. Used in this way, the framework does not function as an abstract explanatory overlay. It functions as a compact structural language for identifying where recursive systems remain aligned with task progress and where execution complexity begins to outpace structural visibility. For hyperscalers and platform operators, that distinction is practical now. The next performance frontier is not only faster inference. It is more structurally intelligible recursion.

The strategic implication is direct. Organizations that can see recursive structure earlier will be better positioned to improve runtime efficiency, reduce hidden cost accumulation, and strengthen the operational readiness of agentic deployments. The relevant question is no longer only whether an

agentic system performs. It is whether the execution structure that produces that performance remains sufficiently visible, coherent, and governable as scale increases.

Agentic systems do not fail because they are wrong. They fail when recursive execution loses structural boundedness.

As agentic infrastructure scales, structural diagnostics become increasingly important for keeping operational understanding aligned with execution complexity.

Acknowledgments: The author acknowledges prior internal architectural work that informed the conceptual development of the diagnostic perspective presented in this paper.

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

Data Availability Statement: No new data were generated in this study. All referenced data are available in the cited publications.

1. Gartner, Inc. (2025). Predicts 2025: Agentic AI — The Evolution of Experience and Productivity. *Gartner Research Report*.
2. Galileo. (2025). The Hidden Costs of Agentic AI: Why 40% of Projects Fail Before Production. *Galileo AI Blog*. https://galileo.ai/blog/hidden-cost-of-agentic-ai
3. Introl. (2025). AI Agent Infrastructure: What Autonomous Systems Require. *Introl Blog*. https://introl.com/blog/ai-agent-infrastructure-autonomous-systems-compute-requirements-2025
4. Stevens Institute of Technology. (2026). The Hidden Economics of AI Agents: Managing Token Costs and Latency Trade-offs. *Stevens Online*. https://online.stevens.edu/blog/hidden-economics-ai-agents-token-costs-latency/
5. McKinsey & Company. (2025). The State of AI in 2025: Generative AI Adoption and Cost Economics. *McKinsey Global Institute*.
6. CX Today. (2026). The Agentic AI Cost Problem: Calculating TCO for Agentic AI. *CX Today*. https://www.cxtoday.com/security-privacy-compliance/the-agentic-ai-cost-problem/
7. Epoch AI. (2025). LLM Inference Prices Have Fallen Rapidly But Unequally Across Tasks. *Epoch AI Data Insights*. https://epoch.ai/data-insights/llm-inference-price-trends
8. Yao, S.; Zhao, J.; Yu, D.; Du, N.; Shafran, I.; Narasimhan, K.; Cao, Y. (2023). ReAct: Synergizing Reasoning and Acting in Language Models. *Proceedings of the International Conference on Learning Representations (ICLR)*.
9. Shinn, N.; Cassano, F.; Labash, B.; Gopinath, A.; Narasimhan, K.; Yao, S. (2023). Reflexion: Language Agents with Verbal Reinforcement Learning. *Advances in Neural Information Processing Systems (NeurIPS)*, **36**.
10. OpenAI. (2026). Introducing GPT-5.3 Codex. *OpenAI Blog*. https://openai.com/index/introducing-gpt-5-3-codex/
11. OpenAI. (2026). OpenAI and Amazon Partnership. *OpenAI Blog*. https://openai.com/index/amazon-partnership/
12. Anthropic. (2026). Claude Sonnet 4.6 Model Card. *Anthropic*. https://www.anthropic.com/claude/sonnet
13. Alibaba Cloud. (2026). Qwen 3.5: 397B MoE with 17B Active Parameters. *Alibaba Cloud Blog*. https://www.alibabacloud.com/blog/qwen-3-5
14. Reuters. (2026). Nvidia Plans New Chip to Speed AI Processing. *Reuters Business*. https://www.reuters.com/business/nvidia-plans-new-chip-speed-ai-processing-wsj-reports-2026-02-28/
15. NVIDIA Developer Blog. (2026). Automating Inference Optimizations with NVIDIA TensorRT-LLM AutoDeploy. *NVIDIA Developer Blog*. https://developer.nvidia.com/blog/automating-inference-optimizations-with-nvidia-tensorrt-llm-autodeploy/
16. Goldman Sachs Research. (2025). Data Center Power Demand: The 6 Ps Driving Growth and Constraints. *GS SUSTAIN Report*.
17. International Energy Agency. (2025). Energy and AI: Energy Demand from Data Centres. *IEA Special Report*. Paris. https://www.iea.org/reports/energy-and-ai
18. Deloitte. (2025). Can US Infrastructure Keep Up with the AI Economy? *Deloitte Insights*. https://www.deloitte.com/us/en/insights/industry/power-and-utilities/data-center-infrastructure-artificial-intelligence.html
19. Liu, X., et al. (2023). AgentBench: Evaluating LLMs as Agents. *Proceedings of ICLR 2024*.
20. Liang, P., et al. (2023). Holistic Evaluation of Language Models. *Transactions on Machine Learning Research*.

21. NVIDIA Research. (2026). SPEED-Bench: A Unified and Diverse Benchmark for Speculative Decoding. *NVIDIA Research Publications*. https://research.nvidia.com/publication/2026-02_speed-bench-unified-and-diverse-benchmark-speculative-decoding
22. ISO-Bench Authors. (2026). ISO-Bench: Can Coding Agents Optimize Real-World Inference Workloads? *arXiv preprint*. https://arxiv.org/html/2602.19594v1
23. Zhou, S., et al. (2023). WebArena: A Realistic Web Environment for Building Autonomous Agents. *Proceedings of ICLR 2024*. https://webarena.dev/
24. OWASP Foundation. (2025). OWASP Top 10 for Agentic AI Security — ASI08: Cascading Failures. *OWASP*. https://owasp.org/www-project-top-10-for-agentic-ai/
25. Wegener, G.H. (2026). 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. *MDPI Preprints*. doi:10.20944/preprints202601.1741.v1
26. Wegener, G.H. (2026). SORT-AI: Runtime Control Coherence in Large-Scale AI Systems. *MDPI Preprints*. doi:10.20944/preprints202601.0298.v1
27. Wegener, G.H. (2026). SORT-AI: Structural Efficiency Recovery in Hyperscale AI Systems. *MDPI Preprints*. doi:10.20944/preprints202602.0015.v1