Article

Cost-Driven Inference Optimization as a Structural Drift Amplifier: A Diagnostic Perspective on Runtime Geometry

Gregor Herbert Wegener

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

Abstract

Your inference costs dropped 80% this year. Your agent task completion reliability dropped 12%. These metrics often appear unrelated. They are not. As AI infrastructure shifts toward inference-dominated economics, optimization efforts increasingly focus on cost per token, throughput, and energy efficiency. This shift is economically rational but architecturally consequential. As inference systems are increasingly optimized for cost efficiency, the control geometry of the system begins to shift: branching width narrows, retry depth shrinks, context persistence becomes less persistent, tool call ordering shifts, and temporal budgets compress. The result is structural drift without model modification—systems behave differently not because the model changed, but because the execution topology surrounding it has been altered. This phenomenon remains largely invisible to standard monitoring practices, which typically measure output quality rather than execution topology. Agent-based systems are particularly sensitive to these effects, as token consumption can multiply 20–30 times under autonomous operation, creating strong economic pressure to constrain precisely the behaviors agents require for reliable reasoning. This paper introduces a diagnostic vocabulary for analyzing cost-induced structural drift, maps the phenomenon to the SORT application catalog, and discusses strategic implications for hyperscale AI operators.

Keywords: inference optimization; structural drift; runtime geometry; control coherence; AI infrastructure; agentic systems; execution topology; cost optimization; hyperscale infrastructure

---

1. The Cost Optimization Paradox

1.1. The Paradox

Your inference costs dropped 80% this year. Your agent task completion reliability dropped 12%. You treat these as separate metrics. They are not.

Across the AI industry, teams are optimizing inference cost efficiency because inference increasingly dominates operational expenditure. Many of these efforts deliver exactly what they promise. Cost per token declines, throughput rises, and utilization improves. In parallel, teams often observe shifts in deployed behavior. Agents show higher variance across similar tasks, reasoning trajectories become less reproducible across runs, and completion patterns change under load, even when the underlying model remains unchanged.

The connection is structural rather than model-centric. Cost optimization does not only change margins. It reshapes the runtime environment in which decisions are formed. The same mechanisms that improve efficiency also modify execution topology. They change how work is scheduled, how intermediate state is retained, how recovery attempts are bounded, and how multi step reasoning unfolds within a given budget. These effects are easy to miss because they sit below the level of model weights and above the level of output metrics.

This paper examines that linkage, and provides a diagnostic frame for discussing it in operational terms.

1.2. The Inference Economy

For most of the modern AI era, training was the dominant economic event. Training large models required large compute clusters, long runtimes, and substantial capital investment. Inference was comparatively lightweight.

This balance has inverted. The inference flip, the point at which operational inference expenditure surpasses initial training costs, is now an operational reality in many deployments [2]. Stanford's 2025 AI Index Report documents that inference costs for GPT-3.5 level performance dropped more than 280 fold between late 2022 and late 2024 [1]. Yet frontier systems with higher test time compute remain expensive to run at scale. The combined result is a sustained industry focus on inference efficiency as a primary lever for economic viability.

Hyperscaler capital expenditure reflects this shift. Projections indicate a 175% surge in data center power demand by 2030 [4]. Energy is becoming a binding constraint. 72% of US power and data center executives now cite grid stress as their leading infrastructure challenge [7]. Under these conditions, inference optimization is not a secondary concern. It is a dominant operational priority that influences how systems are engineered and operated.

1.3. Scope and Framing

This paper treats cost optimization as an architectural force, not only an economic one. The analysis is diagnostic rather than prescriptive. It introduces vocabulary for understanding structural effects without advocating specific interventions.

The focus is runtime control geometry, meaning the structural arrangement of execution pathways, decision points, and state transitions that govern system behavior at runtime, rather than model internals. Weight updates, training dynamics, and mechanistic interpretability are outside scope. The concern is how optimization of the inference environment can shift system behavior without changing model parameters.

The intended audience includes infrastructure economics teams, principal architects, runtime engineers, and AI governance functions within hyperscaler organizations.

2. Economic Drivers of Inference Optimization

The focus on inference efficiency is not arbitrary. It emerges from several structural dynamics that shape large-scale AI infrastructure today: capital expenditure discipline, increasing hardware heterogeneity, energy availability, and the token economics of agentic systems. Each of these dynamics independently encourages optimization of inference workloads. In combination, they establish inference efficiency as a central engineering objective for hyperscale deployments.

2.1. Capital Expenditure Pressure

Hyperscaler investment in AI infrastructure now exceeds $200 billion annually across the major cloud providers. Unlike training clusters, which typically operate in concentrated bursts aligned with model development cycles, inference infrastructure must sustain continuous operation across large user populations.

This operational model places strong emphasis on utilization. Infrastructure must remain active and responsive while serving highly variable demand patterns. As a result, metrics such as cost per token, cost per request, and cost per user session become central operational indicators for infrastructure teams.

These incentives naturally extend across the entire serving stack. Request routing layers, batching policies, speculative decoding strategies, and scheduling mechanisms are continuously refined to improve efficiency. Each improvement contributes to lower marginal cost and higher throughput. In practice, inference systems are therefore in a constant state of economic optimization.

2.2. Hardware Diversification

The hardware landscape supporting inference workloads is becoming increasingly heterogeneous. TPUs, Trainium, Inferentia, and a growing set of custom accelerators are entering production environments alongside established GPU infrastructure. Hyperscalers now operate fleets composed of multiple accelerator classes, connected through routing layers that distribute requests according to availability, latency targets, and cost profiles.

This diversification expands operational flexibility but also increases architectural complexity. Different accelerators expose different memory hierarchies, latency characteristics, and throughput regimes. Routing layers therefore do more than distribute load. They implicitly determine how requests interact with underlying hardware characteristics.

In large fleets, the same request may be executed on different hardware depending on routing decisions at runtime. These variations are typically small and well within expected operational tolerances, yet they illustrate how infrastructure design choices influence the execution environment surrounding the model.

2.3. Energy as a Binding Constraint

Energy availability is increasingly relevant in large-scale inference deployments. Data centers in Northern Virginia already consume approximately 26% of the region's electricity. The International Energy Agency projects that global data center electricity demand could exceed 1,700 TWh by 2035 under high-adoption scenarios [6].

In this environment, energy-aware scheduling becomes part of normal infrastructure operation. Clusters may adjust workload distribution in response to power budgets, thermal limits, or regional grid conditions. These adjustments are typically handled transparently within the infrastructure layer.

From an application perspective, the system continues to operate normally. From an infrastructure perspective, however, the timing and placement of execution may shift in response to energy conditions. Energy efficiency therefore becomes another dimension along which inference systems are continuously optimized.

2.4. Agent-Driven Token Growth

Agentic AI systems introduce an additional dimension to inference economics. Autonomous workflows often involve iterative reasoning, tool invocation, and multi-step planning loops. As a result, token consumption in agentic workflows can be significantly higher than in standard prompt-response interactions. Some studies report increases of 20 to 30 times in total token usage compared to simple generation tasks [9].

These workloads naturally attract optimization efforts. Infrastructure teams look for ways to improve efficiency while maintaining system responsiveness. Scheduling policies, retry budgets, context management, and tool invocation patterns all become potential optimization surfaces.

From a systems perspective, this dynamic illustrates an important characteristic of agent-based architectures: the behaviors that enable complex reasoning also increase infrastructure demand. Optimization therefore focuses on maintaining performance and responsiveness while supporting these richer execution patterns [10].

3. Inference Optimization Mechanisms

Large-scale inference systems rely on a range of optimization techniques that improve efficiency along specific operational dimensions. These mechanisms are widely deployed across modern serving stacks and represent standard engineering practice for operating AI systems at hyperscale. Table 1 summarizes several common techniques, describing their operational function and the local efficiency benefit they provide.

Table 1. Inference Optimization Mechanisms

Mechanism	What It Does	Local Benefit
Speculative Decoding	Uses draft models to propose token sequences that target models verify in parallel	2–3x inference speedup without quality loss
Context Compression	Truncates or summarizes intermediate state to reduce memory footprint	Memory savings, increased batch capacity
Dynamic Routing	Distributes workloads across heterogeneous accelerators based on availability and cost	Fleet utilization, cost optimization
Multi-Request Scheduling	Batches and reorders requests to optimize queue throughput	Improved throughput, reduced latency variance
Retry Policy Tuning	Bounds recovery attempts to control loop costs	Predictable resource consumption, cost stability
Token Truncation	Enforces hard limits on generation length	Latency guarantees and bounded compute usage
Power-Aware Scheduling	Adjusts scheduling policies to stay within energy budgets	Energy efficiency and infrastructure stability

Each mechanism in Table 1 provides a clear local advantage. Speculative decoding, introduced by Leviathan et al., demonstrates how parallel verification can accelerate token generation substantially without reducing output quality [5]. Context compression enables larger effective batch sizes. Dynamic routing improves fleet utilization across heterogeneous accelerators. Multi-request scheduling stabilizes throughput in high-demand environments.

Individually, these techniques represent well-understood and widely adopted infrastructure improvements. Together they form the operational toolkit that enables modern inference systems to scale economically.

The interesting question is not whether these mechanisms are effective—they clearly are—but how their combined operation shapes the runtime environment in which models execute. The following sections examine that interaction at the system level.

4. Runtime Geometry Change

The optimization mechanisms described above do more than improve performance metrics. They also influence the structural environment in which AI systems execute. This section introduces the concept of control geometry and identifies five structural variables that are commonly affected by cost-oriented optimization policies.

4.1. Defining Control Geometry

Control geometry refers to the structural arrangement of execution pathways, decision points, and state transitions that govern system behavior at runtime. Rather than describing performance outcomes, it describes the topology of the execution process itself.

Traditional metrics such as latency, throughput, and error rate describe observable system performance. Control geometry describes the structural conditions that produce those outcomes. Two systems may present similar performance metrics while operating under different execution topologies, and those differences can become visible when workloads, infrastructure conditions, or optimization policies change.

When inference systems are optimized for cost efficiency, elements of the control geometry often shift as well. The system's effective decision space—the range of execution paths, recovery strategies,

and reasoning durations available during runtime—is shaped by scheduling policies, resource limits, and optimization mechanisms embedded in the serving stack.

4.2. The Five Structural Variables

Cost-oriented optimization commonly influences five structural variables that together characterize the control geometry of an inference system:

Branching width describes how many alternative execution paths are explored before committing to a solution. Optimization policies may reduce speculative exploration in order to minimize token expenditure, which can lead to more selective exploration of candidate reasoning paths.

Retry depth describes how many recovery attempts are permitted before an execution loop terminates. Cost-bounded retry policies introduce predictable limits on iteration cycles, helping control resource consumption while shaping how systems recover from intermediate errors.

Context persistence describes how much intermediate state is preserved across reasoning steps. Techniques such as context compression or memory management can reduce the footprint of stored state while influencing how prior reasoning steps remain available to subsequent ones.

Tool call ordering describes the sequence in which external systems are invoked. Scheduling policies that favor faster completion paths may influence the order in which tools are called, affecting the structure of multi-step interactions between agents and external services.

Temporal budget describes how much reasoning time is allocated before execution concludes. Latency constraints and service-level objectives can define practical time budgets for reasoning chains, shaping how long multi-step processes continue before producing an output.

These variables are not performance metrics. Together they describe the structural configuration of the execution environment in which the model operates.

4.3. The Agent Vignette

Consider an agent performing a multi-step software debugging task. In a relatively unconstrained environment, the agent may explore multiple hypotheses about the root cause of a bug, invoke diagnostic tools, iterate across alternative approaches, maintain state about previously attempted solutions, and continue reasoning until the problem is resolved or clearly outside the agent's capabilities.

In a cost-optimized environment, the same agent operates within a different set of structural parameters. Branching width may be reduced, limiting the number of hypotheses explored. Retry policies may bound how many recovery attempts are executed. Context management policies may compress intermediate state. Tool invocation order may be influenced by scheduling priorities. Temporal budgets may define practical limits on how long the reasoning process continues.

The underlying model remains identical in both scenarios. What differs is the structural environment in which the model executes. Changes in control geometry therefore provide one explanation for why systems with unchanged model weights may exhibit different runtime behaviors under different optimization regimes.

5. Why Observability Does Not Detect This

The structural effects described in the previous section are rarely visible through standard monitoring frameworks. This section explains why this is the case.

5.1. The Benchmark Assumption

Most evaluation frameworks assume a stable execution environment. Benchmark suites typically measure model capability under controlled conditions: fixed context sizes, deterministic scheduling, unconstrained retry budgets, and isolated request execution. These assumptions are necessary for reproducible measurement.

Production inference environments operate under very different conditions. Workloads fluctuate, routing decisions adapt dynamically, hardware availability changes, and optimization policies con-

tinuously adjust execution parameters. The environment assumed by benchmarks therefore differs structurally from the environment in which systems actually operate.

Benchmark scores measure model capability in a controlled setting. They do not necessarily describe system behavior under real operational constraints such as cost optimization, heterogeneous hardware fleets, or dynamic scheduling.

5.2. What Monitoring Measures

Operational observability frameworks focus on performance and reliability metrics. Typical monitoring systems collect metrics, logs, and traces describing latency distributions, throughput rates, token consumption, request errors, and output quality signals.

These measurements are essential for operating large-scale inference systems. They allow teams to manage performance, detect failures, and control operational cost.

However, these measurements primarily describe outcomes. They report what the system produced and how efficiently it produced it. They do not directly describe the structural conditions under which execution occurred.

A system may maintain stable latency and throughput metrics while the structural configuration of its execution environment is gradually changing.

5.3. What Monitoring Does Not Capture

Standard observability frameworks are not designed to capture changes in execution topology.

They typically do not record whether identical requests follow different reasoning trajectories under varying load conditions. They do not measure whether speculative exploration width has narrowed due to cost optimization. They do not track how retry policies alter recovery behavior over time. They do not represent how context compression may change the availability of intermediate reasoning state. They also do not capture how queue scheduling decisions may alter the ordering and timing of agent interactions with external systems.

In other words, monitoring systems observe outputs and operational metrics, while the structural changes described in the previous section occur within the execution topology itself.

5.4. The Operational Consequence

The practical consequence is a diagnostic gap. Operators may observe shifts in system behavior or reliability without a clear explanation of the underlying cause.

Industry surveys illustrate the broader operational complexity surrounding AI infrastructure deployment. A McKinsey survey found that 62% of organizations using token-based AI services experienced at least one month of unexpected cost overruns in their first year of implementation [3]. Organizations therefore intensify optimization efforts, while system behavior continues to evolve under those changing constraints.

Because the structural effects occur in the execution topology rather than in the observable performance metrics, the relationship between cost optimization and behavioral change can remain difficult to identify.

The result is not a failure of monitoring systems. Rather, it reflects that existing observability tools were designed to measure performance outcomes, not the structural configuration of the execution environment.

6. Runtime Interaction Effects

The optimization mechanisms described in Section 3 rarely operate independently in production environments. Instead, they interact continuously during runtime. These interactions can produce system-level behaviors that are not directly predictable from the properties of the individual mechanisms.

6.1. Composition Creates Emergence

Each optimization mechanism is individually correct and locally beneficial. Speculative decoding improves token generation throughput. Context compression reduces memory pressure. Dynamic routing improves accelerator utilization. These benefits are well understood and widely adopted across modern inference infrastructure.

When these mechanisms operate simultaneously, however, their composition introduces interaction dynamics that extend beyond component-level analysis. Optimization policies targeting different operational objectives may influence one another indirectly. A scheduler optimizing for queue throughput may interact with retry policies designed to control loop cost. A context compression layer may remove intermediate state that speculative decoding would otherwise exploit. Power-aware throttling may introduce timing variability that influences the sequence of downstream tool calls.

The result is an emergent execution environment in which system-level behavior reflects the interaction of optimization mechanisms rather than the isolated design of any single mechanism.

6.2. Specific Interaction Patterns

Table 2 illustrates several interaction patterns that can arise when optimization mechanisms compose at runtime.

Table 2. Runtime Interaction Patterns

Interaction	Mechanism A	Mechanism B	Emergent Effect
Speculation + Scheduling	Speculative decoding	Queue batching	Branching variance across queue positions
Compression + Routing	Context compression	Hardware routing	Context availability varies across accelerator architectures
Retry + Power	Retry policies	Power throttling	Recovery behavior becomes dependent on cluster power state
Truncation + Agents	Token limits	Multi-step planning	Planning horizon reduction under cost constraints

Each interaction in Table 2 illustrates how locally beneficial mechanisms can influence system behavior when combined. For example, speculative decoding combined with queue scheduling may lead to different branching trajectories depending on queue position. Context compression combined with heterogeneous routing may produce different context availability depending on the accelerator handling the request. Retry policies interacting with power management may alter recovery dynamics depending on cluster energy conditions. Token truncation interacting with agent planning may influence the depth of multi-step reasoning processes.

6.3. The Structural Consequence

These interaction effects accumulate across large-scale inference systems. Individual effects are often subtle when considered in isolation. However, modern inference infrastructure processes millions of requests across heterogeneous hardware fleets under continuously changing load conditions.

Each request therefore experiences a slightly different combination of queue position, routing decision, energy state, and optimization policy. Over time, these variations produce system-level behavioral patterns that reflect the aggregate influence of the optimization environment.

The structural consequence is that system behavior gradually aligns with the constraints and incentives embedded in the optimization regime. Because these dynamics occur at the interaction layer

of the execution environment, they are typically not visible through standard observability frameworks focused on performance metrics.

7. Structural Diagnostics as a Missing Layer

The phenomena described in the previous sections suggest a limitation in current observability practice. What appears to be missing is not additional performance monitoring but a complementary category of observation: structural diagnostics.

7.1. The Measurement Category Gap

The challenge is not measurement precision but measurement category.

Performance metrics answer questions such as: How fast does the system respond? How much does each request cost? How accurate are the outputs? These questions are essential for operating large-scale inference systems, and modern observability frameworks address them effectively.

Structural diagnostics address a different question: how does the execution topology of the system evolve under operational constraints and optimization policies?

This question lies largely outside the scope of current monitoring practices. Increasing the granularity of latency histograms does not reveal changes in control geometry. More detailed cost accounting does not expose shifts in exploration width or retry topology. The underlying phenomena remain difficult to observe because the instrumentation is designed to measure performance outcomes rather than structural execution patterns.

The gap is therefore categorical rather than quantitative.

7.2. What Structural Diagnostics Would Observe

A structural diagnostic layer would focus on how execution paths evolve under changing operational conditions.

Rather than measuring only latency distributions, it would examine how scheduling policies influence which reasoning trajectories requests follow under different load regimes. Rather than measuring only memory utilization, it would observe whether context transformations affect the continuity of intermediate reasoning state. Rather than measuring only throughput improvements, it would examine how speculative execution alters exploration patterns. Rather than measuring only hardware utilization, it would consider how routing across heterogeneous accelerators influences execution topology.

From this perspective, large-scale inference systems can be analyzed as distributed control networks whose behavior emerges from the interaction of scheduling policies, resource constraints, and execution pathways.

7.3. The SORT Diagnostic Perspective

The SORT framework proposes a diagnostic perspective for analyzing structural effects in large-scale inference systems[1]. Rather than introducing new performance metrics, it provides conceptual lenses for examining how system structure evolves under optimization pressure.

Four diagnostic perspectives are particularly relevant in the context described in this paper:

Runtime Control Coherence examines whether execution behavior remains consistent across different runtime regimes. If scheduling policies or cost constraints alter task execution patterns depending on load conditions, the system's control coherence may vary.

Structural Drift Diagnostics examines whether optimization policies gradually alter the decision topology of the system. If identical requests follow different execution paths as infrastructure policies evolve, the system may be experiencing structural drift.

[1] For formal definitions, see [11] and [12].

Inference Pipeline Coherence examines whether interactions between stages of the serving stack maintain a stable execution geometry. If pipeline interactions introduce behavioral variance, coherence across the serving architecture may be affected.

Interconnect Stability examines whether routing across heterogeneous accelerator classes influences reasoning continuity. If infrastructure state changes execution behavior, system reliability becomes partially dependent on hardware topology.

These perspectives are not operational metrics. They function as diagnostic lenses that help identify structural phenomena that traditional performance observability does not directly capture.

8. SORT Application Mapping

The structural mechanisms analyzed in this paper correspond to diagnostic perspectives within the broader SORT application catalog. Rather than introducing isolated concepts, the phenomena discussed here can be positioned within an existing diagnostic framework designed to analyze structural behavior in large-scale AI systems.

Table 3 provides a structured mapping between the mechanisms described in this paper and the corresponding diagnostic applications in the SORT catalog.

Table 3. SORT Application Mapping

Structural Mechanism	SORT Application	Diagnostic Function
Control geometry shift	ai.04 Runtime Control Coherence	Coherence analysis across runtime regimes
Drift without weight change	ai.02 Structural Drift Diagnostics	Detection of execution-layer drift over time
Serving stack interaction	ai.27 Inference Pipeline Control Coherence	Stability analysis across pipeline stages
Hardware-dependent behavior	ai.07 Accelerator Runtime Control	Diagnostics of hardware–execution coupling
Heterogeneous fleet routing	ai.01 Interconnect Stability Control	Cross-accelerator runtime stability analysis
Agent reliability variance	ai.13 Agentic System Stability	Multi-step task coherence diagnostics
Cross-cluster signal aggregation	ai.52 Deployment Drift Signal Aggregation	Distributed structural drift signal detection

Within this mapping, two diagnostic perspectives are central to the phenomena analyzed in this paper: ai.04 (Runtime Control Coherence) and ai.02 (Structural Drift Diagnostics). These applications focus on detecting changes in control geometry and identifying behavioral drift that occurs without modification of model weights.

For agentic workloads, ai.13 (Agentic System Stability) becomes particularly relevant. Multi-step autonomous systems depend heavily on retry topology, exploration width, and planning continuity. Structural constraints introduced by cost optimization therefore have a disproportionate influence on agent reliability.

For heterogeneous infrastructure deployments, ai.01 (Interconnect Stability Control) and ai.07 (Accelerator Runtime Control) provide diagnostic lenses for understanding how routing decisions across accelerator classes may influence execution behavior.

The mapping presented here is not intended as an exhaustive catalog. Instead, it illustrates how the structural dynamics described in this paper connect to a broader diagnostic framework for analyzing large-scale inference systems.

9. Strategic Implications for Hyperscalers

The structural effects described in this paper have practical implications for architecture design, infrastructure strategy, and operational management in hyperscale AI deployments.

9.1. Heterogeneous Accelerator Fleet Considerations

Dynamic routing across heterogeneous accelerator fleets introduces structural variance into inference execution. The same request processed on TPU, GPU, or custom ASIC infrastructure may traverse slightly different execution pathways depending on memory architecture, latency characteristics, and routing policies.

As a result, behavioral consistency across workloads may become partially dependent on accelerator class. Testing performed on one hardware environment does not always fully predict behavior on another.

Fleet management therefore benefits from structural coherence analysis in addition to traditional performance benchmarking. Understanding which workloads are tolerant of hardware-induced variance and which require stable execution topology becomes an important architectural consideration for heterogeneous infrastructure.

9.2. Agent Workloads and Cost Sensitivity

Agentic workloads exhibit particular sensitivity to cost optimization policies. Multi-step agents rely on iterative exploration, retries, and tool interaction in order to solve complex tasks. These behaviors naturally increase token consumption and infrastructure utilization.

Cost optimization policies that constrain iteration depth, retry loops, or planning horizon therefore influence the structural decision space available to agents. This does not necessarily prevent successful task completion, but it can alter how agents approach complex reasoning processes.

For large-scale deployments, agent architecture design may therefore benefit from explicit analysis of cost–reliability tradeoffs at the structural level. Understanding how optimization policies shape exploration depth and recovery behavior becomes particularly relevant for multi-step autonomous workflows.

9.3. Control-Layer Coupling and Infrastructure Migration

Optimization of inference pipelines for a specific runtime environment can create coupling at the execution-control layer. Systems tuned for a particular serving stack may exhibit different runtime characteristics when migrated to alternative infrastructure environments, even when model weights remain identical.

In this sense, infrastructure dependence can arise not only from model compatibility but also from the structural configuration of the execution environment. Execution topology, scheduling behavior, and hardware interaction patterns may all influence runtime behavior.

Migration planning therefore benefits from evaluating structural coherence across environments in addition to traditional capability verification.

9.4. Power-Constrained Inference Scaling

Energy availability is becoming an increasingly important factor in large-scale inference operations. Power-aware scheduling and energy-aware cluster management are likely to remain permanent components of hyperscale infrastructure design.

These mechanisms can influence execution timing, scheduling order, and recovery behavior across distributed inference systems. As a result, system behavior may vary slightly across different power envelopes or cluster load states.

Capacity planning therefore benefits from considering how power-management policies interact with execution topology and scheduling behavior in production environments.

9.5. Optimization Regimes and Runtime Dynamics

Inference optimization typically proceeds through incremental improvements applied across multiple layers of the serving stack. Each individual optimization is locally beneficial and often operationally invisible when considered in isolation.

Over time, however, the interaction of these mechanisms can gradually reshape the structural environment in which models execute. Because these dynamics occur at the level of execution topology, they may not be immediately visible through standard performance monitoring.

For organizations operating large-scale inference infrastructure, structural diagnostics can therefore complement existing observability practices by providing insight into how optimization policies influence execution geometry across evolving runtime environments.

10. Conclusion

The inference economy has become the dominant operational reality of modern AI infrastructure. Organizations are optimizing inference systems aggressively for token efficiency, energy consumption, and infrastructure utilization. By conventional economic metrics these optimizations are highly successful: costs decrease, throughput increases, and hardware fleets operate more efficiently.

However, inference optimization does not occur in an architectural vacuum.

When serving stacks are optimized for tokens, energy budgets, and utilization targets, the structural environment in which models execute also changes. This paper described five structural variables that shape the effective control geometry of inference systems: branching width, retry depth, context persistence, tool call ordering, and temporal budget. These variables influence the decision space available during execution.

Optimization mechanisms that operate across the inference stack interact dynamically at runtime. Their interactions reshape execution topology in ways that are not visible at the level of individual components. The resulting effect can be described as structural drift: systems exhibit behavioral variation even when model weights remain unchanged because the surrounding execution environment evolves.

Current observability frameworks are not designed to capture these dynamics. Benchmarks assume controlled environments that differ from production infrastructure, while operational monitoring primarily measures performance outcomes such as latency, throughput, and cost efficiency. These measurements remain essential for operating large-scale systems, but they do not directly reveal how execution topology evolves under optimization pressure.

The resulting gap is therefore categorical rather than quantitative. Understanding cost-induced behavioral change requires a diagnostic perspective capable of observing execution structure rather than only system outputs.

The SORT framework provides one such diagnostic vocabulary. The structural mechanisms analyzed in this paper correspond to diagnostic perspectives within the SORT application catalog, enabling analysis of runtime control coherence, structural drift, and execution-layer interactions across large-scale inference deployments.

The objective of this analysis is not to discourage optimization. Cost efficiency is a necessary condition for sustainable AI infrastructure. Rather, the objective is to understand how optimization reshapes the control surface of inference systems so that these effects can be observed and managed before they manifest as operational inconsistencies.

In large-scale deployments, structural diagnostics can therefore complement traditional observability by making execution topology visible as inference infrastructure continues to evolve.

Cost optimization is not neutral. It rewrites the geometry of control.

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. Maslej, N.; Fattorini, L.; Perrault, R.; et al. (2025). The AI Index 2025 Annual Report. *AI Index Steering Committee, Institute for Human-Centered AI, Stanford University*. Stanford, CA.
2. 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
3. McKinsey & Company. (2025). The State of AI in 2025: Generative AI Adoption and Cost Economics. *McKinsey Global Survey on AI*.
4. Goldman Sachs Research. (2025). Data Center Power Demand: The 6 Ps Driving Growth and Constraints. *GS SUSTAIN Report*.
5. Leviathan, Y.; Kalman, M.; Matias, Y. (2023). Fast Inference from Transformers via Speculative Decoding. *Proceedings of the 40th International Conference on Machine Learning (ICML '23)*.
6. International Energy Agency. (2025). Energy and AI: Energy Demand from Data Centres. *IEA Special Report*. Paris. https://www.iea.org/reports/energy-and-ai
7. 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
8. Stevens Institute of Technology. (2026). The Hidden Economics of AI Agents: Managing Token Costs and Latency Trade-offs. *Stevens Online Blog*. https://online.stevens.edu/blog/hidden-economics-ai-agents-token-costs-latency/
9. Gartner, Inc. (2025). Predicts 2025: Agentic AI — The Evolution of Experience and Productivity. *Gartner Research Report*.
10. 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
11. Wegener, G.H. (2026). SORT-AI: Runtime Control Coherence in Large-Scale AI Systems. *MDPI Preprints*. doi:10.20944/preprints202601.0298.v1
12. Wegener, G.H. (2026). SORT-AI: Structural Efficiency Recovery in Hyperscale AI Systems. *MDPI Preprints*. doi:10.20944/preprints202602.0015.v1