Article

Runtime Control Layer as a Hidden Performance Variable: Why Model Performance Is Increasingly Determined by Runtime Control Coherence

Gregor Herbert Wegener

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

Abstract

Modern large-scale AI systems increasingly operate within complex runtime environments that extend beyond model architecture itself. In many deployments, model execution is coordinated by an additional layer of scheduling, routing, scaling and optimization mechanisms that determine how and when inference workloads are executed across distributed infrastructure. This paper introduces the concept of a *Runtime Control Layer* to describe this emerging operational layer in AI systems. Within this layer, multiple optimization loops operate simultaneously, including scheduling policies, dynamic batching, speculative decoding controllers, routing decisions and autoscaling mechanisms. These mechanisms are typically designed to optimize specific performance objectives such as throughput, latency or infrastructure utilization. When combined at scale, however, these locally optimized mechanisms form a complex control topology that influences overall system behaviour. Empirical signals from large production deployments suggest that significant performance improvements can often be achieved through runtime control adjustments without modifying the underlying model architecture. The paper proposes a structural diagnostic perspective for analysing runtime control coherence and introduces a vocabulary for describing control interactions within large-scale AI inference environments. Strategic implications for hyperscale AI operators are discussed in the context of infrastructure efficiency, execution predictability and operational transparency.

Keywords: AI infrastructure; runtime control layer; control coherence; inference systems; optimization loops; execution topology; hyperscale computing; structural diagnostics; distributed inference; agentic systems

---

1. Executive Summary

Modern large-scale AI serving systems are increasingly structured around three distinct architectural layers. Beyond the model layer and the inference layer, production deployments now rely on an additional operational layer responsible for coordinating how inference workloads are executed across distributed infrastructure. This layer can be described as the *runtime control layer*. It includes scheduling policies, routing mechanisms, speculative decoding controllers [4], caching strategies, autoscaling decisions, GPU allocation policies, and power-aware execution constraints.

Each of these mechanisms is typically designed to optimize a specific performance objective such as latency, throughput, or infrastructure utilization. In isolation, these optimization mechanisms are locally effective and often essential for operating large-scale inference fleets. In aggregate, however, they form a complex control topology in which multiple autonomous optimization loops interact simultaneously.

When these loops operate without explicit coherence constraints, the resulting system behaviour can become difficult to predict or reproduce. Operational signals observed across large deployments

include performance volatility, uneven GPU utilization, unstable throughput characteristics, and variations in execution behaviour across otherwise similar runs.

Evidence from large production environments suggests that runtime infrastructure adjustments can produce substantial performance improvements even when the underlying model architecture remains unchanged. For example, Meta engineering teams have reported throughput improvements of approximately 35% through infrastructure and runtime adjustments alone. This observation aligns with broader economic trends in the AI ecosystem: commodity inference costs have declined dramatically over time [1], while frontier systems that rely on dynamic test-time compute remain operationally expensive and infrastructure intensive [15].

These developments indicate a structural shift in where performance improvements are generated within deployed AI systems. While model architecture remains central to capability development, operational behaviour is increasingly shaped by the configuration and interaction of runtime control mechanisms.

This paper introduces a diagnostic vocabulary for analysing runtime control coherence and proposes a structural perspective for examining interactions between optimization loops in large-scale inference environments. The phenomenon is mapped to the SORT application catalog [17], and strategic implications are discussed for hyperscale AI operators responsible for operating large distributed inference infrastructures.

2. The Economic Shift

Large-scale AI systems are undergoing a structural economic transition. Early generations of modern machine learning systems were dominated by training costs: the primary capital expenditure occurred during model development, while inference was comparatively lightweight and operationally straightforward.

This balance has shifted. As models are deployed across large user populations and integrated into continuous workflows, inference infrastructure has become the dominant operational cost center. This transition fundamentally changes where optimization pressure is applied within AI systems. Economic constraints that previously shaped model development are now increasingly shaping runtime infrastructure.

The following sections examine several converging industry trends that together explain this shift: the transition to inference-dominated economics, large-scale infrastructure investment and hardware diversification, emerging energy constraints in AI datacenters, and the rapid growth of agent-driven token consumption.

2.1. The Inference Flip

Historically, the most resource-intensive phase of large-scale AI development was model training. Training required large compute clusters operating over extended periods, while inference workloads were comparatively modest and inexpensive to operate.

In many contemporary deployments this relationship has inverted. The point at which cumulative inference expenditure surpasses the cost of training the model itself has been described as the *inference flip* [2]. For systems with large and sustained user populations, inference costs now dominate the lifecycle economics of the model.

Industry data illustrates the scale of the change. The Stanford AI Index 2025 reports that the cost of delivering GPT-3.5 level performance dropped by more than 280-fold between late 2022 and late 2024 [1]. While this reduction reflects rapid improvements in hardware efficiency and software optimization, it also reflects the fact that large volumes of inference are now performed continuously across production systems.

At the frontier of model capability, the trend toward inference-heavy systems is even more pronounced. A meta-analysis of 5,357 accepted papers at ICLR 2026 identified 257 publications focused specifically on test-time compute and inference-phase adaptation [16]. This shift in research emphasis

suggests that compute investment is increasingly moving from pre-training into the inference phase of model operation.

Recent system designs further reinforce this trend. For example, Gemini 3.1 Pro introduces a three-tier reasoning configuration in which inference requests can dynamically allocate low, medium, or high levels of test-time compute depending on task complexity [15]. Deep reasoning configurations can require several minutes of execution time per request, illustrating how inference workloads are evolving from simple forward passes into extended computational processes.

Taken together, these developments indicate that inference efficiency is becoming a central determinant of economic viability for large-scale AI deployments. As inference workloads grow in scale and complexity, increasing optimization pressure is applied to the runtime systems that execute them.

2.2. Capital Expenditure and Hardware Diversification

The economic importance of inference infrastructure is reflected in the scale of capital investment by hyperscale cloud providers. Combined AI-related capital expenditures across major hyperscalers now exceed $200 billion annually, with a growing share dedicated to datacenter infrastructure designed to support continuous large-scale inference workloads.

Unlike training clusters, which typically operate in concentrated bursts during model development cycles, inference infrastructure must sustain continuous operation across large and highly variable user populations. This operational requirement encourages architectural diversification across hardware platforms.

Modern hyperscale AI fleets increasingly combine multiple accelerator classes, including GPUs, TPUs, Trainium processors, Inferentia chips, and a growing set of custom inference accelerators. Requests are distributed across these heterogeneous resources through routing systems that balance latency requirements, cost constraints, and hardware availability.

This hardware diversification trend is likely to intensify. NVIDIA has announced plans for new inference-optimized accelerator designs [11], while deployment frameworks such as TensorRT-LLM AutoDeploy aim to automate model deployment across heterogeneous inference stacks [12].

Strategic partnerships between AI model developers and cloud infrastructure providers further deepen this integration. For example, the OpenAI–Amazon partnership announced in February 2026 includes co-development of a Stateful Runtime Environment on Amazon infrastructure and exclusive third-party cloud distribution for certain enterprise agent products [13]. Such collaborations further entangle model development with runtime infrastructure design.

Heterogeneous hardware environments introduce additional operational complexity. Each accelerator class exposes distinct latency profiles, memory hierarchies, and throughput regimes. Routing systems must therefore do more than distribute requests across available capacity. They implicitly determine how inference workloads interact with underlying hardware characteristics, introducing an additional dimension of runtime control within the serving stack.

2.3. Energy as a Binding Constraint

Alongside economic and architectural pressures, energy availability is emerging as a fundamental constraint on the expansion of AI infrastructure.

Data center power demand is projected to increase substantially in the coming decade. Goldman Sachs estimates that global data center electricity consumption could rise by approximately 175% by 2030 under accelerated AI adoption scenarios [3]. As AI workloads become a larger component of datacenter activity, power availability and grid capacity are becoming critical factors in infrastructure planning.

Industry surveys confirm the growing importance of energy constraints. A Deloitte study reports that 72% of U.S. power and data center executives now identify grid stress as their leading infrastructure challenge [8]. At a global level, the International Energy Agency projects that total data center electricity demand could exceed 1,700 terawatt-hours by 2035 under high adoption scenarios [7].

These pressures are increasingly shaping operational policies within large inference deployments. Energy-aware scheduling mechanisms dynamically adjust workload distribution in response to power budgets, thermal limits, and regional grid conditions. Clusters may shift workloads between datacenters, throttle compute-intensive tasks, or prioritize energy-efficient hardware under constrained conditions.

Each of these adjustments modifies the runtime environment in which inference workloads execute. Energy-aware scheduling therefore introduces another autonomous optimization loop operating below the model layer, further increasing the complexity of runtime control dynamics within large-scale AI systems.

2.4. Agent-Driven Token Growth

A final driver of runtime complexity is the rapid emergence of agent-based AI workflows.

Traditional prompt-response interactions typically involve a single request and a single model response. Agentic systems operate very differently. They often execute multi-step reasoning loops, perform tool calls, explore alternative strategies, and repeatedly interact with external systems before producing a final output.

These behaviors dramatically increase token consumption. Gartner estimates that agentic AI workloads can generate between 20 and 30 times more tokens than conventional prompt-response interactions [9]. As organizations adopt agent-based systems, total inference demand grows accordingly.

Recent model releases illustrate how quickly this transition is unfolding. GPT-5.3 Codex is positioned as an interactive coding agent with strong performance on engineering benchmarks such as SWE-Bench Pro and Terminal-Bench [14]. Claude Sonnet 4.6 emphasizes agent workflows through computer-use APIs and long-running task execution environments, while Claude Opus 4.6 introduces context compaction techniques that support autonomous tasks operating over extremely long context windows.

Open-weight competition is simultaneously intensifying cost pressure within the ecosystem. Models such as Alibaba's Qwen 3.5 combine large parameter counts with mixture-of-experts architectures that activate only a small fraction of parameters per inference step, enabling significantly lower per-token inference costs compared to many proprietary models.

These developments create a structural tension. Agent-based systems require runtime behaviors such as retry loops, exploration strategies, tool interaction, and planning continuity. At the same time, infrastructure operators are under strong pressure to minimize inference cost and maximize hardware utilization.

The runtime control layer mediates this tension by dynamically adjusting how inference workloads are executed across infrastructure resources. Because these adjustments occur within the serving stack rather than within the model itself, their effects are largely invisible to traditional model evaluation metrics.

Industry surveys suggest that this operational complexity may already be affecting deployment outcomes. One analysis estimates that approximately 40% of agentic AI projects fail before reaching production [10]. While many factors contribute to such failures, the coherence of runtime control systems represents a potential structural variable that current diagnostics rarely measure directly.

3. The Hidden Structural Effect

The economic and infrastructural developments described in the previous section do not merely increase operational scale. They reshape the internal structure of AI serving systems. As inference workloads expand and infrastructure becomes more heterogeneous, an additional system layer has emerged within large-scale deployments: the runtime control layer.

This layer is rarely described explicitly in system documentation. Yet it plays a central role in determining how inference workloads are executed across distributed infrastructure. It governs how optimization mechanisms interact, how workloads are distributed across hardware resources, and how operational constraints such as cost, energy, and latency are reconciled in real time.

Understanding this layer is therefore essential for analyzing how modern AI systems behave in production environments.

3.1. The Runtime Control Layer as a System Layer

Contemporary AI deployments can be understood as consisting of three distinct architectural layers.

1. **Model Layer.** This layer includes model architecture, trained weights, and the training processes used to produce them.
2. **Inference Layer.** This layer contains the mechanisms required to execute the model in production environments, including tokenization pipelines, batching systems, and serving frameworks.
3. **Runtime Control Layer.** This layer coordinates how inference workloads interact with infrastructure resources. It includes scheduling policies, routing mechanisms, caching strategies, autoscaling controllers, speculative decoding control, power-aware scheduling, and GPU allocation policies.

The runtime control layer determines when models execute, how tokens are distributed across hardware resources, which infrastructure nodes participate in a request, and how the system responds to transient failures or resource constraints. In large distributed deployments these decisions occur continuously and are typically handled by automated control systems embedded within the serving stack.

Despite its operational importance, the runtime control layer has received comparatively little direct research attention as an architectural system layer. Research activity in machine learning continues to focus primarily on model design and training techniques. For example, an analysis of accepted papers at ICLR 2026 shows extensive attention devoted to training methodologies such as GRPO and to inference-phase adaptation techniques such as test-time compute [16].

In contrast, the structure of runtime control systems themselves remains comparatively underexamined in the research literature.

Industry developments nevertheless suggest that this layer is becoming increasingly significant. NVIDIA's AutoDeploy framework automates many inference optimization decisions that were previously configured manually [12]. Similarly, the OpenAI–Amazon partnership announced in 2026 includes the co-development of a Stateful Runtime Environment designed to manage long-running agent workloads across infrastructure resources [13].

These developments indicate that the industry is progressively formalizing the runtime control layer as a first-class component of AI infrastructure, even if the concept is not yet widely discussed as a distinct architectural category.

3.2. Autonomous Optimization Loops

Within the runtime control layer, multiple optimization mechanisms operate concurrently. Each mechanism is typically designed to optimize a specific operational objective such as latency, throughput, infrastructure utilization, or energy efficiency.

Table 1 summarizes several of the most common optimization loops found in modern inference systems.

Each of these mechanisms is individually rational. They are typically developed by different engineering teams, optimized for different objectives, and deployed incrementally within evolving infrastructure stacks.

Speculative decoding illustrates how these mechanisms interact in practice. Draft-model token proposals can significantly reduce latency by allowing portions of the generation process to be predicted in advance [4]. However, the effectiveness of speculative decoding depends strongly on the surrounding serving environment. NVIDIA's SPEED-Bench evaluation shows that speculative decoding performance varies substantially depending on workload diversity, batching configuration, and serving regime [5].

Table 1. Representative optimization loops in modern inference systems

Optimization Loop	Primary Objective
Inference Scheduler	GPU utilization
Batching System	Throughput
Speculative Decoding Controller	Latency reduction
Routing Layer	Load distribution
Caching Layer	Inference efficiency
Autoscaling Controller	Capacity management
Cost Controller	Cost per token
Power-Aware Scheduler	Energy efficiency

This variability suggests that interactions between optimization loops can materially influence system behavior.

Importantly, these optimization loops typically operate without explicit global coordination mechanisms. They do not share a unified objective function and rarely maintain shared state describing the broader optimization environment. Instead, they interact implicitly through the runtime infrastructure.

As the number of such mechanisms grows, the system contains more autonomous optimization agents than explicit coordination mechanisms. The resulting condition can be described as **control-layer complexity**: a structural situation in which the interaction surface between optimization loops grows faster than the system's capacity to coordinate their combined effects.

3.3. How Cost Optimization Reshapes Runtime Structure

Economic pressure within inference infrastructure does not simply reduce operational margins. It also reshapes how the runtime environment itself behaves.

Cost optimization mechanisms influence several structural characteristics of the execution environment. The following patterns are consistent with the diagnostic vocabulary introduced in [17] and with the structural efficiency analysis discussed in [18].

Context truncation.

Intermediate reasoning state may be compressed or partially discarded in order to reduce memory consumption during long-running inference tasks. Context compaction techniques, such as those introduced in recent agent-oriented systems, explicitly address this constraint by preserving only the most relevant portions of the interaction history.

Early task termination.

Temporal budgets for inference requests are frequently bounded by latency or cost targets. When reasoning processes extend beyond these limits, execution may be terminated earlier than the natural completion point of the reasoning process. Systems such as Gemini 3.1 Pro explicitly expose configurable reasoning depth levels that reflect this tradeoff between computational cost and reasoning time [15].

Tool-call reordering.

In agentic workflows, scheduling policies may prioritize execution paths that reduce overall latency or infrastructure utilization. As a result, the order in which external tools or services are invoked can vary depending on system load and cost constraints.

Retry topology modification.

Retry policies for failed intermediate steps are often bounded by cost or latency budgets. In environments where agent workflows can amplify token consumption by factors of 20–30 [9], these retry limits can significantly influence how systems recover from intermediate errors.

Speculative execution narrowing.

Speculative decoding techniques generate multiple candidate token sequences using draft models. However, the breadth of speculative exploration is typically bounded by infrastructure budgets. Token limits and cost constraints therefore determine the width of speculative branching, rather than purely algorithmic considerations [4].

Taken together, these mechanisms illustrate an important structural observation. Cost optimization does not merely change performance metrics such as latency or throughput. It modifies the geometry of the execution environment in which inference occurs.

The model architecture remains unchanged. The runtime environment through which model computation unfolds evolves in response to infrastructure constraints.

4. Why Benchmarks Don't Catch This

The structural dynamics described in the previous section are largely invisible to conventional evaluation methods used in machine learning. Benchmark suites remain the dominant instrument for measuring model capability, yet they operate under assumptions that differ fundamentally from the conditions found in large-scale production inference systems.

As AI deployments scale and runtime infrastructure becomes more complex, the gap between benchmark evaluation and production behavior becomes increasingly important to understand.

4.1. The Benchmark Assumption

Benchmark evaluation requires controlled experimental conditions. In order to produce reproducible and comparable measurements, benchmark suites assume a stable execution environment in which the model is evaluated.

Typical benchmark conditions include fixed context sizes, deterministic scheduling, isolated request execution, homogeneous hardware environments, and unconstrained retry budgets. These assumptions allow researchers to compare models under consistent conditions and to attribute observed performance differences to model architecture or training methodology.

Production inference environments operate under very different conditions. Real deployments must process fluctuating workloads, dynamically route requests across heterogeneous hardware fleets, and continuously adjust optimization policies in response to infrastructure constraints. Systems may implement autoscaling, cost-aware scheduling, energy-aware workload redistribution, or speculative execution strategies that vary depending on real-time operating conditions.

As a result, benchmark scores primarily measure model capability under controlled execution conditions. They do not necessarily describe how a system behaves once the model is embedded within a complex runtime control environment.

The research community has begun to acknowledge this distinction. NVIDIA's SPEED-Bench evaluation framework explicitly measures speculative decoding performance under realistic serving regimes and heterogeneous workloads [5]. By incorporating production-like serving conditions, the benchmark recognizes that inference performance depends not only on model design but also on the surrounding runtime environment.

Other evaluation frameworks further reflect this shift in perspective. ISO-Bench evaluates whether coding agents can optimize real-world inference workloads rather than simply solve isolated programming tasks [6]. Similarly, FeatureBench introduces evaluation procedures that explicitly account for drift, task coverage, and operational realism in agent-based coding environments.

These efforts suggest an emerging recognition that evaluation must increasingly consider system-level behavior in addition to model capability.

4.2. What Metrics Measure vs. What Changes

Operational monitoring systems and observability tools provide detailed measurements of system performance. Common metrics include token throughput, latency distributions, GPU utilization, benchmark scores, error rates, cost per token, and aggregate throughput.

These metrics provide valuable information about the outputs and efficiency of the system. They describe what the system produced and how efficiently it produced it. However, they do not directly measure the structural configuration of the runtime control environment.

The distinction is important. The runtime control layer described in Section 3 operates by modifying how execution unfolds within the serving stack. Scheduling policies, routing strategies, retry limits, speculative decoding parameters, and resource allocation mechanisms alter the execution topology in which model computation occurs.

Because these structural adjustments occur beneath the level at which most monitoring systems operate, they can evolve gradually without immediately affecting conventional performance indicators.

In practice, this means that a system can simultaneously satisfy all conventional performance criteria — it may pass benchmark evaluations, maintain stable latency distributions, achieve high GPU utilization, and operate within cost targets — while the internal structure of its runtime control environment continues to change.

Expanding benchmark coverage or increasing measurement frequency improves the resolution of performance metrics, but it does not eliminate this structural gap. The limitation is not simply quantitative; it is categorical. Conventional evaluation frameworks measure outputs, whereas the structural dynamics discussed in this paper occur within the control architecture that governs how those outputs are produced.

As a result, existing instrumentation provides only partial visibility into the behavior of large-scale inference systems. Understanding how runtime control mechanisms interact therefore requires additional diagnostic perspectives that focus on the structure of the execution environment itself.

5. The Structural Diagnostic Gap

The previous section demonstrated that benchmark evaluation and operational monitoring primarily measure performance outcomes. They describe how efficiently a system produces outputs, but they provide limited visibility into the structural conditions under which those outputs are generated.

As AI serving systems grow more complex, this distinction becomes increasingly important. Modern inference environments contain multiple interacting control mechanisms whose behavior evolves dynamically under operational constraints. Observability systems can measure the results of these interactions, but they rarely capture the internal structure through which those interactions occur.

This creates a structural diagnostic gap: the mechanisms that govern execution topology remain largely invisible to conventional monitoring and observability tools.

5.1. What Observability Does Not Capture

Modern observability frameworks provide extensive visibility into system performance. Distributed tracing systems, metrics pipelines, and telemetry infrastructures capture detailed information about latency distributions, throughput, resource utilization, and error rates. These measurements are essential for maintaining reliable large-scale infrastructure.

However, these systems are designed primarily to measure performance outcomes rather than the structural configuration of the runtime environment.

Several aspects of runtime behavior typically remain outside the scope of conventional observability instrumentation. For example, monitoring systems rarely record whether identical requests follow different reasoning trajectories under varying load conditions. They do not typically capture

whether speculative exploration width has been narrowed as a result of cost optimization policies, or how retry budgets evolve over time as infrastructure constraints change.

Similarly, observability tools seldom capture how context compression alters the availability of intermediate reasoning state within long-running agent workflows. Queue scheduling decisions may change the ordering and timing of tool calls in complex multi-step tasks, yet these structural variations often appear only indirectly through downstream performance metrics.

In each of these cases, the observable outputs of the system may remain stable while the internal structure of the execution process gradually changes.

This limitation does not reflect a failure of existing monitoring systems. Current observability frameworks were designed to measure operational performance and system health. They are highly effective for diagnosing latency spikes, infrastructure failures, or resource saturation.

The challenge instead lies in the category of measurement being performed. Observability tools measure performance signals, while the structural dynamics discussed in this paper occur within the execution topology itself. Decision paths, control-loop interactions, and state transitions form an internal structure that conventional instrumentation was not designed to record.

A similar categorical distinction has been identified in analyses of cost-induced structural drift in AI systems [18]. In both cases, the observable metrics capture the consequences of structural change rather than the structural configuration that produces those consequences.

5.2. Runtime Geometry Analysis

Addressing the structural diagnostic gap requires a complementary analytical perspective that focuses directly on the organization of execution pathways within the runtime system.

This paper introduces the concept of **Runtime Geometry** as a descriptive framework for analyzing this structure.

Runtime Geometry refers to the structural arrangement of execution pathways, decision points, and state transitions that collectively govern how inference workloads unfold within the serving environment. Rather than measuring performance outcomes, this perspective examines the topology of the execution process itself [17].

Within this framework, runtime behavior can be analyzed across several structural dimensions.

1. **Control Loop Topology**
   This level examines the set of optimization loops active within the runtime environment and maps their interaction surfaces. It identifies which control mechanisms are present, the objectives they pursue, and where their decision boundaries intersect.
2. **Optimization Objective Alignment**
   Different control loops frequently optimize for distinct objectives such as cost efficiency, latency reduction, throughput maximization, or hardware utilization. This level examines how these objectives align or conflict within the broader system architecture.
3. **Interconnect Stress Patterns**
   Interaction points between optimization loops can become structural stress locations within the runtime environment. Under fluctuating workloads or infrastructure constraints, these coupling points may produce increased behavioral variance. Mapping these interaction patterns helps identify where instability regimes may emerge.
4. **Runtime Drift Signals**
   Over time, optimization policies may gradually reshape the execution topology without any modification to the underlying model. Detecting such runtime drift signals provides early visibility into behavioral changes that would otherwise appear only indirectly through performance metrics.

The objective of this analytical perspective is not to replace existing monitoring or observability systems. Instead, it complements them by introducing a structural diagnostic lens through which the internal organization of runtime control systems can be examined.

By focusing on the topology of execution rather than solely on performance outputs, runtime geometry analysis enables the identification of conflicting optimization loops, instability regimes, and emerging performance drift zones before they manifest as operational inconsistencies.

6. SORT Perspective

The structural dynamics discussed in the preceding sections require a vocabulary that can describe how runtime execution environments evolve under operational constraints. The SORT framework provides such a vocabulary. It is used here exclusively as a diagnostic lens for describing structural properties of large-scale AI systems.

The SORT framework introduces no new physical laws, empirical parameters, or additional system degrees of freedom. Its role is descriptive rather than prescriptive. The evaluation criteria within the framework are structural in nature: mathematical consistency, coherence of interacting operators, and minimality of the resulting description. Within the context of this paper, SORT terminology is used to characterize runtime control phenomena that conventional monitoring systems do not explicitly capture.

6.1. Diagnostic Vocabulary

Several terms from the SORT framework are particularly relevant for describing the runtime dynamics examined in this paper.

Control Coherence.

Control coherence describes the degree to which independent optimization loops produce structurally aligned system behavior. When coherence is high, the actions of different optimization mechanisms reinforce one another and produce stable operational regimes. When coherence is low, optimization loops may neutralize or counteract each other, leading to emergent instability or inefficient execution patterns. A formal definition of this concept is provided in [17].

Runtime Geometry.

Runtime geometry refers to the structural arrangement of execution pathways, decision points, and state transitions that determine how inference workloads unfold during system operation. Rather than focusing on performance metrics, this concept describes the topology of the execution process itself. The term was introduced as part of the structural diagnostics framework in [17].

Drift Diagnostics.

Drift diagnostics identify behavioral change arising from modifications to the execution layer rather than changes to the model itself. In this situation, system behavior evolves because the surrounding runtime environment has been altered by optimization policies or infrastructure constraints. The resulting phenomenon can be described as execution-layer drift. A structural analysis of this form of drift is presented in [18].

Interconnect Stability.

Interconnect stability concerns the structural stability of distributed GPU communication and cross-accelerator execution behavior within heterogeneous hardware fleets. When inference workloads are routed across multiple accelerator classes, variations in communication patterns and synchronization behavior can influence the resulting execution topology. This phenomenon is examined in the runtime control analysis described in [17].

Taken together, these terms provide a structured vocabulary for describing runtime behavior in large-scale inference systems. They function as diagnostic descriptors rather than theoretical constructs. Their purpose is to enable consistent discussion of runtime phenomena that conventional observability frameworks typically record only indirectly.

6.2. SORT Application Mapping

The structural dynamics described throughout this paper correspond to several diagnostic applications within the SORT framework. Table 2 summarizes these relationships.

Table 2. Representative mapping between runtime phenomena and SORT diagnostic applications

Structural Phenomenon	SORT Application	Diagnostic Focus
Runtime control incoherence	ai.04 Runtime Control Coherence	Coherence analysis across runtime regimes
Drift without model modification	ai.02 Structural Drift Diagnostics	Execution-layer drift over time
Interconnect stress under optimization	ai.01 Interconnect Stability Control	Cross-accelerator stability analysis
Inference pipeline interaction	ai.27 Inference Pipeline Control Coherence	Stability across pipeline stages
Agent reliability under cost pressure	ai.13 Agentic System Stability	Multi-step task coherence diagnostics
Hardware-dependent behavior	ai.07 Accelerator Runtime Control	Hardware-execution coupling diagnostics
Cross-cluster signal aggregation	ai.52 Deployment Drift Signal Aggregation	Distributed drift signal detection

Within the context of this paper, two applications are particularly central. Application ai.04 (Runtime Control Coherence) and application ai.02 (Structural Drift Diagnostics) directly address the core phenomena examined in earlier sections: runtime control incoherence and behavioral drift occurring without model modification[1].

Agentic workloads introduce additional structural considerations. Application ai.13 (Agentic System Stability) examines how retry topology, exploration width, and planning continuity interact with runtime infrastructure constraints. Under cost optimization pressure, these factors can significantly influence the reliability of multi-step agent workflows.

Heterogeneous infrastructure environments introduce further diagnostic requirements. Applications ai.01 (Interconnect Stability Control) and ai.07 (Accelerator Runtime Control) provide analytical perspectives for examining how routing decisions across accelerator classes influence the resulting execution topology.

Similarly, application ai.27 (Inference Pipeline Control Coherence) focuses on stability across inference pipeline stages where multiple optimization mechanisms interact. Finally, application ai.52 (Deployment Drift Signal Aggregation) addresses the detection of weak structural drift signals that may emerge across distributed deployment environments and cluster boundaries.

This mapping does not represent an exhaustive catalog of SORT applications. Rather, it illustrates how the structural phenomena described in this paper connect to a broader diagnostic framework for analyzing large-scale inference infrastructure. The terminology provides a consistent way to describe runtime control dynamics without introducing additional theoretical assumptions.

7. Strategic Implications for Hyperscalers

The structural dynamics discussed throughout this paper have direct implications for large-scale AI infrastructure operators. As inference workloads grow and runtime control systems become increasingly complex, several strategic considerations emerge for hyperscalers responsible for designing and operating distributed inference environments.

These considerations do not arise from model architecture alone. They reflect the interaction between models, runtime control mechanisms, hardware fleets, and operational constraints such as cost and energy availability. Understanding these interactions is therefore increasingly relevant for infrastructure strategy.

[1] For formal definitions see [17] and [18].

7.1. Vendor Lock-In Through Runtime Optimization

Runtime optimization mechanisms introduce a form of infrastructure coupling that extends beyond traditional model portability concerns. Even when model weights remain identical, systems tuned for a specific serving stack may exhibit significantly different runtime characteristics when deployed in an alternative infrastructure environment.

Optimization policies embedded within the runtime control layer shape how inference workloads interact with hardware resources. Scheduling behavior, retry policies, routing logic, caching strategies, and speculative decoding parameters all influence the execution topology in which model computation occurs. When these mechanisms are optimized within a particular infrastructure ecosystem, migrating the workload to another environment may require substantial reconfiguration of the surrounding runtime architecture.

The OpenAI–Amazon partnership announced in February 2026 illustrates this form of coupling. The co-development of a Stateful Runtime Environment on Amazon infrastructure creates deep integration between model execution logic and the underlying infrastructure stack [13]. In such environments, migration away from the original platform may involve not only porting model artifacts but also redesigning portions of the runtime control architecture.

In extreme cases, infrastructure dependence may emerge as an architectural risk category. Policy or regulatory changes affecting a specific vendor ecosystem can require rapid migration of operational systems. When runtime optimization layers are tightly integrated with specific infrastructure environments, such transitions can become technically complex even if model artifacts themselves remain portable.

From a strategic perspective, this suggests that vendor lock-in may increasingly arise not only from model compatibility or proprietary APIs, but also from the structural configuration of the runtime execution environment.

7.2. Instability in Heterogeneous Accelerator Fleets

Modern hyperscale inference deployments increasingly rely on heterogeneous accelerator fleets composed of GPUs, TPUs, Trainium processors, and specialized inference ASICs. While this diversification improves hardware availability and cost flexibility, it also introduces structural variance into inference execution pathways.

The same request processed across different accelerator classes may follow different execution trajectories depending on memory hierarchy, interconnect bandwidth, latency characteristics, and routing policies. As requests are dynamically distributed across heterogeneous hardware pools, the resulting runtime behavior becomes dependent not only on model computation but also on hardware-specific execution environments.

Recent hardware developments suggest that fleet heterogeneity will continue to increase. NVIDIA's announcement of new inference-focused accelerators reflects an ongoing expansion of hardware specialization within the AI infrastructure ecosystem [11]. Tools such as NVIDIA AutoDeploy aim to simplify model deployment across these heterogeneous environments [12]. While such tools improve operational manageability, they do not eliminate the structural variance introduced by hardware diversity.

At the same time, emerging research trends emphasize dynamic inference workloads. The growing focus on test-time compute in machine learning research [16], combined with adaptive reasoning systems such as Gemini 3.1 Pro's multi-tier reasoning architecture [15], indicates that compute demand per request may vary significantly across workloads.

Routing variable-demand workloads across heterogeneous hardware fleets compounds runtime control complexity. As a result, fleet management decisions increasingly involve structural considerations regarding which workloads tolerate hardware-induced variance and which require more stable execution environments.

7.3. Agent Reliability Under Cost Pressure

Agentic AI systems introduce additional operational complexity into inference environments. Systems such as GPT-5.3 Codex [14] and recent Claude models are designed to perform multi-step tasks involving planning, tool interaction, and iterative reasoning processes.

These capabilities rely on runtime behaviors such as retry loops, intermediate verification steps, and extended reasoning horizons. In many agent workflows, successful task completion depends not only on model capability but also on the continuity of these execution patterns.

Infrastructure cost optimization can influence these behaviors in subtle ways. Policies designed to limit token consumption, reduce latency, or maintain throughput targets may impose bounds on retry attempts, compress intermediate context state, or shorten execution time budgets.

Under agentic workloads, where token consumption may increase by factors of 20–30 compared with standard prompt-response interactions [9], such constraints can have amplified effects on system behavior.

Industry analyses suggest that a significant portion of agent-based AI projects encounter difficulties before reaching production environments [10]. While many factors contribute to such outcomes, the interaction between runtime optimization policies and agent execution patterns represents an important structural dimension of system reliability.

Analyzing these interactions from a runtime control perspective allows infrastructure designers to better understand how cost management policies influence exploration depth, recovery behavior, and overall agent reliability.

7.4. Power Throttling Effects

Energy availability is becoming an increasingly important factor in the operation of large-scale AI infrastructure. As discussed earlier, data center power demand is expected to grow significantly over the coming decade [3,7], and energy constraints are already influencing operational policies within many large deployments.

Inference workloads based on test-time compute can produce highly variable resource demand. For example, high-depth reasoning modes in systems such as Gemini 3.1 Pro may require several minutes of execution time for a single request [15]. When systems dynamically adjust reasoning depth in response to task complexity, the resulting compute demand may escalate non-linearly.

Infrastructure systems must respond to such fluctuations while maintaining stable operation within power and cooling constraints. Power-aware scheduling mechanisms therefore redistribute workloads across clusters, adjust execution timing, or throttle compute-intensive tasks when infrastructure limits are approached.

These adjustments represent normal operational behavior within large distributed systems [8]. However, they also introduce an additional layer of runtime control interaction. Power management policies influence scheduling order, recovery timing, and resource allocation decisions within the serving stack.

From a system perspective, energy constraints therefore act not merely as external limits on infrastructure capacity. They participate directly in shaping the runtime control environment in which inference workloads execute.

8. Conclusion

Modern AI systems increasingly operate across three interacting architectural layers: the model layer, the inference layer, and the runtime control layer. While model architectures and training methodologies remain central to advances in capability, the operational behavior of deployed systems is increasingly shaped by the runtime environment in which those models execute.

This paper has argued that the runtime control layer has emerged as a structurally significant component of large-scale AI infrastructure. Within this layer, multiple autonomous optimization mechanisms operate concurrently. Scheduling systems, routing layers, speculative decoding controllers,

autoscaling policies, and power-aware scheduling mechanisms each pursue locally rational objectives. Their interactions, however, reshape the execution topology through which inference workloads are processed.

These structural dynamics occur largely outside the scope of conventional evaluation and monitoring tools. Benchmark suites measure model capability under controlled experimental conditions, while observability systems measure operational performance signals such as latency, throughput, and resource utilization. Neither class of instrumentation was designed to capture the evolving structure of runtime control interactions.

As a result, large-scale inference systems may experience structural effects that remain difficult to diagnose through conventional metrics alone. Control incoherence between optimization loops, execution-layer drift without model modification, and hardware-dependent behavioral variance can all arise within the runtime environment while conventional performance indicators remain stable.

At the same time, many performance improvements in modern AI deployments increasingly originate from runtime infrastructure optimization rather than model redesign. This shift elevates the strategic importance of the runtime control layer as a determinant of deployed system behavior.

Structural diagnostics provide a complementary perspective for examining these dynamics. By analyzing the topology of execution pathways and the interaction patterns of optimization mechanisms, structural approaches make aspects of runtime behavior visible that traditional monitoring systems record only indirectly.

Within this context, the SORT framework provides a diagnostic vocabulary for describing such structural phenomena. Concepts such as control coherence, runtime geometry, and execution-layer drift enable consistent discussion of runtime dynamics across heterogeneous infrastructure environments.

The mapping presented in Section 6.2 illustrates how the structural phenomena examined in this paper correspond to diagnostic applications within the broader SORT analytical framework.

As AI infrastructure continues to scale, understanding how runtime control mechanisms interact within distributed inference systems will become increasingly relevant for both research and infrastructure design.

As AI systems scale, runtime control coherence becomes an increasingly important determinant of system behavior.

Optimization policies do not merely influence cost or latency; they reshape the structure of the execution environment itself.

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. Goldman Sachs Research. (2025). Data Center Power Demand: The 6 Ps Driving Growth and Constraints. *GS SUSTAIN Report*.
4. 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)*.
5. 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

6. ISO-Bench Authors. (2026). ISO-Bench: Can Coding Agents Optimize Real-World Inference Workloads? *arXiv preprint*. https://arxiv.org/html/2602.19594v1
7. International Energy Agency. (2025). Energy and AI: Energy Demand from Data Centres. *IEA Special Report*. Paris. https://www.iea.org/reports/energy-and-ai
8. 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
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. 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/
12. 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/
13. OpenAI. (2026). OpenAI and Amazon Partnership. *OpenAI Blog*. https://openai.com/index/amazon-partnership/
14. OpenAI. (2026). Introducing GPT-5.3 Codex. *OpenAI Blog*. https://openai.com/index/introducing-gpt-5-3-codex/
15. Google DeepMind. (2026). Gemini 3.1 Pro Model Card. *Google DeepMind*. https://deepmind.google/models/model-cards/gemini-3-1-pro/
16. ICLR 2026 Meta-Analysis. (2026). Accepted Paper Trends: 5,357 Papers Analyzed. *ICLR 2026 Conference Proceedings*.
17. Wegener, G.H. (2026). SORT-AI: Runtime Control Coherence in Large-Scale AI Systems. *MDPI Preprints*. https://doi.org/10.20944/preprints202601.0298.v1
18. Wegener, G.H. (2026). SORT-AI: Structural Efficiency Recovery in Hyperscale AI Systems. *MDPI Preprints*. https://doi.org/10.20944/preprints202602.0015.v1