Article

SORT-AI: Accelerator Runtime Coherence in Heterogeneous AI Inference Infrastructure: A Structural Analysis of Cross-Layer Instability in Large-Scale Systems

Gregor Herbert Wegener

Independent Research & Systems Modeling, Berlin, Germany; gregor.wegener@independent-research-systems-modeling.com

Abstract

Large scale AI inference is shifting from relatively homogeneous accelerator fleets toward structurally heterogeneous infrastructure composed of mixed accelerator generations, disaggregated serving paths, virtualized execution layers, and cross environment placement. This transition introduces a class of instability phenomena that does not arise from component failure in isolation, but from cross layer incoherence between accelerator runtimes, network topology, virtualization boundaries, and migration induced reconfiguration. Classical performance metrics such as utilization, average latency, and tokens per second remain necessary, but they do not capture these structural effects because they observe individual layers rather than the coupled system formed across them. This paper develops a vendor agnostic structural analysis of heterogeneous inference infrastructure and argues that runtime coherence functions as a hidden performance variable that conditions effective capacity, tail latency, and cost efficiency before conventional optimization can succeed. The paper introduces a taxonomy of five instability modes, latency asymmetry drift, memory path incoherence, interconnect induced capacity inaccessibility, virtualization induced control distortion, and migration induced runtime reconfiguration risk. It then maps these modes to four diagnostic domains, accelerator runtime control, structural network scalability, virtualization overhead stability, and structural cloud migration risk. The contribution is analytical rather than prescriptive. It does not propose implementation mechanisms or vendor specific remedies, but provides a research framework for identifying where and why heterogeneous inference systems produce emergent instability under scale.

Keywords: heterogeneous inference, AI infrastructure, accelerator runtime coherence, cross layer instability, structural diagnostics, network scalability, virtualization overhead, cloud migration risk, large scale AI systems

---

1. Introduction

The architectural basis of large scale AI inference is undergoing a structural transition. Earlier deployment regimes were frequently organized around relatively homogeneous accelerator fleets, bounded network assumptions, and comparatively stable runtime conditions. In such environments, performance degradation could often be interpreted through localized constraints such as device saturation, memory pressure, or interconnect bottlenecks considered within a mostly uniform execution fabric. This assumption is becoming progressively less adequate. Contemporary inference systems are increasingly assembled from mixed accelerator classes, heterogeneous memory paths, disaggregated serving stages, virtualized execution layers, and placement patterns that extend across cloud, region, and provider boundaries. Under these conditions, system behavior is no longer determined by isolated component efficiency alone, but by whether the coupled runtime remains structurally coherent across layers.

This shift has multiple drivers. Economic pressure encourages workload placement across different accelerator tiers rather than within a single uniform hardware class. Supply constraints make mixed generation or mixed vendor procurement increasingly common in practice. At the same time, architectural diversification has expanded the number of execution pathways through which inference can be realized, including prefill decode separation, disaggregated serving, remote memory dependency, and hybrid edge cloud deployments. As a result, heterogeneity is not merely an implementation detail. It is becoming a default operating condition of large scale inference infrastructure, particularly in environments shaped by multi cloud requirements, sovereign deployment constraints, or differentiated cost and latency targets. The central problem is therefore not that heterogeneous systems contain different components, but that these components interact through runtime couplings whose aggregate effects are poorly represented by conventional layer specific metrics.

The present paper examines this transition as a structural systems problem. Its focus is not benchmark comparison, hardware ranking, or deployment optimization in a narrow operational sense. Instead, the analysis asks how heterogeneous inference environments generate instability when accelerator runtimes, network topology, virtualization boundaries, and migration induced configuration changes interact under scale. The core claim is that runtime coherence functions as a hidden performance variable in such systems. When cross layer coherence deteriorates, nominal capacity, local utilization, and even apparent throughput may remain visible, while effective system behavior degrades through tail latency inflation, capacity inaccessibility, and rising operational cost without proportional capability gain. These effects are especially difficult to diagnose because they emerge between layers rather than within any single one.

This paper is analytical and diagnostic in scope. It does not present new benchmarks, implementation mechanisms, or vendor specific prescriptions. The objective is to provide a formal structural framing for a class of instability that is gaining practical relevance as AI infrastructure becomes more heterogeneous. The contribution is organized around three elements. First, the paper defines the relevant system class and failure envelope for heterogeneous large scale inference. Second, it identifies the principal structural sources of runtime incoherence across accelerator, network, virtualization, and migration layers. Third, it develops a taxonomy of heterogeneous inference instability and maps that taxonomy to a corresponding diagnostic space. In this sense, the paper extends prior discussion of isolated bottlenecks toward a coupled perspective in which coherence precedes optimization as the primary condition of stable large scale inference.

1.1. The Shift to Heterogeneous Inference Infrastructure

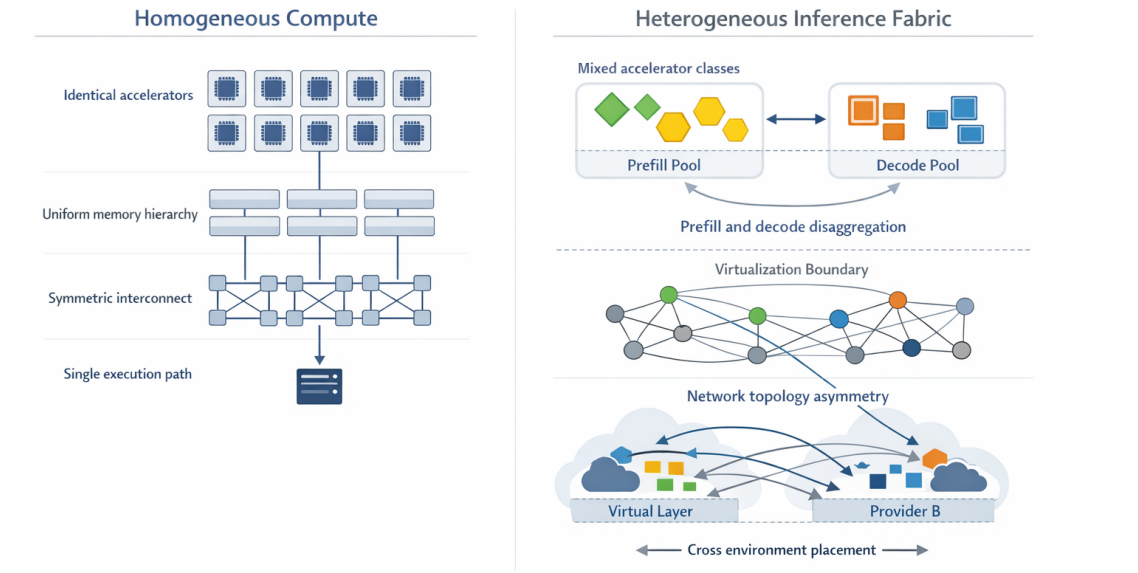

Figure 1. Comparison between a relatively homogeneous compute environment and a heterogeneous inference fabric. The homogeneous regime is characterized by identical accelerators, a uniform memory hierarchy, symmetric interconnect assumptions, and a single dominant serving path. The heterogeneous regime combines mixed accelerator classes, prefill–decode disaggregation, virtualization boundaries, asymmetric memory paths, and cross-environment placement. The figure illustrates the architectural transition that motivates the present analysis.

Modern AI inference has moved beyond relatively homogeneous accelerator fleets toward structurally heterogeneous execution environments composed of mixed accelerator generations, differentiated memory hierarchies, nonuniform interconnect paths, and deployment boundaries that may span multiple clouds or administrative domains. In earlier infrastructure regimes, the simplifying assumption of hardware similarity often permitted performance reasoning through local resource metrics. That assumption weakens once the execution fabric itself becomes compositionally diverse. Mixed accelerator environments do not simply add capacity. They alter the control geometry of the runtime by introducing asymmetries in scheduling behavior, memory access characteristics, communication cost, and latency propagation across the serving path [19,25].

The drivers of this transition are not incidental. Cost optimization increasingly encourages the use of differentiated accelerator tiers rather than exclusive reliance on a single performance class. Supply constraints further reinforce the operational necessity of mixed generation procurement and opportunistic deployment across available infrastructure pools. In parallel, inference architecture itself has become more structurally differentiated. Prefill and decode phases may be separated across distinct resources, memory intensive and latency sensitive stages may be assigned to different execution domains, and hybrid deployment patterns may distribute workloads across edge, regional, and centralized cloud environments [8,9]. These developments expand the reachable design space, but they also multiply the number of cross-layer interactions through which instability can emerge.

Heterogeneity should therefore not be interpreted as a temporary deviation from an otherwise homogeneous norm. In many large-scale deployments it is becoming the operational default, especially where multi-cloud strategy, sovereign control requirements, procurement constraints, or differentiated service classes prevent architectural uniformity [10,52]. What matters is not merely that systems contain diverse components, but that the behavior of the whole increasingly depends on whether these

components remain coherent when composed into a coupled runtime fabric. The relevant question is no longer how a single accelerator performs under ideal conditions, but how heterogeneous resources interact when routing, serving, memory pressure, and topology jointly determine observable system behavior.

The defining challenge of heterogeneous inference infrastructure is thus structural rather than component local. Individual accelerators may operate correctly, local schedulers may optimize according to their own objectives, and measured device utilization may appear acceptable, while the aggregate runtime still exhibits degraded coherence. In such cases, the resulting behavior cannot be adequately understood through a single layer metric because the instability is produced by the interaction surface between layers. This is the point at which classical notions of bottleneck analysis become insufficient. The problem is not one failing component, but the emergence of runtime behavior that is globally inconsistent despite local correctness [25,26].

1.2. Scope and Limitations

This paper presents a structural diagnostic analysis of publicly described architecture classes in heterogeneous AI inference infrastructure. Its aim is to clarify how cross layer incoherence can emerge when accelerator diversity, network topology, virtualization boundaries, and migration related reconfiguration interact under scale. The paper does not evaluate any vendor, product line, or cloud provider as such. References to specific technical systems are used only as public examples of broader architectural classes and should not be interpreted as endorsements, criticisms, or hidden case studies.

The contribution is analytical rather than empirical. No new benchmark campaign, controlled measurement study, or deployment specific performance evaluation is presented here. Accordingly, the claims developed in the paper are not framed as quantified empirical findings, but as structural observations about recurrent interaction patterns that can arise in large scale heterogeneous inference systems. The emphasis lies on diagnostic framing, taxonomic precision, and cross layer interpretation. This distinction is important because the instability modes analyzed here often remain partially invisible to conventional benchmark methodologies even when they are operationally significant.

The paper is also non prescriptive in scope. It does not propose scheduling algorithms, orchestration policies, virtualization controls, or migration mechanisms as implementation solutions. Nor does it claim that a single coordination layer or control technique would universally resolve the identified problems. The purpose is more limited and more foundational. It is to describe a class of infrastructural conditions under which large scale inference can become unstable despite the apparent adequacy of local performance indicators. In this sense, the paper is intended as a research framework for diagnosis rather than as a deployment manual for intervention.

Finally, the analysis is constrained to system classes in which heterogeneity is architecturally consequential. It does not attempt to generalize its claims to all forms of model serving or to small scale inference regimes where single device behavior dominates. The structural phenomena of interest arise only when execution depends on distributed composition across multiple layers, each of which can remain locally correct while the coupled system drifts into incoherence. This bounded scope is deliberate. It preserves analytical clarity and avoids overextending the argument beyond the environments in which the identified instability modes are most likely to matter.

1.3. System Class and Failure Envelope

The target system class of this paper consists of large scale inference deployments whose behavior is shaped by heterogeneous composition rather than by a single uniform execution substrate. These systems typically involve multiple accelerator types or generations, disaggregated serving paths in which prefill and decode may occur on different resources, cloud or multi cloud execution environments, complex placement topologies, and latency profiles dominated by decode stage sensitivity or memory path asymmetry. They may also exhibit substantial KV cache pressure, cross region or cross provider routing, and runtime conditions in which effective capacity depends on communication and control structure as much as on nominal compute availability [6–8].

This definition intentionally excludes small single node and single accelerator inference scenarios. In such settings, performance degradation is often attributable to device local phenomena that can be diagnosed within a comparatively narrow resource model. The present paper is concerned instead with behaviors that emerge from composition at scale, where the serving path traverses multiple interacting layers and where the aggregate system can no longer be interpreted as a simple extension of individual component characteristics. Heterogeneous inference becomes analytically distinctive only when the infrastructure itself contributes to runtime behavior through topology, virtualization, routing, and migration induced reconfiguration.

Within this system class, failure must be defined more broadly than hard crash or explicit service interruption. The relevant failure envelope includes emergent degradation modes such as tail latency inflation, effective capacity loss, throughput collapse under apparently acceptable utilization, cost escalation without proportional capability gain, and non deterministic serving behavior that persists despite the local correctness of constituent subsystems [26,31]. These forms of degradation are structurally important because they often remain below the threshold of catastrophic failure while nonetheless altering the operational meaning of scale. A system may continue to serve requests and remain nominally available, yet do so through an increasingly incoherent runtime in which observable efficiency diverges from actual infrastructural coordination.

The notion of a failure envelope is therefore central to the argument of this paper. It marks the boundary within which heterogeneous inference systems remain technically functional but structurally unstable. This boundary is not determined by whether components continue to execute, but by whether the coupled runtime retains enough coherence for nominal resources to remain effectively usable. Once coherence degrades beyond that boundary, optimization at isolated layers becomes progressively less effective because the system level problem is no longer one of local tuning. It is one of structural inconsistency across the execution fabric itself. The analysis that follows is directed at understanding that inconsistency as a distinct object of AI infrastructure research.

2. The Shift from Homogeneous Compute to Heterogeneous Inference Fabrics

The transition from homogeneous compute infrastructure to heterogeneous inference fabrics marks a structural change in how large scale AI systems must be analyzed. Earlier infrastructure logic was often built around the expectation that additional capacity of the same type would preserve the basic geometry of execution. Under this assumption, scaling could be treated primarily as a quantitative extension of a known substrate. More compute implied more throughput, larger fleets implied higher aggregate serving capacity, and scheduling could rely on a simplified view of nodes as broadly interchangeable. While this abstraction was never exact, it remained analytically useful so long as hardware similarity, communication symmetry, and relatively stable runtime conditions limited the emergence of cross layer divergence.

That assumption becomes increasingly inadequate once inference execution is distributed across mixed accelerator classes, differentiated memory hierarchies, nonuniform interconnect paths, and environment specific virtualization layers. In such systems, scaling no longer consists only in adding units of comparable performance. It changes the structure of execution itself. The runtime becomes dependent on the relation between components rather than on the local behavior of each component taken separately. Accelerator assignment, memory residency, transfer cost, placement locality, and serving stage separation begin to interact in ways that alter effective capacity independently of nominal resource counts. The central analytical shift is therefore from quantity to composition. What matters is not only how much compute is provisioned, but how heterogeneous execution paths reshape the coherence of the system under load.

This section develops that shift in three steps. It first clarifies the assumptions that underlie classical notions of homogeneous scaling. It then examines why heterogeneous accelerators must be understood as active determinants of runtime behavior rather than as passive execution resources. Finally, it shows that large language model inference is internally divided into distinct operational

regimes whose hardware sensitivities differ in a way that makes heterogeneous composition especially consequential. The aim is to establish that heterogeneous inference fabrics are not simply broader deployment environments. They are a distinct infrastructural condition in which system behavior can diverge from traditional scaling expectations even when local resource utilization appears rational.

2.1. Traditional Assumptions of Homogeneous Scaling

Classical infrastructure scaling is built on the expectation that adding capacity of the same type yields approximately predictable improvements in aggregate system performance. This assumption underlies a wide range of operational practices, including fleet level capacity planning, scheduler design, admission control, and cost modeling. If additional nodes are sufficiently similar in compute profile, memory characteristics, and communication behavior, then system level reasoning can proceed through relatively stable abstractions. Capacity can be forecast from resource counts, placement can be optimized under simplified equivalence assumptions, and the performance effect of additional hardware can be estimated without reconstructing the full topology of the execution path [25,29].

The usefulness of homogeneous scaling lies in the simplifications it enables. Nodes can be treated as near interchangeable scheduling targets, memory bandwidth can be modeled as sufficiently uniform for coarse grained planning, communication costs can be approximated as symmetric within bounded deployment zones, and per request resource consumption can be estimated without conditioning every decision on hardware specific variance [27,28]. These assumptions reduce the dimensionality of infrastructure control. They do not eliminate performance bottlenecks, but they help contain them within a resource model that remains tractable at fleet scale. In this sense, homogeneity functions as a stabilizing abstraction for both operations and analysis.

Large scale inference increasingly operates outside that abstraction. Mixed accelerator generations, differentiated accelerator classes, and environment specific execution layers introduce variance that cannot be treated as marginal noise. A deployment that combines different performance envelopes, memory capacities, communication patterns, and virtualization behaviors is no longer governed by a single hardware model. It becomes a coupled system in which local scheduling and placement decisions have consequences that depend on the interaction between hardware, topology, and runtime state [19,20]. What changes is not merely the precision of performance forecasting, but the structural validity of the equivalence assumptions on which homogeneous scaling depends.

The breakdown of homogeneity is analytically important because it introduces qualitative rather than merely incremental change. Once the infrastructure contains heterogeneous execution paths, scheduling is no longer a problem of distributing load across comparable resources. Placement becomes path dependent, resource allocation acquires topology sensitive consequences, and the operational meaning of additional capacity depends on whether that capacity is accessible under the coherence constraints of the serving system [30]. Under these conditions, the traditional language of scale can become misleading. A larger fleet may exist in nominal terms while effective system behavior becomes less predictable, less stable, or less efficiently controllable.

2.2. Heterogeneous Accelerators as Active Runtime Determinants

In homogeneous infrastructure, accelerators are often treated as execution substrates whose primary relevance lies in their local compute characteristics. Heterogeneous inference systems do not permit that simplification. Different accelerators are not interchangeable units of generic throughput. They impose distinct constraints on batch formation, sequence handling, memory access structure, synchronization timing, and the control interfaces through which serving systems interact with hardware [19–21]. As a result, accelerator heterogeneity must be understood not as background variation in available hardware, but as a direct determinant of runtime behavior.

This distinction matters because inference serving is shaped by more than aggregate computational power. The runtime path of a request depends on whether particular requests can be co batched, whether prefill and decode phases can be colocated or should be separated, whether model parallel or pipeline strategies remain feasible under the available memory and communication profile, and how

latency accumulates across the fleet under hardware specific asymmetries [10,12,15]. Once hardware classes differ in the constraints they impose, the serving system itself becomes path dependent on accelerator assignment. The hardware no longer passively receives work. It actively conditions the way work must be organized.

This change is particularly visible in disaggregated or multi pool serving architectures. When different stages of inference are assigned to different accelerator classes, the runtime behavior of the system is no longer reducible to the performance profile of any single pool. Instead, execution depends on whether the relation between pools preserves enough coherence for data movement, scheduling transitions, and state transfer to remain operationally efficient [9,11]. Under these conditions, accelerator assignment is inseparable from control behavior. A request routed through one hardware path may experience a fundamentally different latency and memory trajectory than the same request routed through another, even if both paths are locally well utilized.

The structural consequence is that accelerator heterogeneity expands the effective control surface of the system. Runtime behavior is shaped not only by model structure and software logic, but also by the hardware specific conditions under which serving decisions are made. This introduces a new form of infrastructural sensitivity. Changes in accelerator mix can alter system behavior without any change to model weights, request semantics, or nominal throughput targets. Such behavioral variance is difficult to capture through conventional resource accounting because it arises from the interaction between control logic and execution substrate rather than from the local state of either one alone. In heterogeneous inference, the accelerator is therefore best understood as an active runtime determinant embedded in a coupled control fabric.

2.3. Prefill, Decode, Memory, and Latency as Distinct Regimes

Large language model inference is often discussed as though it were a single computational process with a unified performance profile. In practice, it is composed of multiple operational regimes whose hardware sensitivities differ substantially. At a minimum, this includes the prefill phase, in which the model processes the full input context, the decode phase, in which output tokens are generated autoregressively, and the management of key value cache state, which conditions both memory residency and token generation continuity [6,7,22]. These regimes are not interchangeable in computational character. They are governed by different constraints and therefore respond differently to heterogeneous hardware environments.

The prefill phase is comparatively compute intensive because it processes a large input sequence in parallel. Decode, by contrast, is often more strongly limited by memory bandwidth and state access because generation proceeds token by token while repeatedly consulting accumulated cache state. KV cache management introduces an additional layer of infrastructural dependency through allocation, transfer, retention, and possible eviction dynamics that depend on both memory capacity and inter device communication cost [14,18]. These differences imply that the performance of inference cannot be captured by a single hardware descriptor. Different parts of the serving path are optimized by different hardware characteristics.

Under heterogeneous infrastructure, the distinction between these regimes becomes structurally consequential. A hardware pool that performs well for prefill may be suboptimal for decode. A resource configuration that offers strong compute throughput may still produce poor end to end serving behavior if memory path constraints dominate the token generation regime. Similarly, a system may show satisfactory aggregate tokens per second while hiding latency asymmetries that arise because requests traverse execution paths with mismatched compute to memory ratios [8,9]. Aggregate performance indicators can therefore obscure rather than illuminate the actual structural causes of degradation.

Disaggregated serving architectures make this regime separation explicit by assigning prefill and decode to different hardware pools in order to improve specialization or isolation. This can be operationally effective, but it also introduces new dependencies. KV cache transfer, placement locality, and inter pool communication cost become part of the serving path itself. The resulting latency is

no longer determined solely by the local performance of either pool, but by whether the transition between pools preserves coherence in state movement and runtime timing [8,16]. Heterogeneous inference fabrics therefore expose a fundamental property of large scale model serving. Inference is not a monolithic compute problem. It is a coordinated multi regime process whose stability depends on whether compute, memory, and transfer conditions remain structurally aligned across the execution path.

3. Why Classical Metrics Fail

The expansion of heterogeneous inference infrastructure exposes a fundamental limitation in the measurement conventions that have historically guided performance analysis in large scale computing. Classical metrics remain useful, but their explanatory power depends on an implicit architectural assumption. They are most informative when the system under observation is sufficiently uniform that local measurements can be aggregated into a meaningful account of global behavior. Once inference execution is distributed across mixed accelerators, differentiated memory paths, nonuniform interconnects, virtualization layers, and cross environment placement boundaries, that assumption weakens. The resulting system can remain locally observable while becoming globally difficult to interpret. This is not because performance data disappear, but because the relation between measured variables and effective system behavior becomes structurally indirect.

The core problem is that most standard metrics are layer specific by design. GPU utilization describes the state of an execution substrate. Latency metrics describe response timing at a given observational boundary. Throughput describes aggregate output under a chosen measurement frame. Benchmark scores summarize capability under controlled conditions. Each of these metrics captures something real, but none is designed to represent the coupled interaction surface through which heterogeneous inference instability emerges. In large scale heterogeneous deployments, degradation often does not begin as a visible collapse within one layer. It begins as incoherence between layers whose local signals remain individually plausible. A system may therefore appear healthy in conventional dashboards while already operating inside a structurally degraded regime.

This section develops that claim in two steps. It first identifies the measurement gap that arises when classical performance metrics are applied to heterogeneous inference systems whose behavior depends on cross layer coordination rather than isolated resource efficiency. It then examines why the most consequential effects remain partially invisible under layer local observation. The argument is not that measurement is impossible, nor that conventional observability should be discarded. The argument is narrower and more structural. Classical metrics fail when they are treated as though local visibility were sufficient to explain composite behavior in infrastructures whose operational logic is distributed across interacting layers.

3.1. The Measurement Gap in Heterogeneous Inference

Standard inference metrics provide only partial visibility into the behavior of heterogeneous large scale systems. GPU utilization, average latency, aggregate throughput in tokens per second, and benchmark level task performance all describe important operational dimensions. Their limitation lies not in their validity, but in their observational scope. They primarily measure within layer performance, whereas heterogeneous inference instability is often produced by interaction effects that arise across layers. When execution depends on the relation between accelerator assignment, memory residency, communication topology, virtualization boundaries, and scheduling logic, local metrics become insufficient as explanations of global system behavior [26,37].

This insufficiency is especially clear in the distinction between nominal and effective capacity. A heterogeneous system may report high utilization at the device level while a substantial fraction of its nominal resource pool remains operationally inaccessible for productive work. Synchronization overhead, transfer friction, resource fragmentation, or topology constrained placement can reduce the fraction of capacity that can actually be coordinated into coherent serving behavior. Under such conditions, local activity remains visible, but the operational meaning of that activity becomes

ambiguous. Utilization may remain high precisely because the system is compensating for structural inefficiency rather than because it is converting available hardware into proportional throughput [30, 63].

A similar problem affects latency measurement. Average latency is often operationally attractive because it compresses request behavior into a tractable fleet level signal. In heterogeneous inference systems, however, the decisive performance constraint is frequently determined not by the average path but by the longest structurally exposed path. Tail latency accumulates through the slowest or least coherent interaction segment, whether that segment is induced by cross accelerator transfer, topology dependent decode placement, virtualization overhead, or queueing asymmetry across mixed hardware pools. Average latency therefore conceals the extent to which serving quality may already be dominated by a small but structurally persistent set of degraded execution paths [26,39].

Benchmarks introduce a further form of measurement gap. Most evaluation environments are executed on dedicated or relatively controlled infrastructure designed to reduce confounding variance and maximize comparability. Heterogeneous production inference rarely operates under such conditions. Multi tenant contention, shared interconnect pressure, dynamic scheduling, resource oversubscription, and environment specific orchestration behavior all alter the runtime properties of the serving path. As a result, benchmark performance can remain useful as an indicator of model capability or isolated hardware efficiency while failing to describe the structural behavior of the deployed inference system itself [40,65]. The gap is therefore not only between measurement and reality, but between homogeneous evaluation logic and heterogeneous operational conditions.

The measurement gap in heterogeneous inference should therefore be understood as a structural mismatch between what is observed and what determines effective behavior. The usual metrics still report real signals, but they do so at observational boundaries that no longer coincide with the boundaries at which the most consequential infrastructural effects emerge. This mismatch becomes increasingly important as execution fabrics grow more heterogeneous, because the system level question is no longer simply how efficiently individual resources are used. It is whether those resources can be coordinated coherently enough for local efficiency to remain globally meaningful.

3.2. The Invisibility of Cross-Layer Structural Effects

The most consequential instability phenomena in heterogeneous inference infrastructures often arise between layers rather than within them. They emerge at the interaction surface between accelerator runtime and scheduler, between scheduler and network topology, between virtualization boundary and control plane, or between provider boundary and placement logic. This location of emergence is analytically significant. It means that the relevant degradation mode may have no direct representation within the monitoring framework of any single subsystem, even when each subsystem is extensively instrumented [53,54].

Layer local dashboards are designed according to the operational semantics of the layers they observe. Accelerator telemetry reports device level execution signals. Network monitoring reports link or path behavior within a defined communication domain. Virtualization observability describes tenant isolation, host level overhead, or orchestration level events. Scheduling systems expose queue states, allocation choices, and admission decisions. Each of these observational forms is locally rational. The difficulty is that heterogeneous inference instability often depends on the relation between them. Virtualization overhead may matter only when it interacts with accelerator specific timing sensitivity. Network topology may constrain placement only when serving logic assumes a communication symmetry that no longer holds. Cache transfer cost may become decisive only when decode stage placement is separated from prefill under mixed hardware conditions. These are not within layer failures, but cross layer dependencies whose operational significance is distributed across multiple observation boundaries [55,56].

This partial invisibility is not simply a tooling deficit that improved instrumentation will automatically remove. It reflects a deeper architectural gap between the boundaries at which current systems observe and the boundaries at which heterogeneous execution actually couples. Existing observability

frameworks are typically organized around separable subsystems because those subsystems correspond to administrative ownership, technical interfaces, and implementation layers. Heterogeneous inference fabrics, by contrast, generate instability along interaction boundaries that cut across those separations. The resulting mismatch means that even detailed local telemetry can fail to produce an intelligible system level account of why throughput degrades, why tail latency inflates, or why cost rises without commensurate output improvement [1,62].

The operational consequence is that a composite system can exhibit pervasive degradation without presenting a singular fault signature. Accelerators may appear busy, network links may remain below saturation, scheduling queues may look manageable, and virtualization infrastructure may show no explicit anomaly, while the coupled runtime nonetheless drifts into a state of reduced effective capacity and unstable serving quality. Such cases are difficult to diagnose precisely because no individual layer is necessarily malfunctioning in isolation. The system is degraded by structural inconsistency across the execution fabric rather than by visible failure inside one component domain.

For heterogeneous inference research, this implies that measurement adequacy cannot be judged solely by the granularity of local observability. A system may be richly instrumented and still remain structurally opaque if the decisive effects are produced at boundaries not represented in the measurement architecture. The problem is therefore not the absence of data in a narrow sense. It is the absence of a structural interpretive frame capable of relating locally valid signals to globally emergent behavior. This is why classical metrics fail most clearly in heterogeneous inference environments. They are not wrong. They are incomplete with respect to the interaction geometry that now determines system behavior.

4. Structural Sources of Runtime Incoherence

The transition to heterogeneous inference infrastructure changes not only the scale of execution but also the location at which instability originates. In relatively homogeneous systems, degradation can often be traced to identifiable local bottlenecks, including device saturation, memory exhaustion, or network congestion within a comparatively uniform execution model. In heterogeneous inference fabrics, by contrast, instability frequently originates at the interfaces between components that remain locally functional but become mutually misaligned when composed into a shared runtime. The problem is therefore not only resource insufficiency. It is the loss of coordination across the execution surface through which serving is realized.

This section analyzes where runtime incoherence originates. The discussion is organized around four structural source domains. The first concerns accelerator heterogeneity, where runtime assumptions formed under hardware similarity are extended across non-equivalent execution substrates. The second concerns network and placement coupling, where physically present capacity becomes only conditionally usable because communication topology and placement logic interact. The third concerns virtualization-induced distortion, where abstraction layers alter timing, contention, and resource visibility in ways that change the execution surface itself. The fourth concerns migration-induced reconfiguration, where movement across infrastructure environments rewrites the control geometry under which serving takes place.

Taken together, these domains identify the principal origins of runtime incoherence in heterogeneous inference systems. They are not taxonomic categories in the strict sense and are not intended to classify observed instability outcomes. Rather, they specify the source regions within which incoherence is generated before it appears as degraded throughput, inflated tail latency, or reduced effective capacity. The analytical purpose of the present section is therefore causal localization. It clarifies where structural instability enters the system. Section 5 then formalizes how the resulting instability should be classified once it becomes visible at system level.

4.1. Accelerator Heterogeneity and Control Mismatch

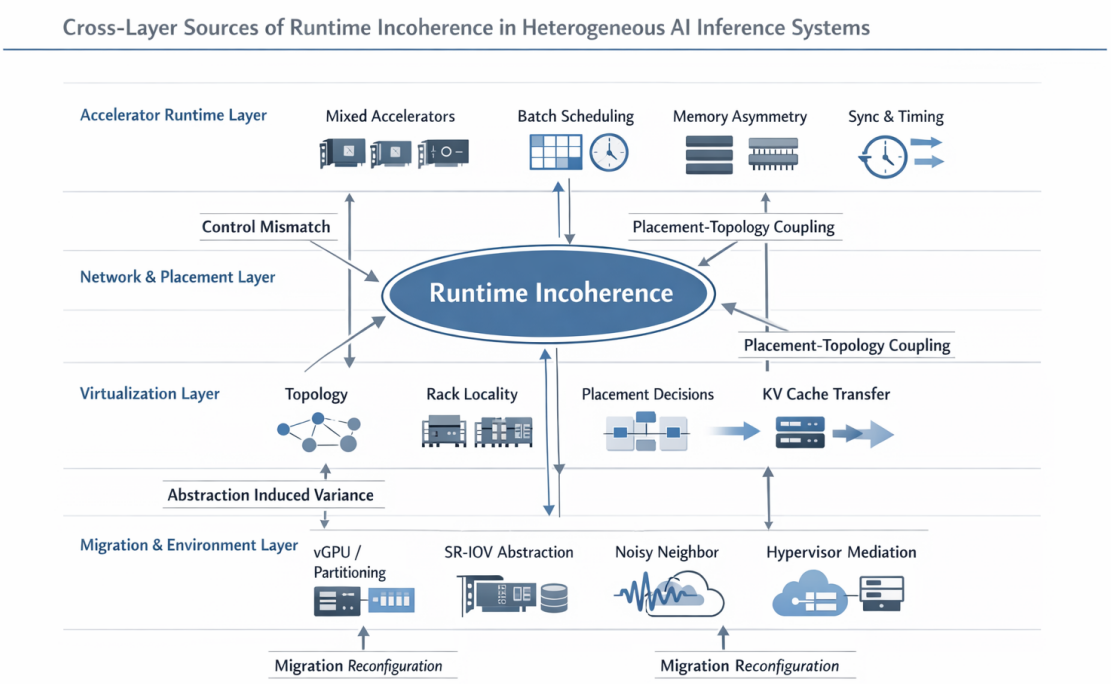

Figure 2. Cross-layer sources of runtime incoherence in heterogeneous AI inference systems. The figure organizes the principal interaction surfaces across accelerator runtime, network and placement, virtualization, and migration or environment layers. The central point is causal rather than taxonomic. Instability originates at the coupling boundaries between layers, not only within any single layer taken in isolation.

A primary origin of runtime incoherence lies in the extension of control logic across accelerators that do not share the same execution assumptions. Different accelerator types encode different timing characteristics, memory-access behavior, batch-formation constraints, synchronization costs, and control-path latencies. Runtime systems developed under homogeneous fleet assumptions can absorb moderate variance within a single hardware family, but they become structurally fragile when the same control assumptions are projected across non-equivalent execution substrates [10,15]. The source of instability is therefore not hardware diversity in itself. It is the mismatch between the control model and the substrate on which that model is applied.

This source domain becomes visible when logically similar execution stages must be coordinated across hardware classes with different queueing dynamics, collective-operation behavior, and state-transition properties. Under such conditions, synchronization ceases to be a neutral coordination mechanism. It becomes a source of structural tension because the hardware-specific assumptions required to maintain alignment are no longer shared across the serving path [33,38]. The resulting incoherence originates at the point where shared orchestration presumes equivalence that the hardware layer does not actually provide.

Memory asymmetry intensifies this source of misalignment. When workloads are distributed across accelerators with different memory bandwidth and capacity regimes, the serving path inherits an internal asymmetry that cannot be reduced to nominal aggregate compute. This is especially significant in inference regimes where decode behavior, cache access, and state continuity are strongly conditioned by memory characteristics. In such cases, the effective ceiling of the composite system is set not by the total amount of compute present, but by the least compatible memory segment on the active execution path [20,21]. The origin of instability thus lies in the divergence between how the runtime expects the path to behave and how the memory structure forces it to behave.

Abstraction layers intended to simplify heterogeneous deployment can further deepen this problem. Unified execution frameworks, translation layers, and driver-mediated orchestration often introduce non-uniform overhead across accelerator classes, creating hardware-dependent control surfaces beneath a common software interface [32,41]. In such cases, the source of incoherence is not only heterogeneity at the device level, but the attempt to govern heterogeneous hardware through abstractions that preserve apparent portability while concealing non-equivalent execution conditions. Accelerator heterogeneity is therefore a causal source domain because it generates instability at the point where hardware diversity and runtime control meet.

4.2. Network and Placement Coupling Effects

A second source domain of runtime incoherence lies in the interaction between communication topology and placement logic. In large-scale heterogeneous inference, capacity is not defined solely by the physical presence of accelerators or memory resources. It is also defined by whether those resources can be arranged into execution paths whose communication properties remain coherent under load. Network topology, rack locality, node adjacency, and scale-out distribution therefore shape the difference between nominal and effective capacity [34,35,37]. The origin of instability in this domain is not the network taken alone. It is the coupling between where work is placed and how the communication substrate can support that placement.

This source domain becomes more consequential under heterogeneity because placement is no longer a generic load-balancing problem. Requests may depend on accelerator-specific affinity, prefill-decode separation may require low-latency transfer across hardware pools, and KV-cache continuity may impose memory-path constraints that cannot be abstracted away by a scheduler operating on resource counts alone [8,9]. A placement decision that is locally rational in compute terms can therefore become globally destabilizing when it implies communication distance, transfer contention, or stage asymmetry that the serving system cannot absorb.

Topology effects become increasingly nonlinear as scale expands. A communication fabric that appears operationally adequate at one deployment size may behave differently once the number of coupled execution paths, routing dependencies, or coordination demands increases. Control planes that remain stable under current load may encounter convergence delays, reduced fault tolerance, or path-level instability as the serving graph becomes larger and more heterogeneous [35,36]. The source of incoherence is thus relational. It appears where the topological assumptions implied by placement no longer match the communication geometry required for coherent execution.

For this reason, network and placement should be analyzed as a unified causal domain. Capacity in heterogeneous inference is not merely allocated and then consumed. It is conditionally realizable depending on whether placement preserves sufficient topological coherence for hardware resources to remain operationally reachable [1,38]. The origin of instability lies precisely in the gap between those two conditions.

4.3. Virtualization-Induced Distortion

A third source domain of runtime incoherence is virtualization-induced distortion. Virtualization changes heterogeneous inference not merely by adding overhead, but by altering the coupling topology between workloads and the physical resources on which they depend. GPU partitioning, virtual GPU layers, SR-IOV based network abstraction, RDMA passthrough arrangements, and hypervisor-mediated scheduling all introduce execution boundaries that differ materially from those of bare-metal environments [24,42]. The origin of instability in this domain lies in that altered boundary structure itself.

This source domain is defined by the fact that virtualization transforms deterministic or bounded hardware behavior into performance distributions whose realized timing and contention depend partly on factors outside the tenant-local workload. Co-tenant activity, hypervisor scheduling choices, arbitration policies for shared resources, and implementation-specific abstraction behavior all shape execution under load [43,44]. The source of incoherence is therefore not a fixed penalty that can simply

be normalized away. It is the introduction of mediation points through which control, timing, and resource visibility are structurally transformed before performance degradation is later observed.

Noisy-neighbor interference illustrates this clearly. A workload may experience degraded serving quality not because its own internal state changed, but because the abstraction layer exposes it to coupling paths generated by concurrent tenant activity. These paths may remain invisible to workload-local telemetry even while they materially alter latency distributions and effective controllability [24,45]. Virtualization becomes a source domain of incoherence because it changes the relation between intent, execution, and observation before instability is classified at the outcome level.

Partitioning mechanisms such as MIG refine this point rather than eliminate it. They can strengthen isolation by providing dedicated memory and compute segments, yet they also impose fixed resource granularity that may not match workload demand [24,46]. The resulting underutilization, overprovisioning, or scheduling rigidity does not originate in application logic alone. It originates in the abstraction boundary through which resources are exposed. Virtualization-induced distortion should therefore be understood as a source of runtime incoherence because it rewrites the execution surface on which higher-level control depends.

4.4. Migration-Induced Reconfiguration Risk

A fourth source domain of runtime incoherence is migration-induced reconfiguration. Migration across providers, regions, or infrastructure classes is often treated as a deployment or portability problem. In heterogeneous inference systems, however, it is more accurately understood as a structural rewriting of the runtime environment itself. When workloads move between infrastructure contexts, the functional interfaces available to the application may remain broadly stable, yet the coupling relations between scheduler, runtime, placement logic, storage path, and communication fabric can change substantially [47,49]. The origin of instability lies in that reconfiguration of control geometry.

This source domain becomes visible when workloads that were implicitly adapted to one environment continue to execute correctly after migration but do so under altered timing, transfer, and locality conditions. Dedicated interconnect assumptions may be replaced by shared network fabrics, local storage paths by network-attached storage, and bare-metal execution by virtualized or semi-abstracted runtime layers [48,50]. The resulting instability does not originate in code-level incompatibility. It originates in the preservation of functionality under changed infrastructural relations.

Cross-cloud and multi-cloud migration make this source domain even more consequential. Different environments expose different accelerator availability profiles, virtualization regimes, network control planes, and latency distributions at both intra-region and inter-region scales [51,52]. A scheduler, cache strategy, or placement heuristic that remains coherent in one environment may become destabilizing in another without any change to model weights or application logic. Migration is therefore a source of runtime incoherence because it changes the environmental assumptions under which serving remains coordinated.

The central point is that migration-induced instability originates before overt failure appears. The system continues to execute, but under a reconfigured set of infrastructural relations that may no longer preserve the coherence on which the original serving behavior depended [27,30]. For heterogeneous inference research, migration belongs in the causal analysis of runtime incoherence because it identifies where instability enters the system through environmental change rather than where it is later observed as degraded performance.

5. Structural Taxonomy of Heterogeneous Inference Instability

Section 4 identified the principal source domains from which runtime incoherence originates. The present section addresses a different analytical question. It asks how the resulting instability should be classified once it becomes visible at system level. The purpose is therefore not to restate the causal origins of incoherence, but to provide a formal vocabulary for describing the characteristic ways in which incoherence manifests as degraded system behavior. In this sense, the taxonomy developed here classifies observed structural instability, whereas Section 4 localized its primary points of origin.

This taxonomy constitutes the principal conceptual contribution of the paper. Its role is diagnostic rather than prescriptive. It does not enumerate product defects, implementation errors, or deployment accidents, nor does it propose remedies. Instead, it identifies recurrent instability modes through which cross-layer incoherence is expressed in heterogeneous inference systems even when constituent components remain locally functional [57,58]. The objective is to distinguish the dominant forms taken by structural degradation once it is operationally manifest. Resolution strategies are intentionally not developed here, because deriving implementation-specific interventions from a structural classification alone would exceed the scope of the argument and collapse distinctions that the taxonomy is meant to preserve.

Five modes are introduced. Latency-asymmetry drift classifies instability expressed through cumulative divergence in execution timing. Memory-path incoherence classifies instability expressed through divergence between visible compute availability and actual memory-constrained serving behavior. Interconnect-induced capacity inaccessibility classifies instability expressed through the gap between provisioned and topologically usable capacity. Virtualization-induced control distortion classifies instability expressed through degraded correspondence between control intent, realized execution, and observed system state under abstraction. Migration-induced runtime reconfiguration risk classifies instability expressed through altered performance distributions and control behavior after environmental transition. These modes are analytically distinct, although they may interact in practice. Their separation enables clearer diagnosis of heterogeneous inference instability without reducing all manifestations to a single undifferentiated notion of degradation.

5.1. Latency-Asymmetry Drift

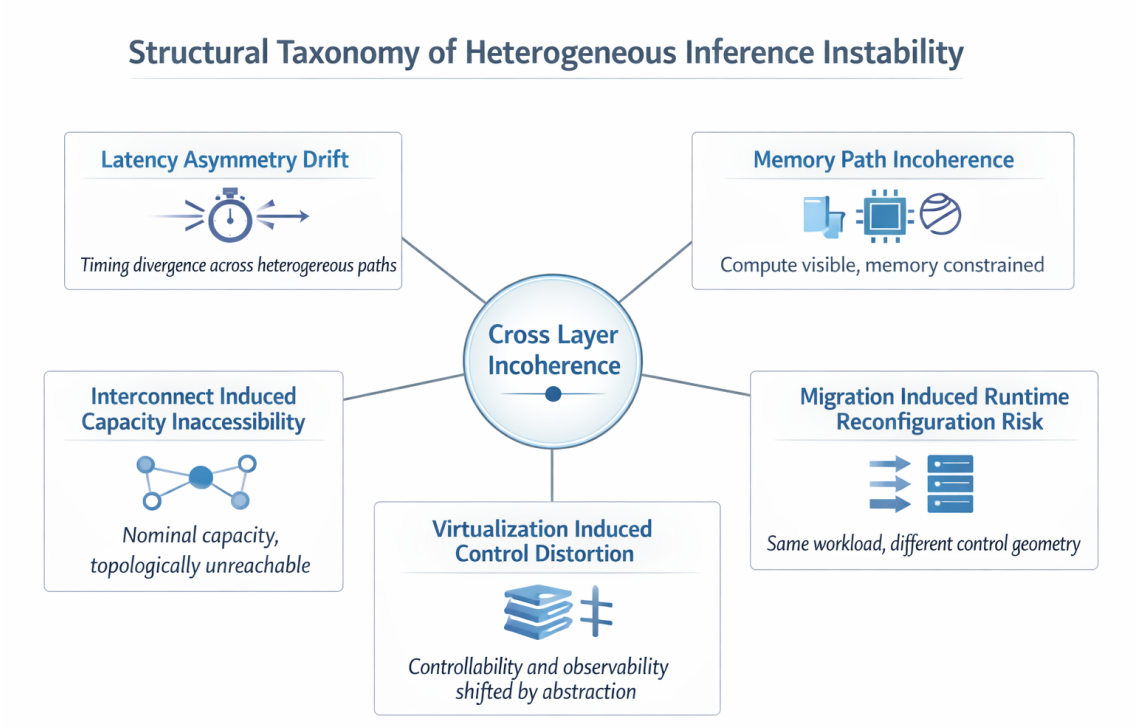

Figure 3. Structural taxonomy of heterogeneous inference instability. The five modes identified in this paper, latency-asymmetry drift, memory-path incoherence, interconnect-induced capacity inaccessibility, virtualization-induced control distortion, and migration-induced runtime reconfiguration risk, are represented as distinct but related expressions of cross-layer incoherence.

Latency-asymmetry drift denotes a condition in which locally efficient execution across heterogeneous hardware paths produces cumulative divergence between scheduling assumptions and realized runtime timing.

This instability mode is expressed when the temporal model implicit in orchestration or scheduling logic no longer matches the timing actually produced by the heterogeneous runtime [12,26]. Its defining property is cumulative divergence rather than discrete fault. No single control action need be incorrect, and no individual device need exhibit anomalous behavior. Instead, repeated execution across non-equivalent latency domains gradually shifts the serving system away from the symmetry assumptions on which coordinated behavior depends.

Operationally, latency-asymmetry drift manifests as inflated tail latency, unstable response ordering, and reduced predictability in end-to-end serving behavior [31,40]. The mode becomes particularly pronounced in disaggregated serving architectures, where prefill and decode execute on different hardware pools and where KV-cache transfer timing introduces an additional source of path-dependent temporal variance [8,14]. The classification is therefore appropriate whenever the dominant expression of incoherence is temporal divergence accumulating across heterogeneous execution paths.

5.2. Memory-Path Incoherence

Memory-path incoherence denotes a condition in which reported compute availability diverges from actual serving capacity because execution is constrained by heterogeneous memory structure, bandwidth, or access latency.

This instability mode is expressed when the system appears to possess available compute capacity while actual serving behavior is limited by the composite memory path through which inference must proceed [6,18]. The defining characteristic is divergence between compute-visible state and memory-determined operational reality. In large language model inference, this is especially consequential because decode-stage behavior and KV-cache continuity depend heavily on memory bandwidth, capacity, and residency rather than on arithmetic throughput alone.

The mode becomes visible when heterogeneous accelerators with different memory hierarchies, bandwidth envelopes, or cache structures share execution responsibility. Under such conditions, the effective capacity of the serving path is bounded by the most constrained or least coherent memory segment rather than by aggregate compute potential [17,21]. The diagnostic difficulty arises because compute-centric monitoring may continue to show healthy utilization while the actual limiting factor lies in cache residency, cross-device state transfer, or memory-path contention visible only when data movement is traced across the full serving pipeline [6,23]. Memory-path incoherence is therefore the appropriate classification when instability is expressed primarily through mismatch between visible compute state and hidden memory-path constraint.

5.3. Interconnect-Induced Capacity Inaccessibility

Interconnect-induced capacity inaccessibility denotes a condition in which provisioned resources remain only partially usable because the communication topology cannot support the execution geometry required for coherent serving.

This instability mode is expressed when capacity that is nominally present in the infrastructure cannot be converted into productive work because the serving graph is topologically misaligned with the communication substrate [34–36]. The defining feature is a widening gap between provisioned and effective capacity that emerges without explicit device failure or overt network collapse. Resources exist, and may appear locally healthy, yet remain only conditionally reachable for coherent execution.

In heterogeneous inference, this mode becomes more pronounced because different serving decompositions and accelerator configurations impose different communication demands, including collective synchronization, point-to-point transfer, and stage-to-stage cache handoff [33,38,66]. A topology that supports one communication pattern efficiently may render another only partially realizable. The resulting manifestation is throughput that fails to scale proportionally with added hardware, not because the devices are absent, but because the interconnect cannot operationalize them at system level [1,37]. The classification therefore applies when the dominant observable effect of incoherence is the inaccessibility of nominal capacity under real communication constraints.

5.4. Virtualization-Induced Control Distortion

Virtualization-induced control distortion denotes a condition in which abstraction layers alter the relation between control intent, realized execution, and observed system state, thereby degrading both controllability and observability.

This instability mode is expressed when hypervisors, virtual GPU schedulers, partitioning mechanisms, or mediated network interfaces cause the same steering signal to produce different runtime effects than it would under more direct physical execution conditions [24,42]. The defining feature is not fixed overhead alone. It is distortion in the mapping between decision and effect. Under such conditions, scheduling, placement, or isolation choices become harder to interpret because execution is mediated by abstraction boundaries whose behavior is only partially visible to the workload.

In multi-tenant environments, this distortion is often coupled to concurrent tenant activity, which introduces non-deterministic variation into both observed performance and control effectiveness [43–46]. The instability mode is therefore manifested when the operator can no longer reliably infer how control actions map to resource behavior, or how measured metrics map to physical execution conditions. Virtualization-induced control distortion is thus the appropriate classification when instability is expressed as degraded correspondence between intent, execution, and interpretation under abstraction.

5.5. Migration-Induced Runtime Reconfiguration Risk

Migration-induced runtime reconfiguration risk denotes a condition in which a workload remains functionally portable across environments while its performance distributions, control behavior, or instability profile change because the underlying control geometry has been reconfigured.

This instability mode is expressed when movement across providers, regions, or infrastructure classes preserves software-level functionality but alters the infrastructural relations under which serving occurs [47,48]. The defining feature is structural rather than functional divergence. The workload continues to execute, yet the target environment encodes different assumptions about network behavior, storage locality, virtualization, accelerator availability, or control-plane interaction.

The operational manifestation is drift in latency distributions, contention structure, tail behavior, or cost efficiency under otherwise comparable demand [49,50]. This mode is especially acute for inference workloads governed by strict service objectives, where even modest changes in timing distributions can produce disproportionate growth in violation rates [8,26]. Migration-induced runtime reconfiguration risk is therefore the appropriate classification when instability appears after environmental transition, not because the workload ceased to function, but because functional continuity concealed a deeper change in runtime geometry.

6. Runtime Coherence as the Hidden Performance Variable

The preceding analysis has identified multiple sources of instability in heterogeneous inference infrastructure and organized them into a structural taxonomy. What remains is to clarify why these modes should not be understood as independent operational irregularities, but as expressions of a deeper systems condition. The central claim of this section is that runtime coherence functions as a hidden performance variable in heterogeneous large scale inference. It is hidden not because it is metaphysical or immeasurable, but because it is not directly represented by the conventional metrics through which AI infrastructure is usually evaluated. Throughput, utilization, average latency, and benchmark performance all describe important system properties. None of them, however, directly captures whether the coupled execution fabric remains sufficiently coordinated for nominal resources to be transformed into stable and economically meaningful serving behavior.

Runtime coherence, as used here, refers to the degree to which control logic, hardware behavior, memory paths, communication topology, and placement decisions remain mutually compatible across the serving system. A coherent runtime is not one in which all layers are identical or perfectly synchronized. It is one in which cross layer interactions preserve enough consistency that local

optimization remains globally intelligible. An incoherent runtime is therefore not defined by the visible failure of a specific subsystem. It is defined by a condition in which individually rational actions, healthy local metrics, and nominal resource availability no longer compose into stable system behavior. This distinction is essential in heterogeneous inference because infrastructure diversity increases both the number of interaction surfaces and the number of ways in which those surfaces can drift apart under load.

In this paper, the term *runtime coherence* is used in a broader sense than *runtime control coherence* in prior work. Earlier usage emphasized logical coupling across scheduling, orchestration, and policy layers. The present paper extends the concept to the full cross-layer coordination surface of heterogeneous inference systems, including accelerator behavior, memory-path structure, network and placement topology, virtualization boundaries, and migration-sensitive environmental reconfiguration. This terminological extension is deliberate. It reflects the fact that, under heterogeneous inference conditions, instability is not confined to logical control interactions alone but emerges across the coupled infrastructural relations through which serving is realized.

The argument developed in this section proceeds in four stages. First, it explains why local optimization across independently managed layers can produce global instability when their interaction structure is not coherently aligned. Second, it examines the multiple control loops that coexist across scheduler, runtime, serving, and placement layers, and shows why their interference becomes more consequential under heterogeneity. Third, it analyzes the phenomenon of hardware dependent behavioral variance, in which the same model and request distribution can exhibit materially different runtime behavior solely because the infrastructure path differs. Fourth, it draws out the structural consequences of incoherence for throughput, tail latency, and effective capacity. Taken together, these steps make visible why runtime coherence should be treated as a first order systems variable rather than as a residual effect of other metrics.

6.1. Why Local Optimization Produces Global Instability

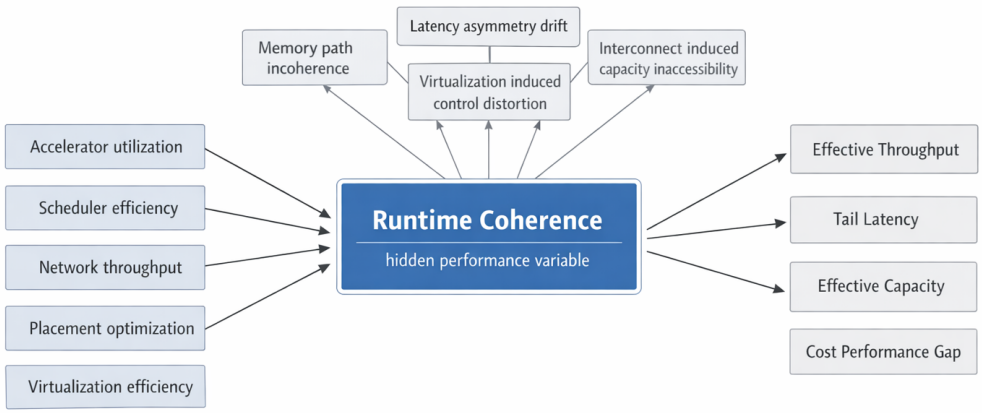

Figure 4. Runtime coherence as the hidden performance variable. Local optimization signals such as accelerator utilization, scheduler efficiency, network throughput, placement optimization, and virtualization efficiency do not directly determine system outcomes. Their effect is mediated by runtime coherence, which conditions effective throughput, tail latency, effective capacity, and the cost-performance gap under heterogeneous inference.

A central systems error in large-scale infrastructure analysis is to assume that improved behavior at the level of individual layers necessarily aggregates into improved behavior at the level of the whole. In heterogeneous inference, this assumption is especially fragile. Accelerator utilization may be improved through more aggressive batching, scheduler efficiency may be improved through tighter packing, network throughput may be improved through path optimization, and placement logic may be improved through more rapid resource assignment, while the composite runtime nonetheless becomes less stable. The reason is that these optimizations act on partially shared system state without necessarily preserving cross-layer consistency [57,58,60].

This is a familiar result in complex systems theory. Locally rational decisions can produce collectively suboptimal or unstable outcomes when the interaction structure between decision agents is insufficiently modeled or insufficiently coordinated [59,61]. Heterogeneous inference infrastructures instantiate this condition in technical form. The system is composed of interacting control domains whose local objectives are not identical and whose effects propagate across one another through hardware-specific timing, topology, memory-path constraints, and abstraction boundaries. Under such conditions, there is no guarantee that optimizing each domain independently will improve the behavior of the coupled system. On the contrary, local success may intensify global incoherence when it sharpens asymmetries that another layer is unable to absorb.

Heterogeneity increases the severity of this problem because it multiplies the number of dimensions along which interaction can become unstable. Each additional accelerator class, virtualization layer, network regime, or provider boundary introduces a new source of conditional behavior that local optimization does not resolve by itself [2,3]. What was once a comparatively bounded scheduling or routing problem becomes a coupled coordination problem in which local decisions alter the assumptions under which other local decisions were made. The resulting instability is not accidental. It is the predictable consequence of optimizing subsystems in an environment whose governing behavior is relational rather than additive.

The implication is that performance improvement in heterogeneous inference cannot be understood solely as the sum of local efficiencies. The critical question is whether those efficiencies remain mutually compatible once composed into a single execution fabric. When they do not, the system enters a regime in which optimization itself becomes a source of instability. Runtime coherence therefore matters because it defines the boundary within which local improvements remain globally productive. Outside that boundary, more optimization at one layer can produce less stability at the level of the whole.

6.2. Control Loops Across Scheduler, Runtime, Serving, and Placement Layers

Large scale inference systems contain multiple coupled control loops that operate simultaneously across different parts of the infrastructure stack. At the accelerator level, batch formation, memory management, and execution timing are shaped by hardware aware scheduling. At the cluster level, orchestration systems govern placement, scaling, and allocation. At the serving level, request routers and load balancers determine how demand is distributed across execution paths. At the provider or infrastructure management level, capacity controllers regulate resource availability, migration, and environment level provisioning [27–29]. Each of these loops is locally meaningful. The challenge arises because they do not operate in isolation.

These loops are separated not only by function, but also by timescale and observational boundary. Accelerator scheduling acts at millisecond granularity, often under very fine resource and queue state assumptions. Cluster orchestration operates over longer intervals shaped by placement and allocation dynamics. Capacity managers act at longer timescales still, with visibility defined more by fleet conditions than by per request behavior [13,30]. In homogeneous settings, such temporal separation may remain manageable because the underlying hardware and communication assumptions are comparatively stable. In heterogeneous environments, however, the state seen by one control loop may change in ways induced by another loop and conditioned by infrastructure characteristics that neither loop fully observes.

Heterogeneous hardware intensifies this coupling because control logic must incorporate hardware dependent conditions into its own decision process. Accelerator type influences feasible batch composition, memory exposure, latency profile, and transfer behavior. These hardware specific features introduce conditional logic and type dependent parameters that expand the dimensionality of the control space beyond what homogeneous fleet abstractions were designed to handle [10,12]. A decision that is optimal under one hardware path may become destabilizing under another, even when implemented by the same nominal control mechanism.

Cross loop interference follows from this expanded control surface. An action taken by one loop can alter the conditions that another loop had been optimizing for, thereby generating feedback dynamics that amplify rather than attenuate perturbation [2,28]. A scheduler that changes batch structure may alter memory pressure in a way that shifts routing effectiveness. A placement controller that improves local resource packing may increase communication asymmetry across serving stages. A migration decision made for capacity reasons may alter virtualization and interconnect behavior sufficiently to invalidate prior latency expectations. In such cases, instability is not the result of absent control. It is the result of multiple active controls interacting without a stable cross layer coherence condition. This is why runtime coherence should be interpreted not as a supplementary systems property, but as the necessary coordination state within which coupled control loops can remain globally constructive.

6.3. Hardware-Dependent Behavioral Variance Without Model Change

One of the defining features of heterogeneous inference infrastructure is that the same model, serving configuration, and input distribution can produce measurably different runtime behavior solely because the infrastructure path changes. The accelerator type that executes the request, the network path over which state is transferred, the virtualization boundary that mediates access to physical resources, and the placement relation between serving stages can all alter realized behavior without any modification to model weights or application logic [9,11]. This phenomenon is structurally important because it means that performance variation cannot be attributed only to workload or algorithmic differences. Infrastructure path itself becomes a determinant of behavior.

Such variance should not be misunderstood as incidental noise or operational imperfection. In heterogeneous systems, it is systematically induced by the material properties of the execution substrate. Different accelerator types expose different timing and memory characteristics. Different communication paths introduce different transfer costs and synchronization conditions. Different virtualization regimes mediate control and observability differently. The consequence is that runtime behavior becomes path dependent even when the semantic task being performed remains unchanged [18,21]. The same computation is therefore not operationally identical across all infrastructure paths. It is transformed by the environment in which it is realized.

This observation has direct consequences for performance characterization. Benchmarks executed on a single hardware class or under isolated test conditions may accurately describe the behavior of the model in that environment, but they do not automatically generalize to heterogeneous deployment regimes. If infrastructure path changes the operational meaning of the same model execution, then single hardware characterization becomes an incomplete basis for deployment level inference about performance [63,64]. What is required instead is infrastructure path aware measurement capable of identifying how different runtime trajectories alter serving quality, latency distribution, and effective capacity.

The broader significance of hardware dependent behavioral variance is conceptual. It reveals that infrastructure is not merely the container within which inference occurs. It is part of the behavioral definition of the deployed system. This does not eliminate the value of model centric evaluation, but it limits its sufficiency. In heterogeneous inference environments, the operational system is the model plus the runtime path through which the model is served. Runtime coherence matters precisely because it determines whether path dependent variance remains bounded and intelligible or expands into instability that conventional model centered metrics cannot explain.

6.4. Structural Consequences for Throughput, Tail Latency, and Effective Capacity

The five instability modes identified in Section 5 do not remain isolated when heterogeneous inference systems operate under scale. They interact multiplicatively and sometimes recursively. Latency asymmetry drift can intensify the impact of memory path incoherence by widening the timing window over which cache transfer becomes destabilizing. Interconnect induced capacity inaccessibility can magnify virtualization induced variance by narrowing the set of topologically viable execution paths. Migration induced reconfiguration can alter the entire coupling structure within which the other four modes unfold [26,40]. The effect is that incoherence accumulates not linearly but compositionally across the execution fabric.

Throughput is therefore best understood in at least two forms. Nominal throughput may be measured as aggregate tokens generated or requests processed over time. Effective throughput is narrower and more operationally relevant. It refers to the subset of throughput realized within acceptable latency and service objective constraints [8,13]. In heterogeneous inference systems, the gap between nominal and effective throughput can widen substantially even when resource utilization appears healthy. This gap reflects a structural efficiency problem rather than a simple lack of compute. The system continues to generate output, but a smaller fraction of that output is produced under conditions that are economically or operationally meaningful.

Tail latency exhibits a similar transformation. In homogeneous systems, tail behavior may be treated as an amplification of ordinary variance. In heterogeneous systems, tail latency is often determined by the slowest structurally exposed execution path rather than by the average path. As heterogeneity increases, the distribution of path latencies widens because requests encounter a larger range of hardware conditions, memory constraints, communication costs, and abstraction boundaries [26,31,39]. The resulting tails are heavier not only because there are more opportunities for delay, but because the system contains more structurally distinct ways in which delay can be produced and propagated.

Effective capacity provides the clearest link between runtime coherence and economic consequence. Provisioned capacity refers to the resources nominally available in the infrastructure. Effective capacity refers to the fraction of those resources that can actually be coordinated into productive serving behavior within the required operational envelope. When runtime coherence deteriorates, effective capacity declines even if provisioned capacity remains unchanged [3,65]. This creates a cost performance gap that cannot be diagnosed by utilization alone. Resources are present, active, and often expensive, yet a decreasing fraction of them can be converted into service that meets the intended quality and timing constraints.

The hidden character of runtime coherence therefore becomes visible through its consequences. It conditions whether nominal throughput becomes effective throughput, whether latency remains bounded or develops heavy structural tails, and whether provisioned resources remain economically usable or drift into partially inaccessible cost burden. These consequences are not secondary effects. They are the operational expression of whether the heterogeneous execution fabric remains coordinated enough for local performance signals to retain system level meaning. Runtime coherence is thus hidden only in the sense that it is not directly represented by standard metrics. Its absence becomes unmistakable in the divergence between what the system appears to contain and what it can actually do.

7. Structural Diagnostics and Application Mapping

The taxonomy developed in Section 5 is intended as a diagnostic classification of heterogeneous inference instability rather than as a standalone vocabulary. Its analytical value increases when each instability mode can be related to a corresponding diagnostic domain capable of examining the relevant structural interactions in a more explicit way. This section provides that mapping. The aim is not to claim that any single diagnostic framework resolves the instability modes identified above. The aim is narrower. It is to show that the cross-layer phenomena isolated in this paper correspond to distinct

analytical domains within the broader SORT-AI research program and can therefore be approached through structured diagnostic decomposition rather than through ad hoc troubleshooting or purely layer-local inspection [4,5].

This mapping serves two related purposes. First, it grounds the taxonomy in prior analytical work by showing that the identified instability classes are not isolated theoretical abstractions, but correspond to already differentiated domains of infrastructural diagnosis. Second, it clarifies that heterogeneous inference instability is not a singular problem with one universal explanatory layer. Different instability modes require different analytical entry points depending on whether the dominant source of incoherence lies in accelerator compatibility, network scaling and placement, virtualization-mediated distortion, or environmental reconfiguration through migration. The mapping is therefore interpretive rather than promotional. It identifies correspondence between the present paper's taxonomy and prior application domains without asserting exhaustive coverage or universal remediation.

The four primary diagnostic domains considered here correspond directly to the four application areas integrated into the present paper. AI.07 concerns accelerator runtime control across heterogeneous hardware fleets and is therefore aligned with instability modes centered on accelerator mismatch and hardware dependent execution divergence. AI.11 concerns structural network scalability risk modeling and aligns with interconnect-induced capacity inaccessibility together with placement-topology coupling. AI.14 concerns virtualization overhead stability analysis and aligns with abstraction-induced variance and control distortion under multi-tenant execution. AI.20 concerns structural cloud migration risk assessment and aligns with the reconfiguration of control geometry that occurs when workloads move across environments. Supporting roles are played by AI.04, AI.01, and AI.27, which sharpen the logical, physical, and end-to-end dimensions of coherence analysis, respectively.

Table 1. Mapping between instability modes, primary source domains, diagnostic domains, and dominant observable consequences.

Instability mode	**Primary source domain**	**Diagnostic domain**	**Dominant observable consequence**
Latency-asymmetry drift	Accelerator heterogeneity and control mismatch	AI.07 Accelerator Runtime Control, supported by AI.04 Runtime Control Coherence	Tail-latency widening, unstable response ordering, divergence between scheduling assumptions and realized timing
Memory-path incoherence	Accelerator heterogeneity and control mismatch	AI.07 Accelerator Runtime Control	Reported compute availability exceeds actual serving capacity, cache-pressure bottlenecks, decode-path degradation
Interconnect-induced capacity inaccessibility	Network and placement coupling effects	AI.11 Structural Network Scalability Risk Modeling, supported by AI.01 Interconnect Stability	Gap between provisioned and effective capacity, non-proportional throughput scaling, topology-bound resource inaccessibility
Virtualization-induced control distortion	Virtualization-induced distortion	AI.14 Virtualization Overhead Stability Analysis	Degraded controllability and observability, noisy-neighbor variance, non-deterministic response to steering decisions
Migration-induced runtime reconfiguration risk	Migration-induced reconfiguration	AI.20 Structural Cloud Migration Risk Assessment	Post-migration latency-shift, altered contention profile, environment-dependent instability without code-level failure

7.1. AI.07: Accelerator Runtime Control

AI.07 addresses the problem of structure compatible control across heterogeneous accelerator fleets, including environments that combine different GPU generations, TPUs, NPUs, or custom accelerator classes. The core analytical premise of this application domain is that heterogeneous hardware does not merely differ in performance magnitude. It differs in execution timing, memory hierarchy, communication behavior, and runtime assumptions, such that control systems formed under homogeneous fleet logic can become incoherent when extended across mixed hardware environments. This makes AI.07 the primary diagnostic counterpart to the instability modes developed in Section 4.1, and especially to the taxonomy dimensions of latency asymmetry drift and memory path incoherence.

Within the present paper, the relevance of AI.07 lies in its focus on the relation between workload structure and hardware composition. The application domain is concerned with structural compatibility analysis between accelerator types for specific workload profiles, runtime incoherence detection across heterogeneous execution paths, memory hierarchy mismatch diagnostics, and communication protocol assessment for inter-accelerator data movement. These diagnostic orientations correspond directly to the mechanisms identified above, namely synchronization mismatch, bandwidth asymmetry, abstraction-layer overhead, and hardware-path-dependent behavioral variance. In this sense, AI.07 provides the closest application-level extension of the paper's claim that heterogeneous accelerators act as active runtime determinants rather than as interchangeable execution substrates.

The mapping should nevertheless be interpreted with precision. AI.07 does not replace the broader framework developed in this paper, because accelerator-level compatibility analysis alone cannot explain network-, virtualization-, or migration-mediated instability. Its significance is more specific. It supplies a diagnostic lens through which the accelerator-originated portion of heterogeneous inference incoherence can be isolated and structurally interpreted. This is especially important in mixed fleet environments, where the local correctness of each accelerator class can obscure the fact that the serving path as a whole has become incompatible at the level of runtime control.

7.2. AI.11: Structural Network Scalability Risk Modeling

AI.11 addresses network scaling risk as an emergent property of interactions among topology design, routing logic, SDN control behavior, and fault-tolerance structure. Its core premise is that network scaling risk is not reducible to component-level performance or isolated link behavior. It emerges from the way topological form and control-plane logic interact under target scale conditions. This application domain therefore aligns directly with the present paper's analysis of network and placement coupling effects and with the taxonomy mode of interconnect-induced capacity inaccessibility.

The diagnostic value of AI.11 for heterogeneous inference lies in its capacity to model the divergence between nominal and effective capacity at the networked system level. The application domain emphasizes multi-dimensional scaling risk assessment across topology, routing, SDN behavior, and failure tolerance, together with interaction-effect modeling that anticipates how apparently acceptable local design choices can generate emergent risk under enlarged system conditions. This is directly relevant to the argument developed in Sections 4.2 and 5.3, where physically present resources become only conditionally usable once placement decisions, communication requirements, and topology constraints are jointly considered.

In the context of heterogeneous inference, AI.11 is particularly important because network structure does not merely transport data between otherwise independent serving components. It actively shapes whether disaggregated execution remains viable. Prefill-decode separation, accelerator-specific affinity, and KV-cache transfer all depend on a communication substrate whose effective behavior may differ substantially from its nominal design intent once the serving graph becomes heterogeneous and scale sensitive. AI.11 therefore provides the principal diagnostic entry point for analyzing when network architecture ceases to be a neutral substrate and becomes a first-order determinant of whether cluster capacity can be realized as coherent inference behavior.

7.3. AI.14: Virtualization Overhead Stability Analysis

AI.14 addresses virtualization-induced instability by treating abstraction layers not as a fixed source of performance overhead, but as a structural transformation of the relation between workloads and physical resources. Its analytical focus spans GPU partitioning, network virtualization, RDMA passthrough regimes, memory isolation, and hypervisor-level mediation under multi-tenant conditions. This makes it the primary diagnostic counterpart to the virtualization-induced distortion analyzed in Section 4.3 and to the taxonomy mode of virtualization-induced control distortion.

The relevance of AI.14 to the present paper lies in its decomposition of virtualization effects into deterministic and stochastic components. The application domain is explicitly concerned with noisy-neighbor interference, cross-tenant coupling paths, SR-IOV and RDMA stability under shared conditions, and the feasibility of performance guarantees under specific virtualization configurations. These concerns match the paper's central claim that virtualization does not merely reduce performance by a constant amount, but changes the variance structure through which serving behavior is realized. When virtualization alters both controllability and observability, the effect is not simply overhead. It is a distortion in the mapping between intent, execution, and measured performance.

This mapping is analytically important because virtualized heterogeneous inference systems are often treated as though the underlying hardware path remained sufficiently transparent for local metrics to preserve their ordinary meaning. AI.14 challenges that assumption by focusing attention on the structural coupling introduced by abstraction layers themselves. In doing so, it provides a diagnostic mechanism for distinguishing bare-metal inefficiency from virtualization-mediated incoherence. The distinction is essential in cloud and multi-tenant AI environments, where the economic rationale of shared infrastructure depends precisely on whether performance guarantees remain structurally credible under abstraction and contention.

7.4. AI.20: Structural Cloud Migration Risk Assessment

AI.20 addresses cloud migration as a structural rather than purely functional problem. Its core premise is that workloads can remain logically portable across environments while becoming operationally unstable because the coupling relations among compute, network, storage, and control planes change during migration. This makes AI.20 the direct diagnostic counterpart to the migration-induced reconfiguration risk analyzed in Section 4.4 and to the taxonomy mode of migration-induced runtime reconfiguration.

Within the present paper, AI.20 is relevant because it formalizes a point that is often underemphasized in infrastructure discourse. Migration does not merely move an application from one location to another. It rewrites the runtime geometry within which that application is served. The application domain therefore emphasizes structural coupling comparison between source and target environments, migration risk quantification based on workload sensitivity to coupling change, phased migration sequencing, and post-migration stability validation. These diagnostic functions correspond directly to the paper's claim that cloud or region changes alter not only average performance, but also latency distributions, contention structures, and control-loop behavior, even when functional interfaces remain intact.

AI.20 is especially important for heterogeneous inference because the migration target may differ from the source environment along multiple dimensions simultaneously: accelerator availability, storage locality, virtualization regime, network fabric behavior, and provider-level orchestration logic. Under such conditions, migration risk cannot be reduced to compatibility testing or simple benchmarking. It must be treated as a structural assessment of whether the target environment preserves the coherence conditions on which the source serving behavior depended. AI.20 provides precisely this diagnostic orientation and therefore anchors the migration dimension of the present paper in a more explicit analytical domain.

7.5. Supporting Diagnostics

The four primary mappings above define the principal correspondence between the taxonomy introduced in this paper and the integrated application domains from which the argument is constructed. Three additional diagnostic domains play supporting roles by sharpening specific layers of the coherence problem without replacing the primary mapping.

First, AI.04, Runtime Control Coherence, addresses logical coupling across scheduling, orchestration, and policy layers. Its supporting value lies in clarifying the cross-loop interference mechanisms discussed in Section 6.2. Where AI.07 focuses more directly on accelerator-runtime compatibility, AI.04 provides a broader control-theoretic lens on how independently operating coordination layers can remain locally rational yet globally inconsistent [2]. It is therefore especially useful in interpreting runtime coherence as a system property rather than as a hardware-local effect.

Second, AI.01, Interconnect Stability, strengthens the physical and topological layer of the present argument. Its focus on interconnect-level coupling effects supports the analysis of network-placement interaction and effective capacity divergence developed in Sections 4.2 and 5.3 [1]. Relative to AI.11, which emphasizes scaling risk across topology, routing, and control, AI.01 offers a more direct framing of how physical interconnect structure conditions the usability of provisioned compute resources.

Third, AI.27, Inference Pipeline Control Coherence, serves as an end-to-end integrating lens. Its relevance lies in connecting accelerator-level, scheduler-level, and serving-level coordination into a single inference pipeline perspective. This makes it particularly useful as a bridging diagnostic domain when the source of incoherence cannot be localized cleanly to one layer and instead emerges through the full serving chain from request routing to execution completion. In that sense, AI.27 does not introduce an additional primary instability class, but complements the present mapping by highlighting that heterogeneous inference instability is often pipeline-distributed even when its strongest signature appears in one local domain.

Taken together, these mappings show that the taxonomy proposed in this paper is not an isolated conceptual artifact. It can be related systematically to a set of diagnostic domains that address distinct portions of the heterogeneous inference problem space. This does not imply resolution by framework declaration. It implies only that the identified instability modes admit structured analytical treatment and that heterogeneous inference incoherence can therefore be examined with greater precision than is possible through layer-local troubleshooting alone.

8. Discussion

The analysis developed in this paper has treated heterogeneous inference instability as a structural problem of runtime coherence rather than as a collection of isolated bottlenecks. This perspective has several implications. First, it suggests that the dominant performance question in large-scale inference is shifting from the efficiency of individual components toward the coordination properties of the composite execution fabric. Second, it implies that heterogeneity should not be interpreted primarily as a temporary deviation from ideal infrastructure uniformity. It is increasingly a persistent condition of deployment, produced by mixed accelerator availability, architectural specialization, virtualization-mediated access to resources, and migration across infrastructure environments. Third, it indicates that the practical significance of heterogeneity cannot be assessed by layer-local metrics alone. The decisive effects arise from how accelerator behavior, memory path structure, network topology, abstraction boundaries, and placement logic interact under scale.

These implications matter because heterogeneous inference is becoming central to the operational form of modern AI systems. The growth of disaggregated serving, multi-pool execution, infrastructure specialization, and cross-environment deployment means that instability can no longer be understood only through local resource constraints. In many cases, the system remains technically functional while drifting into reduced effective capacity, widened tail-latency distributions, and a growing divergence between provisioned resources and usable service output. The discussion below therefore considers

the broader significance of this structural perspective, identifies open research questions that follow from it, and outlines how the paper's taxonomy may be subjected to empirical evaluation.

8.1. Implications for AI Factories and Heterogeneous Inference Platforms

One implication of the present analysis is that the emergence of AI factory architectures amplifies rather than suppresses the coherence problem identified in this paper. As inference becomes a first-class infrastructure workload alongside training, operators are increasingly required to manage mixed accelerator fleets, differentiated serving tiers, and multiple execution pathways within the same datacenter or federated deployment environment [25,52]. This expansion does not simply increase system scale. It multiplies the number of cross-layer dependencies through which runtime incoherence can emerge. AI factories therefore intensify the need for structural analysis because local hardware efficiency is no longer a sufficient proxy for end-to-end serving stability.

Disaggregated inference architecture illustrates this point especially clearly. The separation of prefill and decode across different resource pools is often introduced as a response to the distinct compute and memory demands of large language model inference. This separation can improve specialization and utilization, but it also introduces a persistent dependency on KV-cache transfer, placement locality, and stage-to-stage communication coherence [8,9]. The result is that heterogeneity is both mitigated and reproduced by architectural design. Disaggregation addresses one form of mismatch while simultaneously creating new topology-sensitive interaction surfaces. For this reason, the significance of disaggregated inference cannot be judged only by local gains in stage efficiency. It must also be evaluated in terms of how the resulting serving path preserves or degrades runtime coherence across the composite system.

The same logic extends beyond single-cluster environments to multi-cloud and sovereign-cloud deployment conditions. When workloads are placed across provider boundaries, the heterogeneity problem is compounded by differences in virtualization regimes, accelerator availability profiles, storage path assumptions, control-plane behavior, and network performance distributions [50,51]. Under such conditions, cross-provider placement is not simply a scheduling extension of within-cluster resource allocation. It is a structural coordination problem in which the operational meaning of a placement decision depends on whether the target environment preserves the coupling conditions required for stable inference. The analysis developed here is therefore applicable not only to internal datacenter heterogeneity, but also to federated or policy-constrained infrastructure landscapes in which multiple infrastructure classes must be coordinated without assuming uniform execution behavior.

This perspective also clarifies the significance of infrastructure classes such as CUDA-based cloud environments, TPUs, inference-oriented accelerator families, and custom ASIC deployments. These should not be viewed only as alternative hardware options with different price-performance points. They instantiate distinct coupling topologies with different runtime assumptions, memory-path characteristics, communication behaviors, and abstraction constraints. Movement between them, or composition across them, constitutes a structural reconfiguration of the execution fabric rather than a neutral deployment variation. This is precisely why heterogeneous inference platforms require a systems language of coherence rather than a hardware language of substitution. The relevant question is not merely which resource is faster or cheaper in isolation. It is whether the composite infrastructure remains coordinated enough for nominal hardware diversity to be transformed into stable and economically meaningful serving behavior.

8.2. Open Research Questions

The first major open question concerns cross-layer observability. Existing monitoring systems remain largely organized around separable infrastructure domains, such as compute, networking, storage, orchestration, or application-level service behavior. Heterogeneous inference instability, however, often emerges at the interaction boundaries between these domains rather than within any one of them. This raises the question of what measurement primitives would be required for genuinely structural observability in large-scale inference systems [54,55]. A relevant research agenda would

need to determine which intermediate states, path-level dependencies, and timing relations must be exposed in order to render cross-layer incoherence visible without collapsing the system into an intractable volume of telemetry.

A second open question concerns runtime topology metrics. The analysis in this paper has relied on the distinction between nominal and effective capacity, but this distinction remains underformalized in operational terms. If effective capacity depends on accelerator composition, placement locality, communication structure, and memory-path constraints, then infrastructure analysis requires a more explicit way of representing the topology of usable execution under live serving conditions [1,37]. This suggests the need for runtime metrics that do not merely summarize device activity or fleet throughput, but characterize the degree to which provisioned resources remain structurally reachable for coherent work. Whether such metrics can be computed in real time, and at what level of abstraction, remains an important unanswered question.

A third research question concerns migration-sensitive diagnostics. Migration is often evaluated in terms of functional portability, compatibility testing, or comparative benchmark behavior. The present paper has argued that these criteria are insufficient because migration alters the control geometry within which serving occurs. This raises the question of how migration risk can be quantified structurally before migration is executed, and which source-target relations are most predictive of post-migration instability [47,48]. Answering this would require a framework capable of comparing environments in terms of coupling topology rather than only in terms of individual resource specifications. Such a framework could become important not only for cloud portability, but also for sovereign deployment planning and multi-environment resilience strategies.

A fourth open question concerns coherence-aware benchmarking. Most current benchmark practice remains strongly tied to single-device or homogeneous-cluster evaluation logic. Yet if infrastructure path materially shapes runtime behavior, then benchmark design must eventually account for heterogeneity as part of the measured system rather than as external noise [63,64]. This raises several questions. Should benchmark suites explicitly vary infrastructure path composition. Should they report sensitivity to accelerator mix, placement asymmetry, or virtualization boundary effects. Should infrastructure-path variance be treated as an evaluation dimension analogous to latency or throughput. These questions are important because benchmark adequacy increasingly depends on whether the benchmark reflects the conditions under which deployed inference actually operates.

A fifth research question concerns heterogeneity-aware scheduling theory. Classical scheduler design typically assumes that resources are sufficiently comparable for allocation decisions to be expressed over a simplified equivalence class. Heterogeneous inference weakens that assumption by making the serving outcome path dependent on hardware type, communication structure, memory residency, and abstraction boundary effects [10,12,13]. This suggests that scheduling theory may require a richer representational basis in which the target of scheduling is not a pool of interchangeable units, but a structured composition whose performance properties depend on cross-layer coordination. The problem is therefore not only to improve placement under heterogeneity, but to identify the abstractions within which heterogeneity remains schedulable without generating incoherence faster than optimization can absorb it.

8.3. Outlook on Empirical Validation

The framework proposed in this paper is deliberately pre-empirical. It identifies instability mechanisms and organizes them into a structural taxonomy based on architectural reasoning, publicly described system forms, and inferential synthesis across the literature. It does not claim to present deployment-level measurements or to validate the taxonomy through direct experiments. This limitation is intentional. The paper's purpose is to establish a conceptual and diagnostic structure that can guide later empirical work rather than to infer general conclusions from narrow measurement settings.

Empirical validation would require controlled or semi-controlled studies capable of varying heterogeneous composition along multiple dimensions. At minimum, this would involve systematic variation of accelerator mix, placement topology, virtualization configuration, and migration path

while measuring quantities such as effective capacity, tail-latency distributions, queueing behavior, and cost-performance ratios. Such studies are difficult because they require access to infrastructure that is not only heterogeneous, but heterogeneous at operationally meaningful scale. They also require experimental designs that can separate local hardware effects from interaction effects emerging at the level of the serving path. The resource intensity of this type of work partly explains why many heterogeneous instability phenomena remain analytically underdeveloped despite their operational relevance.

A plausible validation pathway would begin with simulation-based studies or trace-driven modeling. Published workload traces, infrastructure scheduling traces, and validated serving simulators could be used to examine how the instability modes identified here behave under controlled perturbation of topology, heterogeneity, and abstraction conditions [13,30,63]. This would not substitute for real deployment measurement, but it could test whether the proposed taxonomy predicts distinct and reproducible signatures under varying infrastructure assumptions. Such a program could then be extended through targeted cloud-based measurements on heterogeneous instance classes, particularly in environments where mixed accelerator fleets, disaggregated serving, or multi-tenant virtualization are already available.

The taxonomy proposed here is designed to be empirically falsifiable in a limited but meaningful sense. Each instability mode implies observable patterns that should differ from those of simpler homogeneous bottleneck models. Latency asymmetry drift should widen timing distributions as heterogeneity increases. Memory-path incoherence should produce cases in which reported compute availability diverges from actual serving capacity. Interconnect-induced capacity inaccessibility should manifest as non-proportional throughput scaling under added resources. Virtualization-induced control distortion should alter both the variance and the controllability of observed performance. Migration-induced runtime reconfiguration should change performance distributions without requiring code-level failure. These are testable expectations even if their precise quantitative form remains unknown.

The broader significance of empirical validation is therefore not merely confirmatory. It would help determine whether heterogeneous inference instability is best understood as a set of independent operational complications or as a coherent systems phenomenon centered on runtime coordination. The present paper has argued for the second interpretation. That argument now points toward a concrete research program. If future work can show that the proposed instability modes produce measurable and separable signatures under controlled variation, then runtime coherence may become a more explicit object of AI infrastructure science rather than an implicit residual effect inferred only after local metrics fail to explain observed behavior.

9. Conclusion

The analysis developed in this paper has treated heterogeneous inference infrastructure as a structural systems condition rather than as an incidental deployment complication. The core result is that heterogeneous inference changes the meaning of performance management. Once execution depends on mixed accelerator classes, differentiated memory paths, nonuniform interconnects, virtualization boundaries, and migration-sensitive placement, performance can no longer be understood primarily through local optimization of isolated layers. It becomes a problem of maintaining runtime coherence across a coupled execution fabric. This shift is significant because many of the most consequential degradations in large-scale inference do not appear first as device failure, scheduler malfunction, or explicit network fault. They appear as loss of coordination between locally correct subsystems.

9.1. Heterogeneous Inference as a Structural Coordination Problem

The central argument of this paper is that heterogeneous inference infrastructure transforms performance management from an optimization problem into a structural coordination problem. Additional accelerators, wider memory envelopes, or greater nominal network capacity do not in themselves resolve instability when the dominant source of degradation lies in cross-layer incoherence

rather than in isolated scarcity [57,58]. Under heterogeneous conditions, the operational question is not only whether resources exist, but whether they remain mutually compatible enough to support coherent execution under load.

Within this framing, the five instability modes identified in this paper, latency-asymmetry drift, memory-path incoherence, interconnect-induced capacity inaccessibility, virtualization-induced control distortion, and migration-induced runtime reconfiguration risk, should be understood as structurally distinct mechanisms through which coordination breaks down. They are not interchangeable symptoms of a single bottleneck. Each describes a different way in which nominally functional infrastructure can become operationally unstable when composed into a heterogeneous inference fabric. The diagnostic implication is that no single layer-local explanation is sufficient for all of them. Distinct instability mechanisms require correspondingly distinct analytical entry points, even when they interact in practice.

The broader consequence is that heterogeneous inference cannot be reduced to a hardware procurement or deployment diversity issue. It is a systems problem whose relevant object is the execution relation between layers. This is why the present paper has emphasized taxonomy and diagnostic mapping rather than implementation prescription. Before optimization can be meaningfully applied, the structural form of instability must be identified with sufficient precision that local interventions do not further destabilize the coupled runtime.

9.2. Why Coherence Precedes Optimization

A second conclusion follows directly from this analysis. Runtime coherence is not the product of successful optimization. It is the condition under which optimization becomes systemically meaningful. If the execution fabric is already incoherent, then improvements applied to one layer may leave the overall system unchanged or may even intensify instability by sharpening asymmetries elsewhere in the serving path [2,3]. Under such conditions, local metric improvement cannot be treated as evidence of global progress.

This ordering matters because large-scale AI infrastructure is frequently evaluated through utilization, average latency, or aggregate throughput targets that presuppose a sufficiently coherent runtime surface. The present paper has argued that this presupposition is no longer reliable in heterogeneous inference systems. When control loops, memory behavior, topology, virtualization, and migration-sensitive placement interact without stable coordination, optimization loses predictability at system level. Coherence must therefore be established conceptually and diagnostically before optimization can yield reproducible infrastructure-wide gains.

The practical implication is straightforward. Under heterogeneous deployment conditions, investment in structural observability, cross-layer diagnosis, and topology-aware analysis may produce greater marginal benefit than isolated investment in additional compute capacity. This is not because raw capacity has become unimportant, but because the economic meaning of capacity increasingly depends on whether provisioned resources remain effectively usable within a coherent runtime. The sequence is therefore analytically and operationally important: coherence first, optimization second.

9.3. Implications for Future AI Infrastructure Research

The structural challenge identified here is unlikely to diminish. As AI inference continues to scale, and as deployment environments become more diverse across accelerators, providers, regions, and policy regimes, heterogeneous execution will become more common rather than less common [25,52]. This means that the interaction effects analyzed in this paper are likely to move closer to the center of AI infrastructure performance, cost, and reliability research. In such a setting, the critical variables of system behavior will increasingly lie in the coordination properties of execution fabrics rather than in the isolated efficiency of their constituent parts.

Future work should therefore proceed along at least three directions. First, formal methods are needed for reasoning about runtime coherence in heterogeneous systems without collapsing the problem into purely local metrics or overly abstract systems language. Second, empirical methods are

needed for measuring effective capacity, tail-latency sensitivity, and infrastructure-path variance under structural heterogeneity. Third, architectural principles are needed that treat cross-layer coherence as an explicit design target rather than as an emergent byproduct of separately optimized subsystems. These directions are complementary. Together they define a research program in which infrastructure behavior is analyzed at the level where heterogeneity actually becomes operationally decisive.

The structural taxonomy and diagnostic mapping presented in this paper are offered as a foundation for that program, not as its completion. Their intended contribution is to provide a clearer language for identifying where heterogeneous inference instability arises, how it differs across infrastructural conditions, and why conventional optimization logic becomes insufficient once cross-layer incoherence dominates observed system behavior. If heterogeneous inference is to become a stable basis for future AI deployment at scale, runtime coherence must become a first-order object of infrastructure research.

Data Availability Statement: No new datasets were generated or analyzed in this study. The paper is based on structural analysis of publicly described system classes and published literature. Supporting framework materials relevant to the broader SORT research program are archived on Zenodo under DOI 10.5281/zenodo.18094128. Additional validation environment materials are archived at 10.5281/zenodo.18050207. Project materials are maintained at GitHub.

Acknowledgments: The author acknowledges the earlier development of the SORT research architecture, which provided the broader conceptual context within which the present analysis was conducted.

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

Use of Artificial Intelligence: Artificial intelligence tools were used for limited editorial support, including language refinement and LaTeX formatting assistance. The scientific framing, conceptual arguments, structural taxonomy, interpretation of the literature, and all substantive research judgments were developed and verified by the author, who takes full responsibility for the content of the manuscript.

1. Wegener, G. H. (2026). SORT-AI: Interconnect Stability and Cost per Performance in Large-Scale AI Infrastructure—A Structural Analysis of Runtime Instability in Distributed Systems. *Preprints* **2026010161**. DOI:10.20944/preprints202601.0161.v1
2. Wegener, G. H. (2026). SORT-AI: Runtime Control Coherence in Large-Scale AI Systems—Structural Causes of Cost, Instability, and Non-Determinism Beyond Interconnect Failures. *Preprints* **2026010298**. DOI:10.20944/preprints202601.0298.v1
3. Wegener, G. H. (2026). SORT-AI: Structural Efficiency Recovery in Hyperscale AI Systems—A Diagnostic Framework for Throughput, Control, and Orchestration Losses. *Preprints* **2026020015**. DOI:10.20944/preprints202602.0015.v1
4. Wegener, G. H. (2025). The Supra-Omega Resonance Theory (SORT): A Closed Structural Architecture for Cross-Domain Scientific Analysis. *Preprints* **2024111783**. DOI:10.20944/preprints202511.1783.v3
5. Wegener, G. H. (2025). SORT Whitepaper v6: Supra-Omega Resonance Theory—A Projection-Based Structural Framework. *Zenodo*. DOI:10.5281/zenodo.18094128
6. Kwon, W.; Li, Z.; Zhuang, S.; Sheng, Y.; Zheng, L.; Yu, C. H.; Gonzalez, J. E.; Zhang, H.; Stoica, I. (2023). Efficient Memory Management for Large Language Model Serving with PagedAttention. *Proceedings of the 29th Symposium on Operating Systems Principles (SOSP)*, 611–626. DOI:10.1145/3600006.3613165
7. Yu, G.-I.; Jeong, J. S.; Kim, G.-W.; Kim, S.; Chun, B.-G. (2022). Orca: A Distributed Serving System for Transformer-Based Generative Models. *Proceedings of the 16th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 521–538. https://www.usenix.org/conference/osdi22/presentation/yu
8. Zhong, Y.; Liu, S.; Chen, J.; Hu, J.; Zhu, Y.; Liu, X.; Jin, X.; Zhang, H. (2024). DistServe: Disaggregating Prefill and Decoding for Goodput-optimized Large Language Model Serving. *Proceedings of the 18th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. https://www.usenix.org/conference/osdi24/presentation/zhong-yinmin
9. Patel, P.; Choukse, E.; Zhang, C.; Shah, A.; Goiri, Í.; Maleki, S.; Bianchini, R. (2024). Splitwise: Efficient Generative LLM Inference Using Phase Splitting. *Proceedings of the 51st ACM/IEEE International Symposium on Computer Architecture (ISCA)*. DOI:10.1109/ISCA59077.2024.00052

10. Jiang, Y.; Yan, R.; Yao, X.; Zhou, Y.; Chen, B.; Yuan, B. (2024). HexGen: Generative Inference of Large Language Model over Heterogeneous Environment. *Proceedings of the 41st International Conference on Machine Learning (ICML)*. arXiv:2311.11514
11. Jiang, Y.; Fu, F.; Yao, X.; He, G.; Miao, X.; Klimovic, A.; Cui, B.; Yuan, B.; Yoneki, E. (2025). Demystifying Cost-Efficiency in LLM Serving over Heterogeneous GPUs. *arXiv preprint*. arXiv:2502.00722
12. Mei, Y.; Cao, R.; Dong, Y.; Li, M.; Lu, S.; Xu, C.; Yang, F.; Lu, S.; Zhang, Z.; Vinayak, R. K. (2025). Helix: Serving Large Language Models over Heterogeneous GPUs and Networks via Max-Flow. *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. DOI:10.1145/3669940.3707266
13. Li, Z.; Zheng, L.; Zhong, Y.; Liu, V.; Sheng, Y.; Jin, X.; Huang, Y.; Chen, Z.; Zhang, H.; Gonzalez, J. E.; Stoica, I. (2023). AlpaServe: Statistical Multiplexing with Model Parallelism for Deep Learning Serving. *Proceedings of the 17th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 663–679. https://www.usenix.org/conference/osdi23/presentation/li-zhouhan
14. Agrawal, A.; Panwar, A.; Mohan, J.; Kwatra, N.; Gulavani, B. S.; Ramjee, R. (2024). Taming Throughput-Latency Tradeoff in LLM Inference with Sarathi-Serve. *Proceedings of the 18th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. https://www.usenix.org/conference/osdi24/presentation/agrawal
15. Griggs, T.; Liu, X.; Yu, J.; Kim, D.; Chiang, W.-L.; Cheung, A.; Stoica, I. (2024). Mélange: Cost Efficient Large Language Model Serving by Exploiting GPU Heterogeneity. *arXiv preprint*. arXiv:2404.14527
16. Hu, C.; Huang, H.; Xu, L.; Chen, X.; Xu, J.; Chen, S.; Feng, H.; Wang, C.; Wang, S.; Bao, Y. (2025). Inference without Interference: Disaggregate LLM Inference for Mixed Downstream Workloads. *ACM Transactions on Architecture and Code Optimization*. DOI:10.1145/3732941
17. Sheng, Y.; Zheng, L.; Yuan, B.; Li, Z.; Ryabinin, M.; Chen, B.; Liang, P.; Ré, C.; Stoica, I.; Zhang, C. (2023). FlexGen: High-Throughput Generative Inference of Large Language Models with a Single GPU. *Proceedings of the 40th International Conference on Machine Learning (ICML)*. arXiv:2303.06865
18. Pope, R.; Douglas, S.; Chowdhery, A.; Devlin, J.; Bradbury, J.; Heek, J.; Xiao, K.; Agrawal, S.; Dean, J. (2022). Efficiently Scaling Transformer Inference. *Proceedings of Machine Learning and Systems (MLSys)*, **5**. arXiv:2211.05102
19. Jouppi, N. P.; Young, C.; Patil, N.; Patterson, D.; Agrawal, G.; Bajwa, R.; et al. (2017). In-Datacenter Performance Analysis of a Tensor Processing Unit. *Proceedings of the 44th Annual International Symposium on Computer Architecture (ISCA)*, 1–12. DOI:10.1145/3079856.3080246
20. Jouppi, N. P.; Kurian, G.; Li, S.; Ma, P.; Nagarajan, R.; Nai, L.; et al. (2023). TPU v4: An Optically Reconfigurable Supercomputer for Machine Learning with Hardware Support for Embeddings. *Proceedings of the 50th International Symposium on Computer Architecture (ISCA)*, 1–14. DOI:10.1145/3579371.3589350
21. Dao, T.; Fu, D. Y.; Ermon, S.; Rudra, A.; Ré, C. (2022). FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness. *Advances in Neural Information Processing Systems (NeurIPS)*, **35**, 16344–16359. arXiv:2205.14135
22. Dao, T. (2023). FlashAttention-2: Faster Attention with Better Parallelism and Work Partitioning. *Proceedings of the International Conference on Learning Representations (ICLR)*. arXiv:2307.08691
23. Gholami, A.; Yao, Z.; Kim, S.; Mahoney, M. W.; Keutzer, K. (2021). AI and Memory Wall. *RiseLab Technical Report*. https://medium.com/riselab/ai-and-memory-wall-2cb4265cb0b8
24. NVIDIA Corporation (2024). Multi-Instance GPU User Guide. *NVIDIA Documentation*. https://docs.nvidia.com/datacenter/tesla/mig-user-guide/
25. Barroso, L. A.; Hölzle, U.; Ranganathan, P. (2018). *The Datacenter as a Computer: Designing Warehouse-Scale Machines*, 3rd ed. Morgan & Claypool: San Rafael, CA, USA. DOI:10.2200/S00874ED3V01Y201809CAC046
26. Dean, J.; Barroso, L. A. (2013). The Tail at Scale. *Communications of the ACM*, **56**(2), 74–80. DOI:10.1145/2408776.2408794
27. Burns, B.; Grant, B.; Oppenheimer, D.; Brewer, E.; Wilkes, J. (2016). Borg, Omega, and Kubernetes. *ACM Queue*, **14**(1), 70–93. DOI:10.1145/2898442.2898444
28. Schwarzkopf, M.; Konwinski, A.; Abd-El-Malek, M.; Wilkes, J. (2013). Omega: Flexible, Scalable Schedulers for Large Compute Clusters. *Proceedings of the 8th ACM European Conference on Computer Systems (EuroSys)*, 351–364. DOI:10.1145/2465351.2465386
29. Verma, A.; Pedrosa, L.; Korupolu, M.; Oppenheimer, D.; Tune, E.; Wilkes, J. (2015). Large-Scale Cluster Management at Google with Borg. *Proceedings of the 10th ACM European Conference on Computer Systems (EuroSys)*, Article 18. DOI:10.1145/2741948.2741964

30. Tirmazi, M.; Barker, A.; Deng, N.; Haque, M. E.; Qin, Z. G.; Hand, S.; Harchol-Balter, M.; Wilkes, J. (2020). Borg: The Next Generation. *Proceedings of the 15th ACM European Conference on Computer Systems (EuroSys)*, Article 30. DOI:10.1145/3342195.3387517
31. Ananthanarayanan, G.; Kandula, S.; Greenberg, A.; Stoica, I.; Lu, Y.; Saha, B.; Harris, E. (2010). Reining in the Outliers in Map-Reduce Clusters Using Mantri. *Proceedings of the 9th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 265–278. https://www.usenix.org/conference/osdi10/reining-outliers-map-reduce-clusters-using-mantri
32. Rajbhandari, S.; Rasley, J.; Ruwase, O.; He, Y. (2020). ZeRO: Memory Optimizations Toward Training Trillion Parameter Models. *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC20)*. DOI:10.1109/SC41405.2020.00024
33. Narayanan, D.; Shoeybi, M.; Casper, J.; LeGresley, P.; Patwary, M.; Korthikanti, V.; Vainbrand, D.; Groeneveld, P.; Puri, N.; Hooper, S.; Alon, I.; Catanzaro, B. (2021). Efficient Large-Scale Language Model Training on GPU Clusters Using Megatron-LM. *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC21)*. DOI:10.1145/3458817.3476209
34. Al-Fares, M.; Loukissas, A.; Vahdat, A. (2008). A Scalable, Commodity Data Center Network Architecture. *ACM SIGCOMM Computer Communication Review*, **38**(4), 63–74. DOI:10.1145/1402958.1402967
35. Singh, A.; Ong, J.; Agarwal, A.; Anderson, G.; Armistead, A.; Bannon, R.; et al. (2015). Jupiter Rising: A Decade of Clos Topologies and Centralized Control in Google's Datacenter Network. *ACM SIGCOMM Computer Communication Review*, **45**(4), 183–197. DOI:10.1145/2829988.2787508
36. Greenberg, A.; Hamilton, J. R.; Jain, N.; Kandula, S.; Kim, C.; Lahiri, P.; Maltz, D. A.; Patel, P.; Sengupta, S. (2009). VL2: A Scalable and Flexible Data Center Network. *ACM SIGCOMM Computer Communication Review*, **39**(4), 51–62. DOI:10.1145/1592568.1592576
37. Guo, C.; Yuan, L.; Xiang, D.; Dang, Y.; Huang, R.; Maltz, D.; Liu, Z.; Wang, V.; Pang, B.; Chen, H.; Lin, Z.-W.; Kuber, V. (2015). Pingmesh: A Large-Scale System for Data Center Network Latency Measurement and Analysis. *ACM SIGCOMM Computer Communication Review*, **45**(4), 139–152. DOI:10.1145/2829988.2787496
38. Liu, Z.; Aaraj, N.; Awadallah, A.; Volos, H. (2019). Understanding the Design of Communication Libraries for Distributed Deep Learning. *arXiv preprint*. arXiv:1901.03855
39. Li, J.; Sharma, N. K.; Ports, D. R. K.; Gribble, S. D. (2014). Tales of the Tail: Hardware, OS, and Application-Level Sources of Tail Latency. *Proceedings of the ACM Symposium on Cloud Computing (SoCC)*, Article 9. DOI:10.1145/2670979.2670988
40. Harchol-Balter, M. (2013). *Performance Modeling and Design of Computer Systems: Queueing Theory in Action*. Cambridge University Press: Cambridge, UK. DOI:10.1017/CBO9781139226424
41. Shoeybi, M.; Patwary, M.; Puri, R.; LeGresley, P.; Casper, J.; Catanzaro, B. (2019). Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism. *arXiv preprint*. arXiv:1909.08053
42. Barham, P.; Dragovic, B.; Fraser, K.; Hand, S.; Harris, T.; Ho, A.; Neugebauer, R.; Pratt, I.; Warfield, A. (2003). Xen and the Art of Virtualization. *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP)*, 164–177. DOI:10.1145/945445.945462
43. Zhang, J.; Zheng, X.; Shi, L. (2018). A Survey on GPU Virtualization. *Journal of Computer Science and Technology*, **33**(2), 352–371. DOI:10.1007/s11390-018-1818-x
44. Amaral, M.; Polo, J.; Carrera, D.; Mohomed, I.; Unuvar, M.; Steinder, M. (2017). Performance Isolation Analysis in Multi-Tenant Cloud Environments. *IEEE Transactions on Cloud Computing*, **5**(3), 396–409. DOI:10.1109/TCC.2015.2424886
45. Luo, Z.; Xing, J.; Yin, J.; Li, B. (2022). Orion: Interference-aware, Fine-grained GPU Sharing for ML Applications. *Proceedings of the 17th ACM European Conference on Computer Systems (EuroSys)*, 370–386. DOI:10.1145/3492321.3519544
46. Li, B.; Samsi, S.; Gadepally, V.; Tiwari, D. (2022). MISO: Exploiting Multi-Instance GPU Capability on Multi-Tenant GPU Clusters. *Proceedings of the 13th ACM Symposium on Cloud Computing (SoCC)*. DOI:10.1145/3542929.3563510
47. Jamshidi, P.; Ahmad, A.; Pahl, C. (2013). Cloud Migration Research: A Systematic Review. *IEEE Transactions on Cloud Computing*, **1**(2), 142–157. DOI:10.1109/TCC.2013.10
48. Schad, J.; Dittrich, J.; Quiané-Ruiz, J.-A. (2010). Runtime Measurements in the Cloud: Observing, Analyzing, and Reducing Variance. *Proceedings of the VLDB Endowment*, **3**(1–2), 460–471. DOI:10.14778/1920841.1920902
49. Iosup, A.; Yigitbasi, N.; Epema, D. (2011). On the Performance Variability of Production Cloud Services. *Proceedings of the 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, 104–113. DOI:10.1109/CCGrid.2011.22

50. Leitner, P.; Cito, J. (2016). Patterns in the Chaos—A Study of Performance Variation and Predictability in Public IaaS Clouds. *ACM Transactions on Internet Technology*, **16**(3), Article 15. DOI:10.1145/2885497
51. Chadha, M.; Ilsche, T.; Bielert, M.; Schöne, R. (2023). How Does the Task Environment Affect Performance Comparisons of Cloud Instances? *Proceedings of the 16th IEEE/ACM International Conference on Utility and Cloud Computing (UCC)*. DOI:10.1145/3603166.3632135
52. Patterson, D.; Gonzalez, J.; Le, Q.; Liang, C.; Munguía, L.-M.; Rothchild, D.; So, D.; Texier, M.; Dean, J. (2022). Carbon Emissions and Large Neural Network Training. *arXiv preprint*. arXiv:2104.10350
53. Barham, P.; Donnelly, A.; Isaacs, R.; Mortier, R. (2004). Using Magpie for Request Extraction and Workload Modelling. *Proceedings of the 6th Symposium on Operating Systems Design and Implementation (OSDI)*, 259–272. https://www.usenix.org/conference/osdi-04/using-magpie-request-extraction-and-workload-modelling
54. Sambasivan, R. R.; Zheng, A. X.; De Rosa, M.; Krevat, E.; Whitman, S.; Stroucken, M.; Wang, W.; Xu, L.; Ganger, G. R. (2011). Diagnosing Performance Changes by Comparing Request Flows. *Proceedings of the 8th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 43–56. https://www.usenix.org/conference/nsdi11/diagnosing-performance-changes-comparing-request-flows
55. Sigelman, B. H.; Barroso, L. A.; Burrows, M.; Decandia, P.; Dean, J.; Erlingsson, Ú.; Ghemawat, S.; Hsieh, W.; Jaspan, C.; Kuropatkin, N.; Lucovsky, M.; Macy, J.; Romer, T.; et al. (2010). Dapper, a Large-Scale Distributed Systems Tracing Infrastructure. *Google Technical Report*. https://research.google/pubs/dapper-a-large-scale-distributed-systems-tracing-infrastructure/
56. Oppenheimer, D.; Ganapathi, A.; Patterson, D. A. (2003). Why Do Internet Services Fail, and What Can Be Done About It? *Proceedings of the 4th USENIX Symposium on Internet Technologies and Systems (USITS)*. https://www.usenix.org/conference/usits-03/why-do-internet-services-fail-and-what-can-be-done-about-it
57. Bar-Yam, Y. (1997). *Dynamics of Complex Systems*. Addison-Wesley: Reading, MA, USA. ISBN: 978-0-201-55748-1.
58. Mitchell, M. (2009). *Complexity: A Guided Tour*. Oxford University Press: Oxford, UK. ISBN: 978-0-19-512441-5.
59. Holland, J. H. (1995). *Hidden Order: How Adaptation Builds Complexity*. Addison-Wesley: Reading, MA, USA. ISBN: 978-0-201-44230-5.
60. Scheffer, M.; Bascompte, J.; Brock, W. A.; Brovkin, V.; Carpenter, S. R.; Dakos, V.; Held, H.; van Nes, E. H.; Rietkerk, M.; Sugihara, G. (2009). Early-Warning Signals for Critical Transitions. *Nature*, **461**(7260), 53–59. DOI:10.1038/nature08227
61. Newman, M. (2010). *Networks: An Introduction*. Oxford University Press: Oxford, UK. DOI:10.1093/acprof:oso/9780199206650.001.0001
62. Saltzer, J. H.; Reed, D. P.; Clark, D. D. (1984). End-to-End Arguments in System Design. *ACM Transactions on Computer Systems*, **2**(4), 277–288. DOI:10.1145/357401.357402
63. Jeon, M.; Venkataraman, S.; Phanishayee, A.; Qian, J.; Xiao, W.; Yang, F. (2019). Analysis of Large-Scale Multi-Tenant GPU Clusters for DNN Training Workloads. *Proceedings of the USENIX Annual Technical Conference (ATC)*, 947–960. https://www.usenix.org/conference/atc19/presentation/jeon
64. Dubey, A.; Jauhri, A.; Pandey, A.; et al. (2024). The Llama 3 Herd of Models. *arXiv preprint*. arXiv:2407.21783
65. Meta Engineering (2024). Taming Tail Utilization of Ads Inference at Meta Scale. *Meta Engineering Blog*. https://engineering.fb.com/2024/07/10/production-engineering/tail-utilization-ads-inference-meta/
66. Huang, Y.; Cheng, Y.; Bapna, A.; Firat, O.; Chen, D.; Chen, M.; Lee, H.; Ngiam, J.; Le, Q. V.; Wu, Y.; Chen, Z. (2019). GPipe: Efficient Training of Giant Neural Networks Using Pipeline Parallelism. *Advances in Neural Information Processing Systems (NeurIPS)*, **32**. arXiv:1811.06965