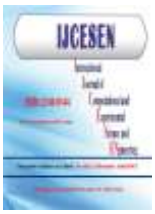
International Journal of Computational and Experimental Science and ENgineering (IJCESEN)

Vol. 12-No.1 (2026) pp. 1428-1438
http://www.ijcesen.com



ISSN: 2149-9144

Research Article

AI Runtime Infrastructure: Establishing a Foundational Layer for Distributed AI Systems

Ashutosh Shanker*

Palo Alto Networks Inc., USA
* **Corresponding Author Email:** ashutoshs.connect@gmail.com - **ORCID:** 0000-0002-5007-2990

Article Info:

DOI: 10.22399/ijcesen.5049
Received : 11 January 2026
Revised : 01 March 2026
Accepted : 10 March 2026

Keywords

AI Runtime Infrastructure,
Distributed Systems Architecture,
Model Orchestration,
Heterogeneous Accelerator Management,
Cloud-Native AI Execution

Abstract:

Architecturally, the AI Runtime Infrastructure, or AIRI, is a foundational layer of distributed architecture designed to enable the execution of large-scale AI workloads. Most modern distributed architectures, heavily influenced by cloud-native design principles, are designed for stateless, deterministic, synchronous, and microservices-based workloads. As such, they are not designed to manage efficiently the stateful, probabilistic, and adaptive workloads that AI execution entails. AIRI is proposed as a runtime layer and reference architecture providing application-agnostic support across compute, storage, and networking infrastructure. It supports core runtime responsibilities such as model lifecycle management, orchestration of heterogeneous accelerators, cross-model coordination, and inference-time policy enforcement. In addition, the architecture includes control-plane capabilities such as model-aware routing, which aid efficiency and governance, as well as data-plane capabilities including feature servers, embedding infrastructure, and vector search. Engineering challenges include multi-model coherence, runtime safety, model-aware scheduling, dynamic batching, and fairness scheduling in multi-tenant environments. As with virtualization and container orchestration in previous generations of computing, AIRI establishes AI workloads as first-class distributed system workloads that require a dedicated runtime and layered abstractions for optimal performance. It eases the scalable, reliable, and efficient deployment of generative models, multimodal systems, and agentic architectures in diverse cloud-native environments. This paper presents a layered architectural model for AIRI, identifies key engineering challenges, and discusses implications for future distributed systems infrastructure.

1. Introduction and Conceptual Framework

AIRI is proposed as a reference architectural layer rather than a single concrete implementation. Its purpose is to formalize recurring runtime responsibilities observed across large-scale AI deployments and to define clear boundaries between general-purpose cloud infrastructure and AI-specific execution concerns.

AIRI adds a new architectural layer to the stack of distributed systems. This allows organizations to run, scale, and manage AI workloads in production. Existing distributed systems are built around compute, storage, and networking abstractions for deterministic software execution. While modern AI workloads introduce requirements such as stateful model orchestration, heterogeneous accelerator management, vector-based retrieval, and continuous evaluation loops, these capabilities are typically composed from multiple layers rather than treated as a cohesive runtime concern. AIRI represents the architectural convergence of these functional requirements into an infrastructure layer. This layer runs above cloud-native infrastructure stacks and below cloud-native application workloads to deliver consistent, scalable, and reliable AI infrastructure on any type of compute environment. With the emergence of unifying metric architectures for AI infrastructure, the need for cross-layer taxonomies is likely to grow, covering performance, efficiency, and cost metrics for the entire AI execution stack, with customary infrastructure metrics failing to holistically and effectively capture the behavior of AI workloads [1].As discussed earlier, each generation of distributed systems introduced abstractions that reduced operational complexity for dominant workload types. AIRI continues this progression by

addressing execution patterns unique to modern AI workloads. With serverless computing, the focus moves to business logic, and infrastructure is handled by the platform much more extensively than before. Each new generation reduces the cost of scaling and operational complexity. AIRI continues this evolution, as AI workloads have characteristics which do not fit existing abstractions and implementations of distributed systems: non-linear compute paths, memory and function-intensive operations, multi-modal data, and variable patterns of execution. The technical implications of modern AI serving systems require architectural models that extend beyond customary distributed systems models, as the industry builds applications using models that require coordinated execution across diverse compute resources [2].

AIRI occupies a unique place in the distributed systems landscape, alongside MLOps frameworks that help with training pipelines, versioning, and deploying ML models. However, they primarily focus on the build and deployment phases to guarantee correct and performant runtime behavior. Model serving platforms optimize throughput and latency for inference of individual models, but do not orchestrate across multiple models. While inference engines specialize in low-level execution optimizations for particular models and frameworks, they do not solve resource allocation, policy enforcement, and other cross-model problems. Beyond these concerns, AIRI frames the AI model itself as a first-class distributed system entity, requiring dedicated runtime abstractions for scheduling, orchestration, reliability, and observability across multiple models and hardware environments, providing a solution to the architectural gap between legacy cloud-native stack infrastructures and modern AI workloads. This leads to adaptive scaling, orchestration of multimodal models, heterogeneous compute environments, and policy-driven execution of LLMs and agent-based systems. These factors require a rethink of the design space of infrastructure layers for AI execution environments, in terms of performance metrics, resource consumption, and operating cost patterns, as they differ from other types of application workloads [1].

2. Architectural Inadequacies of Traditional Distributed Systems

The characteristics of workloads common in AI do not align neatly with the scaling assumptions and operational semantics of traditional distributed system architectures. Cloud-native platforms largely optimized for microservices assume short-lived request-response execution, limited per-request state, and comparatively predictable compute and I/O behavior. AI inference violates several of these assumptions, particularly around state persistence, memory locality, and runtime variability. Model weights occupy large, long-lived memory regions, often resident on accelerators, while inference maintains runtime state such as key-value caches for attention and decoding. Many deployments also require persistent conversational state or context reuse across multiple inference turns. They often exhibit probabilistic outputs and variable execution characteristics, where latency and resource consumption change with context length, decoding strategy, and output size, even for similar inputs. This variability complicates capacity planning and frequently results in elevated tail latency under mixed or bursty workloads. Systems may also be adaptive, modifying tool usage, model selection, or execution steps based on input complexity, context window size, or runtime conditions. These behaviors are difficult to express or optimize using general-purpose container orchestration alone.

Updating a model typically requires coordinated rollouts across distributed serving instances, compatibility with dependent models, checkpoint synchronization, and graceful transitions during version changes. This cross-model orchestration is even more important today in modern AI applications, where the output of one model is the input to downstream models, forming dependency graphs that service meshes can route between, but do not manage semantically in terms of shared state, version compatibility, or coordinated execution. These pipelines have additional requirements for more complex routing, context propagation, and failure-handling logic than typical microservice-based applications. These are exacerbated by the differences in accelerator types, which may include heterogeneous accelerators such as GPUs, TPUs, and FPGAs, each with distinct memory, compute, and communication characteristics. General-purpose resource schedulers typically lack first-class awareness of model-specific requirements such as memory bandwidth sensitivity, collective communication patterns, or accelerator affinity. Furthermore, resource requirements for AI inference are highly variable, making static resource allocation inefficient and complicating scheduling decisions in shared environments [3].

A further limitation arises in observability. Monitoring stacks are designed primarily around infrastructure-level telemetry such as CPU utilization, memory usage, and network throughput, and often lack model-aware signals such as token

generation rates, cache hit ratios, or quality-related indicators. Another class of serving challenges arises when organizations treat models as applications. This often results in duplication of model-serving logic across applications, inconsistency in model deployment, and insufficient resource isolation and observability coverage for models. It can lead to inefficient utilization of hardware, unpredictable tail latency, and operational overheads that increase non-linearly with the number of deployed models, thereby adversely affecting the reliability and scalability of production-grade AI systems [4].

3. Technical Requirements Driving AIRI Emergence

The rapid increase in model scale and architectural complexity has fundamentally altered the infrastructure requirements for AI execution. The shift toward large foundation models requires coordinated execution across thousands of accelerators in distributed environments. This introduces challenges related to large aggregate memory footprints, distributed model state, and coherence across many compute nodes, and broader architectural aspects such as memory hierarchies and high-bandwidth interconnects, and requires orchestrating model, pipeline, and tensor parallelism efficiently at runtime. Large language models illustrate these challenges clearly, particularly in multilingual and multimodal deployments across heterogeneous hardware, including dynamic compute adaptation during runtime, distributed attention, and orchestration of inference across components with significantly different computational characteristics. These challenges are difficult to address through incremental extensions to existing distributed systems and motivate the need for AI-specific runtime infrastructure [5].

Generative and multimodal inference workloads are latency-sensitive, particularly in interactive settings such as multi-step reasoning and conversational systems, where end-to-end response time is critical. The presence of variable compute paths and token generation complexities makes it challenging to optimize such workloads. Inference workloads are characterized by highly non-uniform execution patterns regarding processing times based on the input task complexity, the context lengths of inputs, and the expected output lengths. Unlike other web applications, the latency of inference requests in AI applications can vary considerably. For example, inference latency may range from milliseconds for simple requests to several seconds for long-context or long-generation workloads. In the case of context specificity, recently deployed models require orchestration of sequential and parallel model calls, conditional execution based on intermediate decisions, or conditional model invocation, where specialized models are selected dynamically based on task characteristics. These characteristics complicate traditional autoscaling strategies that assume a relatively uniform request cost. Just as optimizing transformer networks via multi-query attention requires changes to runtime infrastructure to exploit reduced memory bandwidth and improved decoder efficiency, architectural advancements similarly motivate runtime-level changes [6].

Additional challenges arise in managing distributed memory bandwidth and accelerator compute resources at scale, due to the need for long-running, high-throughput movement of data through the memory hierarchy, accelerator arrays, and storage, and in multi-tenant isolation and quality-of-service, due to the need for long-running model deployments with different latency and throughput needs and different resource constraints. Containerization and namespace isolation provide limited guarantees when workloads contend for accelerator memory bandwidth and shared compute resources for AI workloads competing for memory bandwidth, accelerator resources, and hardware accelerators. Modern agentic systems dynamically decompose tasks into subtasks, conditionally call multiple models, and adapt the execution flow based on the intermediate results. These non-deterministic execution graphs are poorly aligned with static service mesh frameworks and predefined orchestration patterns used for customary microservices [5] and request-response workflows.

3.1 Illustrative Application: Multi-Model Agentic Customer Support Assistant

To concretize the technical requirements and architectural challenges described above, consider an enterprise-grade, AI-powered customer support assistant deployed in a regulated industry such as financial services or healthcare. The system supports multi-turn conversations, retrieves domain-specific knowledge, invokes specialized models, and enforces strict safety and latency constraints.

3.1.1 Execution Characteristics

A single user request may involve the following runtime steps:

Initial intent interpretation: A general-purpose language model parses the user query and determines intent and confidence.

- Conditional model selection: Based on the inferred intent, the system dynamically selects one or more specialized downstream models, such as a domain-specific reasoning model, a retrieval model for policy documents, or a summarization model for response synthesis.
- Context retrieval and augmentation: Relevant documents are retrieved using vector-based similarity search and injected into the model context. Context windows must be managed carefully to respect token limits and latency budgets.
- Iterative refinement: The assistant may invoke additional model calls conditionally, based on intermediate confidence scores or partial responses.
- Inference-time policy enforcement: Content filters, access controls, and rate limits are applied consistently across all model invocations to ensure compliance and safety.
- Response generation and observability: The final response is generated while capturing model-aware metrics such as token throughput, cache utilization, and end-to-end latency for monitoring and debugging.

3.1.2 Why Traditional Cloud-Native Infrastructure Falls Short?

When implemented using traditional microservices and container orchestration alone, this application faces several challenges. Execution paths are dynamic and non-deterministic, making static service graphs and autoscaling policies ineffective. State is shared across model invocations, including conversational context and retrieved embeddings, which are not naturally handled by stateless request-response services. Resource demand varies widely per request, depending on context length and downstream model selection, complicating capacity planning. Semantic coordination across models, such as version compatibility and shared context, is not managed by service meshes, which operate at the network level. Observability is incomplete, as infrastructure-level metrics do not capture model-specific performance or quality signals. As a result, teams often implement ad hoc orchestration logic within application code, leading to duplication, inconsistent policy enforcement, and increasing operational complexity as the number of models grows.

3.1.3 How AIRI Changes the Execution Model

Under an AI Runtime Infrastructure, inference routing dynamically selects models and hardware based on runtime context, latency objectives, and accelerator availability. Cross-model coordination is handled by the inference fabric, which maintains shared context and execution ordering across model calls. Model-aware scheduling and batching optimize accelerator utilization while respecting per-request latency constraints. Policy enforcement is applied uniformly at inference time, independent of application logic. Model-aware observability provides insight into token-level behavior, cache efficiency, and quality degradation across the entire workflow. This shifts complex coordination logic out of application code and into a dedicated runtime layer, allowing the application to focus on business logic while the infrastructure manages execution efficiency, safety, and scalability.

3.1.4 Why This Illustrates the Need for AIRI?

This example highlights a fundamental difference between AI workloads and traditional cloud-native applications: the unit of execution is no longer a single service invocation, but a dynamic graph of coordinated model executions with shared state and variable resource demand. AIRI formalizes these requirements at the infrastructure layer, enabling consistent, scalable, and governable execution of such applications across heterogeneous environments.

4. Layered Architectural Model of AIRI

The layered architecture of AI Runtime Infrastructure separates control-plane and data-plane responsibilities, with well-defined integration points into existing cloud-native infrastructure. The control plane includes model lifecycle management, inference routing, and policies to enforce the execution of AI workloads at the system level. Model lifecycle management encompasses the versioning, rollout, and retirement of AI models on distributed infrastructure, as well as their availability and consistency throughout the transitions. Inference routing directs requests based on model properties, hardware affinity, locality, and batching considerations. Policy enforcement mechanisms implement runtime safety constraints, content moderation, rate limiting, and access control in a manner that is consistent and transparent across model invocations without requiring application-layer implementation. They are the foundational governance layer for reliable, safe, and compliant AI operation in heterogeneous deployment environments. Scalable serving systems need to optimize for throughput and provide latency objectives, and they need to be able to elastically scale in response to changing workloads both temporally and spatially [7].

The data plane consists of infrastructure to run use cases in an AI workload, including running real-time feature pipelines on feature servers, storing high-dimensional vector representations in embedding clusters, and supporting semantic search and context-window building for generative models via vector-augmented retrieval. Feature servers provide low-latency access to transformed and aggregated features for online inference workloads to enable transformations, aggregations, and temporal functions, and are tuned for low-latency online inference. Embedding systems manage collections of vector embeddings produced by multiple models and consumed across applications. Combined with highly specialized solutions for efficient similarity search, they can be integrated into inference pipelines to perform retrieval-augmented generation and ground generative models in factual knowledge bases and domain-specific document collections. At the level of the model, the runtime orchestration layer implements adaptive scaling strategies based on workloads and QoS objectives. The heterogeneous compute abstraction enables the execution of a model across a wide variety of accelerators, while abstracting over hardware-specific details such as tensor placement, memory allocation, and inter-device communication, and exposing a single execution interface.

Inference fabric enables cross-model coordination across multi-model pipelines and agentic workflows, maintaining shared context and execution ordering. It also maintains dataflow correctness through multiple model invocations with dependent computations and shared context. Observability subsystems provide model-aware monitoring for inference execution, which captures metrics such as token generation rates, attention, cache usage, quality measures, and standard infrastructure telemetry. Runtime behavior tracking supports the detection of model drift, performance degradation, and anomalous inference behavior that may require further investigation. The long history of deep learning has shown that using custom hardware accelerators and distributed training frameworks also calls for their own runtime system abstractions that effectively manage compute and the characteristics of neural workloads [8]. Integration with existing cloud-native architectures establishes clear boundaries between AI-specific runtime responsibilities and general-purpose infrastructure patterns (such as container orchestration, service mesh, and storage abstraction). AIRI builds upon existing heterogeneous infrastructure investments while providing AI-specific runtime capabilities for execution patterns not well supported by general-purpose infrastructure.

5. Engineering Challenges and Research Frontiers

Multi-model coherence and graph execution optimization represent a key frontier in the evolution of AI systems. As AI workflows evolve from homogeneous single-model inference to heterogeneous multi-model AI workflows made up of multiple specialized models, these workflows will increasingly exhibit inter-model dependencies, conditionalized execution paths, and context shared among multiple models. New optimizations will need to preserve consistency over model boundaries, synchronize states, minimize end-to-end inference latency, and maximize resource efficiency. Runtime optimization must address dynamic AI workloads in which execution paths cannot be fully represented statically and are determined by intermediate results, model confidence, and runtime conditions. Runtime safety and inference-time controllability are emerging research areas, given the increasing demand for AI systems to be used in safety-critical scenarios with formal guarantees on the model's behavior, output properties, and constraints. Traditional testing and validation strategies are often inadequate for stochastic generative models, as their output space cannot be entirely characterized a priori. Runtime enforcement techniques exist to detect and avoid undesirable outputs, enforce content policies, and maintain compliance with specific behavioral goals during runtime.

Model-aware scheduling and placement strategies extend beyond traditional resource allocation techniques, as they have to exploit model properties such as memory access patterns, computation graphs, communication costs, or sensitivity to hardware variations. Dynamic batching and compute pooling algorithms specifically solve the problem of serving heterogeneous request distributions with varying latency, input size, and computation requirements. Such batching is constrained by the throughput–latency trade-off as well as the workload's temporal variability and request burst patterns that differ from typical web service traffic. Compute pooling can also allow expensive accelerator resources to be better shared across heterogeneous models and tenants. However, this requires arbitration mechanisms that minimize contention, respect isolation contracts, and enforce fairness across workloads. Heterogeneous hardware abstraction further complicates processor, memory, and programming model design to include various accelerator types; AI infrastructures now comprise

not only multiple types of computing units, but also memory hierarchies and programming models that should be expressed uniformly across the standardized runtime abstractions.

Agentic systems, which utilize adaptive execution environments, pose new reliability challenges that differ from those of customary systems. Non-determinism, dynamic task decomposition, and iterative refinement loops within a single run execution complicate the detection of failures, the recovery process, and the consistency guarantees. At the model execution level, token-aware queuing and context window management are required for language models with fixed contexts and variable-length input and output. Finally, fairness scheduling and model interference mitigation are the challenges of providing consistent performance for models hosted together in a multi-tenant environment. Resource contention patterns and interference remain a major challenge for co-located models. Together, these challenges define a research frontier that must be addressed to produce an AIRI infrastructure layer that meets the reliability, performance, and operational requirements for enterprise applications of next-generation AI technologies.

6. Future Directions and Implications for Distributed Systems

Standardization pathways for AIRI as an operational layer in the cloud-native ecosystem would represent a first step toward creating common abstractions for managing AI workloads across heterogeneous infrastructure and execution contexts. Defining common interfaces, protocols, and operational semantics would enable the portability of AI applications across multiple cloud providers, reduce vendor lock-in, and support ecosystem development around common runtime primitives. More granular standardization efforts targeting model packaging formats, inference serving protocols, observability schemas, resource description languages, and orchestration APIs can open up new avenues for co-designing application frameworks and runtime infrastructure. These elements deeply impact how AI workloads interact with the runtime infrastructure that hosts them. AI-native applications have stateful conversations over multiple turns with persistent memory, dynamic invocation of plugins and functions, and self-adaptive feedback loops. Achieving this functionality benefits from tighter integration with the runtime infrastructure that understands, formats, monitors, and optimizes components for specific patterns.

Within model-level distributed behavior management, key considerations include coordination, shared state, consistency enforcement, and emergent adaptation of a fleet of AI models. For infrastructure teams, a number of platform engineering decisions arise around monitoring performance and quality, enacting safety policies, and controlling costs as they manage a diverse portfolio of models. Infrastructure teams will increasingly require expertise in machine learning operations, optimizer and accelerator management, distributed system design, and AI safety engineering. An evolution toward AI-first runtime guarantees and systemic abstractions can provide distributed systems with more explicit contracts on model execution guarantees (e.g., inference latency distributions, availability of context windows, and enforcement of safety properties). For applications to trust these assurances, they must be composable at system boundaries, and they have to be verifiable through runtime monitoring mechanisms, allowing applications to rely on infrastructure-level assurances without introducing additional defensive logic at the application layer. This requires performance, efficiency, and cost characteristics of the entire stack of AI infrastructure to be considered and shaped [1].

The placement of AIRI within the field of distributed systems thus strengthens its position as an architectural layer (analogous to virtualization, container orchestration, and serverless compute in previous generations), as a new computational model requiring new runtime abstractions, not merely as a new application domain to be served by existing general-purpose infrastructure. The memory orchestration techniques above also show how AI workloads are driving infrastructure evolution. Hardware and software systems are being built to address the access patterns, capacity, and performance requirements of large model inference [3]. As AIRI matures, new systems may emerge as first-class citizens for AI workloads, including networking fabrics, storage systems, and programming models. As with other models in computing, such as time-sharing systems, distributed systems, and cloud computing, such innovations historically led to changes throughout the computer systems stack, including new layers of computer systems architectural abstraction that remained in place until the next model shift occurred.

***Table 1**: Architectural Limitations of Traditional Cloud-Native Distributed Systems for AI Workloads [3, 4]*

System Characteristic	Traditional Distributed Systems	AI Workload Requirements	Architectural Impact
State Management	Short-lived request-response execution with limited persisted state	Large, long-lived model weights and runtime state, such as attention key-value caches	Misalignment with memory locality and state coordination requirements
Execution Determinism	Relatively predictable execution and resource consumption	Probabilistic outputs with highly variable latency and resource demand	Capacity planning complexity and elevated tail latency
Resource Allocation	Assumed homogeneity in compute and memory allocation	Input and model-dependent memory footprints and execution duration	Inefficient static allocation and scheduling complexity
Model Lifecycle	Independent service versioning and deployment	Coordinated model rollouts with dependency and compatibility constraints	Cross-model orchestration overhead
Cross-Component Orchestration	Service-level routing and connectivity	Multi-model pipelines with shared state and conditional execution paths	Lack of semantic coordination and context propagation
Hardware Abstraction	Limited awareness of hardware heterogeneity	Heterogeneous accelerators with distinct memory and communication characteristics	Suboptimal placement and resource utilization
Observability	Infrastructure-level telemetry (CPU, memory, network)	Model-aware metrics such as token throughput, cache utilization, and quality signals	Limited visibility into inference behavior and performance
Deployment Pattern	Application-centric deployment units	Model-centric runtime entities	Operational duplication and scalability bottlenecks

Table 2: *Technical Requirements Driving AI Runtime Infrastructure Emergence [5, 6]*

Technical Requirement Category	Specific Challenges	Infrastructure Impact	Traditional System Inadequacy
Model Complexity Scaling	Coordination of a large number of accelerators and the distributed model state	Distributed execution strategies, sophisticated memory hierarchies	Incremental extensions to existing systems are insufficient
Multilingual/Multimodal Coordination	Dynamic compute balancing, distributed attention mechanisms	Runtime support for varying computational density and modality	Heterogeneous hardware orchestration gaps
Latency Sensitivity	Interactive response times for multi-step reasoning chains	Stringent performance requirements	Traditional autoscaling strategies are insufficient for variable request cost
Variable Compute Paths	Processing time varies by input complexity, context length	Highly non-uniform execution patterns	Static scheduling assumptions break down
Token Generation Dynamics	Unpredictable generation sequences and durations	Quality-of-service guarantee challenges	Unpredictable tail latency and capacity planning complexity
Multi-Model Pipelines	Sequential and parallel model invocations with conditional branching	Sophisticated workflow orchestration requirements	Service meshes lack semantic model-level coordination
Agentic Execution Patterns	Dynamic model selection based on task requirements	Adaptive execution strategy implementation	Poor alignment with static orchestration patterns
Memory Bandwidth Management	High-throughput data movement across hierarchies	Sustained bandwidth across accelerator arrays	Limited isolation guarantees under shared accelerator contention
Multi-Tenant Isolation	Diverse latency and	Quality-of-service	Container-level

	throughput requirements	enforcement mechanisms	isolation provides limited guarantees
Dynamic Task Decomposition	Conditional model invocation with adaptive flow	Non-deterministic execution graph management	Predefined orchestration patterns lack flexibility

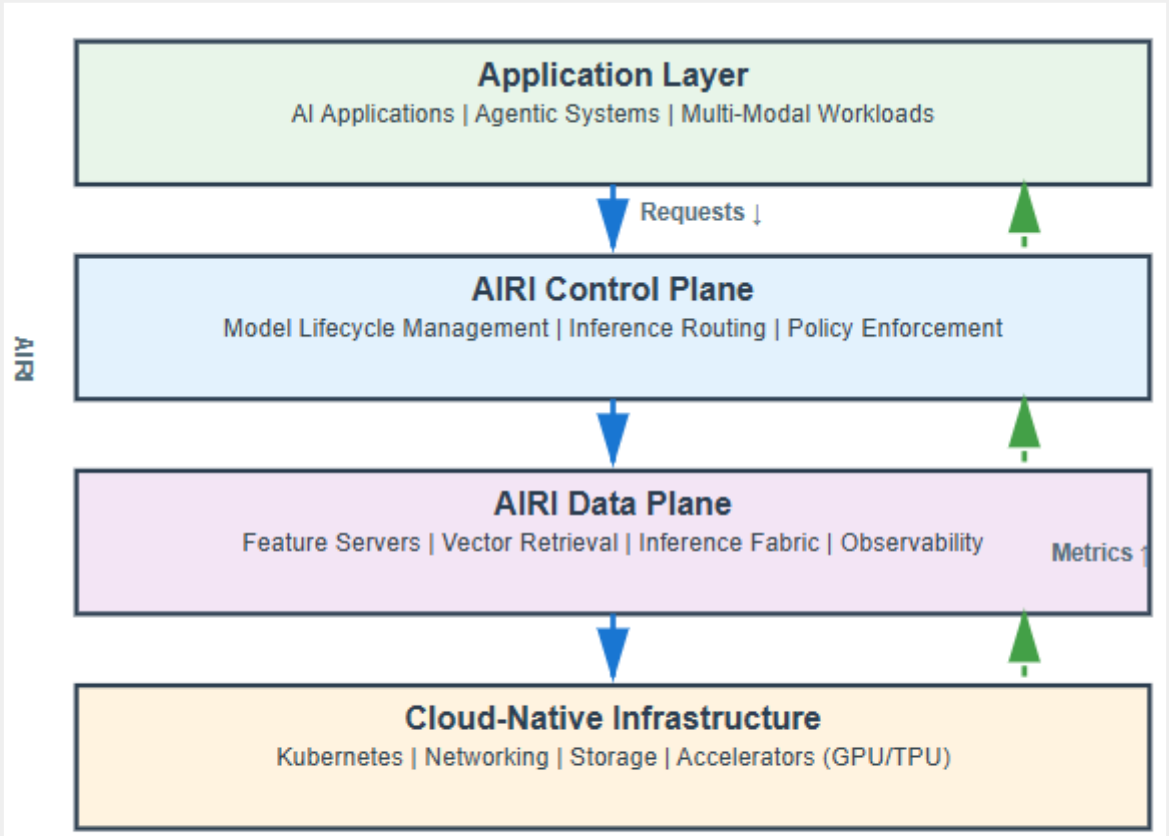


Figure 1: AI Runtime Infrastructure as a Layered Runtime Model

AIRI provides AI-specific runtime capabilities between application workloads and cloud-native infrastructure for model orchestration and governance.

Table 3: *Layered Architectural Components of AI Runtime Infrastructure [7, 8]*

Architectural Layer	Component	Primary Responsibilities	Key Capabilities	Integration Points
Control Plane	Model Lifecycle Management	Deployment, versioning, rollout strategies, and model retirement	Consistency maintenance, availability assurance, and graceful transitions	Container orchestration platforms
	Inference Routing	Intelligent request distribution, hardware affinity optimization	Latency minimization, throughput maximization, locality preferences	Service mesh and request routing layers
	Policy Enforcement	Safety constraints, content filtering, rate limiting, and access control	Transparent governance, runtime compliance	Cloud-native security layers
Data Plane	Feature Servers	Real-time feature pipeline maintenance	Transformations, aggregations, temporal computations	Data source connectors
	Embedding Systems	High-dimensional vector representation management	Efficient storage, retrieval, and similarity	Vector database systems

			computation	
	Vector-Powered Retrieval	Semantic search and context augmentation	Retrieval-augmented generation, knowledge grounding	Knowledge base repositories
Runtime Orchestration	Adaptive Scaling	Demand-responsive scaling with QoS objectives	Dynamic resource allocation, workload adaptation	Cloud autoscaling systems
	Heterogeneous Compute Abstraction	Unified execution across diverse accelerators	Tensor placement, memory allocation, and inter-device communication	Hardware abstraction layers
Inference Fabric	Cross-Model Coordination	Multi-model pipeline management	Dataflow consistency, dependent computation handling	Application frameworks
	Agentic Workflow Support	Dynamic execution path management	Shared context maintenance, conditional invocation	Agent orchestration systems
Observability	Model-Aware Monitoring	Inference-specific metric capture	Token rates, attention patterns, cache utilization, quality indicators	Monitoring infrastructure
	Runtime Behavior Tracking	Drift detection, performance monitoring	Anomaly identification, degradation alerts	Telemetry aggregation systems

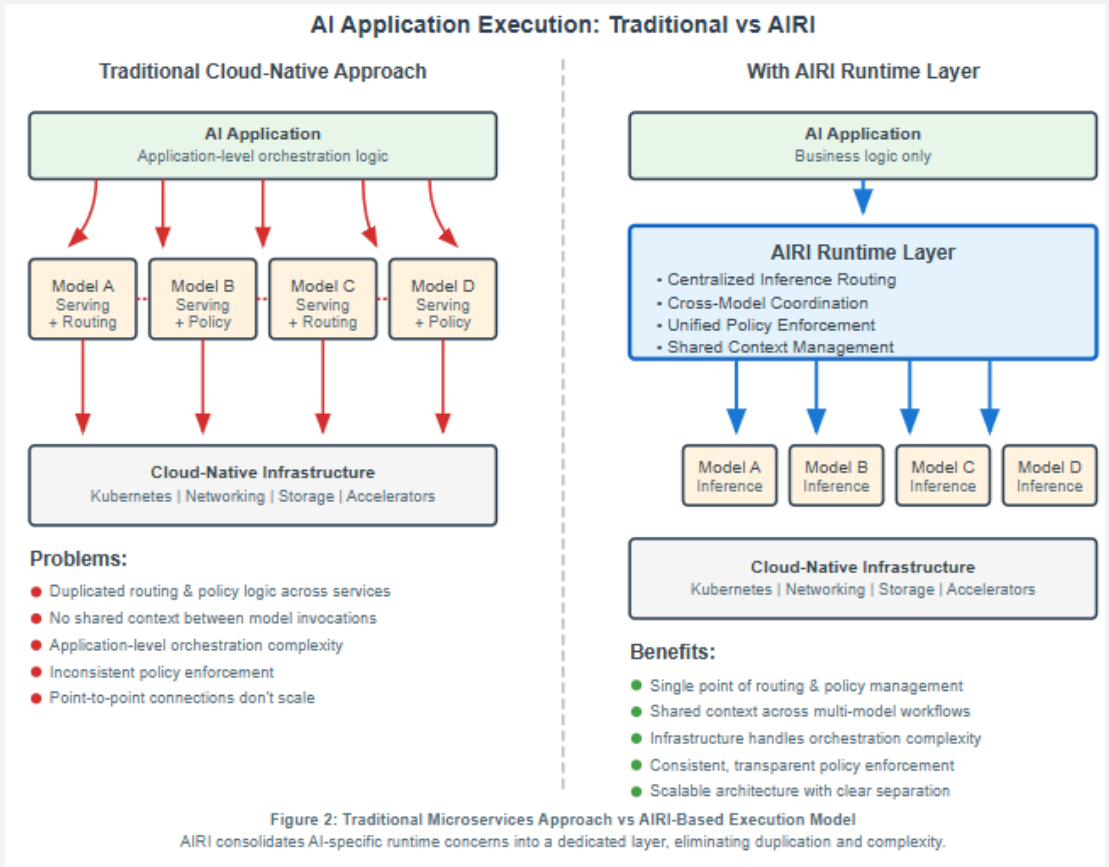


Figure 2: Traditional Microservices Approach vs AIRI-Based Execution Model
AIRI consolidates AI-specific runtime concerns into a dedicated layer, eliminating duplication and complexity.

Table 4: *Engineering Challenges and Research Frontiers in AI Runtime Infrastructure [1, 3]*

Challenge Domain	Technical Complexity	Research Frontier
Multi-Model Coherence	State propagation across	Coordination mechanisms

	boundaries	
Graph Execution Optimization	Dynamic path determination	Adaptive scheduling
Runtime Safety Mechanisms	Verifiable behavior guarantees	Inference-time enforcement
Model-Aware Scheduling	Hardware-sensitive placement	Deep characteristic understanding
Dynamic Batching Strategies	Heterogeneous request streams	Temporal variability adaptation
Compute Pooling Techniques	Multi-tenant accelerator sharing	Arbitration mechanisms
Heterogeneous Hardware Abstraction	Unified runtime across processors	Coherent abstraction frameworks
Agentic System Reliability	Non-deterministic behavior	Failure detection strategies
Token-Aware Queuing	Context window optimization	Intelligent request scheduling
Fairness Scheduling	Multi-tenant performance	Interference mitigation

7. Conclusions

AIRI adds a new layer to the distributed systems taxonomy to address key mismatches between traditional cloud-native infrastructure assumptions and modern AI workload characteristics. By formalizing AIRI as a first-class infrastructure abstraction, organizations can enable more systematic approaches to challenges, such as stateful model orchestration, heterogeneous accelerator management, multi-model pipeline orchestration, and controllability at inference time. The layered architectural model cleanly separates control-plane and data-plane responsibilities, clarifying execution, governance, and observability concerns. It is designed to integrate with existing container orchestration and service mesh systems. Model-aware scheduling, dynamic batching, runtime safety, and fairness scheduling remain open engineering challenges in realizing the layered architecture model. To improve portability across infrastructure providers, standardization pathways may be needed, which could enable the development of an ecosystem of software leveraging standard runtime primitives. The proposed development of AIRI is consistent with the evolution of earlier computing models discussed in this paper, in the domains of hardware, networking, storage, and system software. As artificial intelligence workloads become a major category in enterprise computing, AIRI represents a potential infrastructure abstraction, analogous to virtualization or serverless computing, that may influence how AI workloads are executed, scaled, and governed in cloud-native environments.

Author Statements:

- **Ethical approval:** The conducted research is not related to either human or animal use.
- **Conflict of interest:** The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper
- **Acknowledgement:** The authors declare that they have nobody or no-company to acknowledge.
- **Author contributions:** The authors declare that they have equal right on this paper.
- **Funding information:** The authors declare that there is no funding to be acknowledged.
- **Data availability statement:** The data that support the findings of this study are available on request from the corresponding author. The data are not publicly available due to privacy or ethical restrictions.
- **Use of AI Tools:** The author(s) declare that no generative AI or AI-assisted technologies were used in the writing process of this manuscript.

References

[1] Q. He, "A Unified Metric Architecture for AI Infrastructure: A Cross-Layer Taxonomy Integrating Economics, Performance, and Efficiency," 2025. [Online]. Available: https://arxiv.org/pdf/2511.21772

[2] Xupeng Miao et al., "Towards Efficient Generative Large Language Model Serving: A Survey from Algorithms to Systems," ACM Computing Surveys, Volume 58, Issue 1, 2025. [Online]. Available: https://dl.acm.org/doi/10.1145/3754448

[3] Jiamin Li et al., "FengHuang: Next-Generation Memory Orchestration for AI Inferencing," arXiv:2511.10753v1, 2025. [Online]. Available: https://arxiv.org/pdf/2511.10753

[4] Jovan Stojkovic et al., "Rearchitecting Datacenter Lifecycle for AI: ATCO-Driven Framework," arXiv:2509.26534, 2025. [Online]. Available: https://arxiv.org/pdf/2509.26534

[5] Rohan Anil et al., "PaLM 2 Technical Report," arXiv:2305.10403, 2023. [Online]. Available: https://arxiv.org/abs/2305.10403

[6] Noam Shazeer, "Fast Transformer Decoding: One Write-Head is All You Need," arXiv:1911.02150, 2019. [Online]. Available: https://arxiv.org/abs/1911.02150

[7] Zedong Liu et al., "ElasticMM: Efficient Multimodal LLMs Serving with Elastic Multimodal Parallelism," arXiv:2507.10069v1, 2025. [Online]. Available: https://arxiv.org/html/2507.10069v1

[8] Jeffrey Dean, "The Deep Learning Revolution and Its Implications for Computer Architecture and Chip Design,". [Online]. Available: https://arxiv.org/pdf/1911.05289