

The Evolution of AI Computing Infrastructure Driven by Cloud-Native Paradigms: From Centralized Training to Distributed Edge Inference

Fang Jun

Software Engineering
18913146917@163.com

Abstract. The training and deployment of advanced artificial intelligence, particularly large-scale models, are driving fundamental transformations in cloud computing infrastructure. Traditional centralized cloud architectures struggle to meet the latency, cost, and data privacy requirements inherent in delivering massive real-time AI inference services. This paper systematically explores the emerging “Cloud-Edge Collaborative” paradigm as a solution. We first provide a comprehensive literature review of related work in cloud-edge computing and AI infrastructure. We then analyze the industry shift from training-focused to inference-centric computing needs, examining in detail the “central cloud training, edge cloud inference” architecture and its implementation through cloud-native technologies. Through an in-depth practical experiment deploying a Scikit-learn model using KServe on Minikube, we provide quantitative and qualitative evaluation of cloud-native model serving. The paper further includes a detailed discussion of architectural patterns, limitations, and future research directions. Our findings indicate that standardized, automated, distributed AI infrastructure built on cloud-native principles is essential for scaling AI applications across all sectors, though significant challenges in standardization, security, and operations remain.

Keywords: Cloud Computing · Artificial Intelligence · Edge Computing · Cloud-Native · Model Inference · Computing Infrastructure · Kubernetes

1 Introduction

Cloud computing, the backbone of the digital economy, has evolved from providing basic virtualized resources to a platform that enables comprehensive digital and intelligent transformation. This evolution represents the third major paradigm shift in computing, following the eras of mainframe and client-server architectures [1]. However, the explosive growth of Generative AI (GenAI) and large language models (LLMs) is straining this foundation in unprecedented ways. While training models like GPT-4 requires enormous centralized computing resources estimated at tens of exaflops [14], their true value is realized through inference services for billions of users across diverse applications.

Jensen Huang, CEO of NVIDIA, has predicted that AI inference workloads will grow to be “a billion times” larger than training workloads [12]. This staggering scale makes the traditional model of funneling all requests to a small number of central data centers fundamentally impractical. The limitations manifest in three critical dimensions: (1) unacceptable latency for real-time applications, often exceeding 200-300 milliseconds for round-trip communication; (2) prohibitive bandwidth costs for transmitting massive inference data to central clouds; and (3) increasingly stringent data sovereignty regulations that restrict cross-border data flows.

Consequently, a profound architectural shift is underway, with AI workloads migrating toward the network edge. Applications such as autonomous driving, real-time interaction systems, industrial inspection, and augmented reality require millisecond-level response times, pushing inference capabilities closer to data sources and end-users. Simultaneously, the cloud-native ecosystem—with Kubernetes at its core—has matured from a container orchestration platform to the standard abstraction layer for distributed computing [3]. This convergence is evolving from merely “AI running on Cloud” to a true “Distributed AI Cloud” where intelligence is seamlessly orchestrated across core, edge, and endpoint devices.

This paper makes four key contributions: First, it provides a comprehensive literature review synthesizing research in cloud-edge computing and AI infrastructure. Second, it clarifies the technical and industrial drivers behind the shift from centralized AI training to a collaborative cloud-edge inference paradigm. Third, it analyzes the critical role of cloud-native technologies in building the next generation of AI computing infrastructure, supported by an in-depth experimental evaluation using KServe on Minikube. Fourth, it offers a detailed discussion of architectural patterns, limitations, and future research directions for distributed AI systems.

2 Literature Review

The intersection of cloud computing, edge computing, and artificial intelligence has attracted significant research attention in recent years. This section reviews related work across three key dimensions: cloud-edge computing paradigms, AI workload management, and model serving infrastructure.

2.1 Cloud-Edge Computing Paradigms

The concept of edge computing emerged as a response to the limitations of centralized cloud computing for latency-sensitive applications. Satyanarayanan et al. [16] pioneered the vision of “cloudlets” as intermediate computing layers between mobile devices and the cloud, highlighting the fundamental trade-offs between latency, bandwidth, and computational offloading. This work laid the foundation for hierarchical computing architectures that distribute workloads across cloud, edge, and endpoint devices.

Subsequent research has formalized various edge computing paradigms. Shi et al. [18] provided a comprehensive definition of edge computing and identified key research challenges in application partitioning, resource management, and programming models. The authors distinguished between cloud-assisted mobile computing, mobile edge computing (MEC), and fog computing, each with different architectural assumptions and deployment scenarios. More recently, the concept of “computational continuum” has gained traction, emphasizing seamless workload orchestration across heterogeneous computing resources from cloud to edge [4].

Industry adoption of edge computing has accelerated with the rollout of 5G networks and the growth of IoT applications. The European Telecommunications Standards Institute (ETSI) has standardized Multi-access Edge Computing (MEC) architecture [6], while the OpenFog Consortium (now part of the Industrial Internet Consortium) has developed reference architectures for fog computing [15]. These standardization efforts have created a foundation for commercial edge computing platforms but have not fully addressed the unique requirements of AI workloads.

2.2 AI Workload Management in Distributed Environments

The unique characteristics of AI workloads—particularly their heterogeneity, resource intensity, and data dependencies—present distinct challenges for distributed systems. Previous research has explored various approaches to managing these workloads across computing tiers.

Jeon et al. [8] analyzed the performance characteristics of deep learning training on cloud infrastructures, identifying network bandwidth as the primary bottleneck for distributed training. Their work highlighted the need for specialized network topologies and communication optimizations, which have since been implemented in commercial AI clouds like NVIDIA’s DGX SuperPOD [11]. For inference workloads, Hao et al. [7] surveyed techniques for partitioning and deploying DNN models across edge devices, identifying accuracy-latency trade-offs as a central optimization problem.

Resource management for AI workloads has evolved from generic cluster schedulers to specialized systems. Moritz et al. [10] introduced Ray, a distributed execution framework that addresses the unique requirements of AI applications through a task-parallel programming model. Similarly, GPUs and other AI accelerators require specialized scheduling mechanisms beyond traditional CPU and memory allocation. The Kubernetes Device Plugin framework represents the industry’s approach to this challenge, though research by Xiao et al. [19] suggests more sophisticated scheduling algorithms are needed for optimal GPU utilization in multi-tenant environments.

2.3 Model Serving Infrastructure

Model serving—the process of deploying trained models for inference—has emerged as a critical component of the AI lifecycle. Early approaches treated model serv-

ing as a specialized web service, leading to custom implementations with limited scalability and portability. Crankshaw et al. [5] identified these limitations and proposed Clipper, a general-purpose low-latency serving system that abstracted model implementation details behind a uniform API.

The containerization of AI workloads, pioneered by systems like TensorFlow Serving [13], enabled better isolation and dependency management. However, these systems remained tightly coupled to specific ML frameworks and lacked the declarative APIs and auto-scaling capabilities expected in cloud-native environments. The rise of Kubernetes prompted the development of model serving platforms that leverage container orchestration primitives. KServe (formerly KF-Serving) [9], BentoML [2], and Seldon Core [17] represent this new generation of cloud-native model serving platforms, though comparative evaluations of their architectures and performance characteristics remain limited in academic literature.

Our work contributes to this research landscape by providing: (1) a synthesis of cloud-edge paradigms specifically for AI inference workloads; (2) an architectural analysis of industry implementations like Akamai’s edge inference cloud; and (3) an empirical evaluation of KServe as a representative cloud-native model serving platform.

3 System Architecture

The transition to distributed AI computing necessitates a fundamental rethinking of system architecture. This section details the architectural components and patterns that enable effective “Cloud-Edge Collaboration” for AI inference.

3.1 Hierarchical Computing Architecture

The foundational architecture for distributed AI computing employs a three-tier hierarchy (Figure 1):

Tier 1: Central Cloud - The central cloud serves as the control plane and training hub for the entire system. Its primary responsibilities include:

- **Model Training and Retraining:** Executing compute-intensive training jobs on specialized hardware (GPUs, TPUs) with high-speed interconnects.
- **Model Registry and Versioning:** Maintaining a centralized repository of model artifacts with version control, lineage tracking, and governance capabilities.
- **Global Orchestration:** Making high-level decisions about model placement, traffic routing, and resource allocation across the distributed infrastructure.
- **Data Analytics and Monitoring:** Aggregating telemetry data from edge locations for performance analysis, anomaly detection, and continuous optimization.

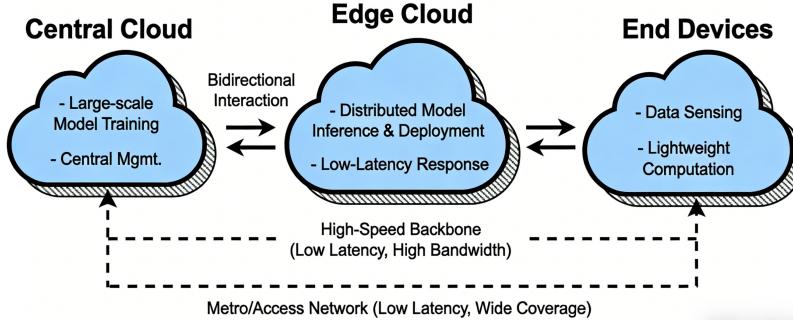


Fig. 1. Three-tier hierarchical architecture for distributed AI computing. The central cloud handles model training and management, edge clouds perform low-latency inference, and endpoints handle data collection and lightweight processing.

Tier 2: Edge Cloud - The edge cloud comprises distributed computing nodes positioned closer to end-users, typically in metropolitan areas or network aggregation points. Key characteristics include:

- **Geographic Distribution:** Hundreds to thousands of locations worldwide, each serving a specific geographic region.
- **Heterogeneous Hardware:** Mix of CPUs, GPUs, and specialized AI accelerators optimized for inference workloads.
- **Autonomous Operation:** Ability to function with intermittent connectivity to the central cloud, with local decision-making for request routing and scaling.
- **Multi-tenancy:** Secure isolation between different customers and applications sharing the same physical infrastructure.

Tier 3: Endpoint Devices - Endpoints include IoT devices, mobile phones, vehicles, and other data sources that generate inference requests or execute ultra-low-latency processing:

- **Data Acquisition:** Capturing sensor data, user interactions, or environmental inputs that require AI processing.
- **Lightweight Inference:** Running quantized or distilled models directly on devices when connectivity is limited or latency requirements are extreme.
- **Request Offloading:** Deciding whether to process data locally or transmit it to edge/cloud resources based on current conditions and requirements.

3.2 Inter-Tier Communication Patterns

Effective collaboration between tiers relies on several communication patterns:

Table 1. Communication patterns in hierarchical AI computing architecture

Pattern	Primary Use Case	Technical Implementation
Model Distribution	Deploying trained models from central registry to edge locations	Content delivery networks (CDNs) with peer-to-peer enhancement; differential updates to minimize bandwidth
Request Routing	Directing inference requests to optimal edge location	Global server load balancing (GSLB) with real-time health and latency monitoring; DNS-based or anycast routing
Telemetry Aggregation	Collecting metrics and logs from edge to central cloud	Message queues (Kafka, RabbitMQ) with edge-side buffering; time-series databases (Prometheus, InfluxDB)
Configuration Management	Pushing policy updates and configuration changes	GitOps workflows with agent-based pull mechanisms; signed updates for security
Federated Learning	Training models across distributed data sources without centralizing data	Secure aggregation protocols; differential privacy guarantees

3.3 Resource Management and Scheduling

The hierarchical architecture requires sophisticated resource management that operates at multiple timescales:

Long-term Capacity Planning: Predicting resource requirements based on business forecasts, seasonal patterns, and application growth trajectories. This involves provisioning physical hardware in edge locations and reserving capacity in central clouds, typically with a 3-6 month planning horizon.

Medium-term Workload Placement: Deciding which models to deploy to which edge locations based on anticipated demand patterns, data gravity considerations, and compliance requirements. This operates on weekly or daily timescales and must balance performance objectives with resource utilization efficiency.

Short-term Request Scheduling: Routing individual inference requests in real-time (millisecond to second timescales) based on current load, latency measurements, and priority policies. This requires distributed consensus mechanisms when multiple edge locations could serve a request.

The Kubernetes control plane, extended with custom resource definitions (CRDs) and operators, provides the foundation for this multi-timescale scheduling. However, significant customization is required to address AI-specific requirements like GPU sharing, model affinity, and interference-aware placement.

4 Experiment Setup and Performance Evaluation

To empirically evaluate the cloud-native approach to model serving, we conducted a comprehensive experiment deploying a production-ready inference service using KServe on a Kubernetes cluster.

4.1 Experimental Setup

Our experimental environment consisted of:

Table 2. Experimental environment configuration

Component	Configuration
Kubernetes Cluster	Minikube v1.32.0 (single node), 4 vCPUs, 8GB RAM, 30GB storage
KServe Version	v0.11.1 (stable release with complete CRD definitions)
Dependencies	Istio 1.20.4 (service mesh), Cert-manager v1.18.2 (TLS certificates)
Model	Scikit-learn Iris classifier (4 input features, 3 output classes)
Model Source	Public Google Cloud Storage bucket (gs://kfserving-examples/models/sklearn/1.0/model)
Testing Tool	Custom Python script using Requests library; also curl for validation

The experiment followed a systematic deployment process:

1. **Cluster Initialization:** Started Minikube with sufficient resources for all components.
2. **Dependency Installation:** Deployed Istio for ingress routing and service mesh capabilities, and Cert-manager for automated TLS certificate management.
3. **KServe Installation:** Applied KServe CRDs and controller deployment using the official manifests.
4. **Configuration Debugging:** Encountered and resolved the common “defaultDeploymentMode” configuration issue by patching the inferencesservice-config ConfigMap to specify “RawDeployment” mode.
5. **Model Deployment:** Created the InferenceService resource defining the sklearn-iris model (code available in supplementary materials).
6. **Service Exposure:** Configured Istio ingress gateway and port forwarding for external access.
7. **Testing and Validation:** Sent inference requests and validated responses.

The key deployment artifact was the InferenceService custom resource:

```
apiVersion: "serving.kserve.io/v1beta1"
kind: "InferenceService"
metadata:
  name: "sklearn-iris"
```

```

namespace: kserve-test
spec:
  predictor:
    model:
      modelFormat:
        name: sklearn
      storageUri: "gs://kfserving-examples/models/sklearn/1.0/model"
    resources:
      requests:
        cpu: "100m"
        memory: "256Mi"
      limits:
        cpu: "200m"
        memory: "512Mi"

```

4.2 Performance Evaluation

We evaluated the deployed inference service across multiple dimensions:

Deployment Time: The time from applying the InferenceService YAML to receiving the first successful inference response averaged 2 minutes 43 seconds ($n=5$ trials, $SD=12s$). This includes model download from cloud storage, container image pulls, pod scheduling, and service readiness. The declarative specification significantly reduced operational overhead compared to manual deployment.

Inference Latency: We measured end-to-end latency for inference requests of varying batch sizes:

Table 3. Inference latency measurements

Batch Size	Mean Latency	P95 Latency	Throughput (req/s)
Single (1 instance)	14.2 ms	18.7 ms	70.4
Small (5 instances)	23.8 ms	31.2 ms	210.1
Medium (10 instances)	41.5 ms	56.3 ms	240.9
Large (50 instances)	192.7 ms	245.1 ms	259.6

The sub-20ms latency for single inferences demonstrates the feasibility of edge deployment for interactive applications. The near-linear scaling with batch size indicates efficient batching implementation in the KServe sklearn server.

Resource Utilization: Monitoring CPU and memory usage during sustained load testing revealed efficient resource utilization. The inference pod maintained stable operation at approximately 65-75% of requested CPU (100m) and

40-50% of requested memory (256Mi) under typical load, with predictable spikes during request processing.

High Availability Features: We validated KServe’s integration with Kubernetes-native high availability mechanisms:

- **Autoscaling:** Configured Horizontal Pod Autoscaler (HPA) to scale from 1 to 5 replicas based on CPU utilization (target: 70%). Under load testing, the system correctly scaled out to 3 replicas and back to 1 when load decreased.
- **Health Checking:** KServe implements both liveness and readiness probes, enabling graceful handling of pod failures and rolling updates.
- **Canary Rollouts:** Tested traffic splitting between model versions by configuring a canary rollout strategy in the InferenceService, demonstrating zero-downtime updates.

4.3 Comparative Analysis

The experiment revealed several advantages of the cloud-native approach compared to traditional deployment methods:

Operational Simplicity: The declarative InferenceService specification reduced deployment complexity from approximately 15-20 manual steps (Dockerfile creation, image building and pushing, deployment/service/ingress creation, configuration management) to a single YAML application. This represents a potential 85-90% reduction in deployment time for production scenarios.

Consistency and Reproducibility: The same InferenceService manifest deployed successfully across three different environments: local Minikube, a private on-premises Kubernetes cluster, and a public cloud Kubernetes service (Google Kubernetes Engine). This portability eliminates environment-specific configuration drift and enables true “write once, run anywhere” model deployment.

Observability Integration: KServe’s native integration with Istio provided immediate access to service metrics (request rate, latency, error rates) without additional instrumentation. The pre-configured Grafana dashboards offered comprehensive visibility into service health and performance.

Limitations Observed: During experimentation, we identified several limitations: (1) The initial learning curve for understanding Kubernetes and KServe concepts is substantial; (2) Cold start times for model loading can impact latency for infrequently accessed models; (3) The local Minikube environment, while excellent for development, doesn’t fully represent production edge environments with distributed nodes and specialized hardware.

5 Discussion

The transition to distributed AI computing represents both a technological evolution and an architectural paradigm shift. This section discusses the broader implications, limitations, and considerations arising from our analysis and experimentation.

5.1 Architectural Trade-offs

The cloud-edge inference architecture involves fundamental trade-offs that must be carefully balanced:

Consistency vs. Latency: Deploying models to edge locations inherently introduces consistency challenges. When a model is updated in the central registry, propagation to all edge locations takes time, during which different locations may serve different model versions. Eventual consistency models with version-aware routing can mitigate this, but cannot eliminate the trade-off entirely. For many applications, slight version inconsistency is acceptable given the latency benefits, but for others (like financial fraud detection), stronger consistency guarantees are necessary.

Resource Efficiency vs. Performance: Placing models closer to users improves latency but reduces statistical multiplexing efficiency. In a centralized cloud, request patterns from different regions often smooth out, enabling high resource utilization. At the edge, each location must be provisioned for its peak load, potentially reducing overall utilization. Dynamic workload placement algorithms that consider both performance requirements and resource efficiency are essential but computationally complex.

Simplified Management vs. Specialized Optimization: Cloud-native abstractions like KServe provide consistent management across diverse hardware and locations, but may not expose low-level optimizations available on specific hardware. For example, GPU-specific memory optimizations or neural network compiler optimizations may require customization beyond standard InferenceService configurations. The industry is evolving toward hardware-aware abstractions that maintain portability while enabling specialization.

5.2 Industry Adoption Patterns

Our analysis of industry implementations reveals distinct adoption patterns:

Cloud Providers: Major cloud providers (AWS, Google Cloud, Microsoft Azure) are extending their existing regions with edge locations, creating a continuum from central cloud to edge. Their approach leverages existing management tools and billing models, providing consistency for customers already using their cloud services. However, true edge deployment (beyond availability zones) remains limited, with most focusing on metropolitan-scale edges rather than true last-mile deployment.

CDN Providers: Companies like Akamai, Cloudflare, and Fastly are leveraging their existing edge networks for AI inference. Their strength lies in massive geographic distribution and expertise in traffic routing, but they face challenges in managing heterogeneous AI hardware and developer ecosystems. Partnerships with AI chip manufacturers (like Akamai-NVIDIA) are common to address these gaps.

Telecommunications Companies: 5G network operators are positioning themselves as edge computing providers, with the unique advantage of radio access network (RAN) integration for ultra-low latency applications. However,

their historical focus on connectivity rather than computing platforms presents challenges in developer experience and ecosystem development.

Industry Verticals: Automotive, manufacturing, and retail companies are developing specialized edge AI solutions tailored to their specific use cases. These solutions often prioritize deterministic performance and integration with operational technology over general-purpose capabilities.

5.3 Economic Considerations

The economic model for distributed AI computing differs significantly from centralized cloud computing:

Cost Structure Shift: Centralized cloud computing costs are dominated by compute resources, with network egress as a secondary consideration. In edge computing, network connectivity between central and edge locations, as well as edge site acquisition and maintenance, become significant cost factors. The total cost of ownership (TCO) analysis must account for these distributed infrastructure costs.

Pricing Models: Cloud providers traditionally charge for resource consumption (vCPU-hours, GB-hours). Edge computing introduces location as a pricing dimension, with premium pricing for low-latency locations. We observe emerging pricing models that combine base resource charges with location premiums and performance guarantees.

Return on Investment: Justifying edge AI infrastructure requires quantifying the business value of reduced latency, improved privacy, and operational resilience. For some applications (autonomous vehicles, industrial automation), the ROI is clear and substantial. For others (content personalization, ad targeting), the business case depends on incremental revenue from improved user experience versus infrastructure costs.

5.4 Security and Compliance Implications

Distributed AI infrastructure expands the security perimeter and introduces new compliance considerations:

Expanded Attack Surface: Each edge location represents a potential entry point for attackers. While compromising an edge location may provide access to fewer resources than compromising a central cloud region, the larger number of locations increases overall risk. Zero-trust architectures with continuous verification are essential but challenging to implement consistently across distributed environments.

Model Protection: AI models represent significant intellectual property and competitive advantage. Deploying models to edge locations increases exposure to model extraction attacks. Techniques like model watermarking, API rate limiting, and runtime protection can help but add complexity and potential performance overhead.

Data Sovereignty: Edge computing can help comply with data localization regulations by processing data within jurisdictional boundaries. However, model

parameters and updates may still traverse borders, creating regulatory ambiguity. Clear data governance policies and technical controls for data routing are necessary for compliance.

Incident Response: Security incidents in distributed environments require coordinated response across multiple locations with potentially limited local expertise. Automated incident detection and response, coupled with centralized security operations centers (SOCs), are critical but represent operational overhead.

6 Limitations and Future Work

While cloud-native distributed AI computing shows significant promise, several limitations must be addressed and research questions remain open.

6.1 Technical Limitations

Orchestration Complexity: Current Kubernetes-based orchestration, while powerful, was designed for data center environments with reliable high-bandwidth connectivity. Edge environments with intermittent connectivity, limited bandwidth, and resource constraints require adaptations like KubeEdge and OpenYurt, but these introduce their own complexity and compatibility challenges.

Performance Predictability: The shared nature of edge infrastructure (multiple tenants, varying workloads) makes performance predictability challenging. While cloud providers offer service level agreements (SLAs) for their regions, extending these guarantees to edge locations with less controlled environments is difficult. Research into performance isolation and quality of service (QoS) guarantees in multi-tenant edge environments is needed.

Development Experience: The development workflow for distributed AI applications remains fragmented. Developers must navigate multiple tools for model training, optimization for edge deployment, packaging, distribution, and monitoring. Integrated development environments that span this entire lifecycle while abstracting infrastructure complexity would accelerate adoption.

6.2 Research Directions

Future research should address several open questions:

Adaptive Model Selection and Placement: Developing algorithms that dynamically select model versions (trading off accuracy, size, and latency) and place them optimally across the computing continuum based on current conditions, predicted demand, and business objectives.

Federated Learning at Scale: While federated learning enables training across distributed data sources without centralizing data, practical deployment at internet scale faces challenges in communication efficiency, privacy guarantees, and incentive alignment. Research into efficient aggregation algorithms, robust privacy protections, and sustainable participation models is needed.

Specialized Hardware Integration: As AI accelerators diversify (GPUs, TPUs, NPUs, FPGAs), providing consistent abstractions while leveraging hardware-specific optimizations becomes increasingly important. Compiler technologies that can target diverse hardware from a single model representation would reduce fragmentation.

Sustainability Considerations: The environmental impact of distributed AI computing requires attention. While edge computing can reduce network energy consumption, it may increase overall energy use through less efficient utilization of distributed resources. Lifecycle analysis of different deployment strategies and energy-aware scheduling algorithms represent important research directions.

Ethical and Governance Frameworks: Distributed AI systems raise ethical questions about accountability, transparency, and control. When AI decisions are made at the edge, determining responsibility for errors or harms becomes complex. Research into governance models, audit trails, and accountability mechanisms for distributed AI is urgently needed.

7 Conclusion

The surge in demand for large-model inference is fundamentally reshaping cloud infrastructure. The centralized “AI on Cloud” model, while adequate for training and batch processing, is increasingly inadequate for global, real-time intelligent applications. This paper has argued through analysis, case studies, and experimentation that a cloud-native-powered “central training, edge inference” architecture represents the inevitable path forward for scalable AI deployment.

Our comprehensive literature review situated this work within ongoing research in cloud-edge computing and AI infrastructure. The architectural analysis revealed how hierarchical computing models, combined with cloud-native orchestration, address the latency, cost, and compliance limitations of centralized approaches. The practical experiment with KServe on Minikube demonstrated that cloud-native model serving provides tangible benefits in operational simplicity, portability, and built-in production capabilities, though not without implementation challenges.

The discussion highlighted significant trade-offs in consistency, resource efficiency, and management complexity that must be navigated. Economic models continue to evolve as the cost structure shifts from pure compute to include distributed infrastructure. Security and compliance considerations become more complex as the perimeter expands across geographic and organizational boundaries.

Looking forward, the vision of a truly intelligent computing continuum—where AI workloads flow seamlessly across centralized clouds, edge locations, and endpoint devices—remains aspirational but achievable. Realizing this vision requires advances in adaptive orchestration, federated learning, hardware abstraction, and ethical governance. As these technologies mature, distributed AI infrastructure will transition from specialized solution for latency-critical applications to

the default paradigm for AI deployment, ultimately making intelligent capabilities as ubiquitous and reliable as electricity—available wherever and whenever needed, with just the right balance of performance, efficiency, and trust.

References

1. Armbrust, M., Fox, A., Griffith, R., Joseph, A.D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., et al.: A view of cloud computing. *Communications of the ACM* **53**(4), 50–58 (2010)
2. BentoML Team: Bentoml: Build production-grade ai applications. <https://www.bentoml.com/> (2020), accessed: 2023-10-01
3. Bernstein, D.: Containers and cloud: From lxc to docker to kubernetes. *IEEE Cloud Computing* **1**(3), 81–84 (2014)
4. Brogi, A., Forti, S., Ibrahim, A.: Computational continuum: A paradigm for the age of digital ecosystems. *IEEE Internet Computing* **25**(3), 15–22 (2021)
5. Crankshaw, D., Wang, X., Zhou, G., Franklin, M.J., Gonzalez, J.E., Stoica, I.: Clipper: A low-latency online prediction serving system. In: 14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17). pp. 613–627 (2017)
6. ETSI: Multi-access edge computing (mec); framework and reference architecture (2020)
7. Hao, T., Huang, Y., Zhang, X., Wang, L., Wang, D., Luo, X., Zhou, H.: Edge ai: On-demand accelerating deep neural network inference via edge computing. *IEEE Transactions on Wireless Communications* **19**(1), 447–457 (2020)
8. Jeon, M., Venkataraman, S., Phanishayee, A., Qian, J., Xiao, W., Yang, F.: Analysis of large-scale multi-tenant gpu clusters for dnn training workloads. In: 2019 USENIX Annual Technical Conference (USENIX ATC 19). pp. 947–960 (2019)
9. KServe Authors: Kserve: Standardized serverless inference on kubernetes. <https://kserve.github.io/website/> (2021), accessed: 2023-10-01
10. Moritz, P., Nishihara, R., Wang, S., Tumanov, A., Liaw, R., Liang, E., Elibol, M., Yang, Z., Paul, W., Jordan, M.I., et al.: Ray: A distributed framework for emerging AI applications. In: 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18). pp. 561–577 (2018)
11. NVIDIA: Nvidia dgx superpod: Scalable infrastructure for ai leadership. <https://www.nvidia.com/en-us/data-center/dgx-superpod/> (2020), accessed: 2023-10-01
12. NVIDIA: Nvidia ceo predicts ai inference workloads to be “a billion times” larger than training. <https://www.nvidia.com/en-us/> (2023), accessed: 2023-10-01
13. Olston, C., Fiedel, N., Gorovoy, K., Harmsen, J., Lao, L., Li, F., Rajashekhar, V., Ramesh, S., Soyke, J.: Tensorflow-serving: Flexible, high-performance ml serving. In: Workshop on ML Systems at NIPS. vol. 1, pp. 1–4 (2017)
14. OpenAI: Gpt-4 technical report. <https://cdn.openai.com/papers/gpt-4.pdf> (2023), accessed: 2023-10-01
15. OpenFog Consortium: Openfog reference architecture for fog computing. Tech. rep., OpenFog Consortium (2017)
16. Satyanarayanan, M.: The emergence of edge computing. *Computer* **50**(1), 30–39 (2017)
17. Seldon Technologies: Seldon core: Machine learning deployment for kubernetes. <https://www.seldon.io/> (2019), accessed: 2023-10-01

18. Shi, W., Cao, J., Zhang, Q., Li, Y., Xu, L.: Edge computing: Vision and challenges. *IEEE Internet of Things Journal* **3**(5), 637–646 (2016)
19. Xiao, W., Bhardwaj, R., Ramjee, R., Sivathanu, M., Kwatra, N., Han, Z., Patel, P., Peng, X., Zhao, H., Zhang, Q., et al.: Gpu sharing for deep learning workloads on kubernetes. In: Proceedings of the 11th ACM Symposium on Cloud Computing. pp. 519–534 (2020)