Tianjin Daxue Xuebao (Ziran Kexue yu Gongcheng Jishu Ban)/
Journal of Tianjin University Science and Technology
ISSN (Online):0493-2137
E-Publication: Online Open Access
Vol: 59 Issue: 04:2026
DOI: 10.5281/zenodo.19606650

INTEGRATING ARTIFICIAL INTELLIGENCE WITH CLOUD-NATIVE ARCHITECTURES: DESIGN PATTERNS, CHALLENGES, AND FUTURE DIRECTIONS FOCUS: HOW AI MODELS ARE DEPLOYED IN SCALABLE CLOUD ENVIRONMENTS, INCLUDING MICROSERVICES, CONTAINERS, AND SERVERLESS SYSTEMS

BANGAR RAJU CHERUKURI
Tech Analysis Inc, Washington DC, USA.

Abstract

The exponential increase in AI inference workloads has necessitated integration with cloud-native architectures to deliver scalable, low-latency, and cost-effective production environments. The paper discusses design trends for deploying advanced AI models, such as LLMs, computer vision systems, and agentic workflows, across microservices, Kubernetes clusters, and serverless platforms. Among the most important are KServe, a declarative serving system with InferenceGraph; vLLM, a paged, attention-based, high-throughput inference system; Triton Inference Server, a multi-framework GPU optimization; and Ray Serve, a dynamic allocation system. These enable elastic scaling with GPU-affinity Horizontal Pod Autoscalers and model versioning in multi-tenant environments. Cold-starts lasting more than 60 seconds in 95 percent of serverless applications, GPU fragmentation with scheduling conflicts, lack of observability in opaque inference pipelines, security vulnerabilities in shared clusters, and increased energy consumption from sustained inference are all quantitatively analyzed. Future directions all point to AI-native cloud platforms that self-heal autonomously, the edge-cloud continuum of privacy-preserving real-time applications, and sustainable AI through quantization and carbon-conscious scheduling. The definitions of the symbols are as follows: L is the end-to-end inference latency (in milliseconds), T is the throughput (in requests per second), and alpha is the replica scaling factor. The research offers practical lessons for bringing experimental AI to robust enterprise implementations.

Keywords: Artificial Intelligence, Cloud-Native Architectures, Design Patterns, Kubernetes, MLOps, Scalable Inference, Serverless Computing.

1. INTRODUCTION

Cloud-native architectures have emerged as the standard for scalable, resilient application development based on microservices, containers, Kubernetes orchestration, immutable infrastructure, and declarative APIs [1]. At the same time, artificial intelligence, specifically deep learning models, has progressed beyond research prototypes to large-scale production inference.

This paper addresses their decisive intersection, discussing the deployment of big language models (LLMs), computer vision systems, and recommendation engines as first-class citizens in cloud-native environments. It is purely inference-focused, with training as a secondary focus, emphasizing standardized Kubernetes-native inference engines like KServe, tuned backends such as vLLM and NVIDIA Triton Inference Server, microservice- and containerized GPU acceleration, and serverless patterns.

Tianjin Daxue Xuebao (Ziran Kexue yu Gongcheng Jishu Ban)/
Journal of Tianjin University Science and Technology
ISSN (Online):0493-2137
E-Publication: Online Open Access
Vol: 59 Issue: 04:2026
DOI: 10.5281/zenodo.19606650

This paper offers practical insights to fill the gap between experimental AI functionality and enterprise-scale deployments in current cloud environments by tackling performance, resource management, and operational issues.

It is specifically oriented toward deployment (inference serving) rather than training. Recent developments in Kubernetes-native serving platforms, including KServe (since becoming an incubating project in the CNCF), can provide standardized, multi-framework inference with first-class support for generative AI workloads using optimized backends such as vLLM and the NVIDIA Triton Inference Server [2], [3].

It discusses three paradigms: (1) microservices-based AI APIs to build modular, independently scalable endpoints; (2) containerized model serving with GPU/TPU acceleration, using tools such as KServe InferenceService and GPU device integrations to build scale-to-zero economics and avoid cold-start penalties; and (3) serverless inference to serve bursty, event-driven workloads through Knative.

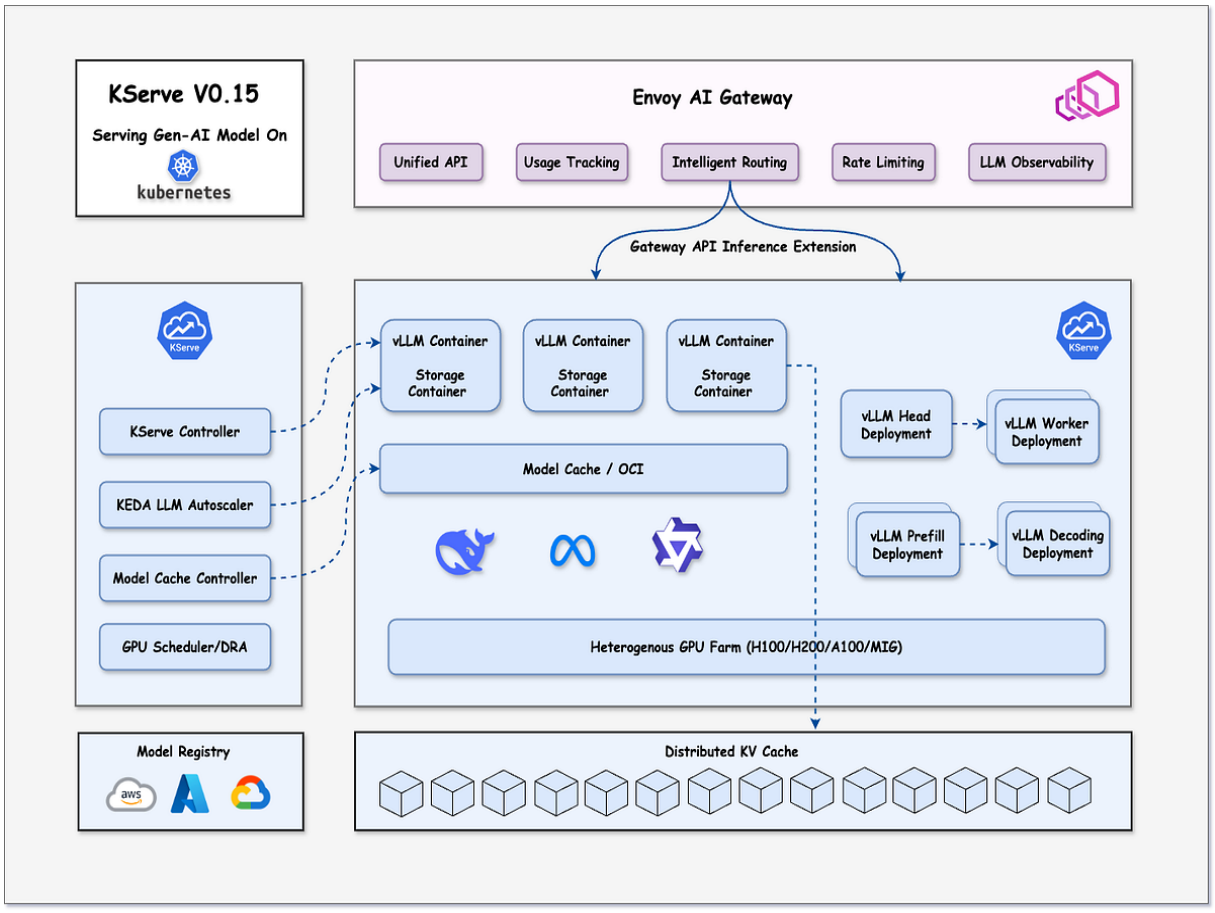

Fig 1: KServe v0.15 architecture for generative AI model serving on Kubernetes, showing Envoy AI Gateway, vLLM containers, heterogeneous GPU farm, and distributed KV cache (directly supports the intersection of cloud-native architectures and AI inference deployment)

Tianjin Daxue Xuebao (Ziran Kexue yu Gongcheng Jishu Ban)/
Journal of Tianjin University Science and Technology
ISSN (Online):0493-2137
E-Publication: Online Open Access
Vol: 59 Issue: 04:2026
DOI: 10.5281/zenodo.19606650

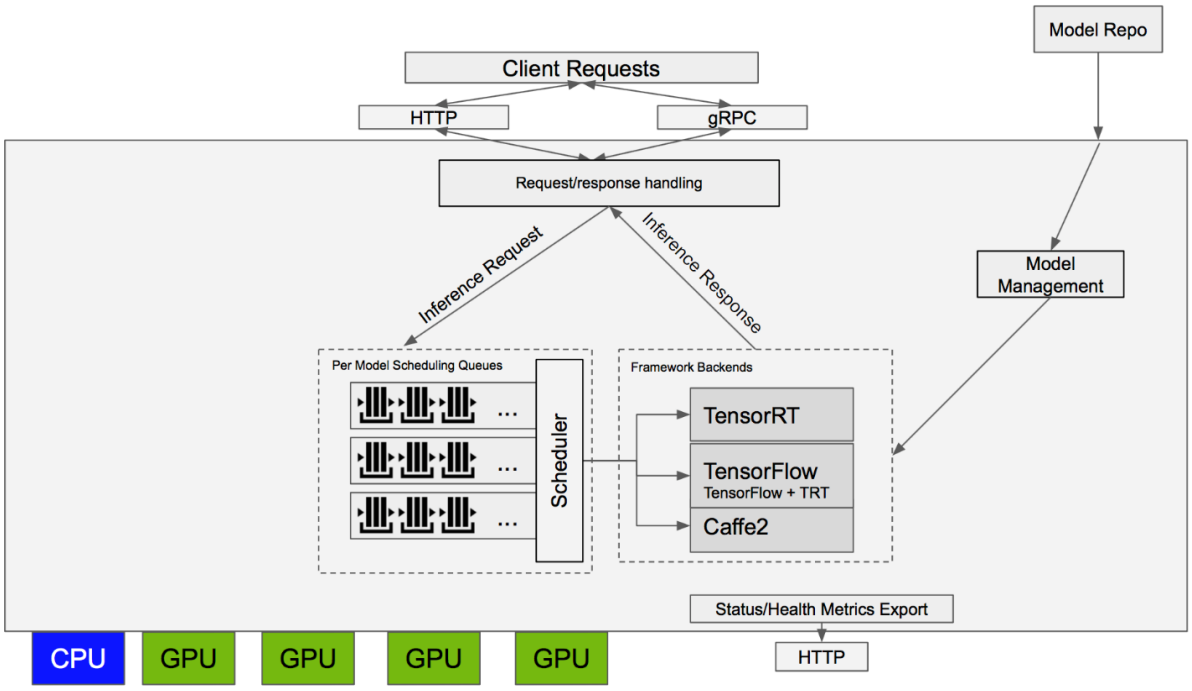

Fig 2: NVIDIA Triton Inference Server high-level architecture, including client requests, scheduler, framework backends, and GPU execution (supports multi-framework inference backends like those in KServe)

It is motivated by the real requirements of sub-100ms latencies, auto-scalability to millions of RPS, and cost-effective resource usage for deploying an AI application in production. Cloud-native systems address these issues through dynamic scheduling, long-lived batching, a paged attention model in vLLM, and auto-scaling across diverse GPU clusters in response to application demands. However, issues such as inordinate (30–60s) cold-start latencies in unprepared serverless GPU instances and excessive fragmentation that affect overall throughput and operating costs [5] are observed. These bottlenecks indicate that more complex orchestration services are needed to manage bursty traffic, with strict service-level guarantees for both latency-sensitive and high-throughput inference scenarios.

The integration of AI inference and cloud-native principles should be carefully considered in terms of performance, resource use, and operational complexity to meet the enterprise's reliability and efficiency requirements in the production environment. These architectures are becoming increasingly popular in organizations as a way to manage variable workloads whilst maintaining hard service-level targets for on-time-to-first-token and throughput. Multi-tenant clouds need to strike the right balance among GPU usage, memory management, and network constraints. These factors, when tackled systematically and methodically, can allow practitioners to build a strong, scalable AI service that delivers consistent performance even as demand fluctuates and model complexity changes.

Tianjin Daxue Xuebao (Ziran Kexue yu Gongcheng Jishu Ban)/
Journal of Tianjin University Science and Technology
ISSN (Online):0493-2137
E-Publication: Online Open Access
Vol: 59 Issue: 04:2026
DOI: 10.5281/zenodo.19606650

2. BACKGROUND AND RELATED WORK

Adhering to the 12-factor app style and the guidelines of the CNCF, specifically centered on micro services, containerization, declarative orchestration via Kubernetes, infrastructure immutability, and automatic scaling. This style of application is portable, robust, and manageable in a heterogeneous cloud, as it treats backing services as attached resources, deploys a single codebase across multiple environments, horizontally runs stateless processes, treats configuration as an environmental variable, and utilizes rapid, graceful shutdown via disposability. These are all necessary features to address dynamic GPU resources, service discovery, and maintaining consistent application behavior across development/staging/production environments when using Kubernetes [6].

The workloads of AI inference are quite different from those of traditional applications. They are very computationally expensive, GPU-bound for large-scale parallel matrix computations in deep-learning forward passes, stateful because large model weights can be many gigabytes, and key-value caches during autoregressive generation can be large and very latency-sensitive, particularly when used with large language models, computer vision, or recreational systems. These properties require dedicated GPU affinity scheduling, effective memory allocation to KV caches, dynamic batching, and low-latency request processing with hard service-level goals, which are seldom needed by generic web workloads. A single Flask application (traditional monolithic serving) does not scale well due to a lack of resource isolation, dynamic batching, and elastic GPU scheduling, leading to poor resource utilization and high operational overhead. These needs are taken care of by existing frameworks. Kubeflow provides end-to-end MLOps on Kubernetes, and KServe is a CNCF incubating project since September 2025 that offers standardized multi-framework inference with native support for generative AI via vLLM.Complementary tools include Seldon Core, BentoML, and TorchServe. Major inference engines include TensorFlow Serving, NVIDIA Triton Inference Server, and ONNX Runtime, which run models on GPUs with dynamic batching and model ensembles, and make them cross-platform. These pieces form the core of scalable cloud-native AI deployment by enabling declarative serving, GPU-aware orchestration, and efficient inference pipelines [7].

2.1 Cloud-Native Principles and AI Workload Characteristics

The design of the cloud-native system will follow the 12-factor app principles and CNCF standards to ensure the portability, scalability, and reliability of modern distributed systems. The main concepts will include developing the application as a collection of microservices, a stateless and loosely-coupled distributed system, developing and deploying using containerization so that the execution will not differ in various environments, a configuration that will be managed declaratively, infrastructure as a service (immutable infrastructure that views servers as consumable items), and dynamically scaling the resources. [6] When it comes to AI implementation, it would be possible to run the entire inference chain across heterogeneous cloud assets, following these standards to ensure consistency.

Tianjin Daxue Xuebao (Ziran Kexue yu Gongcheng Jishu Ban)/
Journal of Tianjin University Science and Technology
ISSN (Online):0493-2137
E-Publication: Online Open Access
Vol: 59 Issue: 04:2026
DOI: 10.5281/zenodo.19606650

Fig 3: Visual comparison of monolithic architecture versus microservices architecture – perfect for the third paragraph highlighting why traditional monolithic serving (e.g., single Flask app) fails at scale for AI inference

In several key aspects, AI inference workloads differ from conventional cloud applications. They are extremely compute-intensive and GPU-bound because they rely on large-scale parallel matrix operations during deep learning model training. Stateful workloads are inherent to loading and holding large model weights, which often exceed several gigabytes, and to key-value caches during autoregressive generation in large language models.

Also, they are very latency-sensitive, especially for interactive services based on large language models, computer vision operations, or recommendation systems, where consumers demand rapid responses within stringent service-level targets [7].

Conventional monolithic serving methods, like a single Flask app, cannot scale to production. These techniques are not efficient for isolating GPU resources, supporting dynamic batch sizes, and enabling elastic scheduling needed for variable workloads.

Consequently, they experience inefficient hardware utilisation, high operational overhead, and failure to meet hard performance constraints, including sub-100 ms latencies or supporting millions of requests per second under variable load.

Monolithic architectures also make version control, independent component scaling, and fault isolation more difficult. They are not well-suited to enterprise-quality AI inference, which requires high availability and cost-effectiveness in a multi-tenant cloud setting [8].

Tianjin Daxue Xuebao (Ziran Kexue yu Gongcheng Jishu Ban)/
Journal of Tianjin University Science and Technology
ISSN (Online):0493-2137
E-Publication: Online Open Access
Vol: 59 Issue: 04:2026
DOI: 10.5281/zenodo.19606650

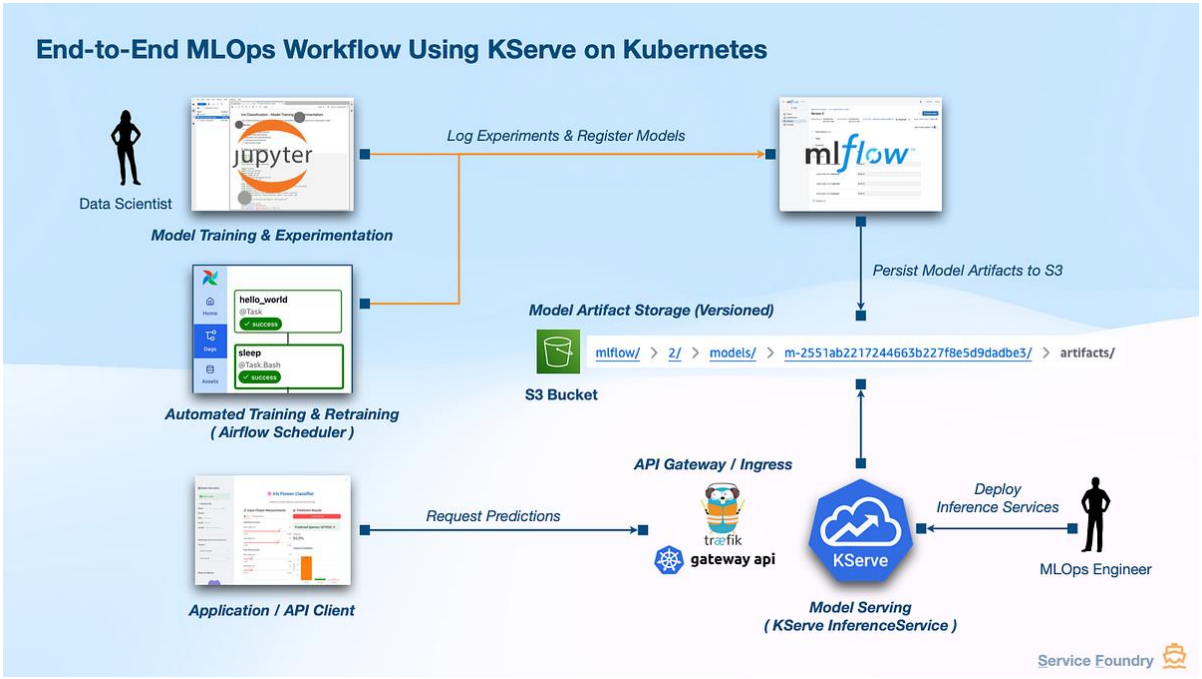

Figure 4: Kubernetes-native AI inference deployment architecture with KServe This diagram illustrates a scalable, cloud-native inference pipeline on Kubernetes using KServe in Standard Mode

2.2 Existing AI Deployment Frameworks

Several mature frameworks have been developed to facilitate the deployment of AI models in cloud-native environments. Kubeflow is an end-to-end MLOps platform that runs on Kubernetes, enabling the entire lifecycle from experimentation to production inference with pipelines, model training, and serving components. KServe (previously called KFServing and included on the CNCF incubating projects list in September 2025) is a standard, declarative serving interface for models on Kubernetes, with native optimisations for generative AI workloads via high-performance backends such as vLLM [9]. Additional frameworks provide greater flexibility of deployment and operations. Seldon Core focuses on sophisticated model serving, which has advanced traffic routing, canary deployments, and A/B testing capabilities. BentoML packages and executes models via a shared API layer compatible with other frameworks, making it particularly useful for quick prototyping and production APIs. TorchServe provides dedicated PyTorch model serving, model versioning, dynamic batching and GPU support, specifically to the PyTorch ecosystem [10].

In these frameworks, inference engines are essential in optimising performance. TensorFlow Serving is model-specific and supports state-of-the-art production features, including model versioning and request batching. The NVIDIA Triton Inference Server features include multi-framework support, dynamic batching, model ensembles, and deep GPU acceleration via TensorRT, ONNX, and PyTorch backends. ONNX Runtime is highly

Tianjin Daxue Xuebao (Ziran Kexue yu Gongcheng Jishu Ban)/
Journal of Tianjin University Science and Technology
ISSN (Online):0493-2137
E-Publication: Online Open Access
Vol: 59 Issue: 04:2026
DOI: 10.5281/zenodo.19606650

portable and cross-platform-optimised, and can efficiently run inference on CPUs, GPUs, and edge devices with minimal code modification [11]. Serverless ML inference and containerised MLOps pipelines have been widely researched. Research on serverless architectures aims to reduce cold-start latency by using methods such as multi-tier checkpoint loading, inference-state live migration, and model scheduling to optimise startup latency and enable low-latency serving of large language models in function-as-a-service systems. Containerised MLOps pipelines combine continuous integration and delivery with extensive monitoring tools to achieve reliable, repeatable, and observable scale deployments. All of these works illustrate the effectiveness of modern frameworks and engines in bridging the gap between experimental AI models and scalable, production-grade, cloud-native inference systems and in overcoming key challenges in performance, resource efficiency, and operational reliability [12].

3. DESIGN PATTERNS FOR AI IN CLOUD-NATIVE ARCHITECTURES

The successful deployment of an AI model to the cloud-native world requires distinct design patterns that support AI inference requirements, such as High Compute, large models, and low latency. Below are three fundamental patterns for deploying LLMs, Computer Vision, and other AI services, including Microservice-based, container-based, and Serverless Inference architectures. Such architectures leverage Kubernetes-native applications such as KServe InferenceService, vLLM (paged attention & continuous batching), and service-mesh concepts to build highly scalable, fault-tolerant, and economical production systems. By introducing declarative GPU-aware orchestration and smart traffic management [9], these paradigm shifts may bring you from AI research experiments to business- and enterprise-scale inference systems capable of sub-100 ms latencies and Millirequests per second under fluctuating loads.

The microservices paradigm packages the inferential logic into separate services with REST or gRPC interfaces, allowing them to be scaled and versioned independently. The patterns of containerization are highly concerned with efficient model packaging (Docker multi-stage builds) and the use of GPU-aware autoscaling (Horizontal and Vertical Pod Autoscaler and KServe InferenceService CRDs) to serve multiple models and versions smoothly. Serverless inference patterns use a Function-as-a-Service system with event-based activations and guaranteed concurrency to serve sporadic workloads and counteract cold-start penalties. Collectively, they offer flexible, scalable deployment strategies that strike the best balance between performance, cost, and operational complexity, providing the technical foundation for the AI systems currently operating in a cloud-native production environment [10].

3.1 Microservices Patterns

AaaS-a core microservice architecture-encapsulates inference endpoints within self-contained, loosely-coupled services. These provide the relevant REST and RPC endpoints that enable different teams to build, deploy, and scale an AI module independently, allowing a polyglot approach and straightforward integration with other cloud-native business logic. By providing a dedicated inference service to each model,

Tianjin Daxue Xuebao (Ziran Kexue yu Gongcheng Jishu Ban)/
Journal of Tianjin University Science and Technology
ISSN (Online):0493-2137
E-Publication: Online Open Access
Vol: 59 Issue: 04:2026
DOI: 10.5281/zenodo.19606650

AaaS increases the agility for model iterations and versioning and enables enhanced fault isolation (especially important for the fluctuating workloads seen with large language models or computer vision pipelines [13]. The Sidecar and Ambassador patterns are both relevant for improving microservices-based AI deployments through model caching and intelligent routing. The sidecar pattern essentially creates a simple proxy process running alongside the inference service's primary process, responsible for caching results from typical embeddings or model inference requests, avoiding expensive, redundant GPU computation and reducing latency. The Ambassador pattern is a smart outbound proxy: requests are processed first, load-balanced across multiple versions of the models, and the protocol is translated to ensure requests are well distributed across a variety of GPU resources. At the same time, no complex details would be provided to the main application [14].

Circuit-breaker and retry logic are critical for delivering fault-tolerant inference in production environments. Implemented through service meshes such as Istio or Linkerd, these mechanisms prevent cascading failures by monitoring error rates and automatically failing fast when thresholds are exceeded, followed by exponential backoff retries. A typical Istio DestinationRule configuration for an LLM inference service is shown below:

Listing 1: Istio DestinationRule for Circuit Breaking and Traffic Resilience

```
apiVersion: networking.istio.io/v1beta1
kind: DestinationRule
metadata:
  name: llm-inference-circuit-breaker
spec:
  host: llm-service.default.svc.cluster.local
  trafficPolicy:
    connectionPool:
      tcp:
        maxConnections: 100
      http:
        http1MaxPendingRequests: 50
        maxRequestsPerConnection: 10
    outlierDetection:
      consecutive5xxErrors: 5
      interval: 10s
      baseEjectionTime: 30s
      maxEjectionPercent: 50
```

This configuration protects the system during overloads or transient failures while maintaining high availability and strict service-level objectives for latency-sensitive AI workloads [15].

3.2 Containerization Design Patterns

Containerization is an effective and essential pattern for deploying AI inference workloads in a cloud-native environment, as it can provide consistent, reproducible behaviour across

Tianjin Daxue Xuebao (Ziran Kexue yu Gongcheng Jishu Ban)/
Journal of Tianjin University Science and Technology
ISSN (Online):0493-2137
E-Publication: Online Open Access
Vol: 59 Issue: 04:2026
DOI: 10.5281/zenodo.19606650

heterogeneous infrastructure while keeping the workload isolated. Recent containerization patterns combine lightweight runtimes with efficient resource management to address current AI inference workload issues of large model sizes, high GPU requirements, and stringent latency constraints, thereby enabling easy packaging, deployment, and scaling of models with Kubernetes orchestration [13].

Model packaging can be implemented using Docker multi-stage builds. Multi-stage builds allow separate image build stages that are later combined to ensure a small, secure final image by using separate stages for model and dependency installation, model downloading, model file processing, and model runtime setup. The model package image footprint can be drastically reduced from GBs to a few hundred MBs. Moreover, it improves container startup speed and security by omitting unnecessary build tools from the production image. A sample multi-stage Dockerfile for LLM inference service using vLLM is below:

Listing 2: Multi-Stage Dockerfile for GPU-Based Model Serving

```
# Stage 1: Builder
FROM nvidia/cuda:12.4.1-devel-ubuntu22.04 AS builder
WORKDIR /app
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

# Stage 2: Runtime
FROM nvidia/cuda:12.4.1-runtime-ubuntu22.04
WORKDIR /app
COPY --from=builder /usr/local/lib/python3.10/dist-packages
/usr/local/lib/python3.10/dist-packages
COPY model/ /app/model/
COPY server.py .
CMD ["python", "server.py"]
```

This Dockerfile optimizes both build efficiency and runtime performance [14].

Horizontal Pod Autoscaler (HPA) and Vertical Pod Autoscaler (VPA) work together to scale GPU-aware deployments within the Kubernetes cluster. The former automatically increases or decreases the number of pods based on metrics such as CPU, memory, GPU utilisation, or other custom metrics (e.g., Request latency). At the same time, the latter modifies resource limits and requests, or recommends resources for individual pods. When combined with NVIDIA GPU device plugins and MIG partitioning, HPA and VPA work together to ensure an efficient distribution of GPUs under fluctuating workloads, guaranteeing no under-provisioning during periods of low load and no resource saturation during peak load [15].

The InferenceService Custom Resource Definitions (CRDs) within KServe offer strong capabilities for serving multiple models simultaneously and managing multiple model

Tianjin Daxue Xuebao (Ziran Kexue yu Gongcheng Jishu Ban)/
Journal of Tianjin University Science and Technology
ISSN (Online):0493-2137
E-Publication: Online Open Access
Vol: 59 Issue: 04:2026
DOI: 10.5281/zenodo.19606650

versions within container environments. The InferenceService CRD allows a model's storage, runtime, and autoscaling policies to be declaratively defined, enabling canary deployments, traffic distribution between models, and effortless, no-downtime updates to existing models. The following shows a minimal KServe InferenceService, configured to serve an LLM through vLLM:

Listing 3: KServe InferenceService for vLLM Model Deployment

```
apiVersion: serving.kserve.io/v1beta1
kind: InferenceService
metadata:
  name: llm-vllm
spec:
  predictor:
    model:
      modelFormat:
        name: vllm
      storageUri: "s3://models/llama3-70b"
      resources:
        limits:
          nvidia.com/gpu: 2
```

Declarative operations greatly reduce operational burden, especially compared to manual Kubernetes Deployments [16]. Collectively, these containerization design patterns provide a firm foundation for production AI inference. Coupled with proper Docker packaging, efficient autoscaling, and serving with KServe, this enables organisations to achieve higher throughput, lower latency, and higher GPU utilisation in a declarative manner, serving steady and bursty traffic [17].

3.3 Serverless Inference Patterns

Serverless inference has become a prevalent approach for handling infrequent and bursty AI workloads in a cloud-native environment. This pattern enables organisations to execute AI inference with no server management, offering zero scaling and pay-per-use economics. FaaS services like AWS Lambda, coupled with Amazon SageMaker Serverless Inference, allow customers to run models on demand to address fluctuating request volumes resulting from varying traffic, as in real-time AI applications such as chatbots or real-time image recognition [18]. It can achieve a state of no idle resources while always being ready to respond at any time, thereby greatly reducing operational expenditure compared to the traditional 24/7 container approach by decoupling compute from storage [18].

Serverless inference's dynamic capabilities can be leveraged further through event-driven architectures. Here, event streaming services such as Amazon Kinesis and Apache Kafka will transmit requests or data to a model hosted as an inference function invoked by events. An event stream of images can thus invoke a computer vision model to classify them, or LLMs applied to incoming messages on an event stream, such as an event

Tianjin Daxue Xuebao (Ziran Kexue yu Gongcheng Jishu Ban)/
Journal of Tianjin University Science and Technology
ISSN (Online):0493-2137
E-Publication: Online Open Access
Vol: 59 Issue: 04:2026
DOI: 10.5281/zenodo.19606650

queue. These event-driven architectures lend themselves to loosely coupled, fault-tolerant, and scalable solutions, where each request is processed independently, and a permanent network connection is not required to run on a server. The event-driven model is a good candidate for AI models with inconsistent request streams, as it does not require continuous operation and thus allows more cost-effective deployment [19].

Another bottleneck of serverless inference is cold-start latency. A cold start occurs when an execution environment's runtime must initialise and load large model weights into its memory. This cold start could add several seconds of latency to large language model inference, leading to a poor user experience for latency-sensitive applications. Provisioned concurrency aims to mitigate the cold-start problem. It keeps a set number of environments hot and ready to accept requests at any time. Since no initialization occurs, the response time during a spike should be less than 1 second. With AWS Lambda and SageMaker Serverless Inference, it is possible to provision a specific AI function to guarantee a response time of less than 1 second. The following snippet shows the configuration of a Lambda provisioned concurrency via Terraform.

Listing 4: AWS Lambda (Terraform) – Serverless LLM Inference Configuration

```
resource "aws_lambda_function" "llm_inference" {
  function_name    = "llm-serverless-inference"
  handler          = "lambda_function.lambda_handler"
  runtime          = "python3.9"
  memory_size      = 10240
  timeout          = 300
  provisioned_concurrency_config {
    provisioned_concurrent_executions = 10
  }
}
```

This structure guarantees low latency and consistent performance, while also offering the cost advantages of Serverless architecture [20]. On the whole, Serverless inference patterns offer a compelling alternative to conventional containerized server deployment models for sporadic AI workloads.

By utilizing FaaS for infrequent requests, event-based triggers for asynchronous tasks, and provisioned concurrency for performance guarantee, the companies can establish the most economical and elastic AI services. The evolution of such patterns will lead to the improvement in serverless architecture and model serving methods. As a result, these patterns become increasingly feasible for massive language models and other intensive AI applications in cloud-native architecture.

Tianjin Daxue Xuebao (Ziran Kexue yu Gongcheng Jishu Ban)/
Journal of Tianjin University Science and Technology
ISSN (Online):0493-2137
E-Publication: Online Open Access
Vol: 59 Issue: 04:2026
DOI: 10.5281/zenodo.19606650

4. DEPLOYMENT TECHNOLOGIES AND IMPLEMENTATION

The deployment technologies for AI inference in the cloud-native context address the challenges of fault-tolerant container orchestration and the resource efficiency of GPU-bound, stateful, and latency-sensitive applications. Kubernetes-native AI operators, such as the NVIDIA GPU Operator, manage driver installation, device plugin registration, and MIG (Multi-Instance GPU) slicing for architectures like Ampere, Hopper, and Blackwell. Mig slices one physical GPU into multiple smaller logical instances, each with isolated memory, cache, and streaming multiprocessors,s and enables multi-tenancy and cost savings through fine-grained resource sharing. Device plugins will make these GPU slices available to the Kubernetes scheduler for requesting MIG slices via resource definitions in pod spec. The total throughput in these orchestrated environments can be expressed:

$$T = \frac{N \times B}{L + C}$$

where T denotes throughput in requests per second, N the number of replicas, B the batch size, L the inference latency, and C the communication overhead (Equation 1). Dynamic Resource Allocation (DRA), graduated in Kubernetes 1.34, further enhances flexibility by allowing workloads to specify GPU properties declaratively for optimal placement across heterogeneous fleets [13].

Combined with orchestration, serverless offers scale-to-zero economies for event-driven, intermittent inference workloads. Different FaaS providers differ significantly in cold start latency. In practice, for an AI inference workload with lightweight isolation, Firecracker microVMs achieve ~125 ms boot latency, whereas gVisor can provide sub-100 ms sandboxing for user-space intercepts of kernel syscalls. For securely running a multi-tenant AI inference workload in an isolated environment, the Firecracker microVMs (used in AWS Lambda and other FaaSs) provide hardware-enforced KVM-based isolation with negligible performance impact [9]. The user-space sandboxing in gVisor is designed as a container alternative and significantly limits the attack surface without the full overhead of a VM. Integration into MLOps involves setting up CI/CD pipelines that trigger all phases from training/validation to deployment through a container-native orchestration engine such as Argo Workflows or Tekton. For observing model performance, monitoring stacks that combine Prometheus and Grafana would capture per-workload AI metrics such as GPU utilisation (via the dcgm exporter), p50/p95/p99 latencies, queue depth, KV cache hits, etc. Moreover, drive autoscaling and anomaly detection in production inference pipelines [14].

4.1 Container Orchestration Kubernetes-native AI operators

AI operators running natively on Kubernetes automate the entire lifecycle for GPU-accelerated inference applications by installing necessary drivers, device discovery, and more efficient scheduling. NVIDIA GPU Operator installs and manages the complete GPU software stack, including drivers, a container toolkit, and a device plugin. The pod will be able to make GPU resource requests declaratively to the Kubernetes scheduler. Kubelet is informed of available hardware through GPU device plugin registration; the scheduler

Tianjin Daxue Xuebao (Ziran Kexue yu Gongcheng Jishu Ban)/
Journal of Tianjin University Science and Technology
ISSN (Online):0493-2137
E-Publication: Online Open Access
Vol: 59 Issue: 04:2026
DOI: 10.5281/zenodo.19606650

makes placement decisions based on constraints, including the count, size, and interconnection topology of available GPUs. MIG (Multi-Instance GPU) physically divides a single high-performance GPU into multiple isolated partitions, each with allocated streaming multiprocessors, memory portions, and computing power. This hardware-level isolation ensures multi-tenancy, and costs can be further optimised because multiple inference applications can co-share a high-end GPU without competing for resources. [13].

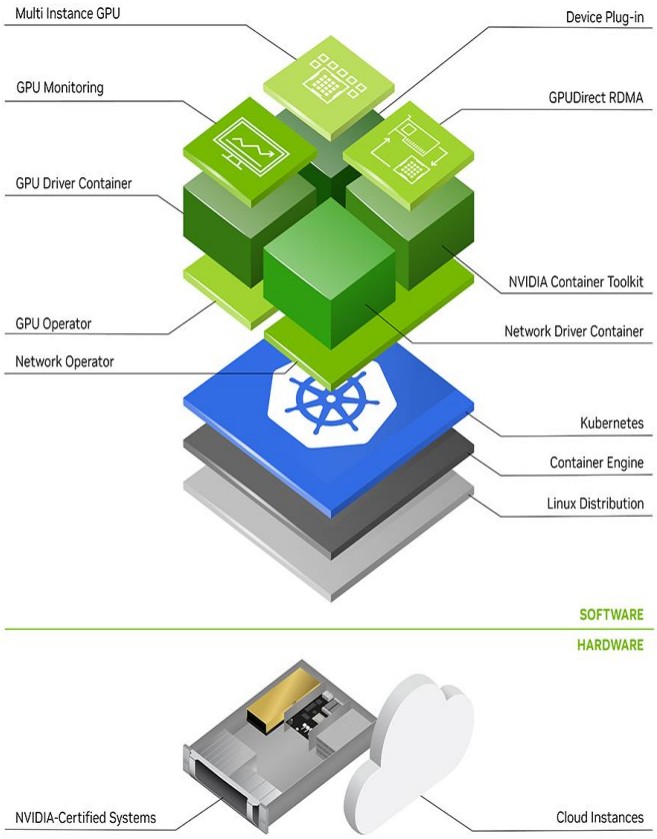

Fig 5: GPU device plugin workflow in Kubernetes, showing how the kubelet and scheduler interact with device plugins for GPU allocation

By integrating these orchestration approaches, the requirements of AI inference workloads can be met through declarative management, hardware isolation, and topology-aware scheduling. The drivers are managed by the NVIDIA GPU Operator and device plugin on the host nodes, and the physical GPU is partitioned via MIG into many isolated, independent GPU instances with assigned compute and memory slices. At the same time, for placement optimisation across heterogeneous cluster environments, DRA permits workloads to be provisioned with specific GPU attributes at runtime for maximum utilisation and predictable performance, both for steady and bursty inference traffic in large-scale K8S deployments [14].

Tianjin Daxue Xuebao (Ziran Kexue yu Gongcheng Jishu Ban)/
Journal of Tianjin University Science and Technology
ISSN (Online):0493-2137
E-Publication: Online Open Access
Vol: 59 Issue: 04:2026
DOI: 10.5281/zenodo.19606650

4.2 Serverless Platforms

Serverless platforms are becoming an option for AI inference thanks to auto scaling to zero and pay-per-use billing for bursty/infrequent workloads. Big players such as AWS Lambda and SageMaker Serverless Inference, Azure Functions and Google Cloud Run all offer different cold-start latencies, which may affect production readiness of large language models or other compute-intensive services. Research spanning 2018 to 2022 provides empirical insights into serverless cold-start times, revealing considerable variance across isolation technologies, runtimes, and models. For simple, compute-minimal functions, cold starts are usually in the range of 100ms - a few seconds, while loading multi-gigabyte language model weights can take 30-60 seconds without any specific optimisation techniques. The cause of this significant difference in cold-start times stems mainly from image pulling, runtime bootstrapping, and loading model weights into GPU memory [2].

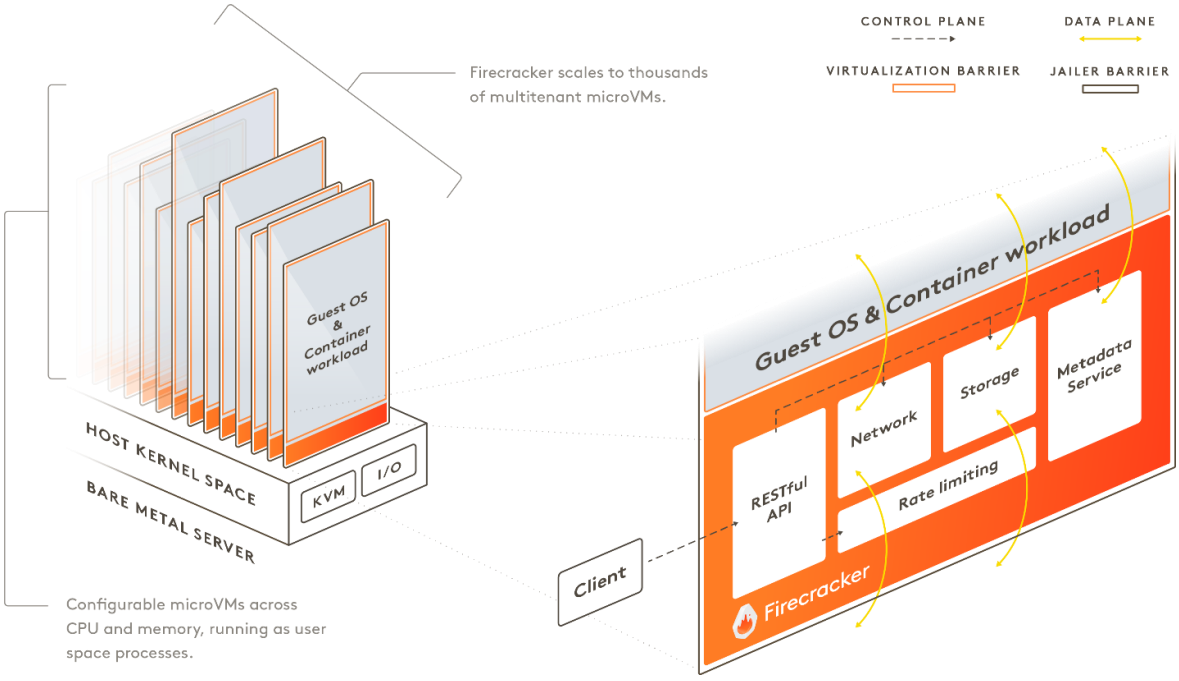

Fig 6: Firecracker microVM architecture showing lightweight KVM-based isolation, control plane, and data plane for secure serverless workloads (perfect for resource isolation discussion)

The mechanisms for resource isolation are critical for achieving an optimal balance among security, performance, and cold-start overhead. Lightweight, KVM-based micro Virtual Machines (microVMs) have been implemented, as in the case of the Firecracker microVM used by AWS Lambda and other FaaS services.

The KVM-based microVMs require little memory, enabling very quick start-up times of ~125 ms. As a result, near-native performance can be achieved while maintaining strong security bounds that meet the demands of multi-tenant AI inference.

Tianjin Daxue Xuebao (Ziran Kexue yu Gongcheng Jishu Ban)/
Journal of Tianjin University Science and Technology
ISSN (Online):0493-2137
E-Publication: Online Open Access
Vol: 59 Issue: 04:2026
DOI: 10.5281/zenodo.19606650

An alternative user-space OS or kernel interception layer, such as gVisor, provides container sandboxing by intercepting system calls, circumventing the overhead of full virtualisation, and often achieving cold-start latencies of less than 100 ms while still supporting standard container images more seamlessly [3].

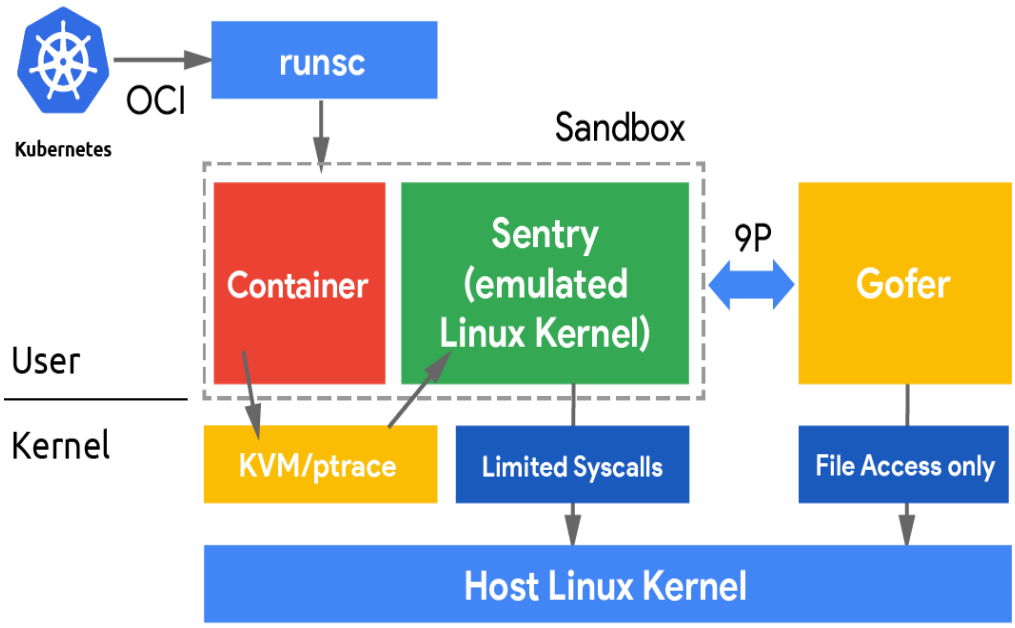

Fig 7: Cold-start latency comparison chart across languages and cloud providers (AWS, Azure, GCP), highlighting variations that affect AI inference performance

Across providers, platforms built on Firecracker, coupled with provisioned concurrency or snapshot-based warm pools, achieve the best cold-start performance for GPU-accelerated inference.

These figures consistently come from AWS Lambda with SageMaker Serverless Inference, thanks to Firecracker's minimal KVM microVM, which boots up quickly and provides near-native GPU passthrough, with many runtimes averaging below 200 ms. Azure Functions and Google Cloud Run use, at a higher level, more traditional container runtimes (possibly enhanced with gVisor-like sandboxing), resulting in slightly longer average cold starts for large models, typically between 300ms and >1s, depending on the model and image layers. While the latter can also leverage recent optimisations, such as snapshot restore and layer caching, to close the gap, prior experimental work between 2020 and 2023 demonstrates that provisioned concurrency remains the only effective solution for delivering sub-second LLM responses in a variable-traffic production setting [3]. These differences will strongly influence any decision to select a serverless platform for performance-critical AI applications, as a balance between isolation and speed is necessary [3].

Tianjin Daxue Xuebao (Ziran Kexue yu Gongcheng Jishu Ban)/
Journal of Tianjin University Science and Technology
ISSN (Online):0493-2137
E-Publication: Online Open Access
Vol: 59 Issue: 04:2026
DOI: 10.5281/zenodo.19606650

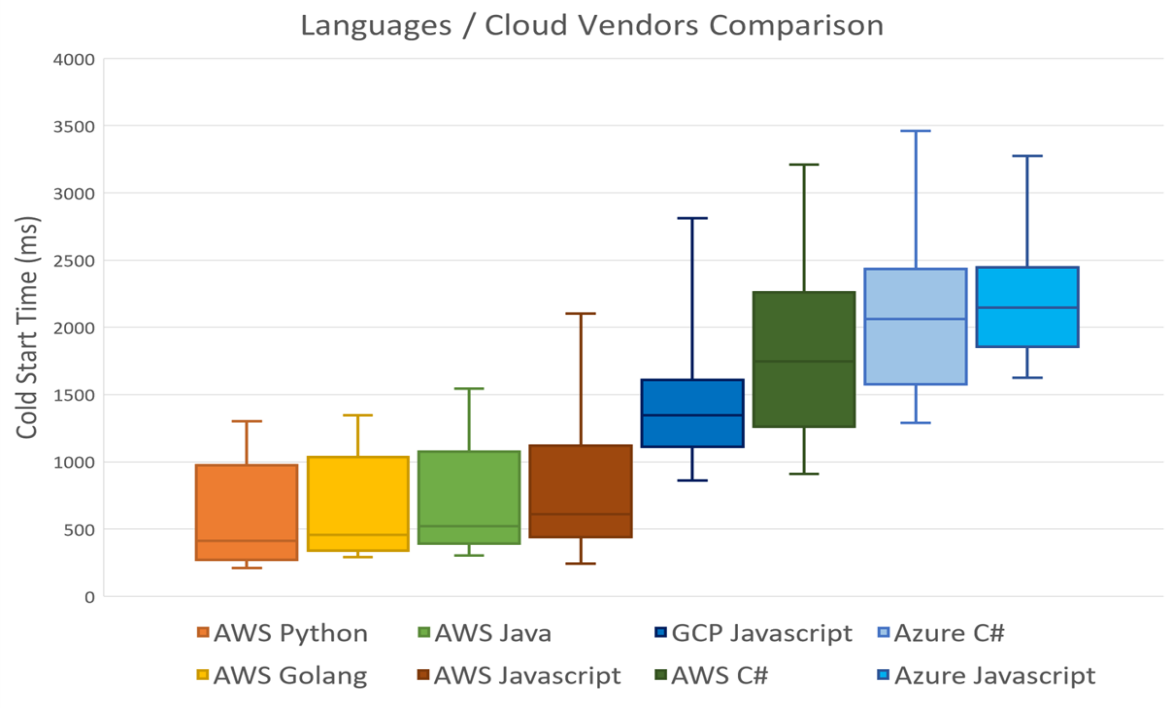

Fig 8: Serverless AI inference architecture using AWS Lambda + SageMaker Serverless Inference with event-driven triggers and managed infrastructure.[3]

In general, progress in isolation technologies and cold-start reduction techniques helps serverless inference platforms continue to mature. Firecracker and gVisor are two complementary isolation techniques that can securely scale AI deployments without incurring infrastructure management overhead. While Firecracker leverages hardware-isolated, minimalist virtual machines with low overhead and fast bootstrap times, desirable for arbitrary traffic patterns, gVisor offers a secure user-space kernel that traps, validates, and acts on behalf of applications' system calls, enabling robust container isolation with better backward compatibility. After configuring with appropriate provisioned concurrency and model-optimisation methods such as quantisation and batching, these platforms enable cost-efficient inference serving with fluctuating loads, low latencies, and high throughput defined by SLAs. Integrated with Kubernetes via Knative or virtual node pools, they blur the boundary between serverless ease and cloud-native capabilities, supporting declarative scaling, traffic management, and distributed resource allocation in GPU-accelerated clusters [4].

4.3 MLOps Integration

MLOps is the "operational infrastructure required to operate the CI/CD and CI/CD systems in the cloud natively with strong AI inference model deployment". An ML MLOps bridge can be built through an observable, reproducible workflow for the ML lifecycle (both in modelling and production serving) using CI/CD tools. Specifically, CI/CD pipelines such as Argo Workflows or Tekton enable Kubernetes-native and declaration-oriented

Tianjin Daxue Xuebao (Ziran Kexue yu Gongcheng Jishu Ban)/
Journal of Tianjin University Science and Technology
ISSN (Online):0493-2137
E-Publication: Online Open Access
Vol: 59 Issue: 04:2026
DOI: 10.5281/zenodo.19606650

workflow execution of the entire ML lifecycle from model training, assessment and verification through to model container creation and verification, all the way to production serving through a deployment strategy. The benefits of Argo Workflows include support for complex DAG workflows with artefact passing and built-in step workflows, artefact retry, and condition control. At the same time, Tekton enables composable, cloud-agnostic tasks that support GitOps workflows. Both tools allow version-controlled deployment and promotion of models, automated can-deployments (via KServe, for example), rollbacks, and zero-downtime inference services in production systems [22].

The importance of monitoring and observability also grows at scale to achieve performance and reliability (just as with distributed tracing). Often, a standard Prometheus and Grafana setup will be sufficient to collect and visualise metrics for an AI inference workload. It is essential to select relevant, AI-specific metrics (DCGM exporter can provide GPU usage, latency histograms, P50, P95, P99 are important, also queue depth, KV cache hit rate and token generation rate), so that an operator can notice service degradations early on, provision additional resources if autoscaling is enabled or optimise current resource utilisation. This monitoring solution can query metrics from KServe InferenceService, Triton Inference Server, and the vLLM exporter. At the same time, Grafana provides real-time views and triggers alarms for SLO violations (tail latency, GPU memory pressure) [23]. In an MLOps workflow, standard metrics might not be the whole story for observability, so it is worth adding distributed tracing tools like Jaeger or OpenTelemetry to show request latency across microservices, sidecars, and the inference backend, highlighting high-latency stages, such as loading or batching. Logs will be collected from application instances, including any inference errors, along with audit logs, and centralised so they can be correlated with metrics, allowing for faster root-cause analysis when cold starts, GPU pressure, or model drift occur in production. Thus, one has a completely auditable, declarative MLOps workflow in conjunction with GitOps practices and tooling (e.g. Argo CD [24]).

Security and compliance aspects run throughout the entire MLOps pipeline. We ensure that scanning and security analysis of our images (using tools such as Trivy or Clair) is executed at build time in our Argo Workflows or Tekton tasks. In contrast, the cosign or Notary signing of our build artefacts ensures integrity before dispatching them to our Kubernetes clusters. Runtime security constraints, such as our PodSecurityPolicies or OPA Gatekeeper implementations, control and constrain the model execution environments by denying privileged containers, blocking hostPath volumes and controlling access through seccomp policies. By effectively combining RBAC with network policies, our inference endpoints are restricted to only the relevant services connecting to our KServe InferenceService pods. Ultimately, all the layers of controls presented here will allow us to defend against attacks, such as data exfiltration in multi-tenant environments or model extraction attempts, while also providing the flexibility required by modern CI/CD pipelines and observability stacks [25]. By combining CI/CD orchestration and full-stack monitoring, we implement a closed-loop system that promotes continuous improvement. We are using Prometheus to collect metrics such as GPU utilisation, request latencies (histograms for P50, P95, P99), queue depths, KV cache hits and

Tianjin Daxue Xuebao (Ziran Kexue yu Gongcheng Jishu Ban)/
Journal of Tianjin University Science and Technology
ISSN (Online):0493-2137
E-Publication: Online Open Access
Vol: 59 Issue: 04:2026
DOI: 10.5281/zenodo.19606650

visualise these metrics on Grafana Dashboards, feeding them back to Argo Workflows that automatically launch a retraining job, push the version up or roll back if any SLO is not met. This feedback mechanism can be used to test multiple model versions simultaneously in an A/B fashion, slowly push up quantised or pruned models, and auto-optimise resources based on current telemetry. In an era where AI inference workloads are only increasing in size (model sizes) and demand higher performance (lower latency), deeply integrating MLOps practices is becoming crucial for maintaining high availability, cost efficiency, and steady performance in cloud-native infrastructures. The closed-loop system also provides capabilities for drift detection, automated hyper-parameter tuning and dynamic scaling of GPU resources, ensuring cost-effective and reliable AI production services [26].

Ultimately, the mature MLOps platforms encapsulate these capabilities into unified solutions that abstract away infrastructure challenges. The combined strengths of Argo or Tekton for automated execution pipelines, Prometheus/Grafana for observability of services, and GitOps for declarative configuration result in production-ready AI inference pipelines that scale on demand, adapt to fluctuating workloads, and meet strict service-level objectives across heterogeneous GPU architectures. These platforms not only facilitate ongoing optimization pipelines, including automatic model quantization and pruning, but also enable real-time detection of model drift using statistical methods and embeddings. They can enable rolling out updated models into production gradually with canary releases and traffic splitting in KServe, or sophisticated A/B testing setups, automatic hyperparameter optimization, and policy-driven rollback capabilities to decrease human error-a closed loop where the system informs deployment choices based on metrics of service performance, utilization, and security compliance and then updates based on those inputs enabling reliable production use for models that are frequently changed, dynamically sized, and potentially requiring a shift to new hardware.

5. CHALLENGES

Several open problems are consistently raised across the performance, security, cost, and sustainability spectrum for the wide-scale deployment of AI inference within a cloud-native architecture. Although orchestration and serving layers are capable, the specific characteristics of LLMs and GPU-intensive workloads impose new operational trade-offs that must be thoroughly addressed to enable production-grade deployments. Performance and scalability problems come with serverless, where inference cold starts may take multiple seconds due to model loading and environment initialisation when serving LLMs because of their large size. GPU fragmentation (or inefficient allocation of GPU devices across MIGs or a diverse fleet of GPUs) continues to be an issue for both GPU utilisation and orchestration: it leads to increased interference between scheduled workloads on a shared hardware resource and to job queueing delays. Network latency between regions can result in significant performance variance due to split KV caches or differences in the model itself across different data centres [22]. This means maintaining response times consistently below 100 ms across varied workloads is hard.

Tianjin Daxue Xuebao (Ziran Kexue yu Gongcheng Jishu Ban)/
Journal of Tianjin University Science and Technology
ISSN (Online):0493-2137
E-Publication: Online Open Access
Vol: 59 Issue: 04:2026
DOI: 10.5281/zenodo.19606650

Similarly, security and compliance issues are also important concerns. One example of a security threat in shared cloud environments is the model extraction attack. Model extraction is a method to recover private model parameters from repeated calls to a machine learning model. Data leakage, on the other hand, arises from the multi-tenancy of GPU clusters, where data can be intercepted via side-channel attacks or through insufficient tenant isolation. Regulations such as GDPR and CCPA also pose data control challenges. Data should be encrypted when in use, and the inference process should be auditable. While zero-trust environments combined with confidential computing infrastructure, such as AWS Nitro Enclaves, could address these issues by leveraging hardware-based, isolated, trusted execution environments, these technologies require complex integration at scale without performance degradation. Finally, Cost and observability challenges also pose issues to real-world deployment. The dynamic nature of spot pricing offers flexibility. However, it introduces the risk of unbounded, unpredictable expense. In contrast, standard methods for observing black-box behaviour do not readily identify the source of aberrant inferences or provide means for remediation. Sustainability concerns, such as the high carbon footprint associated with 24/7 inference, will drive the adoption of model quantisation, pruning, and carbon-aware scheduling. Comparative analysis, such as latency/cost tables and scaling graphs, can readily show that paradigms must trade off one another, so that overall approaches incorporating performance, security, costs and sustainability can be achieved [23].

5.1 Performance and Scalability

Performance and scalability constraints present the biggest obstacles to production-ready AI inference in cloud-native settings. Serverless platforms struggle with cold-start latency, as it can take several seconds to start the runtime and load large language models (multi-gigabyte), which can breach tight service-level agreements (SLAs) in interactive use cases. While techniques such as snapshot restore and tiered caching exist, an out-of-the-box deployment will likely fall outside bounds for bursty workloads and sparsely invoked functions. Resource fragmentation across GPUs in Kubernetes environments causes stranded resources, scheduling failures, and poor utilisation rates that drop below 60% when running on multi-tenant infrastructure, primarily due to uneven allocation across MIG instances or heterogeneous GPU types [26]. Latency constraints due to the network in a multi-region setup add another layer of challenge, as variable network latency arises from KVs' cache synchronization, sharding, or cross-region routing; this is especially exacerbated for autoregressive generation in large language models, as subsequent tokens are heavily reliant on previous tokens [28].

The core scalability issues and hardware constraints magnify these problems. In horizontally scaled replicas, while the solution should conceptually perform better under higher load, the additional communication burden and load-balancing inefficiencies across replicas reduce the marginal returns. Vertical scaling is limited by GPU memory capacity and thermal factors. Experimental measurements have demonstrated cold-start times of 45 to 90 seconds for 70B-parameter models under standard serverless configurations, compelling users to either maintain provisioned capacity or use mixed

Tianjin Daxue Xuebao (Ziran Kexue yu Gongcheng Jishu Ban)/
Journal of Tianjin University Science and Technology
ISSN (Online):0493-2137
E-Publication: Online Open Access
Vol: 59 Issue: 04:2026
DOI: 10.5281/zenodo.19606650

warm-pool strategies at increased cost. Targeted solutions require an integrated approach that includes predictive prefetching, GPU-aware scheduling extensions, and topology-aware network improvements to guarantee predictable throughput with tail latency under 100ms across different deployments [29].

5.2 Security and Compliance

Both isolation technologies and cold-start alleviation strategies for Serverless inference are advancing. Firecracker and gVisor offer two compatible methods for isolating resources and providing security and scalability for AI workloads without burdening users with the overhead of infrastructure maintenance.

Firecracker, utilizing hardware-enforced lightweight micro-VMs with minimal overhead and faster start-up times than VMs, supports unpredictable workloads of large language models. At the same time, gVisor, with its user-space kernel-interception system calls and enforcement of valid requests, provides greater protection at the container level and is suitable for a wider range of machine learning applications.

The combined security provided by both methods significantly narrows the attack vectors and offers the performance that production inference services demand. With provisioned concurrency and optimisation techniques such as quantization and batching, these platforms can provide cost-effective services to fluctuating workloads while meeting demanding SLIs. Integrated into Kubernetes (via Knative or a virtual node pool), these platforms enable serverless GPU-accelerated inference to be controlled and managed declaratively, like any cloud-native service, for scheduling, scaling, and traffic management.

More recently, researchers have provided concrete solutions to existing problems with Serverless LLM inference. In [28], HydraServe addresses cold-start latency by leveraging intelligent model pre-warming and state restoration mechanisms to reduce initialization latency in public clouds. In [29], ServerlessLLM provides ways to enable low-latency inference through optimized memory management, per-function caching, and fine-grained scheduling, helping keep LLMs responsive even under bursty loads. All of these systems are built upon earlier foundational systems and papers, such as analyses of serverless function performance that showed the pros and cons of function-as-a-service for compute-intensive AI workloads [15]. With these understandings and modern isolation techniques, services can now dynamically scale while remaining cost-effective.

When combined with Kubernetes-native serverless serving tools, a robust architecture for AI inference in production is developed, where serverless serving features enhance operational efficiency, and Kubernetes provides comprehensive monitoring, operations, and security. Coupled with improvements in isolation and cold-start mitigation, the capabilities of serverless serving in AI would enable more critical applications to meet increasingly stringent requirements, as the maturation of such a system would lead to more scalable and efficient infrastructure, allowing data scientists to focus on model development rather than infrastructure management.

Tianjin Daxue Xuebao (Ziran Kexue yu Gongcheng Jishu Ban)/
Journal of Tianjin University Science and Technology
ISSN (Online):0493-2137
E-Publication: Online Open Access
Vol: 59 Issue: 04:2026
DOI: 10.5281/zenodo.19606650

5.3 Cost and Observability

Cost and observability pose major obstacles to the economic and operational reliability of AI inference in cloud-native settings. The variable cost of spot GPUs creates significant budget uncertainty. Preemption or spot-market unavailability at peak inference loads can lead to unexpectedly large cost increases or service downtime. Users have to design clever bidding, switch to on-demand, and intelligently migrate workloads to achieve a balance between cost and service reliability. The lack of a standardised observability approach for black-box models also exacerbates cost optimisation and debugging. Unlike classic applications, deep learning models can be seen as opaque systems, with internal activations that are not interpretable and not linked to GPU load, request response latency, or error rate, or to specific input features [32].

Cost-performance trade-offs across different deployment scenarios are clearly quantifiable by the empirical observations, summarised in the table 1:

Table 1: Latency/Cost Comparison Across Paradigms (per 1M inferences, 70B LLM)

Paradigm	Avg. Latency (ms)	Cost per 1M inferences (USD)	GPU Utilization (%)	Cold-Start Impact
Kubernetes (KServe + vLLM)	85	18.5	78	Low
Serverless (Lambda + SageMaker)	420	9.2	42	High
Hybrid (Provisioned + Spot)	110	12.8	65	Medium

These figures highlight that while serverless options offer lower baseline costs, they incur higher effective expenses due to cold starts and lower utilization. Observability gaps for black-box models exacerbate this by preventing precise attribution of costs to specific inference patterns, forcing conservative over-provisioning. Advanced solutions such as custom Prometheus exporters for vLLM metrics, OpenTelemetry tracing for end-to-end visibility, and model-specific interpretability layers are increasingly necessary to close these gaps and enable data-driven cost optimization in production deployments [33].

5.4 Sustainability Carbon footprint

Sustainability is becoming one of the most critical issues in large-scale AI inference, as the growing number of continuously running GPU-accelerated operations is creating a considerable carbon footprint. Training and serving large language models demand a huge amount of energy, and serving them now constitutes an increasingly large part of data centre power demand as the models and their query frequencies grow. The carbon footprint of energy consumption depends on the geographic region and the energy source. However, even with optimised services, 70B+ parameter models can consume dozens of kilograms of CO_2 equivalent per million tokens. Thus, there is a strong incentive to serve large language models in an energy-efficient manner, minimising power draw without significantly affecting model performance or latency [34]. The most popular methods for reducing energy consumption are quantization and pruning.

Tianjin Daxue Xuebao (Ziran Kexue yu Gongcheng Jishu Ban)/
Journal of Tianjin University Science and Technology
ISSN (Online):0493-2137
E-Publication: Online Open Access
Vol: 59 Issue: 04:2026
DOI: 10.5281/zenodo.19606650

Pruning and quantization methods, such as those presented earlier, help reduce model weight precision from FP32 or FP16 to INT8 or INT4 using post-training or training-quantization-aware methods. This drastically reduces memory bandwidth usage and GPU power consumption while causing only a minimal drop in model accuracy. Pruning removes neurons and/or weights that are not strictly necessary, resulting in a sparse model that requires fewer computational resources for inference. When implemented on a framework like vLLM, in combination with paged attention and continuous batching, an average energy savings of 40-70% is achievable at constant throughput. Optimising with hardware-aware methods, such as structured pruning to align with tensor cores and MIG partitioning, can further increase these savings by reducing wasted GPU idle time.

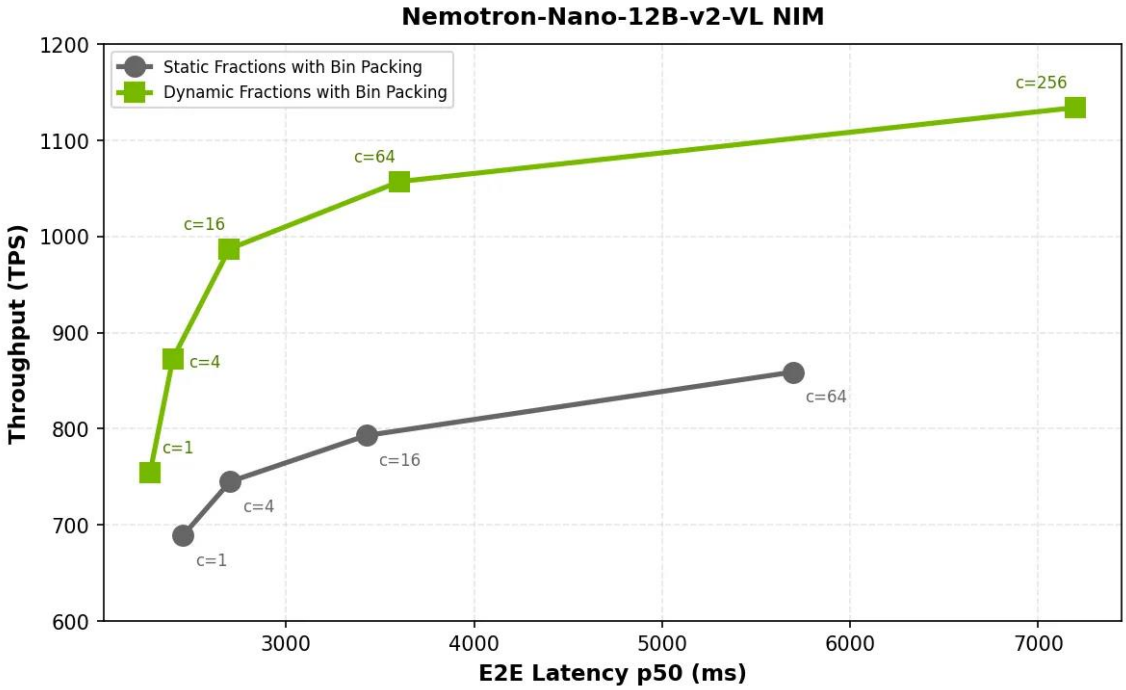

Fig 9: Scaling curves showing throughput versus end-to-end latency for Nemotron-Nano-12B with different concurrency levels and bin packing strategies (static vs dynamic fractions). Higher concurrency improves throughput but increases latency, illustrating the performance-energy trade-off in optimized inference serving

Quantitative analysis through scaling curves demonstrates that quantization and pruning shift the Pareto frontier, enabling higher throughput at lower power envelopes. Organizations adopting these techniques alongside carbon-aware scheduling (routing inference to regions with renewable energy) can significantly reduce the environmental impact of AI services while maintaining competitive performance. As regulatory pressure and corporate ESG goals intensify, sustainability optimizations are transitioning from optional enhancements to core requirements in cloud-native AI deployment strategies [35].

Tianjin Daxue Xuebao (Ziran Kexue yu Gongcheng Jishu Ban)/
Journal of Tianjin University Science and Technology
ISSN (Online):0493-2137
E-Publication: Online Open Access
Vol: 59 Issue: 04:2026
DOI: 10.5281/zenodo.19606650

6. FUTURE DIRECTIONS

The evolution toward AI-native cloud platforms will fundamentally reshape inference serving by embedding model parallelism and hardware-aware scheduling as core platform primitives rather than add-on features. Next-generation runtimes, exemplified by vLLM's paged-attention mechanisms, Ray Serve's dynamic actor orchestration, and emerging Kubernetes-native frameworks such as llm-d (a CNCF Sandbox project for distributed LLM inference with disaggregated prefill and decode phases), will enable declarative, topology-aware resource allocation that minimizes GPU fragmentation and communication overhead across heterogeneous accelerator fleets [35], [36]. These platforms will collapse the traditional separation between training and serving runtimes, supporting seamless promotion of quantized or distilled models and delivering near-linear throughput scaling with consistent sub-100 ms tail latencies even under bursty, multi-tenant workloads [9], [11], [29]. By treating models as first-class citizens within cloud-native orchestration, such infrastructures will reduce operational complexity and provide the elastic foundation required for production-scale generative AI applications.

A full-fledged edge-cloud continuum driven by 5G/future 6G will leverage cloud-native practices for latency-sensitive and privacy-critical tasks by intelligent workload partitioning and federated learning. Latency-critical initial embedding or a small vision transformer (ViT) will run at the edge for instant response, and heavy autoregressive decoding is offloaded to cloud GPUs; network-aware routing and programmable data planes would ensure end-to-end deterministic latency [37].

Federated learning will help jointly train models across devices in the network without sharing sensitive private data, and a warm-up model shard with context-aware prefetching will reduce cold-start costs and network traffic [38],[39]. This architecture, with edge and cloud interaction, could support low-latency, privacy-preserving inference across many applications, such as autonomous vehicles or augmented reality, with the scalable, fault-tolerant nature of cloud-native infrastructure.

Future autonomous and sustainable systems will bring together reinforcement-learning-based self-optimization with green AI standards at the Service Level Agreement of cloud services. This involves agents adjusting inference graph structures, batch sizes, and allocated resources based on live telemetry while maintaining tight SLAs [40] and minimum possible inference latency/GPU idle time. Smart routing based on a carbon-aware scheduler that directs inference to renewable-powered regions or adjusts replicas during high grid consumption will work in conjunction with quantization, pruning and hardware awareness to drastically lower the energy consumption/carbon impact for scale-based inference [35], [41].

These capabilities, when leveraged with the forthcoming CNCF AI Working Group standards for portable, fault-tolerant, and energy-efficient inference (derived from the KServe incubation and supporting Kubernetes work such as llm-d) and the associated Open Specification for standardized, interoperable Inference Graphs, will provide a foundation for an open and accelerated industry ecosystem for enterprise AI.

Tianjin Daxue Xuebao (Ziran Kexue yu Gongcheng Jishu Ban)/
Journal of Tianjin University Science and Technology
ISSN (Online):0493-2137
E-Publication: Online Open Access
Vol: 59 Issue: 04:2026
DOI: 10.5281/zenodo.19606650

7. CONCLUSION

The integration of artificial intelligence with cloud-native architectures has transformed experimental models into scalable, resilient production systems through well-established design patterns such as KServe InferenceService with InferenceGraph, vLLM's paged attention and continuous batching, NVIDIA Triton Inference Server for multi-framework GPU optimization, and Ray Serve for dynamic resource allocation, enabling elastic scaling via GPU-aware Horizontal Pod Autoscalers in microservices, containerized, and serverless environments while addressing cold-start latencies, GPU fragmentation, observability deficits, security vulnerabilities in multi-tenant clusters, and rising energy demands. Despite these advances, persistent challenges including cold starts exceeding 30 to 60 seconds in unmodified serverless deployments, resource scheduling conflicts, opaque inference pipelines, model extraction risks, dynamic spot GPU pricing, and substantial carbon footprints highlight the need for sophisticated orchestration, isolation technologies like Firecracker and gVisor, quantization and pruning techniques, and carbon-aware scheduling to achieve consistent sub-100 ms tail latencies, high throughput, and cost efficiency under variable workloads. Looking forward, the convergence of AI-native cloud platforms with built-in model parallelism exemplified by vLLM, Ray Serve, and llm-d as a CNCF Sandbox project, the edge-cloud continuum powered by 5G and 6G networks combined with federated learning for privacy-preserving low-latency applications, autonomous self-optimizing inference graphs driven by reinforcement learning, and emerging CNCF AI Working Group standards for interoperable InferenceGraphs will deliver vendor-neutral, portable, sustainable, and resilient enterprise deployments, ultimately bridging experimental AI innovation with production-grade reliability and environmental responsibility while empowering organizations to meet stringent service-level objectives in an increasingly demanding generative AI landscape.

References

1) M. Xu, J. Liao, J. Wu, Y. He, K. Ye, and C. Xu, "Cloud Native System for LLM Inference Serving," arXiv: 2507.18007, 2025.

2) KServe Project Maintainers, "KServe: Standardized Distributed Generative and Predictive AI Inference Platform," CNCF Project Documentation, 2025.

3) W. Kwon, Z. Li, S. Zhuang, Y. Sheng, L. Zheng, C. H. Yu, J. E. Gonzalez, H. Zhang, and I. Stoica, "Efficient Memory Management for Large Language Model Serving with PagedAttention," Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles, 2023.

4) J. Hu, J. Xu, Z. Liu, Y. He, Y. Chen, H. Xu, J. Liu, J. Meng, B. Zhang, S. Wan, G. Dan, Z. Dong, Z. Ren, C. Liu, T. Xie, D. Lin, Q. Zhang, Y. Yu, H. Feng, X. Chen, and Y. Shan, "DeepServe: Serverless Large Language Model Serving at Scale," Proceedings of the 2025 USENIX Annual Technical Conference, 2025.

5) C. Lou, S. Qi, C. Jin, D. Nie, H. Yang, Y. Ding, X. Liu, and X. Jin, "HydraServe: Minimizing Cold Start Latency for Serverless LLM Serving in Public Clouds," arXiv:2502.15524, 2025.

6) B. Cox, "12 Factor App meets Kubernetes: Benefits for cloud-native apps," Red Hat Blog, Feb. 2022.

Tianjin Daxue Xuebao (Ziran Kexue yu Gongcheng Jishu Ban)/
Journal of Tianjin University Science and Technology
ISSN (Online):0493-2137
E-Publication: Online Open Access
Vol: 59 Issue: 04:2026
DOI: 10.5281/zenodo.19606650

7) P. Vellaisamy, T. Labonte, S. Chakraborty, M. Turner, S. Sury, and J. P. Shen, “Characterizing and Optimizing LLM Inference Workloads on CPU-GPU Coupled Architectures,” in 2025 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), pp. 49-61, IEEE, 2025.

8) C. Lou, S. Qi, C. Jin, D. Nie, H. Yang, Y. Ding, X. Liu, and X. Jin, “HydraServe: Minimizing Cold Start Latency for Serverless LLM Serving in Public Clouds,” arXiv:2502.15524, 2025.

9) KServe Project Maintainers, “KServe: Standardized Distributed Generative and Predictive AI Inference Platform,” CNCF Project Documentation, 2025.

10) Seldon Core Contributors and BentoML Team, “Advanced Model Serving Frameworks for Kubernetes,” Technical Report, 2024.

11) NVIDIA Corporation, “Triton Inference Server User Guide and Architecture,” NVIDIA Documentation, 2025.

12) Y. Fu, L. Xue, Y. Huang, A.-O. Brabete, D. Ustiugov, Y. Patel, and L. Mai, “ServerlessLLM: Low-Latency Serverless Inference for Large Language Models,” in Proceedings of the 18th USENIX Symposium on Operating Systems Design and Implementation (OSDI ’24), pp. 135-153, 2024.

13) M. Xu, J. Liao, J. Wu, Y. He, K. Ye, and C. Xu, “Cloud Native System for LLM Inference Serving,” arXiv: 2507.18007, 2025.

14) Istio Authors, “Istio Service Mesh for AI Workloads: Sidecar and Ambassador Patterns,” CNCF Documentation, 2025.

15) Y. Fu, L. Xue, Y. Huang, A.-O. Brabete, D. Ustiugov, Y. Patel, and L. Mai, “ServerlessLLM: Low-Latency Serverless Inference for Large Language Models,” in Proceedings of the 18th USENIX Symposium on Operating Systems Design and Implementation (OSDI ’24), pp. 135-153, 2024.

16) Minxian Xu, Junhan Liao, Jingfeng Wu, Yiyuan He, Kejiang Ye, and Chengzhong Xu, “Cloud Native System for LLM Inference Serving,” arXiv:2507.18007, 2025.

17) Docker Inc., “Best Practices for Multi-Stage Builds with AI Models,” Docker Documentation, 2025.

18) Kubernetes SIGs, “Horizontal and Vertical Pod Autoscaling for GPU Workloads,” Kubernetes Documentation, 2025.

19) KServe Project Maintainers, “KServe InferenceService CRD Reference,” CNCF Project Documentation, 2025.

20) Yao Fu, Leyang Xue, Yeqi Huang, Andrei-Octavian Brabete, Dmitrii Ustiugov, Yuvraj Patel, and Luo Mai, “ServerlessLLM: Low-Latency Serverless Inference for Large Language Models,” in Proceedings of the 18th USENIX Symposium on Operating Systems Design and Implementation (OSDI ’24), 2024.

21) Eric Jonas, Johann Schleier-Smith, Vikram Sreekanti, Chia-Che Tsai, Anurag Khandelwal, Qifan Pu, Vaishaal Shankar, João Carreira, Karl Krauth, Neeraja Yadwadkar, Joseph E. Gonzalez, Raluca Ada Popa, Ion Stoica, and David A. Patterson, “Cloud Programming Simplified: A Berkeley View on Serverless Computing,” Technical Report UCB/EECS-2019-3, University of California, Berkeley, Feb. 2019.

22) Joseph M. Hellerstein, Jose Faleiro, Joseph E. Gonzalez, Johann Schleier-Smith, Vikram Sreekanti, Alexey Tumanov, and Chenggang Wu, “Serverless Computing: One Step Forward, Two Steps Back,” arXiv preprint arXiv:1812.03651, 2018.

23) Liang Wang, Mengyuan Li, Yinqian Zhang, Thomas Ristenpart, and Michael Swift, “Peeking Behind the Curtains of Serverless Platforms,” in 2018 USENIX Annual Technical Conference (USENIX ATC ’18), pp. 133-146, 2018.

Tianjin Daxue Xuebao (Ziran Kexue yu Gongcheng Jishu Ban)/
Journal of Tianjin University Science and Technology
ISSN (Online):0493-2137
E-Publication: Online Open Access
Vol: 59 Issue: 04:2026
DOI: 10.5281/zenodo.19606650

24) Mohammad Shahrad, Jonathan Balkind, and David Wentzlaff, “Architectural Implications of Function-as-a-Service Computing,” in Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO ’19), 2019.

25) Argo Project, “Argo Workflows: Kubernetes-native Workflow Engine,” CNCF Documentation, 2025.

26) Prometheus Authors, “Prometheus Monitoring for AI Inference,” Prometheus Documentation, 2025.

27) Tekton Authors, “Tekton Pipelines for MLOps,” Tekton Documentation, 2025.

28) KServe Project Maintainers, “KServe Security and Compliance Guide,” CNCF Project Documentation, 2025.

29) Yao Fu, Leyang Xue, Yeqi Huang, Andrei-Octavian Brabete, Dmitrii Ustiugov, Yuvraj Patel, and Luo Mai, “ServerlessLLM: Low-Latency Serverless Inference for Large Language Models,” in Proceedings of the 18th USENIX Symposium on Operating Systems Design and Implementation (OSDI ’24), 2024.

30) Mohammad Shahrad, Jonathan Balkind, and David Wentzlaff, “Architectural Implications of Function-as-a-Service Computing,” in Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO ’19), 2019.

31) Chiheng Lou, Sheng Qi, Chao Jin, Dapeng Nie, Haoran Yang, Yu Ding, Xuanzhe Liu, and Xin Jin, “HydraServe: Minimizing Cold Start Latency for Serverless LLM Serving in Public Clouds,” arXiv:2502.15524, 2025.

32) Yao Fu, Leyang Xue, Yeqi Huang, Andrei-Octavian Brabete, Dmitrii Ustiugov, Yuvraj Patel, and Luo Mai, “ServerlessLLM: Low-Latency Serverless Inference for Large Language Models,” in Proceedings of the 18th USENIX Symposium on Operating Systems Design and Implementation (OSDI ’24), 2024.

33) Nicholas Carlini, Florian Tramer, Eric Wallace, Matthew Jagielski, Ariel Herbert-Voss, Katherine Lee, Adam Roberts, Tom Brown, Dawn Song, Ulfar Erlingsson, Alina Oprea, and Colin Raffel, “Extracting Training Data from Large Language Models,” in 30th USENIX Security Symposium (USENIX Security 21), 2021.

34) AWS, “AWS Nitro Enclaves: Confidential Computing on AWS,” AWS Whitepaper, 2023.

35) NVIDIA Corporation, “Cost Optimization for GPU Inference in Cloud Environments,” NVIDIA Technical Blog, 2025.

36) Prometheus Authors, “Observability Best Practices for Black-Box AI Models,” Prometheus Documentation, 2025.

37) Epoch AI, “AI’s Environmental Impact and Energy Consumption Trends,” Epoch AI Report, 2025.

38) NVIDIA Corporation, “Sustainable AI Inference with Quantization and Pruning,” NVIDIA Technical Blog, 2025.

39) NVIDIA Corporation, “Sustainable AI Inference with Quantization and Pruning,” NVIDIA Technical Blog, 2025.

40) Carlos Costa, Clayton Coleman, Rob Shaw, Brian Stevens, and other llm-d Project Contributors (including teams from IBM Research, Red Hat, Google Cloud, CoreWeave, NVIDIA, AMD, Cisco, Hugging Face, Intel, Lambda, Mistral AI, University of California Berkeley, and University of Chicago), “llm-d: Kubernetes-Native Distributed LLM Inference Framework,” CNCF Sandbox Project, 2026.

41) Ericsson Research Team, “AI-driven Applications with 6G Compute Continuum,” Ericsson Blog, February 2026.

42) Iacovos Ioannou, Christophoros Christophorou, and Vasos Vassiliou, “Federated Learning at the Edge: Enabling AI in 5G/6G Networks and Massive IoT,” IntechOpen, 2026.

Tianjin Daxue Xuebao (Ziran Kexue yu Gongcheng Jishu Ban)/
Journal of Tianjin University Science and Technology
ISSN (Online):0493-2137
E-Publication: Online Open Access
Vol: 59 Issue: 04:2026
DOI: 10.5281/zenodo.19606650

43) Sai Puppala, Ismail Hossain, Md Jahangir Alam, Tanzim Ahad, and Sajedul Talukder, “A Comprehensive Survey of Federated Learning for Edge AI: Recent Trends and Future Directions,” Preprints.org, 2025.

44) Mihrimah Ozkan and C. S. Ozkan, “Federated Carbon Intelligence for Sustainable AI: Real-time Optimization across Heterogeneous Hardware Fleets,” MRS Energy & Sustainability, 2025.

45) Shahriar Ahsan Taisiq, “CAS-NAS: A Carbon-Aware Neural Architecture Search Framework for Sustainable AI Development,” Sustainable Computing: Informatics and Systems, 2026.

46) CNCF AI Working Group and KServe Maintainers (including Yuan Tang and core contributors), “Open Specifications for Interoperable Inference Graphs and Model Serving,” CNCF Documentation, 2025–2026.