# Building a Unified Multi-Cloud AI Fabric: Cloud-Native Patterns for Portable and Composable ML Services

**Santosh Pashikanti**

**Abstract:**
**Enterprises that build AI/ML platforms at scale are increasingly forced into multi-cloud strategies—sometimes by design (best-of-breed services, regulatory constraints, M&A), sometimes by accident. While Kubernetes and containers promise workload portability, the reality for AI/ML workloads is far more complex: data gravity, GPU scarcity, heterogeneous managed AI services, and fragmented MLOps tooling make "build once, run anywhere" difficult to realize in practice.**
**In this paper, I propose a unified multi-cloud AI fabric: an opinionated but vendor-neutral architecture that standardizes how AI/ML workloads are built, deployed, and operated across AWS, Google Cloud, and Microsoft Azure using containers, Kubernetes, and cloud-native abstraction layers. Building on recent work in cloud-native AI, Kubernetes-based ML platforms (e.g., Kubeflow), and distributed serving frameworks, the fabric defines a layered architecture with consistent patterns for portable training pipelines, composable inference graphs, cross-cloud traffic steering, and policy-driven governance. CNCF+1**
**I describe system requirements and design principles for such a fabric, including portability, composability, resilience, data locality, GPU efficiency, and security. I then present a reference architecture spanning EKS, AKS, and GKE, and walk through an implementation and case study of a global recommendation and fraud-detection platform. An evaluation compares this fabric against a single-cloud baseline along dimensions of migration effort, time-to-deploy, failover RTO, and cost utilization. Finally, I discuss trade-offs and outline future directions, including AI-native control planes, WASM-based runtimes, and cross-cloud vector databases. My goal is to provide a practical blueprint that other architects and ML platform teams can adapt, rather than yet another theoretical multi-cloud diagram that never survives contact with production.**

**Index Terms: Multi-cloud, Kubernetes, AI/ML, cloud-native, MLOps, portability, composable services, EKS, AKS, GKE, Kubeflow, Ray, KServe, SageMaker, Vertex AI, Azure Machine Learning.**

## I. Introduction

The last five years have fundamentally changed how we think about cloud and AI. Cloud-native technologies—containers, Kubernetes, and declarative infrastructure—have become the default substrate for modern applications. At the same time, AI/ML has shifted from experimental projects to revenue-critical workloads that demand enterprise-grade reliability, security, and governance. Cloud-native and AI/ML are no longer separate stories; they are now deeply intertwined. CNCF+1

In parallel, multi-cloud has moved from buzzword to reality. Organizations run workloads on multiple cloud providers to access specialized AI hardware and managed services, negotiate better pricing, meet data residency requirements, and reduce concentration risk. For AI/ML specifically, teams often want to train models where GPUs are available (or cheaper), but serve them close to users for latency or regulatory reasons. Multi-cloud is especially attractive when combining strengths: for example, training on GPU-dense infrastructure in one provider, but serving alongside existing applications and data in another. Portworx+1

Kubernetes is frequently presented as the answer to multi-cloud. By providing a consistent container-oriented API over compute, it does indeed improve workload portability across on-premises and public clouds. Google Cloud+1 But for AI/ML workloads, Kubernetes alone is insufficient. Real-world AI platforms depend on:

- Platform-specific managed AI services (Amazon SageMaker, Vertex AI, Azure Machine Learning)
- GPU scheduling and autoscaling

- Distributed training and serving frameworks (Kubeflow, Ray, KServe, Triton, etc.) Platform9+1
- Data platforms, feature stores, observability stacks, and security controls that are rarely identical across clouds

In my experience, teams start with the intention of "just using Kubernetes everywhere" and quickly find themselves with three slightly different platforms, each tightly coupled to its cloud, tooling, and IAM model. The result is often *accidental complexity* instead of strategic multi-cloud.

This paper aims to move from ad-hoc multi-cloud to a *unified multi-cloud AI fabric*—a systematic way to build portable and composable ML services across AWS, GCP, and Azure, without pretending that all clouds are the same.

**Contributions**

This paper offers:

1. **A requirements model** for a multi-cloud AI fabric, grounded in portability, composability, resilience, and data-aware design.
2. **A layered reference architecture** spanning EKS, AKS, and GKE, integrating Kubeflow, Ray, model registries, feature stores, and multi-cluster networking.
3. **Practical patterns** for integrating cloud-native ML runtimes with managed AI services (SageMaker, Vertex AI, Azure ML on Kubernetes). AWS Documentation+2Microsoft Learn+2
4. **A case study and evaluation approach** that platform teams can reuse to justify such an architecture in their organizations.

## II. Related Work and Industry Landscape

### A. Cloud-Native AI and Kubernetes for ML

The Cloud Native Computing Foundation (CNCF) has explicitly recognized AI/ML as a dominant workload on cloud-native infrastructure, outlining both opportunities and gaps in its Cloud Native Artificial Intelligence whitepaper. CNCF Similarly, surveys and market reports show rapid adoption of Kubernetes for AI workloads across industries. thesciencebrigade.com+1

Multiple practitioners and vendors argue that Kubernetes is becoming the de facto platform for machine learning due to its elasticity, ecosystem, and support for containerized tools across the ML lifecycle. Platform9+1 However, they also highlight challenges: networking complexity, GPU scheduling, and the operational burden of multi-cluster, multi-cloud deployments. F5, Inc.+1

### B. Kubernetes-Based ML Platforms

Kubeflow and related projects provide Kubernetes-native building blocks for ML pipelines, training, and serving. Kubeflow Pipelines focuses on container-based, portable ML workflows; KServe addresses standardized high-scale model serving; and Kubeflow's model registry and dashboards offer end-to-end visibility. Kubeflow+2Kubeflow+2

Enterprise distributions and managed offerings extend Kubeflow to hybrid and multi-cloud environments, emphasizing any-cloud operation while acknowledging the need to integrate with specific underlying storage, identity, and network services. Canonical+1

In parallel, distributed compute frameworks such as Ray provide a cloud-agnostic layer for training and serving, explicitly supporting deployment to multiple public clouds and Kubernetes clusters. Ray+1

### C. Multi-Cloud ML Strategies

Academic and industry work on multi-cloud ML pipelines highlights portability benefits and design patterns for decoupling model logic from underlying infrastructure. Early frameworks focused on portable pipelines for scientific workloads. PMC More recent industry articles emphasize model portability via containerization, common storage formats, and cloud-agnostic DevOps/MLOps pipelines. zeet.co+2Medium+2

However, most literature and blogs either:

- stay at a high level ("avoid lock-in, use containers") or
- zoom in on a single cloud (e.g., "Kubernetes + SageMaker", "GKE + Vertex AI") without offering a cohesive multi-cloud control plane. Amazon Web Services, Inc.+2Medium+2

### D. Managed AI Services and Kubernetes

All three major cloud providers now offer tight integration between Kubernetes and managed ML services:

- **AWS**: SageMaker Operators for Kubernetes and Kubeflow components allow training and inference jobs to be orchestrated from EKS. AWS Documentation+1
- **Azure**: Azure Machine Learning supports Kubernetes compute targets and AzureML Kubernetes extensions for hybrid and multicloud scenarios. Microsoft Learn+1
- **GCP**: Vertex AI integrates with GKE, allowing models and pipelines to be deployed via containerized workloads or managed endpoints. Google Cloud Documentation+1

These integrations are powerful but proprietary. They tend to increase lock-in unless wrapped in fabric-level abstractions.

### E. Gaps

Across the landscape, recurring gaps include:

- No common **multi-cloud AI "fabric" architecture** that spans all three major clouds with practical integration patterns.
- Limited discussion of **cross-cloud data realities** (feature stores, vector databases, model artifacts).
- Sparse guidance on **operational metrics and evaluation** of multi-cloud AI architectures beyond cost or theoretical resilience.

This paper addresses these gaps by proposing a concrete, implementable fabric that combines Kubernetes, portable ML toolchains, and cloud-native abstractions while still leveraging provider-specific strengths.

## III. System Requirements and Cloud-Native Design Principles

Drawing from real implementations and the literature, I define the following requirements for a unified multi-cloud AI fabric.

### A. Functional Requirements

1. **Portable training pipelines**
   - Define ML workflows once, parameterize the target cluster/cloud.
   - Support heterogeneous accelerators (GPU types, TPUs) and storage backends.
2. **Composable inference services**
   - Build inference graphs that can span multiple models and microservices.
   - Allow mixing of self-hosted models (e.g., on KServe/Ray Serve) with managed endpoints (SageMaker, Vertex AI, Azure ML). Ray+2AWS Documentation+2
3. **Multi-cluster, multi-cloud deployments**
   - Treat each cloud as a "fabric zone" with one or more Kubernetes clusters.
   - Provide a unified control plane for configuration, policies, and observability. F5, Inc.+1
4. **Data-aware orchestration**
   - Place training and serving jobs close to data sources when possible.
   - Support feature store synchronization and cross-cloud replication where necessary. Portworx+1
5. **Lifecycle-complete MLOps**
   - Support experimentation, training, evaluation, deployment, rollback, monitoring, and drift management in a uniform way. Kubeflow+1

### B. Non-Functional Requirements

1. **Portability**
   - Minimal code changes when moving workloads between clouds.
   - Use containers and open standards (e.g., OCI images, OpenAPI/GRPC, ONNX) wherever realistic. LinkedIn+1
2. **Composability**
   - Treat models and pipelines as building blocks that can be combined into higher-order services (e.g., RAG + recommendation + fraud scoring).
3. **Resilience and failover**
   - Survive regional or provider outages via cross-cloud failover.

- Explicitly define RTO/RPO targets per workload class. F5, Inc.

4. **Performance and efficiency**
- High GPU utilization via autoscaling, bin-packing, and workload placement. SpringerOpen+1

5. **Security and compliance**
- Policy-as-code for network, identity, and data movement across clouds.
- Fine-grained multi-cluster network segmentation, especially for AI workloads that often process sensitive data. F5, Inc.+1

**C. Cloud-Native Design Principles**

To satisfy these requirements in a realistic, "battle-tested" manner, I adopt the following principles:

1. **Containers as the unit of portability**
   All training and inference components are packaged into OCI container images, including feature engineering, trainers, evaluators, explainers, and canary validators.
2. **Kubernetes as the control substrate**
   EKS, AKS, and GKE are used as the common substrate; but we treat "Kubernetes" not as a silver bullet, rather as a programmable platform for composing higher-level abstractions. Google Cloud+1
3. **Declarative everything (GitOps)**
   Infrastructure (Terraform/Crossplane), platform (Helm, Kustomize), and ML resources (CRDs for pipelines, models, endpoints) are expressed declaratively and reconciled via GitOps tools such as Argo CD or Flux. blog.upbound.io+1
4. **"Least common denominator + progressive enhancement"**
   At the fabric layer, we standardize on portable primitives (KServe, MLflow, Kubeflow Pipelines, Ray). At the zone layer, we selectively use cloud-specific enhancements (SageMaker training jobs, Vertex AI pipelines, AzureML compute) where they add clear value, but hide that behind abstraction interfaces.
5. **Separation of concerns between fabric and workloads**
- Fabric teams own clusters, platforms, policies, and shared services.
- ML product teams own pipelines, models, and business logic, deployed onto the fabric through well-defined interfaces.

6. **Observability and SRE-driven operations**
- Unified logging, metrics, traces, and model telemetry across clouds, via a common stack (e.g., Prometheus, OpenTelemetry, Loki, Tempo) or a vendor platform. F5, Inc.+1

**IV. Architecture**

I call the proposed architecture the **Unified Multi-Cloud AI Fabric (UMAF)**. It is logically organized into four layers and multiple "zones" (one per cloud/provider).

**A. High-Level Layers**

1. **Infrastructure & Cluster Layer**
- **AWS**: Amazon EKS clusters across multiple regions, integrated with VPC, PrivateLink, and Amazon S3.
- **GCP**: GKE clusters with Cloud Storage, Artifact Registry, and Cloud Load Balancing.
- **Azure**: AKS clusters with Azure VNet, Azure Blob Storage, and Azure Key Vault. Google Cloud Documentation+2Microsoft Learn+2

These clusters are provisioned via Terraform or a cluster API and registered into a central "cluster registry" (a simple database or GitOps catalog).

2. **Fabric Control Plane**
   Shared components deployed to each cluster but configured centrally:
- **GitOps controllers**: Argo CD/Flux for declarative sync.
- **Service mesh**: Istio or Linkerd for mTLS, traffic routing, and policy enforcement across services.
- **Multi-cluster gateway**: Kubernetes Gateway API + vendor-specific L7 load balancers (ALB, GCLB, Azure Application Gateway) for cross-cloud ingress and failover. F5, Inc.+1
- **Observability stack**: Prometheus, Grafana, Loki/Elastic, OpenTelemetry collectors.

3. **AI/ML Platform Layer**
   Deployed largely uniformly across clouds, with selective provider integration:
   - **Workflow and orchestration**
     - Kubeflow Pipelines for portable, container-based ML workflows. Red Hat Developer+1
     - Ray for distributed training, hyper-parameter search, and online inference. Ray+1
   - **Experiment tracking & registry**
     - MLflow for experiments/metrics and artifact tracking.
     - Kubeflow Model Registry or an equivalent for versioned models and deployment metadata. Kubeflow+1
   - **Model serving**
     - KServe for standardized inference services on Kubernetes.
     - Ray Serve and/or Triton Inference Server for complex graphs and GPU-intensive serving. Ray+1
   - **Managed AI integration adapters**
     - SageMaker Operators for Kubernetes (AWS zones). AWS Documentation+1
     - Vertex AI connectors for training/serving on GCP. Google Cloud Documentation+1
     - Azure ML Kubernetes extension and compute for Azure zones. Microsoft Learn+1

4. **Data & Feature Layer**
   - **Feature store** (Feast or similar) with backends in each cloud (e.g., BigQuery, DynamoDB, Cosmos DB) but a unified schema and API.
   - **Object storage abstraction** for training data and model artifacts, implemented via per-cloud buckets (S3, GCS, Blob) plus a logical "model catalog" in MLflow and the registry. Portworx+1
   - **Vector search** engines for embeddings (e.g., Elastic, OpenSearch, managed offerings), deployed per zone but synchronizing critical indexes where cross-cloud queries are required. Elastic+1

5. **Application & API Layer**
   - Microservices exposing model predictions over REST/gRPC.
   - RAG and agentic AI services that orchestrate multiple models, retrieval, and business rules at runtime.
   - Tenant-aware routing to specific zones based on geography, regulatory domains, or SLAs.

## V. Implementation and Case Study

To make the architecture concrete, I'll describe a representative implementation: a global retail company building a unified AI platform for **personalized recommendations** and **real-time fraud detection**.

### A. Business and Technical Context

- Customers and transactions span North America, Europe, and APAC.
- Historical clickstream and catalog data is consolidated primarily in GCP (BigQuery), while transactional payment data resides in AWS and Azure for legacy reasons and regulatory constraints.
- GPUs are more cost-effective and available in select regions across clouds; the organization wants the flexibility to burst training workloads wherever capacity is available.

From a business standpoint, we defined targets such as:

- +5–10% lift in recommendation CTR
- 30–50% reduction in fraud-related losses
- Cross-cloud failover with < 10-minute RTO for core inference endpoints

### B. Foundation: Multi-Cloud Clusters and Networking

1. **Cluster provisioning**
   - Use Terraform modules to provision:
     - 2× EKS clusters (us-east-1, eu-west-1)
     - 2× GKE clusters (us-central1, europe-west1)
     - 1× AKS cluster (westeurope)
   - GPU-enabled node pools (e.g., NVIDIA A10/A100 or equivalent) where needed. Google Cloud Documentation+1

2. **Networking**
   - Establish site-to-site IPsec VPN or cloud interconnects between VPC/VNet networks to allow secure cross-cloud traffic.
   - Deploy a mesh (Istio) per cluster; configure multi-cluster gateways to support failover routing based on active health checks. F5, Inc.+1
3. **Cluster registry and GitOps**
   - Maintain a YAML catalog of cluster endpoints, labels (region, cloud, GPU support), and capabilities in Git.
   - Install Argo CD in one "management" cluster per cloud, federated via project conventions and shared repos.

**C. AI Platform Rollout**

1. **Kubeflow and Ray**
   - Deploy Kubeflow to the primary EKS and GKE clusters using vendor-neutral manifests.
   - Deploy Ray clusters on top of EKS/GKE/AKS via Helm charts, with autoscaling policies tuned per cloud. Kubeflow+2Ray+2
2. **Managed AI integration**
   - In AWS zones, install SageMaker Operators for Kubernetes, enabling Kubeflow Pipelines to submit training jobs to SageMaker for large-scale, spot-optimized experiments. AWS Documentation+1
   - In GCP zones, allow certain pipeline steps to call Vertex AI custom training and use managed endpoints for large foundational models. Google Cloud Documentation+1
   - In Azure, attach AKS clusters to Azure ML workspaces as Kubernetes compute targets for both training and online endpoints where needed. Microsoft Learn+2Microsoft Learn+2
3. **Feature store and data pipelines**
   - Implement a feature store (e.g., Feast) with cloud-specific backends:
     - GCP: BigQuery + GCS for clickstream-derived features.
     - AWS: DynamoDB + S3 for payment risk features.
     - Azure: Cosmos DB + Blob Storage for regional loyalty data. Portworx+1
   - Use Kafka / Pub/Sub / Event Hubs connectors to stream key events into the feature store across regions.
4. **Model tracking & registry**
   - Standardize on MLflow for experiment tracking, deployed centrally but accessible across zones (backed by a replicated database and object storage).
   - Use Kubeflow Model Registry to define deployment candidates, their lineage to training runs, and target fabrics (e.g., us-global, eu-low-latency). Kubeflow+1

**D. Training Pipelines**

A typical **recommendation model pipeline** in Kubeflow Pipelines includes components for:

1. Data extraction and feature engineering (BigQuery + feature store).
2. Embedding generation and candidate generation (e.g., using a two-tower model on GPUs).
3. Reranking model training (GBDT or deep learning).
4. Offline evaluation and bias checks.
5. Model registration to MLflow and Kubeflow Model Registry.

The pipeline spec includes a target_zone parameter (aws, gcp, azure) and label selectors referring to clusters with particular GPU capabilities.

This means that **the same pipeline**:

- Can run predominantly in GCP when BigQuery workloads dominate.
- Can shift to EKS using SageMaker training jobs during GPU scarcity in GCP, without changing the pipeline logic, only the underlying components. AWS Documentation+2Red Hat Developer+2

**E. Serving and Composable Inference Graphs**

For **real-time recommendations**:

- Use KServe to expose a gRPC endpoint recommendation-service in each zone.
- Under the hood, KServe routes to:

- A candidate generator model on Ray Serve (for high-throughput embedding retrieval and ANN search).
- A reranker model container. Ray+2Medium+2

For **fraud detection**:

- Deploy a multi-model endpoint using KServe or a custom container that hosts multiple models (e.g., rules engine + anomaly detection + deep model). On AWS, we can also integrate with SageMaker multi-model endpoints for specific workloads while using the same contract at the fabric level. AWS Documentation+1

Both services publish metrics and traces to a shared observability stack, and are fronted by the multi-cluster gateway with canary rules and regional routing policies.

### F. Operationalization

- Rollouts are handled via GitOps: merge to main triggers a new version of the model deployment manifest; Argo CD syncs it into selected zones with automatic progressive rollout.
- SLOs are defined per endpoint (latency, error rate, fraud capture rate). SREs use these to manage canaries and cross-cloud failover policies (for example, automatically shifting traffic from GKE to EKS if GKE's latency exceeds a threshold). F5, Inc.+1

## VI. Evaluation and Results

Because multi-cloud AI fabrics can be complex to justify, I find it useful to frame evaluation in terms that business and engineering stakeholders both care about.

### A. Evaluation Dimensions

1. **Portability**
   - Time and effort to move a representative ML service from one cloud to another (e.g., GCP → AWS).
   - Number of changes required in code vs configuration.
2. **Time-to-deploy and iteration speed**
   - Lead time from a merged pipeline change to availability in all target zones.
   - Frequency of safe deployments per week.
3. **Resilience**
   - Measured RTO/RPO for critical inference endpoints under simulated regional failure.
4. **Resource efficiency**
   - GPU utilization across clouds, before and after the fabric.
   - Cost of running training and inference relative to a single-cloud baseline. PhilArchive+1
5. **Operational complexity**
   - Number of duplicated platform patterns.
   - Mean time to detect (MTTD) and mean time to recover (MTTR) from incidents.

### B. Illustrative Comparison

In one reference implementation, we compared:

- **Baseline**: three independently built single-cloud AI stacks, each with its own pipelines, serving layer, and monitoring.
- **Fabric**: UMAF architecture described above, with shared patterns and components.

Representative (illustrative) outcomes:

| Metric | Baseline (Single-cloud Stacks) | UMAF (Multi-Cloud Fabric) |
|---|---|---|
| Service migration effort (person-days) | 10–15 per service | 1–3 (config and manifests) |
| Deploy lead time (code → prod) | 3–5 days | < 1 day |
| RTO for cross-region failover | 1–2 hours | < 10 minutes |
| Average GPU utilization | 40–50% | 65–75% |
| Distinct CI/CD patterns | 3–4 | 1–2 |

These numbers are not universal truth; they are consistent with what I have seen across several real-world engagements when teams move from fragmented stacks to a unified fabric and adopt GitOps, standardized ML tooling, and multi-cluster routing.

## VII. Discussion

### A. Benefits

1. **Pragmatic portability**

The fabric does not require "identical" environments; instead, it standardizes the parts that matter most to ML teams—pipelines, model packaging, and serving contracts—while abstracting cloud-specific details behind operators and adapters.

2. **Better resource utilization**

By treating GPUs across clouds as a shared pool (within the limits of data residency), teams can opportunistically shift large training jobs to where capacity is cheapest or available, without rewriting pipelines. SpringerOpen+1

3. **Resilience as a first-class property**

Cross-cloud failover becomes an explicit design goal with measurable RTO/RPO, rather than an afterthought for a disaster-recovery document nobody tests. F5, Inc.+1

4. **Unified governance and compliance**

With policy-as-code and a shared mesh, enforcing encryption, mTLS, and data access policies becomes more consistent even as workloads span providers. F5, Inc.+1

### B. Costs and Trade-offs

1. **Operational complexity and skill requirements**

Running multiple Kubernetes clusters, service meshes, and ML platforms across clouds is not trivial. Teams need skills in SRE, platform engineering, and MLOps, not just data science.

2. **Data gravity and latency**

While models can move, data often cannot. Careful design is required to avoid chatty cross-cloud calls or expensive data transfers. Sometimes, the right answer is to move models to data, not the other way around. Portworx+1

3. **Partial vendor lock-in is inevitable**

Even with a fabric, managed services are attractive for certain workloads. The goal is to *consciously* accept these dependencies and wrap them appropriately, not to pretend that total neutrality is possible.

4. **Tool sprawl**

AI/ML tooling is evolving rapidly; adding multi-cloud multiplies options. A disciplined platform roadmap (e.g., a CNCF-style landscape tailored to your organization) is essential. CNCF Landscape+2blog.upbound.io+2

### C. When a Multi-Cloud AI Fabric *Isn't* the Right Answer

There are scenarios where a single-cloud strategy or regional failover within one provider might be more appropriate:

- Early-stage startups with limited platform capacity.
- Highly regulated environments where cross-cloud traffic is not feasible.
- Workloads where performance and data locality strongly favor a single provider.

The architecture described here is most compelling for organizations that already operate in multiple clouds or expect to do so for the foreseeable future.

## VIII. Conclusion

Multi-cloud is no longer an edge case for AI/ML; it is increasingly the norm. Kubernetes, containers, and cloud-native tooling give us the raw ingredients for portability and composability, but they do not automatically produce a coherent multi-cloud AI platform.

In this paper, I proposed a **Unified Multi-Cloud AI Fabric** architecture that:

- Uses Kubernetes as a common substrate across AWS, Azure, and GCP.
- Builds a shared AI/ML platform layer with Kubeflow, Ray, KServe, MLflow, and feature stores.

- Integrates managed AI services from each provider behind fabric-level adapters.
- Emphasizes declarative, GitOps-driven operations, observability, and policy-as-code.
- Demonstrates, via a case study, how this fabric can improve portability, resilience, and resource efficiency compared to fragmented single-cloud stacks.

My intent is not to claim a perfect reference model but to offer a *practical starting point* that can be adapted to different organizational constraints. With a disciplined platform strategy, the right abstractions, and an honest understanding of data and operational realities, a unified multi-cloud AI fabric can turn multi-cloud from a liability into a competitive advantage.

**REFERENCES:**

[1] Cloud Native Computing Foundation, "Cloud Native Artificial Intelligence Whitepaper," Mar. 2024. CNCF
[2] P. K. Anand, "Machine Learning in the Cloud: Best Practices and Use Cases," 2024. PhilArchive
[3] C. A. Ellis *et al.*, "A Cloud-based Framework for Implementing Portable Machine Learning Pipelines for Neural Data Analysis," *Frontiers in Neuroinformatics*, 2019. PMC
[4] K. Senjab *et al.*, "A Survey of Kubernetes Scheduling Algorithms," *Journal of Cloud Computing*, 2023. SpringerOpen
[5] G. Google, "How Kubernetes Takes Container Workload Portability to the Next Level," Google Cloud Blog, May 2016. Google Cloud
[6] Komodor, "Why Kubernetes Is Becoming the Platform of Choice for Running AI/MLOps Workloads," Jan. 2025. Komodor
[7] Kubermatic, "AI and Machine Learning Integration into Kubernetes: Trends, Challenges, and Best Practices," Jun. 2024. kubermatic.com
[8] A. Dettori, "Handle MLOps Across Multiple Cloud Providers Using Kubeflow," *dev.to*, Oct. 2021. DEV Community
[9] Kubeflow Project, "Kubeflow and Kubeflow Pipelines Documentation," Accessed 2025. Kubeflow+1
[10] Amazon Web Services, "SageMaker AI Operators for Kubernetes," AWS Documentation, 2020–2024. AWS Documentation+1