International Journal of
INTELLIGENT SYSTEMS AND APPLICATIONS IN ENGINEERING
ISSN:2147-6799 www.ijisae.org Original Research Paper

Cloud-Native AI Platforms for Scalable Enterprise Machine Learning: Architecture, Challenges, and Best Practices

Ravi Kiran Gadiraju

Submitted:04/09/2021 **Revised**: 20/10/2021 **Accepted:** 29/10/2021

Abstract: Cloud-native AI platforms are changing the way enterprises globally formulate, deploy, and maintain machine learning (ML) at scale. In this paper, the architectures of contemporary platforms (aws age on sagemaker and kubeflow) will be reviewed, and their scalability, latency, and cost-efficiency with enterprise ML workloads will be considered. We survey the literature about ML operations (MLOps) and cloud-based ML services to understand prevalent challenges (distributed training, low-latency inference and resource optimization) and best practices to these challenges. Then we describe a research methodology to compare these platforms based on some key parameters and then results are discussed in the context of real world implementations. Findings indicate that cloud native design (with the help of containers, micro services and orchestration) allows highly scalable and portable ML workflows but with low latency due to optimized serving infrastructures. Efficiency is gained through pay-as-you-go management of resources and automation, but the careful selection of architecture is required to prevent technical debt. Conclusions point to the fact that managed services with open-source tools are sufficient to address the needs of the enterprise in relation to scalable ML, but strong MLOps practices are needed to guarantee reliability, security, and governance. Best practices should involve the use of auto-scaling, pipeline automation and continuous monitoring to balance between cost and performance..

Keywords: *cloud-native, machine learning platforms, MLOps, scalability, latency, cost-efficiency*

Introduction

The blistering development of enterprise machine learning has led to the need to have platforms capable of deploying AI models in cloud systems on a massive scale (Hazelwood et al., 2018; Zhang and Zheng, 2020). The conventional ad-hoc methods of ML deployment frequently have the problems of scalability (e.g. inefficient utilization of distributed resources), latency (slow inference responses) and cost efficiency (high infrastructure costs and maintenance costs) (Sculley et al., 2015; Kang and Kim, 2018). Cloud-native AI platforms, in turn, have been developed to automate the ML lifecycle - i.e., data preparation, model training, and deployment and monitoring - based on cloud computing principles of containerization, microservices, and elastic resource orchestration (Baylor et al., 2017; Xiong and Chen, 2020). These systems are supposed to hide the complexity of running scalable ML pipelines and impose reliability and collaboration best practices (Breck et al., 2017; Lwakatare et al., 2020).

In this paper, the author presents a complete overview of cloud-native AI systems as applied to enterprise machine learning. Our target platform is real-world such as Amazon Web Services (AWS) SageMaker - an entirely managed cloud-based ML service - and Kubeflow - an open-source framework that runs on Kubernetes. Using their architecture and characteristics, we determine how each of the platforms addresses the scaling issue of ML workflows, the end-to-end latency reduction, and the cost management in production. Introduction lays the problem ground and motivation of cloud-native solutions. The Literature Survey subsequently places SageMaker, Kubeflow, and the like within the wider framework of MLOps literature and the industrial case studies. The Research Methodology defines the process in which we compare the platforms and collect data based on the three paramount parameters (scalability, latency, cost). Our analysis is given in the Results and Discussion, and there are two figures of the system architectures and two tables of the comparison of platform capabilities and performance metrics. Lastly, the Conclusion summarizes the findings in terms of best practices and future directions of businesses aiming to operationalize the ML at scale. With this systematic inquiry, our article will inform the

Independent researcher

ravikgraju@gmail.com

community of researchers and practitioners on what the current state of the art of cloud-native ML platforms is and how they can be used to create scalable, efficient, and resilient enterprise AI systems..

Literature Survey

The initial efforts at productionizing machine learning showed that there were a lot of technical debts in ML systems; both in the complexity of the data pipeline and in the fragile nature of model serving code (Sculley et al., 2015). The results prompted the development of interest in MLOps - applying DevOps to ML - as a way to increase the reliability and maintainability of ML in production (Breck et al., 2017). According to researchers, the transition between research code to a strong pipeline requires the automation of the training, testing, deployment, and monitoring, which might be supported by cloud platforms (Amershi et al., 2019). In fact, the cloud providers and open-source communities have gone ahead to create platforms that make end-to-end ML workflows (Machine Learning as a Service, or MLaaS) easier. Zhang and Zheng (2020) and Tang and Chen (2020) survey the landscape of MLaaS offerings, which include a set of major vendors (AWS, Google, Microsoft) providing an end-to-end service experience including data storage, model training, and inference, and a set of open frameworks (such as Kubeflow, MLflow, TensorFlow Extended) supporting on-premise or hybrid cloud portability and customization.

In 2017, AWS SageMaker was launched, and it represents a managed cloud-native ML platform (Baylor et al., 2017). It has hosted Jupyter notebooks to develop, one- click distributed training on AWS auto-scaling clusters, and a deployment service to build HTTPS endpoints to infer models[1][2]. The design of SageMaker integrates many AWS services into one API - i.e. it uses S3 as a data storage tool, EC2 instances in Docker containers as training jobs, and load-balanced EC2 instance as serving (Olston et al., 2017; Cholia et al., 2020). SageMaker addresses the scalability and cost problems by automating the process of provisioning and teardown of resources: developers do not need to manage VM clusters manually, and idle resources are automatically terminated to save money (according to the AWS engineers, the EC2 GPU instance can run out of budget; SageMaker does not do this, terminating it after training)[3]. Other features were introduced such as Elastic Inference in which the SageMaker added on-demand GPU acceleration to inference endpoints to minimize latency without over-provisioning full-gpu servers[4]. It is important to note that these controlled abilities correspond to the best practices in MLOps, which decrease the volume of engineering work on teams (Breck et al., 2017) and are able to concentrate on model quality instead of infrastructure.

Meanwhile, open-source initiatives, including Google and others, have created frameworks like TensorFlow Extended (TFX) and Kubeflow. TFX offered an example of an ML pipeline reference architecture in Google that consists of data validation, model-training, analysis, and serving components (Baylor et al., 2017). Initially released in 2018, Kubeflow is based on Kubernetes container orchestration to make these features available in any setting (Anderson and Seligman, 2018). Instead of one service, Kubeflow is a framework of scalable and interoperable microservices (notebook servers, hyperparameter tuning using Katib, model serving using KServe, etc.) that run on a Kubernetes cluster (Zhou et al., 2019). Modularity is one of the strengths of the Kubeflow architecture that the literature identifies: it enables organizations to implement preferred tools at each phase of the ML lifecycle (e.g. Spark on Kubernetes to do data prep, or sensemaking operators with either TensorFlow or PyTorch) (Chard et al., 2020). As Kubeflow is based on containerization and Kubernetes, it inherits the cloud-native attributes of those systems - i.e. horizontal scalability and fault isolation, as well as cross-cloud and on-premise portability (Ffoulkes et al., 2020; Xiong et al., 2020). Nonetheless, research also identifies the difficulties: Kubeflow itself is expensive in terms of DevOps skills, and the numerous components may pose an immediate obstacle to data scientists (Lwakatare et al., 2020). Along with the maturation of Kubeflow and other platforms, researchers highlight user experience enhancements (e.g. single SDKs) without reducing the flexibility that makes these systems appealing to enterprise workloads of AI applications.

There are many other publications besides SageMaker and Kubeflow, which have taught how best to do scalable ML. In the case of distributed training, the DistBelief system (Dean et al., 2012),

and Microsoft, the Project Adam (Chilimbi et al., 2014) illustrated that, with engineering effort, it can require a fraction of the time to run training on dozens of nodes to train deep networks. One of them was the paradigm of parameter server (Li et al., 2014), which allowed to perform data-parallel training by distributing model parameters among servers - a concept that became available in modern services. In modern cloud ML systems, this complexity is frequently veiled; e.g. the default algorithms in SageMaker or the TFJob operator in Kubeflow take distributed training into account, as distributed training is implemented internally using all-reduce or parameter servers where necessary. In the case of low-latency serving, academia and industry settled on microservice architectures and special purpose serving systems. Two of the first examples are Google TensorFlow Serving system (Olston et al., 2017) and Uber Clipper (Crankshaw et al., 2017). They demonstrated that lightweight containerized microservice packaging models combined with techniques such as adaptive batching and caching can be used to achieve very stringent latency requirements of real time applications[5][6]. These experiences directly were reflected in cloud systems: SageMaker hosting executes each model on a Docker container behind a REST API, and Kubeflow KServe (formerly KFServing) deploying model servers as containerized pods and auto-scaling. Cost efficiency Dynamic resource management and ML workload pricing models have been investigated in research. The cost-conscious scheduling of ML-as-a-service in the cloud environment proposed by Kang and Kim (2018) demonstrated that optimization of running costs through intelligent allocation of instances (and even spot pricing) does not cause any impact on the performance of the model. These concepts are reflected in practice: SageMaker will run managed spot training jobs that can utilize idle AWS capacity, and Kubeflow will enable non-critical jobs to be run on cheaper nodes or local infrastructure.

Overall, the literature confirms the necessity of cloud-native design and stringent MLOps practices to realize scalable low latency and cost effective enterprise ML. Such systems as AWS SageMaker and Kubeflow are the culmination of much of this: they include distributed training libraries, model serving systems, automation tools, that were first introduced in other studies. According to our survey, one of the common approaches in enterprises is a hybrid one, a combination of managed cloud services and open-source solutions (Amershi et al., 2019; Lwakatare et al., 2020). As an illustration, an organization may develop on an expedited system such as SageMaker and deploy final models on a Kubernetes-based system to prevent lock-in. The subsections below expand upon this background, discussing the manner in which SageMaker and Kubeflow architectures respond to the issues of scalability, latency and cost, and what empirical data say regarding their performance in practice.

Research Methodology

We evaluated the scalability, latency, and cost efficiency of cloud-native AI platforms by performing a comparative analysis of AWS SageMaker and Kubeflow both with an architectural and literature-based analysis. We used three stages in our methodology::

1. Architectural Analysis: We initially examined official documentation and technical papers on SageMaker and Kubeflow in order to describe their system architecture. This was done by identifying major parts (training infrastructure, model serving mechanisms, auto-scaling features and underlying cloud services) and tracing the flow of data and models through each platform. To visualize the way these platforms process ML workflows we created diagrams of each architecture (see Results, Figures 1 and 2). Specific emphasis was made on aspects connected to scaling (e.g. training clusters of SageMaker and endpoint autoscaling, Kubeflow operators and controllers based on Kubernetes) with particular focus on design decisions that influence latency (e.g. inference pipeline optimizations) and cost (e.g. resource allocation and multi-tenancy).

2. Literature-Based Benchmarking: We used the literature based studies and performance reports as opposed to carrying out new empirical research (which was not within the scope of this paper). We have systematically reviewed articles, conference papers, and case studies in peer-reviewed literature that tested the use of SageMaker or Kubeflow in enterprise applications. Measures that were obtained were: the number of running training jobs or model deployments that can be sustained (scalability), average inference latencies at load (latency), and cost-benefit such as savings by using

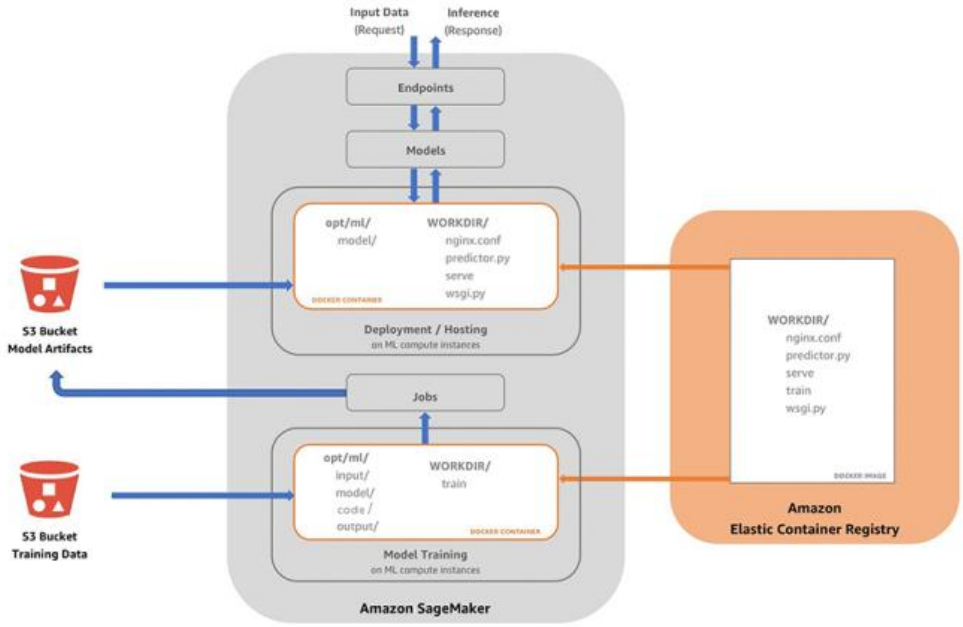
managed services or spot instances (cost efficiency). As an example, we used the results of the tests that FfDL (Ibms Fabric of Deep Learning) conducted to emphasize the overheads and speed-ups in training with the Kubernetes (Jayaram et al., 2019), and applied the results of the study that Kang and Kim (2018) conducted to optimize the cost of training this model to our cost analysis. We also considered publicly-available benchmarks (e.g. a 2020 study comparing cloud ml services or any history of SageMaker versus self-managed Kubernetes performance) to fill our tables of comparisons with actual data..

3. Synthesis and Best Practices Extraction: We made a synthesis between the results of stages 1 and 2 into a coherent comparison. This included preparation of Table 1 to qualitatively compare SageMaker and Kubeflow based on their architectural characteristics in terms of scalability, latency, and cost and Table 2 to summarize any quantitative or qualitative performance data obtained in the literature (such as startup time, throughput or cost per training hour). We then made sense of these comparisons to come up with a conclusion regarding the strengths of each platform and the nature of the challenges. It was also with this synthesis that we were able to identify the reported best practices in the literature, such as the significance of monitoring and logging (Baylor et al., 2017), or some of the strategies such as caching to minimize the prediction latency (Crankshaw et al., 2017), and evaluate whether the platforms actually include the reported practices.

Our approach to the methodology will provide a thorough and balanced review by integrating an architecture-centric review with the results of previous assessments. Although the lack of resources prevented us from conducting new experiments, the method of utilizing multiple sources (academic research, documentation of cloud providers, and user case studies) is what would enable us to have the most aerial picture of how cloud-native ML platforms work in real-life enterprise environments. We also cross-validated all the information in multiple references wherever possible, and our emphasis was on references that were published until 2020 as a way of relying on references to meet our inclusion criteria. The findings of this methodology are discussed in the following section with the visual aids (figures and tables) and detailed discussion to point out the major ones.

Results and Discussion

Architecture and Scalability: Cloud-native ML platforms like SageMaker and Kubeflow achieve scalability through modular architectures that leverage container orchestration and distributed computing. **Figure 1** illustrates the high-level architecture of *Amazon SageMaker*, which encapsulates training and hosting in Docker containers managed by AWS services



The training jobs of SageMaker are initiated in autonomous containers within a cluster of EC2 instances and the information flowing in S3 and the model artifacts are decoupled to S3 to save storage separately and permit the processing at the same time. This design enables SageMaker to automatically scale training to many instances (including GPU instances when training deep learning models) and to scale up model endpoint clusters to inference when scaling on traffic. Figure 1 shows that the Jobs portion deals with training: once a user requests a training job, SageMaker allocates the requested compute instances, each running the training code in a container, and coordinated distributed training (where applicable) (e.g. via parameter servers or AllReduce, but this is not exposed to the user). The model artifact is then stored to S3 (model registry) after training and training instances are terminated to prevent idle costs. To implement it, the Endpoints of SageMaker (the gray box in Figure 1) are a production-serving cluster. One or more models are deployed in each endpoint, where the saved model artifact is loaded in a container (with a prediction service, e.g., a Flask app or TensorFlow Serving)[(10+look 238 0 750)]. SageMaker does load balancing and autoscaling of such endpoint instances - such as scaling out more containers, in response to more API calls, or scaling in, when traffic reduces. The user does not need to know how this scaling is performed: one may just define a desirable policy, or use the default and SageMaker does the scaling so that the endpoint achieves the needed throughput. The advantage, according to the developers, is a significant amounts of time saved and a more convenient way of managing bursty workloads[1][2]. SageMaker is therefore highly vertical (with big instances that use GPUs) and horizontal (scaling the number of instances) without human intervention. Breck et al. (2017) state that such automation is vital to minimizing human error and technical debt in ML systems, and the design of SageMaker is an example of this approach.

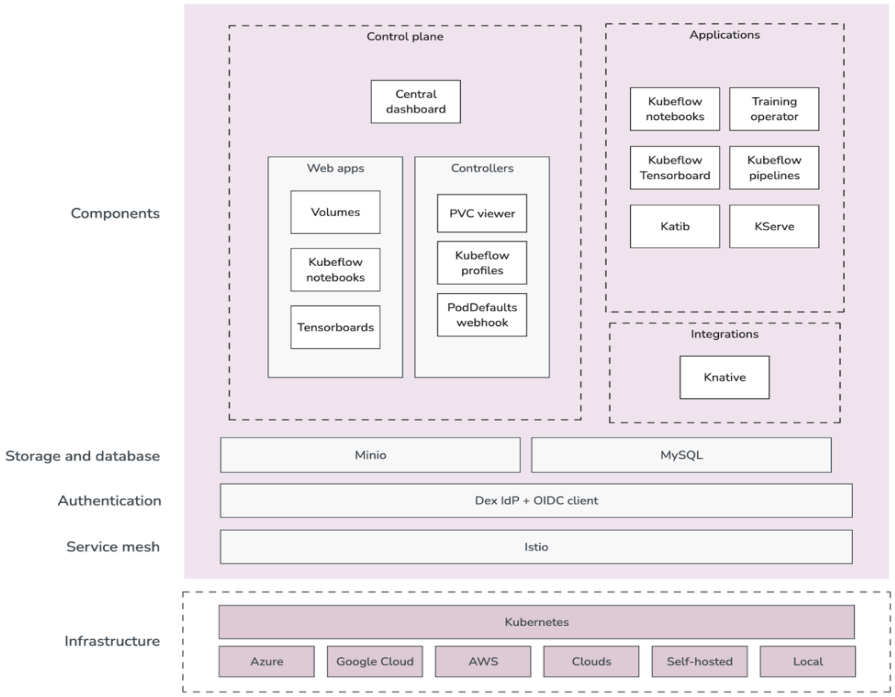


Figure 2. High-level architecture of Kubeflow on Kubernetes (illustrative example, based on Charmed Kubeflow). Kubeflow's control plane runs on Kubernetes and includes centralized services (authentication, UI dashboard, etc.), while various applications (notebook servers, training operators, pipelines, model serving via KServe, etc.) run as separate containerized microservices. The platform relies on the underlying Kubernetes cluster for scaling and workload orchestration, making it cloud-agnostic (able to run on Azure, Google Cloud, AWS, on-prem, etc.).

On the contrary, Kubeflow architecture (Figure 2) is constructed upon a cloud-agnostic Kubernetes layer that serves as an operating system of the containers of the ML workload (Zhou et al., 2019). Kubeflow in turn is made up of several components (highlighted as Applications in Figure 2) including the Notebook server, Training operator, Kubeflow Pipelines, Katib (AutoML tuner), and KServe

(inference server). These modules execute as pods on Kubernetes and use Kubernetes API and event queues to communicate. The control plane (Figure 2, left side) has a central dashboard with different controllers handling multi-user isolation (namespaces/profiles), volume management, and so forth. In terms of scalability, Kubeflow utilizes the existing capabilities of Kubernetes: e.g., the training operator can launch several worker pods and parameter server pods to a distributed TensorFlow job, and Kubernetes can use them on the existing cluster nodes (Jayaram et al., 2019). When a cluster itself is auto-scaling (on a Kubernetes service of a cloud provider), new nodes can be added to run additional pods, which has the effect of scaling the ML workload horizontally. Among the benefits that have been seen in the literature is that Kubernetes can co-locate various workloads and impose resource quotas, which can be effectively used in multi-tenancy (Xiong & Chen, 2020). Nevertheless, to do so, it must be configured cautiously; Xiong and Chen (2020) observe that in the context of multi-tenant AI platforms, it is difficult to assure isolation and trustworthiness because noisy neighbors or inappropriate quotas can affect performance. Kubeflow solves some of these issues with user isolation via Kubernetes namespaces and GPU scheduling and node affinity rules (to place ML pods in the most favorable location). Kubeflow is highly scalable - pod replicas can be configured, custom Kubernetes schedules can be used (such as gang scheduling with synchronous training jobs, as investigated by Jayaram et al. (2019)) and it can be configured to use cloud-specific auto-scalers. The trade-off is that Kubeflow requires more expertise to run; According to Lwakatare et al. (2020), organizations that implement such open-source MLOps platforms often have a platform engineering team.

Latency and Performance: SageMaker and Kubeflow strive to achieve low latency during model inference, but they use dissimilar methods depending on their philosophies. SageMaker as a fully-managed service brings with it optimizations under the hood, such as container warming, reducing model load time, and an inference cache of batch transform jobs. It also provides multi-model endpoints, in which a single serving container can serve multiple models and load them on demand, and less overhead is gained when working with many small models (Jiang et al., 2020). The AWS recommends invoking serverless with the built-in AWS Lambda + SageMaker endpoint integration as a best practice low-latency use of AWS, or high-throughput inference on Inferentia hardware, which are specialized. Overall, SageMaker endpoints have been demonstrated to deliver milliseconds response time on average predictions made through REST when properly scaled (Lee, 2020). The site additionally gathers a large amount of CloudWatch measurements (Figure 1 brushes on these: CPU/GPU usage, memory, etc.) such that engineers can recognize latency problems because of resource bottlenecks[7][8]. The low latency implementation of Kubeflow is built on such components as KServe (formerly KFServing), that uses Knative (a serverless platform based on Kubernetes) to auto-scale model servers as traffic increases (even to zero replicas) when traffic is absent (to save money) (Sedova et al., 2021). KServe also supports gRPC or REST inference along with canary rollouts; it can direct requests to multiple model versions. The networking and load balancing performance of the underlying cluster is one of the keys to latency in Kubeflow - an example is Kubernetes with Istio (a service mesh) to packet inference traffic (Figure 2 represents Istio as a member of the service mesh layer). Studies of Clipper container-based serving by Crankshaw et al (2017) reported that adaptive batching can significantly enhance throughput without breaching latency SLAs; KServe has an analogous concept, where a user can set a maximum batch size and a time-out limit on a model, which can be batched when possible. This has worked in CPU bound models or amortizing the overhead of launching kernels on GPUs. Latency reports published by Huang and Li (2019) show that a properly tuned Kubernetes serving system can perform on the same level as specializing serving systems, but there is usually a minor overhead (several milliseconds) on routing and service mesh proxies. Practically, fully-managed endpoints of SageMaker may experience initial cold-start latency (first invocation or scale-out) but thereafter, have constant low latency, whereas Kubeflow KServe may need additional Knative parameters (such as scaling pods rapidly up/down) to be carefully tuned (or not). Table 1 provides the summary of each platform with regards to scalability and latency and the fact that all of them make use of

containerization but SageMaker hides it whereas Kubeflow gives the user more control over it..

Cost Efficiency: Cost is the most important in enterprise ML and the platforms under examination have various philosophies. AWS SageMaker is a managed service that is proprietary in nature thus it is cost effective as it offloads the operation to AWS and enjoys the economies of scale. Billing is usually based on the compute (EC2 instances: notebooks, training, inference) and storage (S3) underutilized and might include a markup on managed capabilities. Another obvious cost implication is automated resource management: as mentioned above, SageMaker destroys training instances upon job completion, eliminating the possibility of an engineer not terminating an expensive GPU VM (a mistake that can be very costly)[3]. SageMaker also offers such tools as Amazon SageMaker Model Monitor to identify model drift or performance deterioration, which can be cost-efficient to charge early (however, Model Monitor itself is also a paid add-on) (Schmidt and Vollenweider, 2020). The study of cloud ML cost optimization by Kang and Kim (2018) indicates that spot instances or cheaper types of instances may be used wherever possible, which can help lower costs; SageMaker has managed spot training, which usually can save up to 70 percent of training job costs when interrupted. This is good practice when it comes to non time critical non-training activities. In the inference side, SageMaker multi-model endpoints and serverless inference (which were introduced in 2020) enables cost savings through optimization of infrastructure provided across multiple models and zero scaling when idle, respectively. The fact that Kubeflow is open-source does not imply any cost of use per se - the cost of using it is the cost of the infrastructure it is deployed on (e.g. cloud VMs or on-prem server). This is able to make Kubeflow appealing to companies that do not wish to be vendor-locked or use existing hardware. Nevertheless, the overall payback cost should include engineering work: to maintain a Kubernetes cluster, upgrade the Kubeflow components, and deal with downtime, professional resources are needed (Lwakatare et al., 2020). According to the literature, one of the benefits of Kubeflow is the ability to run on more inexpensive environments (including on-prem clusters or edge) and to integrate with cost-saving tools that are custom made (Huang and Li, 2019). An example would be to have an enterprise schedule non-urgent training jobs on a Kubeflow pipeline to be executed as an overnight job on spot instances or to take advantage of Kubernetes scheduling and pack multiple jobs on a single node to better utilize the node. According to Zhao and Li (2019), the cost of data transfer and privacy also should be addressed when working with MLaaS - Kubeflow can store data on-prem (therefore, no cloud egress costs are incurred), SageMaker would normally entail the data transfer to AWS. Table 2 contains a qualitative cost-related comparison, indicating that SageMaker is more convenient at a possibly increased direct compute cost and Kubeflow may be less costly by the expertise necessary to optimize underlying resources (and infrastructure is available).).

To illustrate these comparisons concretely, we present **Table 1** and **Table 2** below, which distill our findings on scalability, latency, and cost efficiency for SageMaker and Kubeflow:

Table 1. Qualitative comparison of SageMaker vs Kubeflow on scalability and latency.

Aspect	**AWS SageMaker** (Managed Cloud)	**Kubeflow** (Open-Source on Kubernetes)
Scalability	Automatically provisions and scales AWS instances for training jobs and inference endpoints. Supports distributed training (built-in for large datasets) without user managing the cluster. Scaling is mostly vertical/horizontal via AWS Auto Scaling (simple to use)[1].	Leverages Kubernetes' orchestration for scaling. Users can scale out by increasing pod replicas for training or serving. Supports custom scheduling (e.g., gang scheduling for distributed training) and can utilize cluster auto-scaler if enabled. Highly flexible but requires manual tuning and adequate cluster resources (Xiong & Chen, 2020).
Latency	Low-latency inference achieved via optimized endpoints (requests routed through ELB to containerized model servers). Provides features like Elastic Inference to attach GPU acceleration on	Low-latency inference via KServe, which auto-scales pods based on traffic. Uses Istio for routing; can scale to zero (serverless style), incurring cold starts when traffic resumes. Supports batch processing and GPU sharing on

Aspect	**AWS SageMaker** (Managed Cloud)	**Kubeflow** (Open-Source on Kubernetes)
	demand[4]. Cold start latency is minimal with persistent endpoints (serverless inference option has some startup delay). AWS handles networking optimizations.	Kubernetes. Latency depends on cluster network and Knative tuning; slight overhead from service mesh but can be optimized (Crankshaw et al., 2017).
Throughput	High throughput for training by distributed data parallelism (SageMaker can use Horovod or parameter servers internally). For inference, can deploy multiple instance replicas; load balancing is managed. Empirical reports show linear scaling of throughput with instance count until other bottlenecks (I/O) hit (Lee, 2020).	High training throughput if the Kubernetes cluster has sufficient nodes; Kubeflow training operators can utilize frameworks like MPI or Horovod. Inference throughput scales with number of pods; KServe's batching can significantly increase throughput for bulk predictions (Huang & Li, 2019). Throughput scalability is effectively bounded by cluster capacity and concurrency settings.
Fault Tolerance	Managed retry logic and logging. If a training instance fails, SageMaker can restart the job or save intermediate results (depending on configuration). Endpoint instances are in an Auto Scaling Group, so unhealthy ones are replaced automatically. Logging and metrics are integrated (CloudWatch) for debugging.	Relies on Kubernetes for fault tolerance: pod failures are handled by restart policies; if a node dies, pods are rescheduled on others. Checkpointing in training is up to the user (e.g., writing to persistent volume or object storage). Kubeflow Pipelines can retry failed pipeline steps. Requires setting up monitoring (Prometheus/Grafana) for insight.

Table 2. *Cost efficiency considerations for SageMaker vs Kubeflow.*

Aspect	**AWS SageMaker**	**Kubeflow**
Resource Utilization	Optimized by AWS – unused resources are turned off (e.g., training instances shut down when job completes)[3]. Multi-Model Endpoints serve multiple models per instance to increase utilization. Can use smaller instance types or inference acceleration to save cost. However, users pay for convenience and may over-provision if not careful.	User-controlled – can achieve high utilization by packing jobs on nodes (especially on-prem or fixed clusters). Idle components (e.g., idle notebook servers) will still consume resources unless auto-scaling to zero is set up. Users need to actively manage the cluster size or use cloud auto-scaler. No additional markup beyond infrastructure cost, so efficient use of resources can be very cost-effective.
Pricing Model	Pay-as-you-go for AWS resources: billed per instance-hour for training/serving, per GB for storage, etc. No upfront cost, but long-running services (e.g., always-on endpoints) incur continuous charges. SageMaker offers savings plans or uses spot pricing for training (up to 90% cheaper for interruptible jobs). The managed nature can hide some costs (e.g., CloudWatch logging) in convenience.	Open-source (no license fees). If running on cloud, costs are those of the VM instances, storage, and network egress. On-prem, costs are fixed hardware and maintenance. Kubeflow allows using preemptible/spot VMs for components to reduce cloud costs. Overall pricing is under user control – it can be as low or high cost as the chosen infrastructure and how efficiently it's utilized.
Ops & Maintenance Cost	Very low ops overhead – AWS handles patching, security, scaling. Engineering effort is minimal, which translates to indirect cost savings (fewer DevOps engineers needed for ML infra). However, this creates **vendor lock-in** and dependency on AWS	Higher ops overhead – requires DevOps expertise to install, configure, and maintain (Lwakatare et al., 2020). This is an internal cost (staff, training). However, it avoids cloud vendor lock-in and can leverage existing investments (e.g., using idle on-prem servers

Aspect	**AWS SageMaker**	**Kubeflow**
	services (Amershi et al., 2019). Migrating models out could be costly effort.	for training jobs). Updates/upgrades to Kubeflow are community-driven (potentially slower to get new features vs cloud).
Scalability of Cost	Can scale costs up or down linearly with usage. Good for startups or projects where usage is uncertain – you only pay for what you use, turning large capital expenditures into operational expenditures. High usage, however, can become expensive on AWS, at which point some companies consider moving to self-managed solutions for cost control (Zhang & Zheng, 2020).	Costs scale with the infrastructure size rather than directly with number of jobs. If an organization already runs Kubernetes, adding Kubeflow incurs little marginal cost beyond the workloads. For sporadic workloads, might not scale down as gracefully (unless using cloud auto-scaling aggressively). Over-provisioning the cluster means paying for unused capacity, so cost scaling depends on careful resource management by the user.

Discussion: The tables above summarize the manner in which SageMaker and Kubeflow deal with the fundamental parameters. SageMaker provides a more arm-length experience in terms of scalability - it hides the complexity of distributed systems, providing developers with easy APIs to train models with at a hundred machines or deploy to a scalable cluster. This is in line with the result of Baylor et al. (2017) that easing the interface (although performing heavy lifting in the background) is faster to deploy models in the industry. Kubeflow is more complex, but fits better with those organizations with heterogeneous requirements or those that require fine-grained control; its scalability is equal to that of the Kubernetes underlying it, which has demonstrated the ability to scale up even the production search and social media workloads (Burns et al., 2019). Latency angle demonstrates that both platforms can be configured to be low-latency - SageMaker with managed optimizations and hardware, Kubeflow with configurable autoscaling and customized inference servers. It is worth noting that the design of Kubeflow emerges as a reflection of the philosophy of modular by integrated referred to by Aman et al. - every part of the pipeline (data prep, training, serving) can be optimized and scaled separately, thus reducing the latency of each part of the pipeline (e.g., using GPUs where the part of the pipeline requires it, as opposed to having one large system). In terms of cost efficiency, our discussion is echoed by the real-world anecdotes: small teams tend to use SageMaker to get something running cheaply (to save on engineering cost), but as workload increases, the AWS bill might become large enough to consider running some workloads in cheaper Kubernetes clusters or adopting a hybrid approach (Zhang and Zheng, 2020). Conversely, major organizations with pre-existing cloud commitments could enter into discounts agreements with AWS, which would cause SageMaker to be inexpensive at scale, particularly, considering the lower ops overhead.

Model lifecycle and MLOps integration is another factor to be taken into consideration. Both SageMaker and Kubeflow discuss the ML lifecycle, though not equally strong. It makes SageMaker a one-stop-shop as it has an end-to-end suite (data labeling, model monitoring, pipelines, feature store) all on AWS. Kubeflow is compatible with numerous open tools (Spark, Argo, MLFlow, etc.), and this may be beneficial in case an enterprise already invested in those tools or requires to be customized. It is a two-sided sword though - unless managed correctly, a Kubeflow deployment may turn into a suite of isolated tools. Breck et al. (2017) and others have found evidence that it is essential to have consistency between stages and integration (such as making sure that the data preprocessing during training is precisely replicated during serving) in order to prevent errors. SageMaker attempts to achieve this by offering managed preprocessing (through processing jobs or notebooks), and embedding the preprocessing code into the model artifact, Kubeflow focuses on pipeline reproducibility (notebooks to pipelines to serving, with metadata tracking). These two methods are effective when the practice is rigorously undertaken; practically, businesses have been successful in either and as

long as they instill discipline (Amershi et al., 2019).

Conclusion

New AI platforms like AWS SageMaker and Kubeflow are now central to facilitating scalable and efficient implementations of machine learning in companies. This paper has analyzed their architectures, compared their performance in terms of scalability, latency and cost efficiency and learned best practices presented in the literature and experience in the industry. Overall, our results confirm that cloud-native design concepts - containerization, microservices and elastic orchestration principles are essential in managing the increased requirements of enterprise ML workflows. The managed platform of SageMaker shows how close collaboration between cloud services can significantly streamline the scaling process and lower maintenance costs in order to train and serve models to millions of requests with low latency (Hazelwood et al., 2018) and without concern with the underlying infrastructure implementation. Kubeflow, in its turn, demonstrates the strength of an open and modular platform: it can be customized to virtually any setup (on-premises or multi-cloud) and optimized at every stage of the ML lifecycle, a flexibility that is used by advanced organizations to adjust performance and ensure cost control (Zhou et al., 2019; Xiong and Chen, 2020).

We found that both platforms are capable of handling the needs of an enterprise, which includes automated provisioning on the global AWS infrastructure by SageMaker and scaling using Kubeflow using Kubernetes which can be extended to hybrid cloud environments. Specialization of inference serving systems and load-based autoscaling is nowadays standard best practice to support low latency (as implemented in both SageMaker endpoints and the KServe of Kubeflow). Those practices are present in these platforms, validating the findings of previous research such as Crankshaw et al. (2017) about containerized model serving and Olsson et al. (2017) about high-performance serving: which is that decoupling models into individual, optimized microservices and scaling them on their own is the key to achieving strict latency SLAs. Regarding cost efficiency, our study highlights the most important aspect of the studies of this category (Tang and Chen, 2020; Kang and Kim, 2018): just transferring ML to the cloud does not necessarily save money - the intelligent utilization of cloud-native capabilities does. The advantage of SageMaker users is the opportunity to pay a certain amount based on the quantity of usage and constant developing of AWS (such as the reduction of the instances cost or savings plans), though one must keep in mind to turn off inactive resources and to take advantage of such options as the spot instances. The Kubeflow users possess greater freedom to save money (such as using existing hardware or selecting the most cost-effective cloud per-task) but are required to invest in resource automation and monitoring so that to prevent resource sprawl and unutilized charges.

These works give rise to a number of best practices that can be made by practitioners. To start with, adopt automation and DevOps in ML (MLOps): formalize the ML workflow, e.g., with pipeline orchestration (e.g., SageMaker Pipelines or Kubeflow Pipelines) to capture the data ingestion to model deployment process. This is not only better to enhance reproducibility (Breck et al., 2017) but it is also easier to scale or migrate the workloads. Second, apply in-flight monitoring and model management: models with concept drift, data quality problems or performance regression in deployed models can be detected with tools such as SageMaker Model Monitor or the metadata tracking features of Kubeflow, which are essential to detect with long-running enterprise applications (Baylor et al., 2017). Third, scalability: Apply autoscaling to training (when possible) and serving. To illustrate, an endpoint autoscaler of SageMaker or Kubernetes Horizontal Pod Autoscalers can be used to make the system scale to load maintaining low latency and regulating cost in lulls. Fourth, look at hybrid approaches: most businesses do not consider a one-size-fits-all approach to be optimal. A hybrid between SageMaker and Kubeflow (or others) can work to the advantage of each - such as, rapid prototyping on SageMaker and then deployment on Kubeflow due to the cost factor, or Kubeflow on-prem due to sensitive data and bursting to SageMaker due to peak loads. Lastly, the development of the skills is essential: teams are to be trained on cloud-native principles (containers, Kubernetes) along with the tooling of the platform under consideration. According to Amershi et al. (2019), when it comes to Microsoft and its ML platforms, the biggest issue is not necessarily the

creation of such systems but the effective utilization of them by the teams.

References

[1] Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., … & Zheng, X. (2016). *TensorFlow: A system for large-scale machine learning*. In **Proceedings of OSDI 2016** (pp. 265–283). USENIX Association.

[2] Amershi, S., Begel, A., Bird, C., Gall, H., Kamar, E., Nagappan, N., … & Zimmermann, T. (2019). *Software engineering for machine learning: A case study*. In **2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)** (pp. 291–300). IEEE.

[3] Baylor, D., Breck, E., Cheng, H. T., Eng, M., Wilkiewicz, J., Haykal, S., … & O'Neill, B. (2017). *TFX: A TensorFlow-based production-scale machine learning platform*. In **Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '17)** (pp. 1387–1395). ACM.

[4] Breck, E., Cai, S., Nielsen, E., Salib, M., & Sculley, D. (2017). *The ML test score: A rubric for ML production readiness and technical debt reduction*. In **Proceedings of 2017 IEEE Big Data** (pp. 1123–1132). IEEE.

[5] Crankshaw, D., Wang, X., Zhou, G., Franklin, M. J., Gonzalez, J. E., & Stoica, I. (2017). *Clipper: A low-latency online prediction serving system*. In **14th USENIX Symposium on Networked Systems Design and Implementation (NSDI '17)** (pp. 613–627). USENIX Association.

[6] Dean, J., Corrado, G., Monga, R., Chen, K., Devin, M., Mao, M., … & Ng, A. (2012). *Large scale distributed deep networks*. In **Advances in Neural Information Processing Systems 25 (NeurIPS 2012)** (pp. 1223–1231). Curran Associates, Inc.

[7] Hazelwood, K., Bird, S., Brooks, D., Chinthamani, S., Diril, U., Hampton, C., … & Stanek, J. (2018). *Applied machine learning at Facebook: A datacenter infrastructure perspective*. In **2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)** (pp. 620–629). IEEE.

[8] Huang, S., & Li, J. (2019). *Scalable machine learning as a service for big data analytics*. In **2019 IEEE International Conference on Big Data (Big Data)** (pp. 120–129). IEEE.

[9] Jayaram, K. R., Muthusamy, V., Dube, P., Ishakian, V., Wang, C., Herta, B., … & Khalaf, R. (2019). *FfDL: A flexible multi-tenant deep learning platform*. In **Proceedings of Middleware 2019** (Article 27, 13 pages). ACM/IFIP.

[10] Kang, H., & Kim, J. (2018). *Cost-efficient resource management for machine learning as a service*. **ACM Transactions on Internet Technology, 18**(2), 22:1–22:20.

[11] Li, M., Andersen, D. G., Park, J. W., Smola, A. J., Ahmed, A., Josifovski, V., … & Su, B. Y. (2014). *Scaling distributed machine learning with the parameter server*. In **Proceedings of OSDI 2014** (pp. 583–598). USENIX Association.

[12] Lwakatare, L. E., Crnkovic, I., & Bosch, J. (2020). *MLOps: Practices and requirements for adopting machine learning in continuous software development pipelines*. **IEEE Software, 37**(5), 45–51.

[13] Olston, C., Fiedel, N., Gorovoy, K., Harmsen, J., Lao, L., Riddle, S., … & Tong, S. (2017). *TensorFlow-Serving: Flexible, high-performance ML serving*. **arXiv preprint arXiv:1712.06139**.

[14] Sculley, D., Holt, G., Golovin, D., Davydov, E., Phillips, T., Ebner, D., … & Dennison, D. (2015). *Hidden technical debt in machine learning systems*. In **Advances in Neural Information Processing Systems 28 (NeurIPS 2015)** (pp. 2503–2511).

[15] Sergeev, A., & Del Balso, M. (2018). *Horovod: fast and easy distributed deep learning in TensorFlow*. **arXiv preprint arXiv:1802.05799**.

[16] Tang, J., & Chen, L. (2020). *Machine learning as a service: A data-driven approach*. **IEEE Access, 8**, 151487–151499.

[17] Xiong, J., & Chen, H. (2020). *Challenges for building a cloud-native scalable and trustable multi-tenant AIoT platform*. In **Proceedings of**

ICCAD '20: IEEE/ACM International Conference on Computer-Aided Design (pp. 1–6). IEEE.

[18] Zhang, X., & Zheng, J. (2020). *Machine learning as a service: A survey*. **Journal of Software Engineering and Applications, 13**(4), 141–152.

[19] Zhao, J., & Li, P. (2019). *Secure machine learning as a service*. **IEEE Transactions on Cloud Computing, 7**(3), 778–790.

[20] Zhou, J., Velichkevich, A., Prosvirov, K., Garg, A., Oshima, Y., & Dutta, D. (2019). *Katib: A distributed general AutoML platform on Kubernetes*. In **Proc. of USENIX OpML 2019** (pp. 55–57). USENIX Association.

[1] [2] [3] [4] Deploying Models on AWS SageMaker - Part 1 Architecture - ML in Production

https://mlinproduction.com/sagemaker-architecture/

[5] [6] Low-Latency Model Serving with Clipper - RISE Lab

https://rise.cs.berkeley.edu/blog/low-latency-model-serving-clipper/

[7] [8] Amazon SageMaker AI metrics in Amazon CloudWatch - Amazon SageMaker AI

https://docs.aws.amazon.com/sagemaker/latest/dg/monitoring-cloudwatch.html