NextGen Scientific Publication
American International Journal of Computer Science and Technology

Volume 6 Issue 6, Pg. No. 37- 45, AIJCST-V7I6P104, 2025
https://doi.org/10.63282/3117-5481/AIJCST-V7I6P104

Original Article

Productionizing GPU Inference on EKS with KServe and NVIDIA Triton

* Babulal Shaik

Cloud Solutions Architect at Amazon Web Services, USA.

Abstract:

The increasing use of AI-based applications has brought about a necessity for operationalizing GPU inference on a large scale in production environments. But, the deployment and management of GPU-accelerated machine learning models in the wild are still major challenges that arise from the complexity of infrastructure orchestration, cost management, and model lifecycle automation. This paper is an exploration of a complete framework for productionising GPU inference on Amazon Elastic Kubernetes Service (EKS) with the help of KServe and NVIDIA Triton Inference Server, thus providing a simplified route from model deployment to large-scale checkpointed inference. Amazon EKS provides a controlled Kubernetes base that takes care of the automatic scaling, resilience, and security, whereas KServe makes it easy for the models to be served with the help of standardized APIs and native autoscaling for inference workloads. NVIDIA Triton complements this stack by offering the best performance, single or multi-framework, through GPU optimization, dynamic batching, and model ensemble all very important features to the maximum use of the hardware. They all together form a complete pipeline that is compatible with present-day MLOps practices, thus helping the continuous integration, model versioning, and automated rollouts. This paper also talks about the ways of keeping a balance between the performance and the cost such as GPU sharing, autoscaling policies, and efficient pod scheduling. Using this EKS–KServe–Triton trio, organizations can turn experimental ML models into production-grade, scalable inference services, thus closing the gap between model development and real-world deployment with a cloud-native, cost-optimized approach.

Keywords:

EKS, GPU Inference, KServe, NVIDIA Triton, Kubernetes, MLOps, Model Serving, Autoscaling, Deep Learning, A100 GPU, Performance Optimization, Model Deployment.

Article History:

Received: 16.09.2025

Revised: 20.10.2025

Accepted: 03.11.2025

Published: 15.11.2025

1. Introduction

Deep learning over the last ten years has shifted from being a research curiosity to the main driver of artificial intelligence (AI) applications. The deep neural networks that are powered by now are the ones that handle image recognition, natural language processing, recommendation engines, and fraud detection. Basically, they are everywhere where AI is used, making it possible for AI to go beyond the laboratory. But the computational part of the story has been following the evolution of deep models. To be more precise, GPUs are still the best choice for heavy truncations, while central processing units (CPUs) remain the preferred option for lightweight inference tasks. As a result, the inference part has witnessed a rapid GPU-backed infrastructure adoption for the model deployment that is particularly useful in production environments, where the features such as low latency, high throughput, and cost efficiency are mandatory. Nevertheless, a production environment where GPU inference is scaled has new challenges, and they are significantly bigger than those related to model training.

On the other hand, model deployment in research settings is usually a very simple procedure. Data scientists can achieve this either locally or on cloud-based notebooks by running experiments using static GPU instances. Production, however, is so different as it is harder to support workloads that are not improvable, latency constraints for efficient user-facing, and continuous integration of updated models. Managing this change, also known as the 'last mile' of machine learning, is an extremely difficult task. The issues, container orchestration, GPU scheduling, model versioning, observability, and cost control are just the beginning of what the teams have to deal with. Furthermore, the GPU resources are naturally costly, and improper utilization can elevate cloud spending to a large extent quickly. The capability of the elastic scale of GPU workloads that can be power, depending on the traffic patterns, and GPU model demand, is the key to keeping the production system sustainable.

This is the point at which Kubernetes and, more specifically, Amazon Elastic Kubernetes Service (EKS) fulfill the need for a very strong base. EKS does not only remove the majority of the difficult operations associated with running Kubernetes clusters, but it also allows the data science and MLOps teams to effortlessly deploy, scale and monitor their containerized workloads. The very own capabilities of Kubernetes, such as horizontal pod autoscaling, rolling updates, and declarative configuration, make it an ideal technology for running machine learning inference services. EKS takes it a step further by connecting thoroughly with all the other AWS services like Elastic Load Balancing, Identity and Access Management (IAM), and CloudWatch, which are essential in providing a stable and secure environment for GPU workloads that are also cloud-native. Teams can thus use EKS with GPU node groups, node affinity rules, and autoscaling policies to schedule GPU-powered pods in a cost-efficient manner as well as optimize the performance of the pods by scaling them as desired.

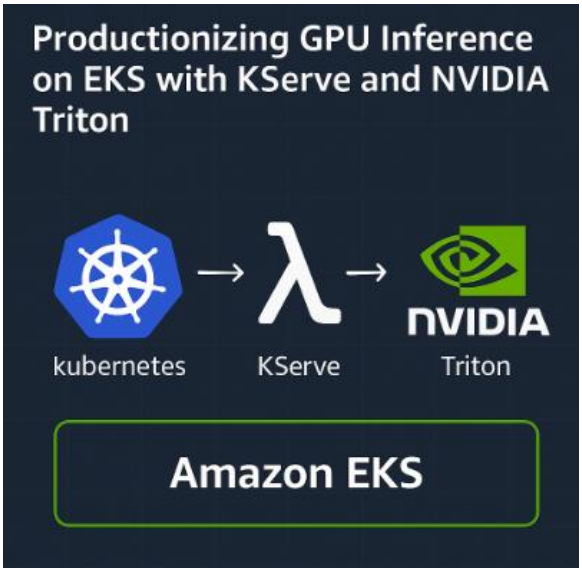

Figure 1. Productionizing GPU Inference on EKS With Kserve And NVIDIA Triton

Although Kubernetes makes the base for deployment and scalability, it is still hard to serve machine learning models efficiently at a large scale. The role of KServe and NVIDIA Triton Inference Server is then coming in. KServe is an open-source project under the Kubeflow project umbrella that helps with model deployment by hiding the complex serving logic behind the standardized interfaces and declarative specifications. The project supports numerous model formats and can be easily integrated with Kubernetes-native features like autoscaling and metrics collection. KServe not only supports MLOps workflows that are production-grade but also enables advanced features such as model canarying, rolling updates, and explainability endpoints to be aligned with them.

On the contrary, NVIDIA Triton Inference Server is the one that puts the focus on the performance and the hardware optimization. Triton supports multiple frameworks for deep learning i.e. TensorFlow, PyTorch, ONNX, and XGBoost, within a single runtime. It furthermore allows for the use of dynamic batching, concurrent model execution, and GPU sharing, which greatly enhances the throughput and reduces the latency. The combination of Triton and EKS when coupled with KServe as the orchestrator delivers the performance axis of the scalability of the inference systems thus allowing for very efficient use of GPU resources and at the same time easy meeting of the service-level objectives (SLOs).

2. The Need for GPU Inference in Production

As machine learning transitions from being just an experiment to a deployment at enterprise scale, the emphasis has been changed to the large models that are to be efficiently served in real-world production systems. Nevertheless, the training process is still very resource-heavy and computationally intensive but it is most often performed only once or a few times. On the other hand, inference is the process of continuously generating predictions, usually in real time, by using a trained model.

2.1. Inference vs. Training: Different Goals, Different Challenges

Model training and inference have completely different goal sets.Training is essentially an offline task, the main aim of which is to enhance the model accuracy by repeatedly passing the data, applying backpropagation, and updating the gradient. This process is usually operated in batch mode and can have longer computing times, as the goal is to raise the model's quality. On the contrary, inference is a real-time or almost real-time operation, where the speed is measured in milliseconds, and the cost per prediction has a direct impact on user experience and revenue.

In training, models run GPUs at or close to their full capacity. This is because the training is heavily computational and consists of large matrix operations. However, after deployment, the same models are often handling single inputs or small batches, which leads to GPU underutilization. Besides that, inference tasks need to be prepared for fluctuating request rates, different model versions, and varied input sizes. These factors make them unpredictable and therefore, they are difficult to be efficiently scaled.

Training procedures could be done on GPU clusters with fixed settings, but inference pipelines require elastic scalability, low latency, and cost-consciousness, which is a combination that calls for both smart orchestration and optimized serving infrastructure.

2.2. Bottlenecks in GPU Inference Workloads

Several bottlenecks arise when GPU inference is deployed in production environments:

- Latency and Throughput Constraints: The inference latency, i.e., the time taken by a model to give an output, is the most direct measure that impacts user-facing applications. For example, in natural language models that are used to power chatbots or real-time translation systems, even a delay of a few hundred milliseconds can spoil the user experience. Consequently, GPU inference must achieve a delicate balance between parallel execution (to expand throughput) and short response times (to maintain responsiveness).
- Memory Footprint and Resource Contention: Deep learning models, particularly transformer-based architectures such as BERT or GPT, are of tremendous memory footprints. Hence it is not a surprise that running multiple models or high concurrency can occupy all the memory on your GPU in no time which eventually will lead to errors or the GPU slowing down. Thus, without proper memory management, most of the time, teams are forced to over-provision GPUs—wasting more resources than necessary and increasing operational costs.
- GPU Underutilization: The paradox is that despite the high cost of GPUs, they are still underutilized or idle most of the time during inference. The idle situation occurs when, among other things, the workloads are not steady, batch sizes are small or requests happen at different locations at different times. The non-use of the GPU accounts for a hidden inefficiency in terms of cost, as the organizations still have to pay the full rent for that compute capacity, which is not fully accessed.
- Deployment and Versioning Complexity: The operational complications of managing different model versions, frameworks, and runtime dependencies are quite significant. In other words, every model comes with its own libraries and preprocessing steps as well as configurations; therefore, the process of deployment pipelines can be really a pain. Containerization mainly aims at bringing standardization to such environments, though it still requires deployment across clusters and their dynamic scaling, which means you need more professional, advanced management, such as Kubernetes-based inference frameworks.

2.3. Common GPU Inference Workloads

GPU inference is good for a variety of production workloads that use deep learning:

- Natural Language Processing (NLP): Trying to understand what human beings say or write has been improved by the so-called transformer models that are used in making speech recognition, summarization of texts, and AI generative by the help of GPUs. Real-time NLP inference requires that both low latency and high throughput are given so that concurrent user sessions can be handled.

- Computer Vision: CNNs (Convolutional Neural Networks) have become the underpinning of one category of tasks named Computer Vision. They can do image classification, object detection, and video analytics only with the help of CNNs. These jobs have to deal with a lot of pixel data; that's why parallel processing on GPUs with their high performance makes it perfect.
- Recommender Systems: Personalized recommendations in e-commerce, streaming platforms, and social media require large-scale matrix factorization and embedding lookups—operations well-suited for GPU acceleration. Anyway, these systems should also be isolated so that they can manage varying user traffic patterns.
- Generative AI and Multimodal Applications: The main characteristic of inference in recent times is heterogeneity brought by combining text, vision, and audio processing along with the growth of big language models and image generation systems. Doing it at scale with all these different types of inference makes the problem of orchestration even more difficult.

The common features of these workloads are that they require predictable performance, low-latency responses, and cost-efficient scaling, which is a mix that calls for careful system design and smart GPU management.

2.4. Balancing Cost and Performance: Elastic Scaling and GPU Sharing

GPUs that are running at their full capacity should not be used continuously. A production environment that is effective should use GPUs in a dynamic manner so as to match the demand of the workload. Elastic scaling, which is the idea, allows for the provision of or the cancellation of GPU pods, depending on the fluctuation of the traffic in real time, hence, the minimum possible time for the GPUs to be left idle and reduced costs.

On the other hand, one more technology that is aimed at better usage of GPU resources is GPU sharing, in which several inference processes or containers that are stored on the same GPU can operate simultaneously. Methods, as an example, fractional GPU allocation and multi-instance GPU (MIG) coming from NVIDIA, give the possibility of dividing GPU resources among different workloads. Consequently, small inference tasks that cannot fill up a whole GPU can operate side by side in an efficient manner, thus saving a lot of the cost that would be used otherwise without performance getting compromised. Thus, cost-performance optimization is not only a matter of reducing the cost; rather, it is about developing reliable, scalable inference pipelines that are in tandem with the service-level objectives.

2.5. Why Kubernetes Matters: Portability and Resource Abstraction

Kubernetes is the de facto container orchestration standard that has been embraced globally by most of the organizations spanning industries. The primary reasons are the benefits one gets from using kubernetes, and the GPU inference workload does not remain an exception at all. In principle, Kubernetes is a way of eliminating the binding between software and hardware or vendors and vendors, thus permitting the users to specify their workloads in YAML files for achieving the required state. This unique functionality of Kubernetes is a real game-changer for the organizations, as it allows them to execute inference services via a hybrid or multi-cloud set-up, which is not only extremely convenient but also ensures zero downtime.

Let's take GPU workloads as an example. Here, Kubernetes device plugins are the channels through which GPU resources reach the containers. Besides, device plugins are the secret agents who take care of managing the scheduling, isolation, and the lifecycle of GPUs. Thus, data scientists and MLOps engineers can focus on just refining the model and logic they are serving instead of low-level infrastructure worries. Furthermore, with the support of Amazon EKS, there could be even more enhancements to the above features because of the managed cluster operations, automated node scaling, and the linkage with AWS monitoring and identity services.

To put it all together, the need for GPU inference in production, best as is the case, doesn’t revolve around the hardware specs alone; rather, the setting of an ecosystem that efficiently manages performance, elasticity, and manageability plays a major role. The upshot of this is that Kubernetes together with MLOps libraries like KServe and NVIDIA Triton does the trick of enabling GPU-powered models to be converted into those production services which are scalable, reliable, and cost-effective.

3. Overview of the Core Components

It is necessary to have a complex orchestration stack that manages the three accounts of elasticity, reliability, and performance simultaneously to be able to deploy GPU-accelerated machine learning inference on a large scale. Such a system is fulfilled by a strong, production-worthy framework built by Amazon Elastic Kubernetes Service (EKS), KServe, and NVIDIA Triton Inference Server. The

roles that these three components perform are very different but at the same time they interact in the GPU inference lifecycle, from infrastructure provisioning and scaling to efficient model serving to the optimization of the hardware execution.

3.1. Amazon EKS for Scalable Deployment

This architecture is built on an Amazon Elastic Kubernetes Service (EKS) basis, which is a fully managed service that eases the operation of Kubernetes clusters in the AWS cloud. Kubernetes alone offers a declarative, portable platform for container orchestration—that is, the automation of deployment, scaling, and management of applications. EKS gives access to this power by taking care of control plane functions where the likes of cluster provisioning, patching, and upgrades are performed, hence allowing the engineering teams to put their time and effort on workload optimization rather than infrastructure management.

Among other things, EKS is really good at exploiting the AWS ecosystem to the maximum for stable and secure operations. It connects AWS Identity and Access Management (IAM), which gives the user control of the access rights, Elastic Load Balancing (ELB), which distributes the incoming traffic of the cluster, and CloudWatch for monitoring cluster performance. These integrations ensure observability, resilience, and compliance—all necessary characteristics for production-level ML inference.

In the case of GPU inference, EKS is compatible with the use of NVIDIA GPU-enabled EC2 instances such as the p3, p4, and p5 families which are designed to be as efficient as possible with machine learning workloads. You can set these kinds of instances as part of certain node groups in your cluster therefore creating an option to separate GPU workloads from CPU-only compute resources. With the help of the NVIDIA device plugin for Kubernetes, the GPUs can be accessed the same way as the schedulable resources are to pods thus allowing for precise allocation and as well as utilization across workloads. In order to be certain that the project is elastic, EKS has teamed up with the Cluster Autoscaler and Karpenter, which adjust the number of worker nodes dynamically. They calculate necessary resources for projects. In case heavy traffic hits the inference, new GPU-backed nodes can be provided without delay. There is a small part of the cluster that is passively waiting for a new task to occur but at the moment the traffic is decreasing so idle nodes will be decommissioned; thus, you will not be charged for that resource.

3.2. KServe: Serverless Model Serving

KServe is an open-source project under the Kubeflow umbrella, designed explicitly for machine learning model deployment and management on Kubernetes. Through a user-friendly interface known as an InferenceService, it simplifies the complexities of serving infrastructure, allowing users to define model deployment, scaling, and exposure, all via simple YAML specifications. Technically, the InferenceService is at the core of the KServe design and Architecture. It is the component that is responsible for the deployment of the model. The InferenceService comprises multiple key components such as

- Predictor: The component from the core is in charge of hosting the model and the whole process of inference. Predictors could rely on backends like TensorFlow Serving, TorchServe, or NVIDIA Triton to run the inference request.
- Transformer: Deals with all the input and output processing, such as tokenization, normalization of data, or decoding of output. Such a design pattern allows the use of complex model pipelines executed inside the Kubernetes ecosystem, thus making it more straightforward and more efficient.
- Explainer: Offers interpretability by suggesting the reasons for the model decisions—a feature that is of great importance in the process of compliance, debugging, and user trust in AI systems.

KServe permits serverless model serving, which means that models are automatically scaled up when requests are received and scaled down to zero when there are no requests. Such a mode of operation, supported by Knative, allows teams to lower their expenses without losing their responsiveness. Moreover, KServe has the capability to carry multi-model serving that gives the possibility of several models in the same inference service to be at rest - the best example is a situation where we have to manage hundreds of small lightweight models concurrently.

3.3. NVIDIA Triton Inference Server

The NVIDIA Triton Inference Server is the main component that helps to achieve the best GPU utilization in the EKS-KServe ecosystem. Triton is an open-source high-performance inference serving platform that allows deployment of various models on one GPU infrastructure without any compatibility issues as it supports multiple deep learning frameworks. Triton's major benefit is that it supports more than one framework i.e. it can serve models that are built with TensorFlow, PyTorch, ONNX Runtime, TensorRT, XGBoost, and even custom Python backends at the same time. Therefore, we do not need separate serving stacks for each model type

and it makes the ML environment simpler to manage. Storing models is done in a model repository, which is a standardized directory structure that Triton keeps checking for updates. This arrangement enables teams to add, update, or remove models on the go without having to restart the inference server, thus facilitating seamless CI/CD integration and continuous deployment. One of the most attractive features of Triton is dynamic batching where several inference requests are cleverly combined to form a single batch at the time of runtime. As a result, a big volume of work is done efficiently without needing to make changes on the client-side, particularly for those models where batching does not have an impact on latency-critical performance. Together with dynamic batching, the concurrent model execution allows several models or even different copies of the same model to be working side by side on different GPUs or GPU streams, thus no compute resources go wasted.

Advanced memory management and optimization for GPUs are among the features that Triton has been upgraded to. The features such as model instance groups and memory pinning are utilized to mix workloads over GPU memory as well as compute cores. Through the support of NVIDIA Multi-Instance GPU (MIG), Triton can create multiple separated partitions from one GPU, thus allowing several lightweight models to share the same physical GPU without any trouble. This function is definitely important in multi-tenant environments or cost-efficient production systems. The monitoring and logging attributes of Triton, for instance, the integration with Prometheus and NVIDIA DCGM (Data Center GPU Manager), give the deepest possible visibility for inference performance, GPU health, and utilization metrics. Such a high level of observability lets teams become data-driven in their decisions regarding the scaling, budgeting, and workload allocation. KServe on EKS, together with Triton, is what makes the perfect high-performance inference backend that is able to provide stable, low-latency predictions at large scales. They are a comprehensive, cloud-native stack, which simplifies infrastructure management whilst delivering the best GPU performance that is customer-friendly and, thus, a big step in the operationalization of AI in production, further solidifying enterprise trustworthiness, agility, and effectiveness.

4. Building the Production Pipeline

To effectively use a GPU for inference in production, a carefully designed pipeline that integrates infrastructure provisioning, model deployment, scaling, monitoring, and continuous delivery seamlessly is necessary. Such an end-to-end system that is modular, elastic, and fully automated can be built with the combination of Amazon EKS, KServe, and NVIDIA Triton Inference Server. The stages of implementing this GPU inference pipeline for production purposes sequentially from cluster setup to model deployment, scaling, observability, and CI/CD automation are described here.

4.1. Infrastructure Setup on EKS

A properly configured Amazon EKS cluster is the base for building a scalable GPU inference pipeline. Both CPU and GPU node groups should be included in the cluster; this way workloads could be effectively divided according to their resource requirements. Generally, GPU nodes are set up with EC2 p4 or p5 instances, carrying the NVIDIA A100 or H100 GPUs that are specialized for deep learning workloads. These nodes can be created by AWS CloudFormation, eksctl, or Terraform; thus, the deployment of the infrastructure can be completely automated and replicable. Workload node scheduling cannot be accurate and specific without node labeling and node tainting. Assigning GPU node labels such as accelerator=gpu or workload=model-serving allows Kubernetes to schedule pods that depend on GPU resources only on these nodes. Taints thus stop non-GPU workloads from being placed on GPU nodes unless explicitly tolerated, thereby ensuring that the expensive GPU resources are saved for inference tasks.

After GPU nodes have been set, the NVIDIA Kubernetes device plugin must be installed. This plugin is the one that shows the GPU resources to Kubernetes groupmates, such as through resource specifications (e.g., resources.limits.nvidia.com/gpu: 1). The plugin also fulfills the function of device discovery and device isolation, thereby letting multiple inference containers that are technically co-located on the GPU hardware share it without conflict if the GPU model allows for such. Having done with the cluster configuration, the EKS cluster is now a GPU-aware orchestration platform that is capable of scheduling, scaling, and managing deep learning inference workloads. This allows one to proceed with the deployment of model-serving workloads using KServe and NVIDIA Triton.

4.2. Autoscaling and Load Balancing

One of the main features that a production inference system must have is the property of elasticity, that is the capability to increase the resources on the system when the load is high and, at the same time, reduce the resources when there is a period of inactivity. KServe has great support for autoscaling which is achieved not only by a Knative Pod Autoscaler (KPA) but also by a Horizontal Pod Autoscaler (HPA) mechanism. The KPA is on the side of request-driven autoscaling. It scales inference pods depending

on a live request concurrency and traffic volume, thus, on the basis of real-time data, new replicas are added automatically when the number of requests increases whereas scaling down to zero is made when there are no active requests. This is a serverless scaling model that can be very beneficial when it comes to applications which demand fluctuation, because in that case the cost is minimized whilst responsiveness is maintained. On the other hand, HPA is a metric-based method and thus you can use it when the scaling per resource utilization is enough. For example, you can scale the pods based on GPU, CPU, or memory utilization or by using a certain custom metric. In the case of GPU workloads, scaling with GPU utilization as a parameter endows one with the advantage that new pods will be started only if the current GPUs are close to being run at full capacity, and in that way, one can maximize and make efficient use of the resources. Istio is responsible for load balancing. It handles the distribution of the traffic across the replicas as well as the managing of the retry logic. Moreover, it gives the feature of weighted routing that can be used in canary deployments, thus, allowing a small percentage of the traffic to be routed to the new model versions before the complete rollout. Hence, risk is reduced and performance benchmarking in real production condition is enabled. The utilization of these autoscaling and load-balancing techniques allows the inference pipeline to not only save the cost but also to be highly available which, in turn, brings the GPU resources to be aligned with the real-time demand patterns in a dynamic manner.

4.3. CI/CD for Model Deployment

To have a model deployment that is frequent and dependable, a CI/CD (Continuous Integration and Continuous Deployment) pipeline is necessary. Currently, MLOps use GitOps tools like ArgoCD or Flux for easy and controlled version in Kubernetes environments besides model deployment. The GitOps workflow saves model parameters, e.g., KServe InferenceService YAML files, in the Git repository. Upon receiving a new model version (e.g., a new model file in S3 or an updated YAML definition), the change is immediately detected by ArgoCD, which then replicates it with the EKS cluster. Hence, a declarative consistency between source control and the live environment is maintained. KServe integration with Istio makes automated blue-green and canary deployments possible. The new model versions are introduced alongside the existing ones and gradually the traffic is switched over to the new version. If such a situation occurs wherein the performance metrics decline or errors are produced, the rollback mechanisms return the previous stable configuration - hence, the downtime and operational risk are at their minimum. Besides, the CI/CD pipeline for model testing and validation should not be overlooked. Validation by the pipeline can be automated before the deployment stage, whereby the script can check the model for accuracy, input/output compatibility, and the desired behavior of the staging clusters. After all the tests are passed, the model is then released to production.

5. Case Study: Deploying a Real-Time Vision Model

5.1. Scenario Overview

The case study narrates the introduction of a real-time object detection service based on the YOLOv5 (You Only Look Once) model using Amazon EKS, KServe, and NVIDIA Triton Inference Server. It explains a GPU inference productionization scenario in the wild with a very simple tool called YOLOv5. A newly developed convolutional neural network for object detection is the first choice of real-time video analytics, self-driving cars, and surveillance systems due to its precision and speed. The goal of this work is the implementation of a YOLOv5 model as an on-the-fly image stream processor which gives bounding boxes and class labels as output with minimal latency. This mentioned deployment satisfies the user needs in terms of obtaining scalability, low latency, and cost-effectiveness via Kubernetes-native orchestration, GPU optimization, and automated scaling.

5.2. Architecture Implementation

The deployment architecture for the YOLOv5 inferencing service is effectively layered on top of Amazon EKS, using GPU-enriched EC2 instances from the p4d family, with configurations that include NVIDIA A100 GPUs. Two separate node groups were created inside the EKS cluster, one for GPU workloads and the other for CPU-based auxiliary services (such as preprocessing and logging). Embedded capacity of the YOLOv5 model was performed externally and the trained files were uploaded into the Amazon S3 bucket. The model-registry implementation was enabled by S3 versioning in order to contain numerous model versions. Each version was marked with metadata that included the accuracy, dataset version, and timestamp, tagging it thereby making rollback and controlled promotion of models to production quite effortless. Taking advantage of Knative and Istio, KServe automatically provisioned the needed services, ingress routes, and load balancing for inference traffic. The Triton backend was tailored to utilize the dynamic batching and concurrency optimization so as to maximize the GPU's computational potential. The requests entered through an Istio ingress gateway that distributed and authenticated the traffic. This modular design thus ensured the whole pipeline to be fault-tolerant, scalable, and easy to MLOps workflow maintenance.

5.3. Cost Optimization Strategies

Although GPU instances can offer tremendous output, their costs can grow quickly if they are not properly used. The cost-performance balance in the YOLOv5 deployment pipeline was the main focus that led to the implementation of various measures.

- Spot Instances for GPU Workloads: Non-critical inference pods were allowed to run on AWS EC2 Spot Instances in the scenario where the deployment of the corresponding tasks on on-demand instances is not obligatory. This step led to instance costs having been cut by up to 70% as compared with on-demand pricing. The pod disruption budgets of Kubernetes and the autoscaler of KServe allowed the smooth passing of control to on-demand nodes when the capacity of the spot had been taken away, thus not interrupting the service.
- Pod Autoscaling Strategies: The setting of autoscaling was adjusted to confirm that the GPU pods would scale up only when the request concurrency goes beyond the established thresholds. Prometheus was used to monitor the necessary parameters of GPU utilization and request latency. When the performance had been kept below 40% for a long time, the pods were automatically scaled down, which was a money-saving way in the non-peak hours. On the contrary, during the periods of high traffic, through the integration of EKS with Cluster Autoscaler, new GPU-backed pods were deployed immediately.
- Multi-Model Sharing on a Single GPU:Triton's model instance groups and Multi-Instance GPU (MIG) support were used to deploy multiple YOLOv5 model variants (e.g., small, medium, and large) on the same GPU. This enabled multi-tenant serving; thus, the need for additional GPU nodes was minimized. Low-resolution streams were used to train lightweight models that were served on shared GPUs concurrently with the high-precision models for better utilization without latency compromise.

These initiatives were combined to cut down the GPU spend by 40-50% and at the same time, there were no big losses in the provision of the service as well as the real-time performance it required. The implementation of the best practices of spot provisioning, intelligent scaling, and multi-model co-location led to the YOLOv5 service hitting the sweet spot of scalability, cost control, and high-performance GPU inference on Amazon EKS.

6. Challenges and Best Practices

While using GPU inference on Amazon EKS with KServe and NVIDIA Triton is a great choice for scalability and performance, the operational challenges can arise significantly. In these cases, being able to solve them appropriately and effectively is the key to preserving the system's reliability, optimizing the usage of resources, as well as ensuring delivery of models at the top of their security and performance level.

- Cold Start Times for GPU Pods: One of the major issues that users encounter is cold start latency or slow startup, when a scaled-down GPU pod needs to be brought to an operational state. In general, the process of getting the GPU node up and running may consume more resources, and besides that, the model loading and runtime initialization may be required, so the startup time can extend to several minutes.
- Model Loading Latency and Caching Strategies: The storage and retrieval of large models such as YOLO, BERT, or GPT derivatives from remote S3 buckets can be a very time-consuming process; thus, model loading will slow down not only the startup process but also the switching of the models. One of the good practices is to use Amazon FSx for Lustre or EFS in order to have the models mounted near the computing nodes so that the access is fast and the required throughput is high. Model caching as well as lazy loading can be set up in Triton so that the models that are trended in memory and that are most frequent in accessing have very short reload times. Moreover, with multi-model repositories, the executing of models running in parallel is possible without the overhead caused by reloading.
- Managing Dependencies and Container Size: Deep learning frameworks, preprocessing libraries, and custom dependencies are the main reasons for the huge size of inference containers. The oversized containers lead to a slow pod startup and image pulls. To solve this problem, the teams need to start using multi-stage Docker builds, isolate model-specific logic from base frameworks, and make use of lightweight CUDA base images. Apart from that, frequent dependency audits and pruning of frameworks are two activities that can bring faster deployments and smaller attack surfaces along.
- Resource Fragmentation and GPU Memory Partitioning: Inefficient GPU utilization may cause resource fragmentation, thus some parts of GPU memory remain unused. The employment of NVIDIA Multi-Instance GPU (MIG) and Triton model instance groups allows the division of one GPU into multiple smaller, separate compute units, whereby the coexistence of multiple lightweight models is possible. Proper resource requests (e.g., nvidia.com/gpu: 0.25) are helpful for the balanced scheduling of the resources and saving of money.
- Security and IAM Roles for Model Access: The issue of deployment security is, unfortunately, not given due consideration most times. The models, when stored in S3 or FSx, should be accompanied with IAM roles as service accounts (IRSA) that

allow only pods that are authorized access and no one else. Apart from that, network policies and Istio mTLS can enhance the security of the communication that is happening between inference services.

7. Conclusion and Future Directions

The setup which combines KServe and NVIDIA Triton Inference Server on Amazon EKS represents a powerful, cloud-native base that can be used to efficiently deploy and scale GPU inference workloads in production. This combined stack makes it possible to discard the troubles associated with the infrastructure while maintaining value in terms of speed, scalability, and cost-effectiveness. In this system, container orchestration and node scaling are managed by EKS, model serving and traffic management is simplified by KServe, and inference execution is achieved through facilities provided by Triton thus enterprises get to deeply realize the potential of deep learning models without any glitches.The architecture serves as a milestone for organizations to take up AI and put it into production. Regular deployment cycles are not only made faster, but real-world AI services also gain from this by being provided with predictable performance, high availability, and cost control. What is more, the use of declarative configurations, autoscaling, and observability tools makes it possible to achieve seamless integration and monitoring of models within MLOps pipelines.The next step in the evolution of GPU inference on Kubernetes, allowing for greater throughput and latency performance, would be future developments like multi-GPU scheduling, model parallelism, and inference caching. It will also involve the use of emerging technologies such as GPU virtualization and distributed inference frameworks for better flexibility of large-scale deployments.Consequently, the first big production-ready move for teams should be to make container builds that are optimized for their purpose, have proactive scaling strategies in place, pay special attention to security, and, of course, do a lot of monitoring. When the EKS-KServe-Triton ecosystem is fully functioning, the organizations are given a tool that can close the gap between AI innovation and the arrival of the reliable, scalable deployment option thus, the intelligent models are converted into production-grade business value.

References

[1] Dhakal, Aditya, Sameer G. Kulkarni, and K. K. Ramakrishnan. "Gslice: controlled spatial sharing of gpus for a scalable inference platform." *Proceedings of the 11th ACM Symposium on Cloud Computing*. 2020.

[2] Véstias, Mário. "Processing systems for deep learning inference on edge devices." *Convergence of Artificial Intelligence and the Internet of Things*. Cham: Springer International Publishing, 2020. 213-240.

[3] Kim, JooHwan, Shan Ullah, and Deok-Hwan Kim. "GPU-based embedded edge server configuration and offloading for a neural network service." *The Journal of Supercomputing* 77.8 (2021): 8593-8621.

[4] True, Thomas, and Gareth Sylvester-Bradley. "An Edge Processing Platform for Media Production." *SMPTE 2022 Media Technology Summit*. SMPTE, 2022.

[5] Minakova, Svetlana, Erqian Tang, and Todor Stefanov. "Combining task-and data-level parallelism for high-throughput CNN inference on embedded CPUs-GPUs MPSoCs." *International Conference on Embedded Computer Systems*. Cham: Springer International Publishing, 2020.

[6] Jani, Yash, and Arth Jani. "Robust framework for scalable AI inference using distributed cloud services and event-driven architecture." (2024).

[7] Lu, Chengzhi, et al. "SMIless: Serving DAG-based Inference with Dynamic Invocations under Serverless Computing." *SC24: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2024..

[8] Wilkins, Grant. "Online Workload Allocation and Energy Optimization in Large Language Model Inference Systems." (2024).

[9] Koubaa, Anis, et al. "Cloud versus edge deployment strategies of real-time face recognition inference." *IEEE Transactions on Network Science and Engineering* 9.1 (2021): 143-160.

[10] True, Thomas, and Gareth Sylvester-Bradley. "COTS (commercial-off-the-shelf) platform for media production everywhere." *SMPTE Motion Imaging Journal* 132.2 (2023): 15-25.

[11] Iusztin, Paul, and Maxime Labonne. *LLM Engineer's Handbook: Master the art of engineering large language models from concept to production*. Packt Publishing Ltd, 2024.

[12] Zhang, Yi, et al. "Flattenquant: Breaking through the inference compute-bound for large language models with per-tensor quantization." *arXiv preprint arXiv:2402.17985* (2024).

[13] Ardestani, Ehsan K., et al. "Supporting massive DLRM inference through software defined memory." *2022 IEEE 42nd International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2022.

[14] Jain, Rishabh, et al. "Pushing the Performance Envelope of DNN-based Recommendation Systems Inference on GPUs." *2024 57th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2024.

[15] Wilkins, Grant, Srinivasan Keshav, and Richard Mortier. "Offline energy-optimal llm serving: Workload-based energy models for llm inference on heterogeneous systems." *ACM SIGENERGY Energy Informatics Review* 4.5 (2024): 113-119.