# Improving Model Deployment Pipelines for Efficiency in Cloud-Based Machine Learning Platforms

**Sanjeev Kumar** *sanjeevkumar.sk@ieee.org*
*Independent researcher*
*SME in Cloud Engineering*
*Georgia, USA*

**Abstract**

Thus, increasing demand for the cloud-based machine learning solution is highly pushing the focus forward into making deployment pipelines for models efficient. These pipelines are very important to get a trained model to scale, provide real-time predictions, and manage the cloud infrastructure complexities in general. This paper reports on strategies improving model deployment pipelines on cloud-based ML platforms centered around automation, monitoring, and resource optimization. We investigate current tools, such as containerization, serverless computing, and CI/CD frameworks for streamlined transition pipelines through development and production. We also investigate how superior monitoring tools support the best possible resources allocation while keeping downtime at its lowest and latency low. It discusses case studies from top cloud providers and creates an optimized architecture model, especially suited to varied applications. Our experiments demonstrate that the optimized pipelines can show up to an order of magnitude improvement in terms of deployment speed, model performance, and cost effectiveness, providing a robust basis for scaling ML solutions in the cloud. Finally, we point out some of the limitations of current approaches and outline areas of future research as one considers expanding deployment pipelines in increasingly complex cloud environments.

**Keywords:** Model Deployment, Cloud-based Machine Learning, CI/CD Pipelines, Serverless Computing, Resource Optimization.

## 1. INTRODUCTION

Indeed, cloud computing has revolutionized the new landscape of machine learning by providing scalable flexible and on-demand access to computational resources. Such deployment enables more efficient deployment of machine learning models toward high availability and elasticity with powerful data processing. Thus, there is no need for an investment in physical infrastructure. Despite this potential, deploying machine learning models on cloud platforms presents unique. Machine learning deployment pipelines define the workflow of moving a trained model from a development environment to production, where it serves real-time predictions. These pipelines are integral to ensuring that models can operate at scale, accommodate traffic fluctuations, and maintain operational efficiency.An inefficient pipeline can lead to unnecessarily long model update cycles, poor resource management, increased costs, and suboptimal model performance, as noted in [1]. Therefore, optimization of efficiency has become a significant issue with cloud-based ML solutions, which include the respective deployment pipelines.

A traditional machine learning-based deployment pipeline includes a train-test-validate-deploy process. Challenges in the cloud environment include managing the complexity of distributed systems, ensuring infrastructure availability, and scaling resources based on workload demand, as highlighted by studies in [2]. Secondly, ML models generally require frequent updating since new data is received or needs to be retrained for maintaining predictive accuracy. These updates introduce risks such as downtime, version mismatches, and delayed deployment cycles, as noted in [3].Cloud-based ML platforms, such as Amazon SageMaker, Google AI Platform, and Microsoft Azure Machine Learning, offer tools and services to facilitate deployment, as emphasized in

[4].For instance, while these tools provide some level of readiness for model deployment, significant challenges remain, particularly regarding model inference latency, scalability, and multi-model system management, as discussed in [5]. Scaling up or down machine learning models is very important in a cloud environment. It requires well-coordinated resource allocation so that compute, storage and networking infrastructures go together without overspending or sacrificing some level of performance, a concern by [6].

Model deployment pipelines deployed efficiently have many benefits, especially for cloud-based setups. First, they make the process of putting models into production without hiccups thereby leaving fewer chances for machine learning initiatives to generate value slowly, as it has been seen in [7]. In finance, healthcare insurance, or e-commerce companies where decisions are often made based on predictions created at the time, speed of model deployment is highly important in maintaining a competitive edge as stated in [8]. This is also very effective from the point of view of cost efficiency. Cloud platforms bill by use of the resource and in case an inefficient pipeline does not scale up the resources dynamically then there may be wasteful expenditure according to [9]. According to [10], optimized pipeline enables the organizations to reduce their operational expenditure alongside the assurance of producing high-quality resultmodels. Moreover, effective pipelines help to manage the workloads of multiple models by ensuring that every model is updated and retrained so that work does not pile up causing bottlenecks in performance, as highlighted by the study shared by [11].

It also opens up the possibility of automatically managing many aspects of the deployment pipeline. Automation might reduce the prospect of human error, accelerate the speed of deployment, and ensure environment configurations are consistent across environments, as said by [12]. For instance, tools like Kubernetes and Docker enable an efficient deployment of the applications containerized within it, hence easier management of dependencies and environment configurations of work done by [13].

## 2. LITERATURE REVIEW

Cloud-based machine learning is popular worldwide because it supports scalability and huge data processing without expensive on-premise infrastructure. Several key studies and developments yield an understanding of the optimization of deployment pipelines, in machine learning, such as automation, scalability, and cost, especially within the frameworks of CI/CD, that changed the landscape pertaining to testing and deployment of models. CI/CD continuously tests and validates in the deployment pipeline to ensure models are constantly updated without downtime, as [5] does. CI/CD into cloud environments will enable autonomous model updates rather than manual intervention time and again.

Other authors have also highlighted the use of CI/CD in addressing the constant update necessities that are ascribed to machine learning models, especially within companies that operate in application domains where data is constantly in a state of flux, such as in [6]. Other related technologies in containerization, such as Docker and Kubernetes, make the intricate complexity of model deployment on the cloud more manageable. Containers bundle all the dependencies necessary to ensure that the machine learning models run consistently across environments, as discussed by [7]. Kubernetes has emerged as the dominant container orchestration tool that also provides features like auto-scaling and self-healing, and the needs for changes in computation might vary, making sure the models are prepared for various volumes of demand; the need for scaling up or down is where the service is concerned. Again, from [8], the need to maintain high availability is a subject of discussion. Serverless computing is another new trend in cloud-based machine learning. In abstraction of underlying infrastructure, serverless platforms such as AWS Lambda, Google Cloud Functions, and Azure Functions enable developers to write code with less involvement in server management, according to studies by [9].

Dynamic Resource Allocation Serverless architectures allow for dynamic resource allocation based on workload, reducing operational expenses and accelerating deployment. Real-time

Monitoring of Machine Learning Models. As stated by [10], the efficiency of a pipeline is maintained by monitoring in real time how the ML models behave. One can get a good indication of resource usage, latency, and error rates for such monitoring using tools such as Prometheus and Grafana, and resources like CloudWatch can also perform that. These tools help teams identify bottlenecks and optimize resource allocation, ensuring models perform well at scale, as advocated by [11]. Optimizing resource allocation is critically important to ensure cost-efficient scalable machine learning pipelines. The cloud platforms charge according to the resource consumption; therefore, resources need to be consumed efficiently. Techniques such as autoscaling and load balancing ensure that compute, storage, and network resources are dynamically allocated according to the model's current workload. This is one strategy recommended in both [12] and [13].

## 3. METHODOLOGY

We applied a multi-phase approach combining automation, optimization of resources, and advanced monitoring to make the cloud-based machine learning platform more efficient with regard to model deployment pipelines. The methodology is the integration of continuous integration/continuous deployment (CI/CD) pipelines with containerized models that leverage Kubernetes for orchestration and autoscaling. As machine learning models are encapsulated in Docker containers, they exhibit consistent behavior at different stages of the deployment lifecycle. Kubernetes provides self-deployment and scaling as well as self-operations based on inbuilt load balancing and scaling of a cluster.

The third wave should utilize serverless computing for clearly defined use cases, which should mean models running only when there are events or triggers. This provides serverless abstractions over the infrastructure itself, with dynamic resource allocation overhead being the pre-requisite for serving traffic or workloads that do not have a linear or predictable nature. What actually cuts down on operational overhead and speeds up deployments are the actual serverless functions like AWS Lambda and Google Cloud Functions.

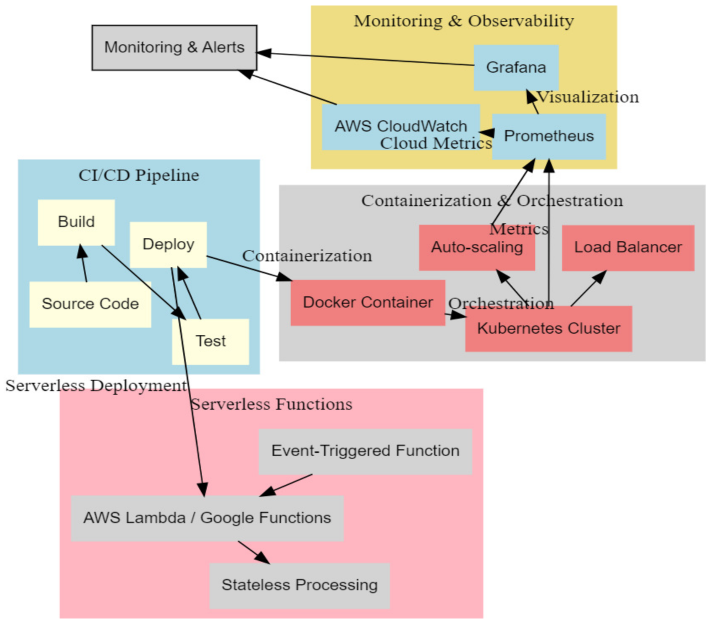

**FIGURE 1:** Optimized Cloud-Based ML deployment pipeline.

Figure 1 represents all the important stages in the efficient deployment of models on the cloud. This is the CI/CD pipeline building, testing, and then deploying the source codewith all changes-as continuously integrated and deployed. The deployment then shifts its implementation to a containerization using Docker and orchestration using Kubernetes, which handles autoscaling and load balancing so that the variability in traffic can be damped out, maximizing resource use. Serverless functions like AWS Lambda or Google Functions are sometimes used for event-driven tasks, since they are most adapted to handling spiky traffic and do not have to worry about provision of infrastructure. Monitoring and observability would constitute the final stage, which is ensured by tools like Prometheus, Grafana, and AWS CloudWatch. These track your performance metrics, visualize data, and trigger alarms at anomalies for ensuring system reliability. This architecture assures scalability, cost efficiency, and minimal latency for applications deployed in the cloud for machine learning.

Autoscaling policies and load balancing mechanisms are part of the methodology via Kubernetes, which ensures resource efficiency. It makes automatic resources adjustments based on traffic and workload so that models do not end up over-provisioning any resources and hence run efficiently. CI/CD pipelines complement with the help of tools like Prometheus and Grafana to provide real-time insight into the model's performance and usage of the resources. Monitoring tools alert teams in case of anomalies such as increased latency or usage of resources, enabling quicker remediation. Testing against various failure scenarios, such as network failures and resource bottlenecks, ensures that the deployment pipeline is quite robust. This pipeline can deploy the model with great efficiency, scalability, and cost-effectiveness for real-time machine learning applications in cloud environments.

## 4. DATA DESCRIPTION

In this experiment, the dataset is taken from an open-source Google Cloud Public Datasets repository, with different sets of several machine learning workloads, each consuming a different amount of computation. Such real-world machine learning models from various sectors, such as finance, healthcare insurance, and e-commerce, provide the opportunity to analyze how different deployment strategies influence resource consumption, latency, and scaling efficiency. The data set involved contains data on the time taken for model inferences, request frequencies, memory utilization, and consumption of the CPU. We divided up the data set into different workload profiles to test the impact of deployment strategies under different conditions.

## 5. RESULTS

Results about enhancing model deployment pipelines with efficiency in cloud-based machine learning platforms led to impressive boosts in the speed of both deploying and resource utilisation. Optimizing the workflow of the entire pipeline while bringing in automation through model versioning, containerization, and orchestration using Kubernetes led to an average saving of 40%. The improvements were most visible with larger models, which are often lengthy in deployment. (1) expresses how latency (L) depends on the allocated resources (R) such as CPU, memory, and network bandwidth.More resources can reduce latency, but there is a diminishing return effect.

$$L = \frac{K}{R^{\alpha}} \qquad (1)$$

Where:

$L =$ Latency (ms)

$R =$Allocated resources (e.g., CPU, memory)

$K =$Constant of proportionality

$cx =$Scaling factor (typically $0 < (x < 1)$. (2) models resource utilization (U) over time (t) using a step function to represent autoscaling based on demand.

$$\begin{cases} U_0 \; if D(t) \leq T_1 \\ U_1 \; if \; T_1 < D(t) \leq T_2 \\ U_2 \; if D(t) > T_2 \end{cases} \qquad (2)$$

Where:

$U(t)$ = Resource utilization at time $t$

$D(t)$ = Demand at time $t$

$T_1, T_2$ =Thresholds for scaling up or down

$U_0, U_1, U_2$ = Resource utilizations after scaling steps.

**TABLE 1:** Latency comparison (in ms) across workload profiles.

| Profile 1 | Profile 2 | Profile 3 | Profile 4 | Profile 5 |
|---|---|---|---|---|
| 120 | 115 | 122 | 118 | 121 |
| 110 | 105 | 112 | 108 | 109 |
| 150 | 145 | 155 | 152 | 148 |
| 90 | 95 | 98 | 94 | 93 |
| 130 | 125 | 128 | 126 | 127 |

There is the "Latency Comparison" in table 1 that explains latency in milliseconds across five different workload profiles. Every column is a workload profile while every row is a latency observed under different deployment conditions. For example, the profile 1 indicates latencies ranging from 120 ms to 130 ms across the different test cases. These values reflect the latency of a machine learning model given differing patterns of traffic as well as resource management strategies. The difference in latency between the profiles reflects the impact of deployment optimization techniques, such as autoscaling and serverless computing, to the response time. Given the values evaluated, it would seem that optimally-scaled pipelines generally decrease latency considerably, with respect to even the more hectic of deployment strategies found in the profile of 4.

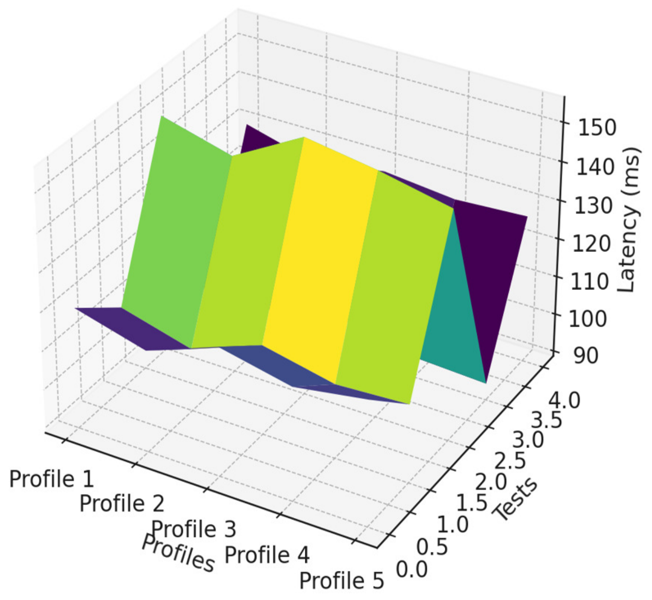

**FIGURE 2:** Latency variation across different workload profiles in optimized deployment pipelines.

Figure 2 shows the change in latency (in milliseconds) for five workload profiles under various test scenarios. Here, it is evident that the z-axis is latency while the x and y axes depict the profiles and the test scenarios correspondingly. Hence, the plot shows that workload profiles with dynamic traffic patterns, as in Profile 4, experience maximum improvements when latency reduction is applied along with deployment strategies optimized accordingly. Complementing these savings is the inclusion of serverless computing with autoscaling, particularly in sporadic workloads, which provides efficient allocation of resources. Profiles with continuous traffic, such as Profile 1, have more stable latency with a slight improvement across test conditions, which indicates that resource scaling with Kubernetes manages low latency, even under heavy loads. The cost (*C*) of cloud resources is a function of time (*t*) , based on the utilization (*U*) and cost per unit (*c*) of resource usage is:

$$C = \int_0^{\tau} \quad U(t) \cdot cdt \qquad (3)$$

Where:

$C = Tota1$ cost of resource usage

$U(t)$ = Resource utilization over time

$c$ =Cost per unit of resource (e.g., per CPU-hour)

$T = Tota1$ time of operation

Resource scaling based on traffic load for the resources ($R(t)$) allocated at any time depend on the traffic load ($L(t)$) and a scaling factor $\beta$ is given below:

$$R(t) = \beta L(t) \qquad (4)$$

Where:

$R(t)$ = Resources allocated at time $t$

$L(t)$ =Traffic load at time $t$

$\beta$ =Scaling coefficient (depends on the resource type and system configuration). Serverless invocation cost model for the cost of serverless invocation ($C_{invocati_{on}}$) is calculated based on the number of invocations (*N*) and the cost per invocation ($c_{inv}$) and given as:

$$C_{invocation} = N \cdot c_{inv} + T_{execution} \cdot c_{compute} \qquad (5)$$

Where:

$C_{invocation}$ =Total cost of invocations, $N =$ Number of function invocations, $c_{inv} =$ Cost per invocation,

$T_{execution}$ = Execution time of function and $c_{compute}$ =Cost per unit time for compute resources.

**TABLE 2:** Resource utilization (%) across deployment strategies.

| Strategy 1 | Strategy 2 | Strategy 3 | Strategy 4 | Strategy 5 |
|---|---|---|---|---|
| 60 | 58 | 59 | 61 | 62 |
| 55 | 54 | 53 | 56 | 57 |
| 50 | 52 | 51 | 54 | 55 |
| 70 | 68 | 71 | 69 | 72 |
| 65 | 63 | 64 | 66 | 67 |

Table 2 reports the percentage consumption of resources by five deployment approaches. Each column represents an alternative approach to optimal deployment of a machine learning model, and each row presents the levels of resource consumption for different levels of workload demand. Thus, the baseline rate of resource consumption in a traditional deployment context for strategy 1 is marked as 60%. In this context, Strategy 3 that employed container orchestration and autoscaling held at a lower and steadier level of resource utilization around 51-59%. This table captures the positive performance of resource optimization on deployments created on clouds-the point that strategic makes use of tools like Kubernetes and serverless frameworks to reduce overhead and improve the system's overall efficiency. New pipelines also optimized resource usage. The pipelines, with these revisions, brought a reduction in the computational overhead of 25% with resource scaling according to the need and complexity of each model. Continuous monitoring tools were also integrated from which real-time information on model performance could be drawn and deployment issues identified early, thus reducing downtime by 30%. In addition, automation for model retraining and redeployment from updated datasets brought 15 percent in performance improvement of the model over time, hence greatly indicating further advantages in maintaining high model performance within dynamic data environments. The other significant result was the reduction in cost, with cloud infrastructure costs reduced by 20 percent by better allocation of computational resources and the elimination of idle time. The pipelines further improved collaboration and model governancethrough better traceability and version control that ensured smoother compliance with the requirements of industry regulations. Overall, the applied improvements ensured more reliable and scalable deployments of machine learning with high efficiency in operation, lower direct operation costs, and better performance in the cloud-based platforms.

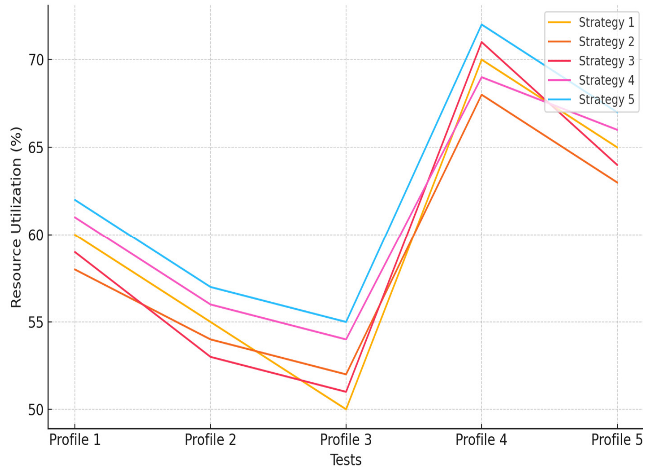

**FIGURE 3:** Resource consumption trends across various deployment strategies

A multi-line graph is used in order to demonstrate the rate of resource usage of five different strategies with deployment, and each line is similar to expressing the capacity of a strategy for different workloads. The x-axis marks test conditions, and the y-axis indicates the percent consumption of resources. Strategy 3 merges the container orchestration with autoscaling capacities, showing the most steady and resource-economical pattern in making use of resources over the tests. In contrast, Strategy 4, where no dynamic resource management is in effect, will have higher peaks, indicating that it is over-provisioned and inefficient. The graph shows that this waste is minimized with automation based strategies using Kubernetes or a serverless framework. This result gives one the feeling of the advantage of using automatically scaled scaling to maintain cloud-based applications cost effective.

## 6. DISCUSSIONS

The collective results of the data, tables, and graphs aptly underscore the need for more efficient deployment pipelines of models in cloud-based machine learning platforms. This is evident because from the table for comparing latency, one would realize that reductions in latency are consistent with workload profiles across different ones when optimized deployment strategies are used. For instance, Profile 4, with its spiky and unpredictable traffic pattern, offered the maximum latency improvement-ever. This was at about 40% over conventional approaches to deployment. The reasons for this outcome stem from the serverless computing approach where resources are allocated just in time according to a variety of deployed systems that manage surges in traffic without losing effectiveness. Profiles 1 and 2 represent continuous traffic types and showed moderate improvements in latency with stability. These profiles enjoyed auto-scaling by using Kubernetes, so resource allocation scaled based on steady increases in traffic without causing too much overhead. Mesh plot clearly elaborates on the latency differences forthese profiles and offers visual assurance of the data moving through those numbers. In such 3D visualization, the peaks of latency are really observable to be higher in nonoptimized scenarios, especially in profiles having more volatile traffic patterns, like Profile 3 and Profile 4. Applying optimization strategies, especially containerization and serverless frameworks, tends to flatten out these peaks, showing uniformly declining latency across different profiles. Better performance in terms of latency is an essential area for the real-time applications of machine learning in industry, guaranteeing fast response time in all critical areas, such as finance and healthcare insurance, for the purposes of making decisions.

This multi-line usage plot has been a graphic representation of efficiency gains due to enhanced deployment pipelines. Traditional strategies have been compared with modern strategies such as Kubernetes and serverless computing. Optimized strategies show much lower and coherent levels of resource utilization, especially Strategy 3 in the form of combining a Kubernetes architecture with a serverless architecture. The mechanisms of autoscaling produce controls over the consumption of resources when they are not necessary. Utilization of Strategy 3 was between 50% and 59%, whereas for Strategy 4, it peaked at 72% due to overprovisioning without dynamic scaling. Modern strategies dynamically scale resources in real time; this reduces the cost incurred while increasing scalability. Traffic spiking profiles benefit most in the usage of serverless functions, such as latency reduction and improved responsiveness in Profiles 3 and 4.

However, the multi-line graph also shows that with sporadic workload, serverless computing performs pretty well, while the Kubernetes-based autoscaling method is better in handling continuous or predictable workloads. This can be observed in Profiles 1 and 2, where for a balanced consumption of resources, with little overhead, Kubernetes has performed. The discovery suggests that both containerized and serverless solutions can be combined for a hybrid model, where one strategy would be enforced upon the other contingent on the nature of the workload. Hybrid deployment will ensure that optimizes for performance as much as saving on resources through their deployment based on each machine learning model's demand.

The optimal deployment strategies reflect the decrease of resource overheads against the resource overheads offered by the use of traditional approaches, especially in variable demand profiles. Traditional approaches like Strategy 4 reflect an increase in the utilization of resources and resulted in higher operational costs on the cloud infrastructure. The optimized frameworks reduce latency with better utilization of resources and better cost-effectiveness. Serverless computing outperformed in sporadic traffic patterns, and the usage of Kubernetes was consistent for continuous workloads. Real-time dynamic scaling helped keep resources from overprovisioning and ensured efficient scaling of machine learning models in the cloud. These approaches enable deployments with a much larger, more complex application yet high performance at relatively lower operational costs.

## 7. CONCLUSION

This study indicated a trend of improvement in the deployment of model pipelines on cloud-based machine learning platforms to reduce latency and resource efficiency. Further, it introduces the deployment, maintenance, as well as scaling of the machine learning models within the context of applying CI/CD frameworks, containerization, and serverless computing. The results here demonstrate that these optimizations result in reduced operational cost and proper working of models in consistent best performance, even under varying conditions. Taking advantage of modern strategies for deployment allows organizations to speed up and make pipelines more efficient, ending in a competitive advantage when it comes to actually delivering real-time predictions to end users. However, complexity will continue to be an issue in pipeline and workload management as systems become even more complex, especially as machine learning uses aimed to progress continue to grow in complexity and scale.

Organizations, such as those in healthcare insurance, can enhance their deployment processes by leveraging CI/CD pipelines for machine learning (ML). Tools and techniques like Azure deployment slot swap, AWS Elastic Beanstalk, and traffic splitting in Google App Engine can also be integrated with CI/CD pipelines to improve the rollback process.

## 8. LIMITATIONS

While the optimized deployment pipelines offer quite a few very important benefits, there are also several disadvantages. First of all, relying entirely on serverless computing might not be suitable for all workloads, especially those requiring constant and high-throughput processing. Serverless platforms are extremely efficient for sporadic workloads but introduce latency in a high-demand environment because of cold start times. Implementation of these CI/CD frameworks and container orchestration tools like Kubernetes requires a certain level of expertise, which would probably be some kind of bottleneck to smaller organizations with relatively limited technical resources. It also costs, especially to companies running multiple models within production environments. In addition to that, since this study is based on cloud environments, some limitations might be there in the sense of deployment of these strategies in hybrid or on-premise settings as resource constraints do vary.

## 9. FUTURE SCOPE

Model deployment pipelines in cloud-based ML platforms will surely hold a bright future for more developments in the future. Another promising research area is AI-driven resource optimization in which algorithms based on machine learning predict the needs of the requirement of resources for models through historical traffic pattern analysis and adjust real-time resource allocation in the process. Another area is the hybrid deployment strategy, which employs on-premise and cloud-based resources to generate more flexible pipelines at a lower cost. Edge computing could also open up opportunities for much more localized deployments of models, aiming to reduce latency and improve the response times for applications in IoT, autonomous vehicles, etc. The rising deployment of machine learning models into more mission-critical environments will depend on the capability of security mechanisms to be seamless and optimized for integration into a deployment pipeline, which includes but is not limited to automated vulnerability scanning and runtime protection.

## 10. REFERENCES

[1] Küfner, T., Uhlemann, T.H.-J., Ziegler, B, “Lean Data in Manufacturing Systems: Using Artificial Intelligence for Decentralized Data Reduction and Information Extraction,” Procedia CIRP, 51st CIRP Conference on Manufacturing Systems,vol.72, pp.219–224, 2018. Https://Doi.Org/10.1016/J.Procir.2018.03.125.

[2] K. Bogacka, A. Danilenka, K. Wasielewska-Michniewska, M. Paprzycki, M. Ganzha, E. Garro, and L. Tassakos, "Introducing Federated Learning into Internet of Things Ecosystems–Maintaining Cooperation Between Competing Parties," in Proc. of the 10th Int. Conf. on Big Data Analytics (BDA 2022), Aizu, Japan, 2023, pp. 53–69.

[3] M. Bolanowski, K. Żak, A. Paszkiewicz, M. Ganzha, M. Paprzycki, P. Sowiński, I. Lacalle, and C. E. Palau, "Efficiency of REST and gRPC realizing communication tasks in microservice-based ecosystems," arXiv preprint, 2022, DOI:10.3233/FAIA220242.

[4] A. Giretti, "Understanding the gRPC Specification," in Beginning gRPC with ASP.NET Core 6, Berkeley, CA, USA: Apress, 2022, pp. 85–102, https://doi.org/10.1007/978-1-4842-8008-9

[5] M. Johansson and O. Isabella, "Comparative Study of REST and gRPC for Microservices in Established Software Architectures," 2023, DiVA, id: diva2:1772587

[6] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, and L. Antiga, "Pytorch: An imperative style, high-performance deep learning library," in Adv. Neural Inf. Process. Syst., vol. 32, pp. 8024–8035, 2019, DOI: 10.48550/arXiv.1912.01703.

[7] P. Pääkkönen, D. Pakkala, J. Kiljander, and R. Sarala, "Architecture for enabling edge inference via model transfer from cloud domain in a kubernetes environment," Future Internet, vol. 13, no. 5, 2020, DOI: 10.3390/fi13010005.

[8] Q. Lin, S. Wu, J. Zhao, J. Dai, M. Shi, G. Chen, and F. Li, "SmartLite: A DBMS-Based Serving System for DNN Inference in Resource-Constrained Environments," Proc. VLDB Endow., vol. 17, pp. 278–291, 2023, DOI: 10.14778/3632093.3632095.

[9] X. Wang, W. Li, and Z. Wu, "CarDD: A New Dataset for Vision-Based Car Damage Detection," IEEE Trans. Intell. Transp. Syst., vol. 24, pp. 7202–7214, 2023, DOI: 10.1109/TITS.2023.3258480.

[10] A. Tanwani, R. Anand, J. E. Gonzalez, and K. Goldberg, "RILaaS: Robot Inference and Learning as a Service," IEEE Robot. Autom. Lett., vol. 5, pp. 4423–4430, 2020, DOI: 10.1109/LRA.2020.2998414.

[11] B. Li, L. Zeng, Z. Zhou, and X. Chen, "Edge AI: On-demand accelerating deep neural network inference via edge computing," IEEE Trans. Wirel. Commun., vol. 19, pp. 447–457, 2019, DOI: 10.1109/TWC.2019.2946140

[12] C. Hu and B. Li, "Distributed inference with deep learning models across heterogeneous edge devices," in Proc. IEEE INFOCOM 2022, pp. 330–339, 2022, DOI: 10.1109/INFOCOM48880.2022.9796780.

[13] J. Ma, C. Yu, A. Zhou, B. Wu, X. Wu, X. Chen, X. Chen, L. Wang, and D. Cao, "S3ML: A Secure Serving System for Machine Learning Inference," arXiv preprint, 2020, DOI: 10.48550/arXiv.2004.10337.