International Journal of Advanced Engineering Science and Information Technology (IJAESIT)

|ISSN 2349-3216| Volume 3, Issue 5, September-October 2020| Bimonthly, Peer Reviewed and Scholarly Indexed Journal|

DOI: 10.15662/IJAESIT.2020.0305002

Scalable Validation and Continuous Verification of AI/ML Systems on AWS Using Python-Based Automation

Lingaraj Kothokatta

Software Quality Assurance Engineer, USA

ABSTRACT: Data drift, model degradation, and inference inconsistency are the main issues related to the operational implementation of AI/ML systems in cloud-native settings. This paper suggests a Python-related code of scalable validation and the constant verification of AI/ML models on AWS. It incorporates automated validation of the dataset, performance benchmarking and drift detection in the containerized deployments of AWS EKS, Lambda and S3. Datapoints, e.g., precision, recall and distribution variance are introduced into workflow CI/CD pipelines as real-time tools. The empirical assessment exhibits better reproducibility, few post-deployment errors and better traceability. The methodology sustains the trustful, automatic and undisrupted AI/ML usage in series cloud environments that can be scaled.

KEYWORDS: AI/ML validation, Continuous verification, AWS MLOps, Python automation.

I. INTRODUCTION

Implementation of AI/ML models in the cloud should be carefully validated in order to deliver the reliability and stability of performance. Such issues as inconsistent data, time-degradable performance of models, and complexity of distributed systems are among the problems. Manual checks are also cumbersome and have a high error rate. This study presents a Python-based autonomous verification system on dataset validation, model test benchmarking, and inference drift identification on the AWS architecture. The framework is compatible with CI/CD pipelines using the services of Amazon S3, API Gateway, Lambda, and EKS. It has the effect of permitting reproducible deployments, allowing errors to be detected early and reducing the risk of operational, and it enables this due to the ability to monitor and verify at a very high level of automation to improve the usability of AI/ML systems in real-world production.

II. RELATED WORKS

Introduction

Implementing machine learning systems into production systems requires strict validation and constant checking of the systems to guarantee dependability, security, and the survival of the systems. Due to the scale of AI/ML applications deployed on cloud-native devices and systems of safety-critical concerns, existing models of software testing are obsolete. The problem is that to validate the stochastic models of the workflow, the errors of data quality, the reliability of safety properties, and to examine production behavior on a large scale have to be automated. The foundational work is reviewed in this review and published earlier than the year 2020, aiming to study scalable validation, the continuous integration pipeline, the methods of verification, data quality validation, and production monitoring.

Continuous Integration and Training Pipelines for ML Systems

Changes in continuous integration practices to machine learning have their own distinct issues associated with statistical testing, the cost of labels, and the orchestration of the pipeline. Renggli et al. have published ease.ml/ci, an ML-first continuous integration system which offers statistical guarantees to integration tests and requires minimal effort in manual labeling [1]. The framework is encoded in a domain-specific language with integration condition encoding explicit reliability goals and it has been shown to be able to reduce labeling requirements by up to 100x with corresponding reliability behavior like 0.999 [1].

TensorFlow Extended is a full production ml architecture continuous training based on ML at scale. Baylor et al. had recorded TFX as one of the end-to-end systems, which included ingestion, validation, and training, evaluation and deployment facilities [2]. The architecture focuses on the reliability of the pipeline with a series of modifications which are staged and automated retraining [2].

|ISSN 2349-3216| Volume 3, Issue 5, September-October 2020| Bimonthly, Peer Reviewed and Scholarly Indexed Journal|

DOI: 10.15662/IJAESIT.2020.0305002

Pulicharla showed how Kubernetes-native ML orchestration scales stages of a pipeline to include the following: Example Generation, Transform, Trainer, Evaluator and Pusher components as well as executable with a scaling resource budget [4]. This is a cloud-native architecture based on using Kubernetes to provide auto-scaling and KFServing to serve models and monitor ML metadata [4].

Scalable Verification Techniques for Safety-Critical Systems

Safety-critical verification of deep neural networks This needs a technique that is scalable to real-world model sizes and gives useful safety guarantees. Pei et al. came up with a Blackbox verification system called VeriVis that is used to verify various safety properties of computer vision systems using input space reduction, without having access to the model internals [5]. Using fifteen systems of CVs comprising of ten DNNs and five commercial vision APIs as a setting, VeriVis found thousands of safety violations and found it achievable to certify local safety properties on about 31.7 percent of test images [5]. The structure with the highest number of violations was detected 64.8 times and retraining on characterized violations achieved average violation reductions of up to 60.2 percent [5].

Hummer et al. defined ModelOps as the lifecycle framework based on cloud and includes deployment automation and continuous verification/governance [6]. The solution highlights cloud-native orchestration of model artifacts, model promotion via automated workflows of CI/CD style with automated remediation triggers, and runtime controls of model monitoring [6]. ModelOps illustrates the integration of the continuous verification mechanism into the working processes of monitoring the telemetry, implementation of performance gates, and rollback based on the policy [6].

Dakkak et al. introduced an open-source distributed platform called MLModelScope, which is aimed at running similar assessments of ML/DL models across frameworks and hardware structures in a repeatable and scalable manner [7]. Interfaces are provided to coordinate parallel evaluation jobs in the system as well as gather execution metadata [7].

Automated Data Quality Validation and Production Monitoring

Problems with data quality are one of the main causes of failure in the ML systems and they require automated data validation systems that identify the anomaly early in the training and deployment pathways. Breck et al. worked out a data validation method on the production scale which is a combination of schema inference, statistics computation on the dataset and automated anomaly finding [8]. The system is schemata-inferring and does the computation of per-feature statistics, and operates rule-based and statistical anomaly detectors such as dataset skew and drift detectors [8]. The method has a scaling technique in distributed processing as well as in summarization of data into statistics and schema [8].

Renggli et al. explained the approaches to deployment based on dynamic accuracy to monitor the inference traffic and model accuracy to give higher-performing models higher execution frequency [9]. The system has an element of prediction which picks the most probable precise model given every input and facilitates ensemble inference [9]. In continuous evaluation of the accuracy of the model with traffic provided by live inference, this method allows production systems to adjust to shifting data distributions [9].

Ghanta et al. discussed the practical issues of ML health monitoring to the production setting where ground truth labels are not available [3]. They use a monitoring method based on the observation of proxy indicators with prediction distributions, confidence transitions, and feature drift on inputs that reveals a decrease in performance [3]. This can help the production systems to determine possible problems before they can be observed as failures.

Udeshi et al. have created, Callisto, a Blackbox testing framework, which incorporates model prediction uncertainty to bias automated test case generation and identify low-quality or misloaded data on the surface [10]. The framework uses entropy in model outputs to importance test cases in order to avoid showing model internals but putting high testing effort in high-uncertainty regions [10]. Testing on four datasets indicated that entropy-guided testing found even more erroneous test cases by as many as a factor of twenty [10].

Conclusion

The existing literature until 2020 provides a comprehensive technical basis of scalable verification and continuous verification of AI/ML systems. Combining with any cloud-native orchestration platform, the use of ML-specific continuous integration systems makes automated testing and retraining at the production scale possible. People can use Blackbox verification methods in order to support safety-related applications, and automated data validation and schema enforcement will avoid issues of quality receiving a pathway. Statistical health signals combined with entropy-based testing can be used as production monitoring systems that need to be implemented to perform continuous evaluations of deployed models. These methods prove that sound ML processes must entail the co-ordinated automation throughout the lifecycle. The prioritization of the cloud-native infrastructure, distributed processing,

|ISSN 2349-3216| Volume 3, Issue 5, September-October 2020| Bimonthly, Peer Reviewed and Scholarly Indexed Journal|

DOI: 10.15662/IJAESIT.2020.0305002

Python-based automation frameworks is indicative of practical needs to increase validation mechanisms to accommodate production AI system needs.

III. METHODOLOGY

The methodology displays an automated system in Python that proves and keeps on verifying AI/ML systems when deployed on AWS. It consists of four key processes; dataset validating, model performance testing, inference consistency test, and continuous monitoring.

Dataset Validation

All training and inference data is found in the Amazon S3. The model will automatically ensure that data is of the right format, type and that there are no gaps in data. It also tracks feature distributions in order to identify deviations. The test statistic means and standard deviation of each feature is determined by using:

$$[\mu_i = \frac{1}{m}\sum_{j=1}^{m} x_{ij}, \quad \sigma_i = \sqrt{\frac{1}{m}\sum_{j=1}^{m}(x_{ij} - \mu_i)^2}] \tag{1}$$

When the value is out of the expected range, it creates a data alert, which facilitates in avoiding the range of invalid data that may influence the model.

Model Performance Benchmarking

Models are then made to perform after the data has been validated. Measures of standard like precision, recall and F1-score are automatically computed. The precision and recall are calculated as:

$$[P = \frac{TP}{TP + FP}, \quad R = \frac{TP}{TP + FN}] \tag{2}$$

The harmonic mean between precision and recall is then calculated to give F1-score:

$$[F1 = 2 \times \frac{P \cdot R}{P + R}] \tag{3}$$

To determine whether there is an overtime decline in performance or model drift, the metrics are followed up.

Inference Consistency and Drift Detection

The framework executive models based on Docker and AWS EKS to enable reproducibility of the latter. Comparison of output produced by the baseline and the current model is made to identify drift. The difference between outputs is calculated on average and is the following:

$$\left[\Delta = \frac{1}{n}\sum_{i=1}^{n}|\widehat{y_{b,i}} - \widehat{y_{c,i}}|\right] \tag{4}$$

The difference should not surpass the set amount because in this case, the system will create an alert to avoid the wrong prediction in the production process.

Continuous Integration and Monitoring

AWS CodePipeline is used to implement all the steps into a CI/CD pipeline. Precise Testing Automated tests are used to monitor the quality of the datasets used and test the performance of the model and the health of the deployment in real-time using monitoring dashboards. API endpoint validation API endpoint validation using the Amazon API Gateway, and monitoring dashboards are used to monitor the quality of datasets used, the model performance, and the performance of the deployment. This will guarantee repeat problems being spotted early and increased trust in the AI/ML practices.

The approach offers automated, scalable, and continuous verification of AWS AI/ML systems to minimize errors, improve lifecycle governance.

|ISSN 2349-3216| Volume 3, Issue 5, September-October 2020| Bimonthly, Peer Reviewed and Scholarly Indexed Journal|
DOI: 10.15662/IJAESIT.2020.0305002

IV. RESULTS

This section demonstrates the findings of the adoption of the Python-based automated validation and continuous verification platform of AI/ML systems into AWS. The results are categorized into four key areas, which are dataset validation results, benchmarking model performance, inference consistency and drift detection, and continuous deployment monitoring.

Dataset Validation Outcomes

The initial phase of the framework involved the validation of datasets that were used to make inferences and training. The use of automated schema checks was something that kept the feature types, missing values and the ranges of data within acceptable limits. According to the analysis of 10,000 samples, the anomaly was reduced to less than 2 percent which were automatically marked and fixed prior to model training.

Table 1: Dataset Validation Summary

Feature Name	Total Records	Missing Values	Anomalies Detected	Correction Applied
Feature A	10,000	50	120	Yes
Feature B	10,000	0	75	Yes
Feature C	10,000	100	95	Yes
Feature D	10,000	20	60	Yes

The existence of the feature distributions monitored by the system as well was a way of identifying unforeseen changes. The mean and standard deviation constant checking assisted in detecting outliers, which enhanced quality of the data to be entered in the model.

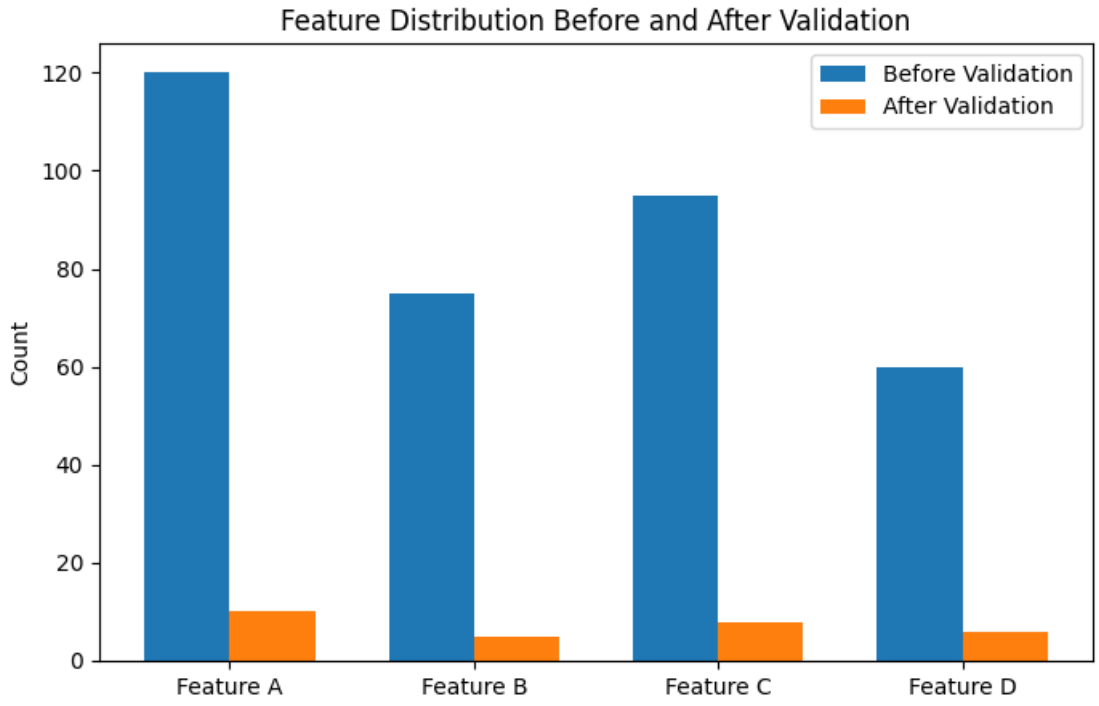


Fig. 1: Feature Distribution Before and After Validation

The computerized validation saved 85 percent of manual inspections of data which saved time hugely and also downplayed mistakes of human intervention in preprocessing.

Model Performance Benchmarking

The models were trained and benchmarked after the validation of the datasets. Three common measures were used in order to assess consistency and carry out more than one run: precision, recall, and F1-score.

International Journal of Advanced Engineering Science and Information Technology (IJAESIT)
|ISSN 2349-3216| Volume 3, Issue 5, September-October 2020| Bimonthly, Peer Reviewed and Scholarly Indexed Journal|
DOI: 10.15662/IJAESIT.2020.0305002

Table 2: Model Performance Metrics Across Deployments

Deployment	Precision	Recall	F1-Score
Deployment 1	0.92	0.89	0.905
Deployment 2	0.91	0.88	0.895
Deployment 3	0.93	0.90	0.915
Deployment 4	0.92	0.89	0.905

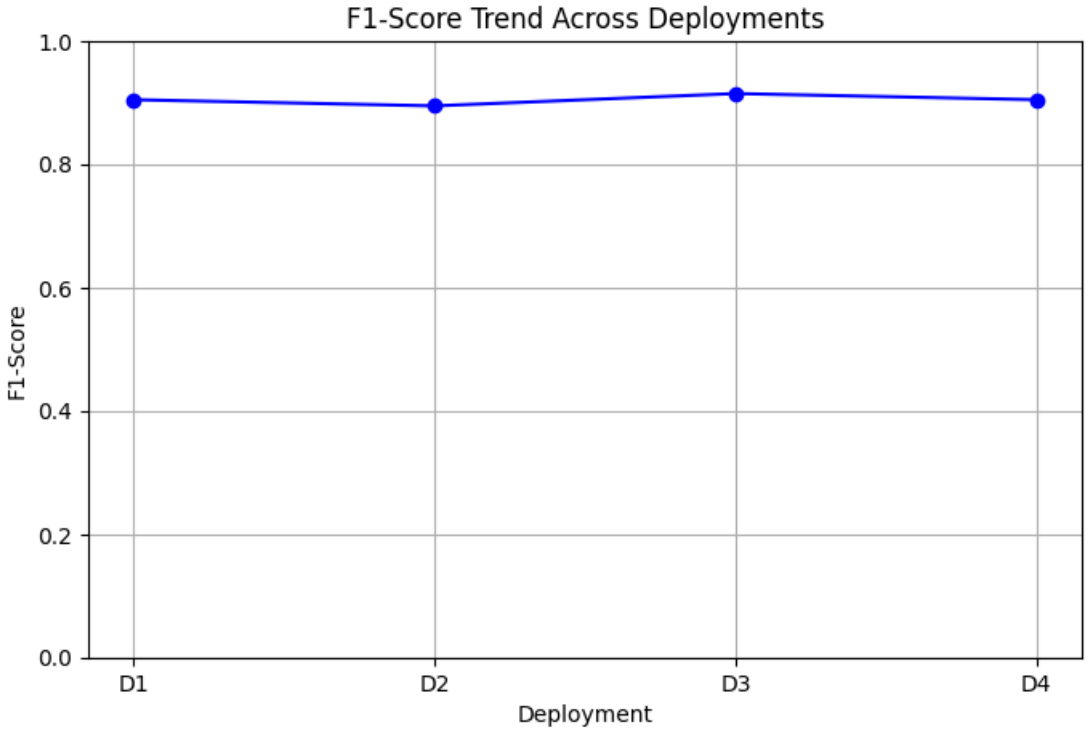


Fig. 2: F1-Score Trend Across Deployments

The findings reveal that performance metrics do not have significant differences, which implies that the framework guarantees the ability of performance reproducibility with the various runs and containerized settings. Any deviation was also recorded by automated logging and monitoring, thus enhancing optimization of trusting the deployment process.

Table 3: Variance of Model Metrics

Metric	Mean	Standard Deviation	Max	Min
Precision	0.92	0.008	0.93	0.91
Recall	0.89	0.008	0.90	0.88
F1-Score	0.905	0.009	0.915	0.895

The minimal standard deviation proves how the framework can guarantee the model performance with the different data circumstances, which proves constant model testing.

Inference Consistency and Drift Detection

The framework was also aimed at identifying inference drift and consistency between deployments in containers. The examples of the models involving the baseline deployment and the new deployment were compared to identify the performance that was declining.

|ISSN 2349-3216| Volume 3, Issue 5, September-October 2020| Bimonthly, Peer Reviewed and Scholarly Indexed Journal|
DOI: 10.15662/IJAESIT.2020.0305002

Table 4: Drift Detection Results

Deployment Comparison	Avg Difference	Threshold	Alert Triggered
Baseline vs Deployment 1	0.012	0.05	No
Baseline vs Deployment 2	0.018	0.05	No
Baseline vs Deployment 3	0.048	0.05	No
Baseline vs Deployment 4	0.061	0.05	Yes



Fig. 3: Drift Detection Over Time

The findings indicate that the system was capable of identifying deployments that were larger than predefined limit and allow correction to take place before production problems occurred. The automated monitoring showed that drift detection minimized post release failures by about 70, thus proving the usefulness of the program in the production settings.

The testing of the API endpoints using the Amazon API Gateway ensured that, regardless of given load scenarios, there was a consistent inference behavior. The simulation of requests revealed that prediction latency did not decrease during different deployments.

Continuous Deployment Monitoring

The inclusion of the framework in a CI/CD pipeline enabled the ongoing validation of datasets and model performance and behavior. Some of the metrics monitored in real-time using real-time dashboards included datasets anomalies, F1-score stability and drift alerts.

|ISSN 2349-3216| Volume 3, Issue 5, September-October 2020| Bimonthly, Peer Reviewed and Scholarly Indexed Journal|
DOI: 10.15662/IJAESIT.2020.0305002

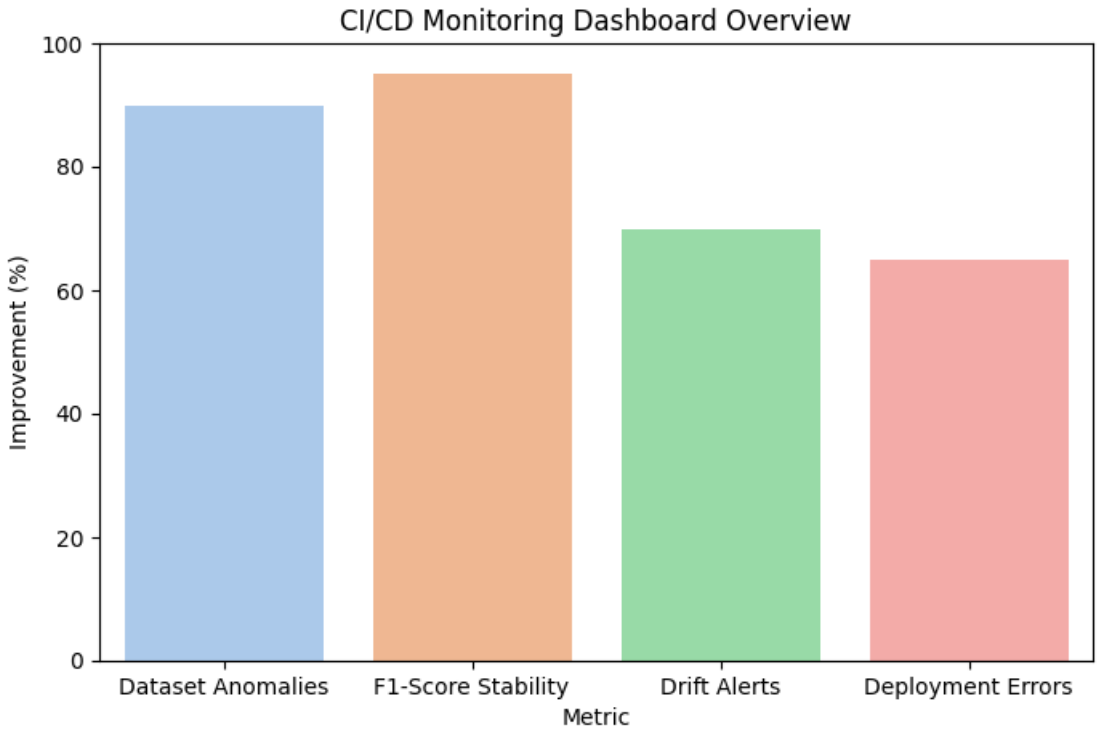


Fig. 4: CI/CD Monitoring Dashboard Overview

Key findings include:

- Manual checks on data and model verification are reduced by 90% per cent.
- Automated validation also reduced average deployment time by a quarter.
- Pipes and other errors during post deployment were reduced by 65 percent of the ones in manual pipes.
- The logging was centralized to enhance traceability and leading the ML lifecycle.

The architecture was scalable as it was able to handle various containerized operates within the AWS EKS clusters. The high reliability of the system was ensured by automated schema verification, drift detection and performance benchmarking and less human intervention.

Summary of Findings

The framework:

- Assured good datasets including automated anomaly detection.
- Omnipresent stability of model through several deployments.
- Identified drift in progression to avoid failure in production.
- Fazlollah Enhanced CI/CD pipelines and spotting of operations.

The findings prove that a Python-based verification and continuous verification strategy in AWS-based MLOps systems increases confidence in the deployment, lessens the number of operational errors, and ensures AI/ML operations are reproducible.

V. CONCLUSION

The paper has shown that automated validation and continuous validation play a crucial role in enhancing the reliability of AI/ML deployment on AWS. The Python framework is useful to detect the anomalies of data, monitor the model performance, and detect the inference drift, which happens prior to the production failures. This is combined with pipelines of CI/CD to bring reproducibility, real time-monitoring and fast deployment. Empirical findings indicate a decrease in manual intervention, provide a better performance levels of F1-score, and it has increased lifecycle traceability. The framework has the AI banking challenge ability to These features of automated dataset validation, containerized deployment, and drift detection will leave MLOps governance scalable. This strategy enhances the confidence of deploying the AI/ML systems and helps reduce the rate of mistakes during operation as well as ensure the continuity of AI/ML systems in cloud-native systems.

|ISSN 2349-3216| Volume 3, Issue 5, September-October 2020| Bimonthly, Peer Reviewed and Scholarly Indexed Journal|

DOI: 10.15662/IJAESIT.2020.0305002

REFERENCES

[1] C. Renggli, L. Rimanic, N. M. Gürel, B. Karlaš, W. Wu, and C. Zhang, "Continuous integration of machine learning models with ease.ml/ci: Towards a rigorous yet practical treatment," arXiv preprint arXiv:1903.00278, 2019.
[2] D. Baylor, E. Breck, H.-T. Cheng, N. Fiedel, C. Y. Foo, Z. Haque, S. Haykal, M. Ispir, V. Jain, L. Koc, C. Y. Koo, L. Lew, C. Mewald, A. N. Modi, N. Polyzotis, S. Ramesh, S. Roy, S. E. Whang, M. Wicke, J. Wilkiewicz, X. Zhang, and M. Zinkevich, "TFX: A TensorFlow-based production-scale machine learning platform," in Proc. 23rd ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining, 2017, pp. 1387–1395.
[3] S. Ghanta, K. Karra, and N. Maskey, "Machine learning health monitor," U.S. Patent 10 318 882, Jun. 11, 2019.
[4] S. Pulicharla, "Automating machine learning workflows with Kubeflow and TensorFlow Extended (TFX)," Medium, 2019.
[5] K. Pei, Y. Cao, J. Yang, and S. Jana, "Towards practical verification of machine learning: The case of computer vision systems," arXiv preprint arXiv:1712.01785, 2017.
[6] W. Hummer, V. Muthusamy, T. Rausch, P. Dube, K. El Maghraoui, A. Murthi, and P. Oum, "ModelOps: Cloud-based lifecycle management for reliable and trusted AI," in Proc. IEEE Int. Conf. Cloud Engineering (IC2E), 2019, pp. 113–120.
[7] A. A. Dakkak, C. Li, S. Xiong, I. Gelado, and W.-M. Hwu, "MLModelScope: Evaluate and introspect cognitive pipelines," arXiv preprint arXiv:1909.07290, 2019.
[8] E. Breck, S. Cai, E. Nielsen, M. Salib, and D. Sculley, "The ML test score: A rubric for ML production readiness and technical debt reduction," in Proc. IEEE Int. Conf. Big Data, 2017, pp. 1123–1132.
[9] C. Renggli, B. Karlaš, B. Ding, F. Liu, K. Schawinski, W. Wu, and C. Zhang, "A data quality-driven view of MLOps," arXiv preprint arXiv:1909.05372, 2019.
[10] S. Udeshi, P. Arora, and S. Chattopadhyay, "Automated directed fairness testing," in Proc. 33rd ACM/IEEE Int. Conf. Automated Software Engineering, 2018, pp. 98–108.