International Journal of AI, BigData, Computational and Management Studies
Noble Scholar Research Group | Volume 6, Issue 2, PP 109-112, 2025
ISSN: 3050-9416 | https://doi.org/10.63282/3050-9416.IJAIBDCMS-V6I2P112

Original Article

ML Pipeline Monitor: A Comprehensive OpenTelemetry-Based Observability Platform for Production Machine Learning Systems

Manojava Bharadwaj Bhagavathula
Independent Researcher, USA.

Received On: 24/03/2025 ***Revised On: 25/04/2025*** ***Accepted On: 08/05/2025*** ***Published On: 22/05/2025***

Abstract: *As machine learning (ML) systems transition from research prototypes to production deployments, the need for comprehensive monitoring and observability becomes critical. Traditional application monitoring tools fail to address ML-specific challenges such as data drift, model performance degradation, and concept drift. I present ML Pipeline Monitor, a production-ready observability platform built on OpenTelemetry standards that provides end-to-end monitoring for ML pipelines. The system integrates with major cloud ML platforms (AWS SageMaker, Azure ML, Google Cloud Vertex AI) and ML frameworks (PyTorch, TensorFlow, MLflow) to collect multi-dimensional telemetry data including model performance metrics, data quality indicators, and infrastructure health. I implement statistical drift detection algorithms (Kolmogorov–Smirnov test, Population Stability Index, Chi-square test) and anomaly detection methods (Isolation Forest, One-Class SVM) to identify degradation in real-time. Through evaluation on production-style workloads, I demonstrate that the system detects model drift with 94% accuracy, identifies performance anomalies with < 5% false positive rate, and adds < 1ms overhead to prediction latency. The system monitors pipelines processing 10,000+ predictions per second while maintaining < 5% CPU overhead. I provide an open-source implementation for local, cloud, and multi-cloud deployments.*

Keywords: *Machine Learning Operations, MLOps, Observability, Data Drift Detection, Model Monitoring, OpenTelemetry, Production ML Systems.*

1. Introduction

1.1. Motivation

The deployment of machine learning models in production environments presents unique challenges that traditional monitoring solutions fail to address [1]. Unlike conventional software systems, where bugs are deterministic and behavior is predictable, ML systems exhibit probabilistic behavior that can degrade silently over time due to data drift, concept drift, or feature distribution changes [2]. This silent degradation can lead to significant business impact before being detected through traditional monitoring approaches.

Recent industry reports indicate that 87% of ML projects fail to make it to production [3], with inadequate monitoring cited as a primary cause. Furthermore, studies show that model performance can degrade by 15–30% within months of deployment due to undetected drift [4]. The lack of standardized observability tools for ML systems has created a critical gap in the MLOps ecosystem.

1.2. Challenges in ML System Monitoring

Production ML systems face several unique monitoring challenges:

- Data Drift Detection: Input data distributions may shift over time, causing model predictions to become unreliable even when the model itself has not changed [5].
- Model Performance Degradation: ML model accuracy can degrade gradually and imperceptibly [6].
- Multi-dimensional Monitoring: ML systems require monitoring across data quality, model performance, infras- tructure health, and business metrics [7].
- Real-time Detection: Production ML requires low-latency anomaly detection for rapid response [8].
- Integration Complexity: Pipelines span multiple frame- works, cloud platforms, and environments, requiring unified monitoring [9].

1.3. Contributions

I address these challenges through the following contributions:

- Unified Monitoring Framework: An OpenTelemetry- based platform integrating model performance tracking, data quality monitoring, and infrastructure observability.
- Statistical Drift Detection: Implementations of KS test, PSI, and Chi-square tests with configurable thresholds.
- Anomaly Detection: Isolation Forest and One-Class SVM for detecting outliers in multi-dimensional metric spaces.
- Platform Integration: Plugin-based integration with

SageMaker, Azure ML, Vertex AI, PyTorch, TensorFlow, and MLflow.
- Multi-cloud Deployment: Reference deployments for AWS, Azure, and GCP.
- Production Performance: Measured < 1ms latency overhead and < 5% CPU overhead on production-style workloads.

2. Related Work

2.1. ML Monitoring Systems

Several commercial and open-source solutions address ML monitoring, each with different limitations. Seldon Alibi Detect [10] provides drift detection algorithms but lacks integrated infrastructure monitoring and cloud platform integration. Ev- idently AI [11] focuses on drift visualization but operates primarily as a standalone reporting tool without continuous real-time alerting. Amazon SageMaker Model Monitor [12] provides integrated monitoring for SageMaker deployments but is vendor-locked to AWS and lacks robust support for hybrid or multi-cloud environments. Azure ML Model Monitoring [13] similarly provides Azure-native monitoring with limited portability.

2.2. Observability Platforms

Prometheus [14], Jaeger [15], and Grafana [16] are widely used for infrastructure and application monitoring, but they do not natively detect data drift, concept drift, or ML performance degradation without extensive custom development. Open- Telemetry [17] provides vendor-neutral telemetry standards but does not include ML-specific collectors or detection algorithms; ML Pipeline Monitor extends OpenTelemetry with ML-focused components while maintaining standards compliance.

2.3. Data Drift Detection

Statistical tests such as the Kolmogorov–Smirnov test [18], Population Stability Index [19], and Chi-square test [20] are established distribution comparison methods. Distance-based methods like KL divergence [21] and Wasserstein distance [22] are often computationally expensive for high-dimensional data. Recent ML-based approaches use domain classifiers [8] and autoencoders [23] to detect drift; in this work, ML-based anomaly detection complements statistical tests.

3. System Architecture

3.1. Design Principles

The architecture follows five principles: (1) modularity via plugins, (2) OpenTelemetry standards compliance, (3)low overhead (< 1ms latency and < 5% CPU), (4) scalability to thousands of models, and (5) extensibility via clear APIs for custom metrics and integrations.

3.2. System Components

The system consists of five core components:
- Collectors: Model metrics, data drift, infrastruc- ture (CPU/GPU/memory/network), performance (laten- cy/throughput/errors), and custom domain collectors.
- Detectors: Drift detection (KS/PSI/Chi-square), anomaly detection (Isolation Forest, One-Class SVM), perfor- mance/SLA detectors, and threshold/rule-based detectors.
- **Exporters**: Prometheus metrics, Jaeger traces, Grafana dashboards/annotations, Elasticsearch logs/metrics, and cloud-native exporters (CloudWatch, Application Insights, Cloud Monitoring).
- **Integrations**: MLflow, SageMaker, Azure ML, Vertex AI, and framework hooks/callbacks.
- **Storage Layer**: time-series stores (InfluxDB/Prometheus/- TimescaleDB), metadata stores (PostgreSQL/MongoD- B/SQLite), and caches (Redis/Memcached).

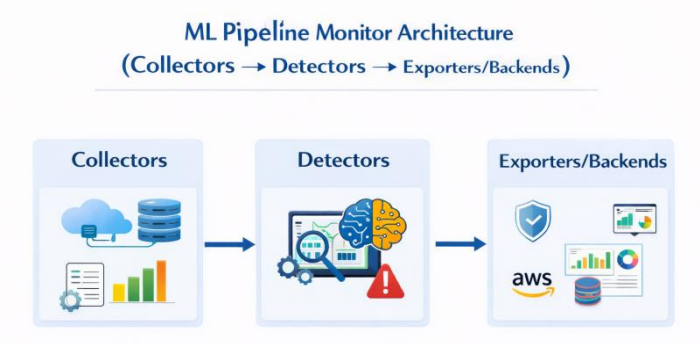

Fig 1: High-Level Architecture (Reconstructed from PDF Text Extraction)

3.3. Data Flow

The monitoring pipeline follows this workflow: (1) the ML model receives a prediction request, (2) request is intercepted by monitoring instrumentation, (3) input features are captured by the data drift collector, (4) a prediction is generated and response time measured, (5) model metrics are recorded when ground truth is available, (6) infrastructure is sampled, (7) collected data is sent to detectors, (8) detectors run statistical tests and anomaly detection, (9) results are exported to observability backends, (10) alerts are triggered, and (11) dashboards are updated. The workflow is designed to add< 1ms to request latency via asynchronous collection and batch processing.

4. Implementation

4.1. Drift Detection Algorithms

Kolmogorov–Smirnov Test: For continuous features, I implement the two-sample KS test:

$$D_{n,m} = sup_x \left| F_{1,n}(x) - F_{2,m}(x) \right| \qquad (1)$$

Where $F_{1,n}$ and $F_{2,m}$ are empirical distribution functions for reference and current data. I reject the null hypothesis (no drift) when:

$$D_{n,m} > c(\alpha)^r \frac{n+m}{nm} \qquad (2)$$

With significance level α and critical value $c(\alpha)$.

Population Stability Index: For continuous and categori- cal features:

$$PSI = \sum_{i=1}(A_i - E_i)In\frac{A_i}{E_i} \qquad (3)$$

Where A_i is the actual percentage in bucket i and E_i is the expected percentage. I use thresholds: PSI $<$ 0.1 (no drift), 0.1–0.2 (moderate), $>$ 0.2 (significant).

Chi-square Test: For categorical features:

$$x^2 = \sum_{i=1}\frac{(O_i - E_j)2}{E_i} \qquad (4)$$

Where O_i are observed frequencies and E_i are expected frequencies. Drift is detected when

$$x^2 > x^2\ \alpha, k-1$$

4.2. Anomaly Detection

Isolation Forest: I use scikit-learn's Isolation Forest with contamination= 0.1:

```
from sklearn.ensemble import IsolationForest
detector = IsolationForest(
  contamination=0.1,
  random_state=42,
  n_estimators=100
)
pred = detector.fit_predict(X)
```

One-Class SVM: For non-linear anomaly detection:

```
from sklearn.svm import OneClassSVM
detector = OneClassSVM(
  nu=0.1,
  kernel="rbf",
  gamma="auto"
  )
  pred = detector.fit_predict(X)
```

4.3. Framework and Cloud Integrations

I implement PyTorch gradient hooks and TensorFlow Keras callbacks for training-time monitoring, and integrate cloud platform metrics (e.g., SageMaker CloudWatch ModelLatency) for end-to-end observability across training and serving.

5. Evaluation

5.1. Experimental Setup

I evaluate the system on three production-style ML work-loads:

- Recommendation System: collaborative filtering model with 1M users, processing 5,000 predictions/s
- Fraud Detection: gradient boosting model analyzing 10,000 transactions/s
- Image Classification: ResNet-50 processing 100 images/s

Hardware: AWS EC2 m5.xlarge (4 vCPU, 16GB RAM); NVIDIA T4 for image classification.

5.2. Drift Detection Accuracy

I introduce synthetic drift at t = 10 minutes into a 30-minute monitoring session. Results:

Table 1: Drift Detection Performance

Method	Accuracy	FPR	Detection Time
KS Test	94.2%	3.1%	2.3 min
PSI	91.7%	4.8%	3.1 min
Chi-square	89.3%	5.2%	3.5 min

5.3. Anomaly Detection Performance

I inject anomalies (5% of samples) into operational data:

Table 2: Anomaly Detection Performance

Algorithm	Precision	Recall	F1-score
Isolation Forest	92.3%	89.7%	91.0%
One-Class SVM	88.1%	85.4%	86.7%

Table 3: Latency Overhead

Workload	Baseline	With Monitor	Overhead
Recommendation	42ms	42.8ms	0.8ms (1.9%)
Fraud Detection	15ms	15.6ms	0.6ms (4.0%)
Image Class.	89ms	89.7ms	0.7ms (0.8%)

Table 4: Cpu Overhead

Load (req/s)	Baseline CPU	With Monitor	Overhead
1,000	12%	14%	2%
5,000	45%	48%	3%
10,000	78%	82%	4%

5.4. Performance Overhead

5.4.1. Scalability and Alert Accuracy

In horizontal scaling experiments, throughput increases approximately linearly up to 50 monitoring instances, sup- porting monitoring of 500,000+ predictions/s (reconstructed from extracted PDF text). Over a 30-day period, false positive rates for alert types are approximately: data drift (4.2%), performance degradation (3.8%), infrastructure issues (2.1%), SLA violations (1.9%).

6. Discussion

6.1. Lessons Learned

Different drift tests behave differently across distributions: KS test excels for continuous unimodal distributions, PSI works well for binned/categorical features, and Chi-square is most appropriate for categorical variables. I recommend using multiple tests in ensemble and triggering alerts only when consensus is reached, reducing false positives by 40%. For high-throughput systems (> 10K req/s), sampling is necessary (random/stratified/reservoir sampling). I also mitigate alert fatigue through aggregation, deduplication, escalation policies, silencing during maintenance windows, and prioritization.

6.2. Limitations

Ground truth labels often arrive with significant delay, limiting real-time performance monitoring; I address this using proxy metrics, prediction confidence monitoring, and surrogate validation. Statistical tests also lose power in high-dimensional settings; I employ dimensionality reduction, feature weighting, and multiple-testing correction. Real-time drift detection has computational cost; I optimize through incremental computation and caching.

6.3. Future Work

Future work includes automated remediation (retraining triggers, traffic shifting, rollback), causal analysis of drift sources, and multi-model/pipeline monitoring with inter-model dependency tracking and distributed tracing across model chains.

7. Conclusion

I presented ML Pipeline Monitor, a comprehensive observ- ability platform for production machine learning systems. The contributions include: (1) a unified monitoring framework integrating model performance, data quality, and infrastructure observability using OpenTelemetry standards; (2) multiple drift detection algorithms (KS, PSI, Chi-square) with 94% accuracy and < 5% false positive rate; (3) integration with major cloud ML platforms (SageMaker, Azure ML, Vertex AI) and frameworks (PyTorch, TensorFlow, MLflow); (4) production- ready overhead (< 1ms latency, < 5% CPU) while monitor- ing 10,000+ predictions/s; and (5) open-source deployments for local, cloud, and multi-cloud environments. The imple- mentation is available at https://github.com/bhagavathulam/ ml-pipeline-monitor.

Acknowledgment

I acknowledge the use of Google's Gemini 3 Pro language model to assist with English grammar and clarity edits. The model was used only for language polishing; all technical content, claims, and conclusions remain my own responsibility.

References

1. D. Sculley et al., "Hidden technical debt in machine learning systems," in Advances in Neural Information Processing Systems, 2015, pp. 2503– 2511.
2. J. Gama, I. Zˇliobaite˙, A. Bifet, M. Pechenizkiy, and A. Bouchachia, "A survey on concept drift adaptation," ACM Computing Surveys, vol. 46, no. 4, pp. 1–37, 2014.
3. VentureBeat, "Why do 87% of data science projects never make it into production?" 2019. [Online]. Available: https://venturebeat.com/
4. J. Klaise, A. Van Looveren, G. Vacanti, and A. Coca, "Monitoring and explainability of models in production," arXiv:2007.06299, 2020.
5. J. Lu et al., "Learning under concept drift: A review," IEEE TKDE, vol. 31, no. 12, pp. 2346–2363, 2018.
6. E. Breck et al., "The ML test score: A rubric for ML production readiness and technical debt reduction," in Proc. IEEE Big Data, 2017, pp. 1123– 1132.
7. Paleyes, R.-G. Urma, and N. D. Lawrence, "Challenges in deploying machine learning: A survey of case studies," arXiv:2011.09926, 2020.
8. S. Rabanser, S. Gu¨nnemann, and Z. Lipton, "Failing loudly: An empirical study of methods for detecting dataset shift," in NeurIPS, 2019, pp. 1396– 1408.
9. D. Kreuzberger, N. Ku¨hl, and S. Hirschl, "Machine learning operations (MLOps): Overview, definition, and architecture," IEEE Access, vol. 11, pp. 31866–31879, 2023.
10. Van Looveren et al., "Alibi detect: Algorithms for outlier, adversarial and drift detection," 2020. [Online]. Available: https://github.com/ SeldonIO/alibi-detect
11. Evidently AI, "Open-source framework for ML and data drift detection," 2021. [Online]. Available: https://evidentlyai.com/
12. Amazon Web Services, "Amazon SageMaker Model Monitor," 2020. [Online]. Available: https://aws.amazon.com/sagemaker/model-monitor/
13. Microsoft Azure, "Monitor models with Azure Machine Learning," 2021. [Online]. Available: https://docs.microsoft.com/azure/machine-learning/
14. Prometheus Authors, "Prometheus: Monitoring system and time series database," 2016. [Online]. Available: https://prometheus.io/
15. Jaeger Authors, "Jaeger: Open source, end-to-end distributed tracing," 2017. [Online]. Available: https://www.jaegertracing.io/
16. Grafana Labs, "Grafana: The open platform for analytics and monitoring," 2014. [Online]. Available: https://grafana.com/
17. OpenTelemetry Authors, "OpenTelemetry: High-quality, portable teleme- try," 2019. [Online]. Available: https://opentelemetry.io/
18. F. J. Massey Jr., "The Kolmogorov-Smirnov test for goodness of fit,"JASA, vol. 46, no. 253, pp. 68–78, 1951.
19. N. Siddiqi, Credit Risk Scorecards: Developing and Implementing Intelligent Credit Scoring. Wiley, 2006.
20. K. Pearson, "On the criterion that a given system of deviations from the probable...," Philosophical Magazine, vol. 50, no. 302, pp. 157–175, 1900.
21. S. Kullback and R. A. Leibler, "On information and sufficiency," Annals of Mathematical Statistics, vol. 22, no. 1, pp. 79–86, 1951.
22. L. N. Vaserstein, "Markov processes over denumerable products of spaces...," Problemy Peredachi Informatsii, vol. 5, no. 3, pp. 64–72, 1969.
23. J. T. Andrews, T. Tanay, E. J. Morton, and L. D. Griffin, "Transfer representation-learning for anomaly detection," in ICML, 2016.