Journal of Information Systems Engineering and Management
2026, 11(2s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

Machine Learning Model Deployment and Monitoring in Google Cloud Platform (GCP): A Scalable and Reliable MLOps Framework

Ancilia Anthony Dmello
University of Michigan, USA

ARTICLE INFO

Received: 27 Feb 2026

Accepted: 03 March 2026

ABSTRACT

Machine learning models have to move beyond the experimental platform to a production platform where they can produce real business value, but this shift is difficult because the infrastructure is complex, and organizational concerns continue to complicate the change, not to mention the ongoing issue of model performance as data distributions change over time. Vertx AI is one of many managed services offered by Google Cloud Platform, which also includes Cloud Run, BigQuery, Dataflow, and Cloud Monitoring that can be used in combination to enable a full range of MLOps practices to deploy, monitor, and maintain machine learning systems at scale. Various architectural designs, such as online prediction models to provide real-time inferences, batch models where the throughput of data is high, and stream architecture where data flows continuously, need to be carefully considered with the deployment of production ML systems. Continuous monitoring is important since the models degrade and require automated detection systems, which can detect that performance is worsening before it affects the users. Automated retraining pipelines with human intervention in critical decisions can help organizations to keep models accurate and also ensure that governance and compliance requirements are upheld. Schema validation, centralized stores of features, security measures, and cost optimization initiatives are best practices that help organizations to design ML systems that are reliable, precise, and economically feasible in their lifecycle of operations.

Keywords: MLOps, Model Deployment, Machine Learning Monitoring, Google Cloud Platform, Continuous Training

1. Introduction

Machine learning models transition from experimentation to production through ML engineering practices collectively known as MLOps. Google Cloud Platform (GCP) provides native, scalable tooling that streamlines model delivery and monitoring across large datasets and high-traffic use cases such as connected vehicles, finance, retail, and predictive maintenance.

The challenge of operationalizing machine learning systems remains one of the most significant barriers in the industry, as evidenced by research showing that the overwhelming majority of data science initiatives fail to reach production environments due to infrastructure gaps, organizational challenges, and the complexity of maintaining model performance over time [1]. Machine learning applications must operate at scale, reliably, and in real-world environments where model performance can degrade due to data drift, concept drift, and operational anomalies. The deployment of machine learning systems involves navigating numerous challenges that extend far beyond algorithm development, including issues related to configuration management, data dependencies, model serving infrastructure, and the continuous monitoring required to detect when models begin to fail in production settings [2]. The accuracy of models usually diminishes with time because of changes in users' behavior, sensor wear and tear, external factors, or market fluctuations. To achieve a stable performance in a dynamic environment, with the statistical characteristics of the incoming data

Journal of Information Systems Engineering and Management
2026, 11(2s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

subject to change, constant monitoring, automatic retraining, and operational observability are necessary.

GCP provides a portfolio of managed services such as Vertex AI, Cloud Run, Cloud Functions, BigQuery, Dataflow, and Cloud Logging that can be combined to provide a well-rounded MLOps ecosystem to support end-to-end deployment, monitoring, retraining, and governance. This paper provides a production-scale architecture and process of deploying ML models to GCP, the practices of continuous integration and continuous delivery, automated monitoring strategies, and the best practices of ML observability and lifecycle management.[6]

2. Key GCP Components and Deployment Architectures

2.1 Vertex AI and Cloud Infrastructure Components

Vertex AI serves as the central hub for ML operations, offering centralized model management, AutoML and custom training capabilities, a comprehensive model registry with versioning support, Feature Store for data consistency across training and serving, batch and real-time prediction endpoints, and pipelines for automation. The platform provides unified tooling that addresses the full machine learning lifecycle, from data preparation and feature engineering through model training, deployment, and ongoing monitoring, with built-in support for both AutoML solutions that automate model development and custom training workflows that give practitioners full control over their modeling approaches [3]. The architecture supports multiple deployment patterns, including online prediction services that respond to synchronous requests with low latency requirements, batch prediction jobs that process large volumes of data asynchronously, and streaming inference pipelines that handle continuous data flows from sources such as IoT devices or real-time transaction systems.

Cloud Run provides a serverless microservice platform for custom prediction services with container-based deployment featuring autoscaling from zero instances, making it ideal for low-latency API requirements where traffic patterns may be unpredictable or highly variable. The managed nature of Cloud Run eliminates the operational overhead of cluster management while providing automatic scaling based on incoming request volume, with the ability to scale down to zero during periods of no traffic to minimize costs [3]. BigQuery serves as a petabyte-scale data warehouse for feature engineering, monitoring metrics storage, and drift analysis, enabling efficient querying of massive datasets for ML operations with its columnar storage format and distributed query execution engine that can scan terabytes of data in seconds. The monitoring capabilities provided by BigQuery extend beyond traditional database metrics to include specialized functionality for tracking data quality, identifying anomalies in feature distributions, and analyzing historical patterns that may indicate model degradation [4]. Dataflow: Dataflow supports streaming and batch pipelines of data preprocessing, data model-serving capabilities, and telemetry transformations, scalability, and data processing at all stages of the ML life cycle using the unified programming model of Apache Beam, which enables the same pipeline code to execute in both batch and streaming modes. Cloud Monitoring and Logging monitors latency, throughput, failures, model health, unforeseen inputs, and operation anomalies, and gives full visibility to the performance of the system with customizable dashboards and monitoring mechanisms that alert so that anomalies can be easily identified and responded to.

2.2 ML Deployment Architectures in GCP

The online prediction architecture begins with models stored in the Vertex AI Model Registry, with Docker images built via Cloud Build, and serving endpoints established through either Vertex AI Prediction for managed hosting or Cloud Run for custom inference logic where specialized preprocessing or postprocessing is required. API Gateway exposes the prediction endpoint to clients while monitoring hooks feed data into BigQuery and Cloud Logging for observability, creating a complete path from model artifact to production API with full instrumentation for performance tracking and debugging. Batch inference leverages Vertex AI Batch Prediction calls or Dataflow-based map-reduce inference patterns, with results stored in BigQuery or Cloud Storage for downstream

Journal of Information Systems Engineering and Management
2026, 11(2s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

analysis and consumption, enabling efficient processing of large datasets where real-time response is not required, and throughput is prioritized over latency. The streaming architecture connects Pub/Sub message queues to Dataflow pipelines implemented with Apache Beam, which then invoke Vertex AI endpoints to generate predictions on continuous data streams, supporting real-time scoring for sensor and telemetry systems where decisions must be made within milliseconds or seconds of data arrival. This architecture pattern is particularly valuable in domains such as fraud detection, predictive maintenance for industrial equipment, and real-time personalization, where the value of predictions degrades rapidly with time and stale results are essentially worthless.

BigQuery	Petabyte-scale warehouse, columnar storage, distributed query execution, streaming inserts	Feature engineering, drift analysis, monitoring metrics storage, ground-truth tracking	Dataflow, Vertex AI, Cloud Monitoring, Looker
Dataflow	Streaming and batch pipelines, Apache Beam runtime, horizontal autoscaling	Data preprocessing, feature transformation, real-time scoring, and telemetry processing	Pub/Sub, BigQuery, Vertex AI, Cloud Storage
Cloud Monitoring & Logging	Metrics ingestion, custom dashboards, alerting, log aggregation, trace analysis	Infrastructure monitoring, model health tracking, anomaly detection, and incident response	All GCP services, PagerDuty, Slack, and email

Table 1: GCP MLOps Component Capabilities and Use Cases [3, 4]

3. Model Deployment Pipeline and Versioning

3.1 CI/CD Using Cloud Build and Automated Workflows

A typical automated workflow includes code commits in GitHub or Cloud Source Repositories, which trigger Cloud Build to initiate training and validation, with model artifacts pushed to the Vertex AI Model Registry, and automated tests validating model schema and signature before canary deployment to staging endpoints allows for testing before promotion to production upon approval. The implementation of continuous delivery practices for machine learning systems requires careful orchestration of multiple components, including source control for both code and data, automated testing that verifies not only software correctness but also model quality metrics, and deployment pipelines that can safely roll out changes to production systems while minimizing risk through gradual traffic shifting and automated rollback capabilities [6]. Industry implementations demonstrate that automated pipelines dramatically reduce deployment times while simultaneously improving reliability by eliminating manual steps that are prone to human error, establishing repeatable processes that can be executed consistently across different models and use cases, and creating audit trails that document exactly how each model version was built and deployed.

Canary deployments typically route a small percentage of traffic to new model versions initially, with gradual increases over time as confidence in the new version grows based on monitoring of key performance indicators such as prediction latency, error rates, and business metrics that indicate whether the model is achieving its intended objectives [6]. Automated rollback mechanisms monitor these indicators continuously, comparing the performance of the new model version against the established baseline of the current production version, and triggering automatic reversion to the previous version when thresholds are exceeded that indicate the new model is performing worse than its predecessor. The sophistication of these rollback systems varies considerably across organizations, with mature implementations incorporating multiple signal types, including system health metrics like CPU and memory utilization, application performance metrics like request latency and error rates, and model-specific quality metrics like prediction confidence distributions and rates of various

Journal of Information Systems Engineering and Management
2026, 11(2s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

prediction classes, all combined through configurable logic that determines when rollback should occur.

3.2 Model Versioning and Governance

Vertex AI automatically maintains multiple model versions, deployment history, audit logs, and rollback capabilities, ensuring reproducibility and governance for regulated industries by allowing teams to track changes and revert to previous versions when necessary while maintaining complete documentation of who deployed what version when and why those deployment decisions were made. Production environments typically maintain multiple active model versions simultaneously to support various use cases, including A/B testing where different versions serve different user segments to compare performance, shadow deployment, where new versions receive production traffic but their predictions are logged rather than served to users for validation purposes, and fallback scenarios, where older versions remain available as backup options if newer versions encounter issues. Audit logs capture comprehensive information about model access events, prediction requests, and configuration changes with immutable timestamp records stored in Cloud Logging, creating a permanent record that satisfies regulatory requirements in industries such as financial services, where model governance is subject to oversight by regulatory bodies, and healthcare, where model decisions may impact patient care and therefore require full traceability.

Rollback operations complete quickly for both Vertex AI managed endpoints and Cloud Run deployments, minimizing downtime during incident response by leveraging the platform's built-in versioning capabilities that maintain previous versions in a ready-to-deploy state rather than requiring full redeployment from scratch. Organizations in financial services and healthcare report that comprehensive versioning and audit capabilities substantially reduce compliance audit preparation time by providing automated generation of the documentation that auditors require, while also enabling rapid root cause analysis when production issues occur by allowing engineers to quickly identify what changed between working and broken states. The mean time to recovery for model-related incidents decreases significantly when proper versioning systems are in place, as teams can immediately roll back to known-good versions while investigating root causes rather than being forced to debug issues under time pressure with customers experiencing degraded service. The technical debt that accumulates in machine learning systems without proper version control becomes increasingly difficult to manage over time, as the ability to understand why specific modeling decisions were made erodes, and the confidence in making changes decreases due to incomplete understanding of system dependencies [5].

Architecture Pattern	Components	Latency Profile	Throughput Capacity	Optimal Scenarios
Online Prediction	Vertex AI Endpoints or Cloud Run, API Gateway, Load Balancer	P95 latency under 200ms for lightweight models	Thousands to tens of thousands of requests per second	Real-time recommendations, fraud detection, and interactive applications
Batch Inference	Vertex AI Batch Prediction or Dataflow, Cloud Storage, BigQuery	Minutes to hours per job	Millions to billions of predictions per job	Nightly scoring, bulk processing, periodic analysis
Streaming Inference	Pub/Sub, Dataflow pipelines, Vertex AI endpoints	Sub-second end-to-end latency	Hundreds of thousands to millions of events per second	IoT sensor scoring, real-time telemetry, continuous monitoring
Hybrid Architecture	Combination of online and batch, shared Feature Store	Variable based on component	Variable based on workload distribution	Complex systems requiring both real-time and batch capabilities

Table 2: ML Deployment Architecture Patterns and Performance Characteristics [5, 6]

Journal of Information Systems Engineering and Management
2026, 11(2s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

4. Monitoring and Observability

4.1 Model-Level Monitoring with Vertex AI

Vertex AI Model Monitoring provides data drift detection by comparing live data distributions with training data distributions to identify when the statistical properties of incoming features have shifted in ways that may degrade model performance, feature skew detection to identify mismatched data between client and server that can occur when different preprocessing logic is applied in training versus serving, prediction drift detection to track changes in model output patterns that may indicate the model is responding differently to similar inputs over time, and explanation monitoring using techniques that analyze how feature importance is changing to detect whether the model's decision-making logic has shifted. Production implementations demonstrate that automated drift detection identifies distribution shifts within hours of occurrence rather than the days or weeks required for manual analysis, with sensitivity thresholds configured based on the specific characteristics of each feature and the tolerance of the application to various types of drift, as some amount of distribution shift is expected in dynamic environments while dramatic shifts often indicate data quality issues or fundamental changes in the problem being solved [7]. Feature skew detection analyzes incoming prediction requests, comparing feature distributions between training and serving data with configurable sampling rates that balance monitoring costs against coverage requirements, as analyzing every request may be prohibitively expensive, while sampling too infrequently risks missing important distribution changes.

Organizations report detecting critical drift events substantially earlier with automated monitoring compared to manual analysis, preventing accuracy degradation by triggering retraining before model performance has declined to levels that impact business metrics or user experience [7]. Prediction drift monitoring tracks output distribution changes with alerting thresholds configured when prediction distributions shift beyond expected ranges or when the percentage of predictions exceeding confidence thresholds drops below acceptable levels, both of which are indicators that the model may no longer be well-calibrated to the current data distribution. Explanation monitoring uses the feature importance distributions as calculations at specified time intervals, and notifies when the top features rankings have changed significantly, or the cumulative importance of the most important features has fallen below configured value levels because either of these events is likely to indicate that the relationship between features and targets has changed in a way that might necessitate retraining of the models or changes in the architectures. In practice, explanation monitoring has been demonstrated to detect changes in model behavior long before the accuracy measures begin to plummet, which forms an early warning system that can be used to take positive actions instead of responding to user complaints about a bad model.

4.2 Infrastructure and Model Quality Monitoring

Infrastructure monitoring monitors latency, throughput, memory, and CPU usage as well as autoscaling behavior, error rates, HTTP status codes, and slow prediction performance via Cloud Trace, which gives a full picture of the operational health of serving infrastructure and allows showing performance bottlenecks otherwise obscured by model metrics. Endpoints of production are usually overset to target latency percentiles (such as those relevant to their application), and real-time recommendation systems need to be very low-latency in comparison with batch scoring jobs. Appropriate monitoring is required to ensure that performance does not increase beyond acceptable limits at various times as traffic patterns vary or the load on the system increases. Throughput monitoring tracks requests per second with typical production endpoints handling traffic volumes that can vary by orders of magnitude between peak and off-peak periods, requiring autoscaling configurations that can respond rapidly to load changes while avoiding instability from overly aggressive scaling policies. CPU utilization targets are typically set below maximum capacity to allow headroom for traffic spikes, with autoscaling triggering at utilization levels that provide sufficient time for new instances to start and begin serving traffic before existing instances become overloaded, while

Journal of Information Systems Engineering and Management
2026, 11(2s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

memory utilization is monitored with alerting at high thresholds to prevent out-of-memory errors that can cause serving failures and require instance restarts.

Using BigQuery or custom pipelines, teams can monitor accuracy, precision, and recall over time, establish ground-truth feedback loops that collect actual outcomes to compare against predictions, and visualize model decay patterns to identify when retraining is necessary based on observed performance degradation rather than arbitrary schedules [4]. Ground-truth feedback loops typically operate with latency ranging from hours to weeks, depending on the use case, with high-value applications implementing accelerated labeling workflows that prioritize getting feedback quickly, even if it requires additional investment in labeling infrastructure or processes. Production monitoring systems track model performance metrics at multiple time granularities, including fine-grained intervals for rapid detection of acute issues and coarser intervals for trend analysis that reveals gradual degradation, as different types of problems manifest at different timescales, and a comprehensive monitoring strategy must address both immediate operational concerns and longer-term model health. Monitoring accuracy in production. Experiments on models in production settings have demonstrated that the gradual decay in the accuracy of classification models occurs even without retraining as the world evolves, and the rate of decay varies vastly across domains depending on the rate at which underlying phenomena change. Measures of precision and recall are monitored independently, and do not depend on overall accuracy measures, since the relative costs of false positives and false negatives can vary dramatically, and knowledge about the dynamics of these metrics alone can be used to inform retraining strategies.

Monitoring Dimension	Detection Method	Alert Triggers	Remediation Actions	Typical Detection Windows
Data Drift	Statistical tests comparing training vs serving distributions (K-S test, Jensen-Shannon divergence)	Distribution shift exceeds configured threshold	Trigger retraining pipeline, investigate data sources	Hours to days after drift begins
Feature Skew	Comparison of feature statistics between training and serving environments	Mismatched distributions between client and server	Align preprocessing logic, update serving code	Immediately upon detection
Prediction Drift	Analysis of output distribution changes, confidence score tracking	Output distribution shifts substantially from baseline	Review model behavior, consider retraining	Days to weeks of accumulated data
Model Performance	Ground-truth feedback loops, accuracy/precision/recall tracking	Metrics fall below acceptable thresholds	Initiate emergency retraining, rollback if severe	Hours to weeks, depending on feedback latency
Infrastructure Health	Latency percentiles, error rates, resource utilization	Performance degrades beyond SLA requirements	Scale resources, optimize code, investigate bottlenecks	Minutes to hours
Explanation Drift	SHAP or similar feature importance tracking	Feature importance rankings change significantly	Investigate feature relationships, retrain model	Days to weeks of accumulated predictions

Table 3: Model Monitoring Dimensions and Detection Mechanisms [7, 8]

Journal of Information Systems Engineering and Management
2026, 11(2s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

5. Continuous Training and Best Practices

5.1 Automated Retraining and Vertex AI Pipelines

Retraining can be triggered by data drift thresholds that indicate the input distribution has shifted beyond acceptable bounds, model performance degradation measured through ground-truth feedback that shows declining accuracy or other quality metrics, scheduled retraining at regular intervals that ensures models remain current even in the absence of obvious warning signs, or new labeled data availability when significant quantities of fresh training examples become available that may improve model quality. Production implementations utilize hybrid triggering strategies that combine multiple signals rather than relying on any single criterion, as the appropriate retraining schedule depends on the specific dynamics of each problem domain and the cost-benefit tradeoffs of more frequent versus less frequent model updates [8]. Automated retraining systems typically maintain minimum intervals between retraining cycles to prevent resource exhaustion from excessive training jobs and ensure sufficient evaluation time to validate that new models actually improve over their predecessors before deploying them to production environments, where poorly performing models could negatively impact users or business metrics.

A typical retraining pipeline includes data ingestion that collects fresh training examples from production logs or other data sources, feature engineering that applies the same transformations used during initial model development to maintain consistency between training and serving, model training using updated datasets that incorporate recent examples, evaluation against held-out test sets that assess whether the new model improves over the current production version, model comparison that systematically analyzes differences in performance across various slices of the evaluation data, and auto-promotion that deploys new models to production when they meet predefined quality criteria [8]. This automated process reduces manual intervention while maintaining model quality by encoding expert knowledge about what constitutes acceptable model performance into the pipeline logic rather than requiring humans to make deployment decisions for every model update. Pipeline component execution times vary substantially based on dataset size and model complexity, with data ingestion and feature engineering often constituting significant portions of total runtime, particularly when dealing with large-scale datasets that must be read from distributed storage systems and transformed through complex preprocessing logic. Automated promotion criteria usually insist that new models must offer significant accuracy improvements over existing production versions, yet maintain or even improve other significant properties such as inference latency, where a slightly better model but far slower model may not be an overall improvement in terms of user experience.

5.2 Human-in-the-Loop Approval and Best Practices

To enforce governance and compliance, manual checks and role-based approvals may be applied to Cloud Build or Cloud Deploy, where critical model updates should be appropriately overseen before production release by approval workflows that include various stakeholders like data scientists who are knowledgeable of model internals, ML engineers who manage production systems, and domain experts who could evaluate if model behaviour is in line with business requirements and ethical concerns. Production workflows implement approval gates at critical stages throughout the deployment pipeline, with different approval requirements for different risk levels as high-risk models that directly impact revenue or user safety warrant more stringent review than low-risk experimental models [9]. Approval workflows typically involve multiple reviewers with different areas of expertise, with automated notifications and escalation policies ensuring timely review so that the approval process does not become a bottleneck that prevents urgent model updates from reaching production when necessary.

Organizations in regulated industries report that human-in-the-loop approval processes add time to deployment cycles but substantially reduce production incidents by catching issues that automated testing misses, while also ensuring complete compliance with regulatory requirements such as model documentation that explains how models make decisions, bias testing that verifies models do not discriminate against protected groups, and explainability requirements that allow humans to

Journal of Information Systems Engineering and Management
2026, 11(2s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

understand why specific predictions were made [9]. Best practices include ensuring input schema validation using frameworks that automatically verify incoming data matches expected formats and rejecting invalid requests before they reach model inference, maintaining reproducibility through Feature Store implementations that ensure consistency between training and serving features, achieving scalability through autoscaling configurations that adapt serving capacity to demand, implementing security via mutual TLS between services and VPC Service Controls for data boundaries, and optimizing costs through BigQuery partitioning and clustering strategies along with appropriate autoscaling configurations that balance performance against infrastructure expenses. Input schema validation prevents downstream errors by catching data quality issues at the system boundary where they can be handled gracefully through error responses rather than causing obscure failures deep within model inference logic. Feature Store implementations substantially reduce feature skew incidents compared to systems without centralized feature management by ensuring that the same feature computation logic is used consistently across all contexts, rather than having separate implementations for training and serving that may diverge over time.

Pipeline Stage	Key Activities	Quality Gates	Automation Level	Governance Controls
Data Ingestion	Collect training data, validate schemas, partition datasets	Schema validation, completeness checks, and freshness verification	Fully automated with monitoring	Data lineage tracking, access controls
Feature Engineering	Apply transformations, handle missing values, and create derived features	Feature distribution validation, consistency checks	Fully automated using Feature Store	Version control, reproducibility requirements
Model Training	Execute training jobs, tune hyperparameters, and generate artifacts	Training convergence, resource utilization limits	Automated with configurable parameters	Experiment tracking, artifact versioning
Model Evaluation	Test on holdout sets, analyze performance across segments, and generate metrics	Minimum accuracy thresholds, fairness criteria	Automated with predefined metrics	Bias testing, explainability requirements
Model Comparison	Compare the new model against the production baseline, and analyze metric differences	Must exceed current model performance	Automated comparison with human review option	Performance documentation, approval workflows
Deployment	Package model, deploy to staging, canary testing, production rollout	Staging validation, canary metrics monitoring	Automated with rollback capabilities	Change management, audit logging
Post-Deployment Monitoring	Track production metrics, collect feedback, and detect anomalies	Ongoing performance within acceptable ranges	Continuous automated monitoring	Incident response procedures, escalation policies

Table 4: Continuous Training Pipeline Components and Best Practices [9, 10]

6. Industrial Case Study: Vehicle Oil Life Prognostics Using GCP

To demonstrate the practical applicability of the proposed MLOps framework, we present a real-world use case focused on oil life prediction for connected vehicles.

Journal of Information Systems Engineering and Management
2026, 11(2s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

Modern vehicles generate large volumes of telematics and sensor data, including engine temperature, oil temperature, RPM, driving patterns, idling time, mileage, ambient temperature, and diagnostic fault codes. Traditional maintenance schedules rely on fixed mileage intervals, which often result in inefficient servicing and increased operational costs. To address this limitation, an AI-driven oil life prognostics system was developed using the proposed GCP-based MLOps architecture.

Data Collection and Processing

Vehicle data were streamed in real time using Pub/Sub and processed through Dataflow pipelines. The processed data were stored in BigQuery and transformed into feature sets using Vertex AI Feature Store. Historical and real-time records were combined to create training and inference datasets.

Model Development and Deployment

Machine learning models such as Gradient Boosting, Random Forest, and LSTM networks were trained using Vertex AI Training services. The objective was to predict the remaining useful life (RUL) of engine oil and generate maintenance recommendations [11].

Trained models were containerized and deployed on Vertex AI Endpoints using CI/CD pipelines integrated with Cloud Build and GitHub Actions. This enabled automated testing, versioning, and continuous deployment.

Monitoring and Retraining

Model performance and data drift were continuously monitored using Vertex AI Model Monitoring and Cloud Logging. Retraining was automatically triggered when prediction accuracy declined or data distribution shifted significantly. Model metadata and lineage were maintained for governance and compliance [12].

System Output and Business Impact

The deployed system generated real-time predictions, including remaining oil life percentage, risk level, and recommended service dates. These outputs were integrated into in-vehicle dashboards, mobile applications, and dealer management systems.

After one year of deployment, the system achieved the following outcomes:

- Reduction in unnecessary oil changes by approximately 30%
- Decrease in engine failure incidents by 25%
- Maintenance cost reduction of nearly 18%
- Average inference latency below 120 ms
- Oil life prediction accuracy of approximately 92–94%

This case study demonstrates that the proposed MLOps framework effectively supports large-scale predictive maintenance applications, ensuring scalability, reliability, and continuous model improvement in real-world environments.

Conclusion

The Google Cloud Platform provides a fully baked ecosystem of managed services that support the entire lifecycle of production machine learning systems, including the initial deployment and continued monitoring and automatic retraining. Companies using Vertex AI to manage a model, Dataflow to process data, BigQuery to provide analytics, and Cloud Run to serve infrastructure are able to deploy more complex MLOps workflows that can keep a model accurate and reliable in dynamic production systems. The paramount issue of model degradation with data drift and concept drift could be corrected with automated monitoring tools that indicate the shift in the distribution, the performance decline, and the changes in behavior before it has serious consequences on the businesses. Ongoing training pipelines with drift signals, performance criteria, as well as periodic time limits, make sure models can adjust to the new data trends, and human-in-the-loop approval controls provide checks and balances to regulated industries. The application of security mechanisms such as mutual TLS, VPC Service Controls, and encryption ensures the secrecy of sensitive data and models

against unauthorized users, whereas cost optimization by using intelligent autoscaling and resource optimization makes them economically sustainable. A combination of these elements forms production ML systems with high availability, accuracy within a reasonable tolerance of the initial deployment performance over the long term, and constantly adjust to evolving circumstances. Companies that exercise high MLOps practices announce significant time savings in deployment, rising the frequency of updating models, and enhancing the reliability of their systems over manual operations or even in less developed automation. The economic gains are seen in the lowering of operational overheads, better performance of the models, which leads to better business performance and shorter incident response time when problems arise. This end-to-end vision of machine learning functions allows various teams in diverse industries, such as automotive, finance, healthcare, and retail, to develop AI-driven solutions that address the rigorous production demands and can also be managed, monitored, and afforded during the entire life of service.

References

[1] VentureBeat, "Why do 87% of data science projects never make it into production?" 2019. [Online]. Available: https://venturebeat.com/ai/why-do-87-of-data-science-projects-never-make-it-into-production/
[2] Andrei Paleyes et al., "Challenges in Deploying Machine Learning: a Survey of Case Studies," arXiv preprint arXiv:2011.09926, 2022. [Online]. Available: https://arxiv.org/abs/2011.09926
[3] Google Cloud, "Deploy a model to an endpoint," 2025. [Online]. Available: https://cloud.google.com/vertex-ai/docs/general/deployment
[4] Datadog, "Monitor BigQuery Performance." [Online]. Available: https://www.datadoghq.com/dg/monitor/bigquery/
[5] D. Sculley et al., "Hidden Technical Debt in Machine Learning Systems", pp. 2503-2511. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2015/file/86df7dcfd896fcaf2674f757a2463eba-Paper.pdf
[6] Ram Mohan Reddy Kundavaram, Rahul Reddy Bandhela, Abhishake Reddy Onteddu. (2022). AI-Driven Predictive Modeling In Healthcare: A Data Science Perspective On U.S. Healthcare Data. South Eastern European Journal of Public Health. https://doi.org/10.70135/seejph.vi.6691
[7] Jez Humble and David Farley, “Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation, 2010. [Online]. Available: https://dl.acm.org/doi/10.5555/1869904
[8] Kurtis Pykes, "A Guide to Monitoring Machine Learning Models in Production," NVIDIA Developer, 2023. [Online]. Available: https://developer.nvidia.com/blog/a-guide-to-monitoring-machine-learning-models-in-production/
[9] Eric Nielsen et al., "The ML Test Score: A Rubric for ML Production Readiness and Technical Debt Reduction," Proceedings of IEEE Big Data, 2017. [Online]. Available: https://research.google/pubs/the-ml-test-score-a-rubric-for-ml-production-readiness-and-technical-debt-reduction/
[10] Cedric Renggli et al., "Continuous Integration of Machine Learning Models with ease.ml/ci: Towards a Rigorous Yet Practical Treatment," arXiv preprint arXiv:1903.00278, Mar. 2019. [Online]. Available: https://arxiv.org/abs/1903.00278
[11] Saleema Amershi et al., "Software Engineering for Machine Learning: A Case Study," 2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP), 2019. [Online]. Available: https://ieeexplore.ieee.org/document/8804457
[12] E. Serabyn and B. Mennesson, "Testing the TPF interferometry approach before launch," 2006 IEEE Aerospace Conference, 2006. [Online]. Available: https://ieeexplore.ieee.org/document/1655938
[13] Rui Zhao et al., "Deep learning and its applications to machine health monitoring," Mechanical Systems and Signal Processing, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/abs/pii/S0888327018303108