International Journal of
INTELLIGENT SYSTEMS AND APPLICATIONS IN ENGINEERING
ISSN:2147-6799 www.ijisae.org Original Research Paper

Enterprise Machine Learning Model Lifecycle Management: A Seven-Phase Framework for Production-Grade MLOps

Maitray Modi

Abstract—Enterprise machine learning adoption has been happening at a pace that, consistently, far outstrips operational infrastructure to support it. Industry surveys between 2020 and 2024 show that between 78 and 87 percent of enterprise ML models never make it to production, a number that has been remarkably steady in the face of the rapid upskilling of algorithms and cloud economics. The production gap isn't caused by the mathematics of ML itself but by the ML model lifecycle, where ML models progress from data acquisition to training, validation, testing, deployment, monitoring, and finally, retirement. In the absence of an ML lifecycle management framework, ML models quickly decay in production. This can result in the considerable erosion of trust in investment in artificial intelligence. We propose a seven-phase ML lifecycle management framework based on cloud-based enterprise platform architectural patterns of Amazon Web Services and the Microsoft Azure cloud platform. It evaluates the platform capabilities, governance controls, and operational automations needed for each stage to operate at scale reliably. That comprises controlled feature stores, experiment tracking infrastructure, progressive deployment patterns and drift detection technology. It discusses MLOps lifecycle maturity governance and legislation in the context of the European Union Artificial Intelligence Act and United States federal AI policy. The framework provides a practical architecture reference point to enterprise engineers and technology leaders to close the divide and support sustainable production value from machine learning investments.

Index Terms—*MLOps, Machine Learning Lifecycle, Feature Store, Model Monitoring, Concept Drift, Continuous Training, Model Governance, Enterprise AI, AWS SageMaker, Azure Machine Learning.*

I. Introduction

This is qualitatively different, not just quantitatively different, from a decade ago, when data science teams were often the analytical think tanks of the organization, handing their insights to a human with a decision-making muscle to evaluate and act using their best judgment as the final arbitration layer. Modern machine learning models are deployed as autonomous decision-making agents within the enterprise operations stack: approving credit, routing customer calls, adding fraud review flags to transactions, automating in-stock replenishment order flows, and customizing product experiences at a population scale too large for even expert human review. These operational models have advanced far beyond the data analytics models which preceded them, but model engineering practices have not evinced a similar maturation and advancement with respect to model development, maintenance, and retirement.

Independent Researcher, USA

One of the greatest sources of structural waste in enterprise technology is the gap that exists between the development of a machine learning model and its deployment to production. Surveys conducted between 2020 and 2024 by Gartner, Algorithmia and the McKinsey Global Institute estimate that between 78% and 87% of enterprise machine learning models are never deployed [1]. In the subset of models that make it to production, 60 percent are estimated to have an important performance degradation within six months, a drop that is often not detected until a downstream business metric has changed. This leads to a compounding, mostly hidden misallocation of engineering resources. The models that required months of data-science work now mostly sits in model registries or data scientists' notebooks without generating any value in production, while the capabilities they were meant to enable remain unjustified.

This is not an error mode unique to their algorithm, and even the models themselves are typically capable of solving the problems that they are tasked with. The fundamental issue here is the lack of

infrastructure for machine learning life cycle management. Without a strong environment around data pipelines, feature engineering, experiment reproducibility, evaluation rigor, deployment safety, monitoring coverage, and retirement discipline, machine learning is just a sequence of experiments rather than a compounding engineering capability. Each model is built afresh, informally evaluated, and, if deemed sufficient, manually deployed and monitored only until the engineer who built it moves on to their next project [4].

The article proposes a seven-phase machine learning model lifecycle management framework that addresses the issue of the missing structural governance in most enterprise machine learning programs. It is based on architectures of production ML systems on Amazon Web Services and Microsoft Azure and MLOps maturity models from platform providers and independent researchers. We have designed this as an engineering reference architecture, not a prescriptive taxonomy. Building on the lifecycle stages, Section III addresses the foundations of the data. Sections IV, V, VI, and VII respectively address the deployment, monitoring, governance, and organizational stages. Section IX concludes the paper.

II. The Seven-Phase ML Lifecycle: Conceptual Framework

The seven-phase framework organizes the complete operational arc of an enterprise machine learning model into a governed, instrumented sequence: (1) data ingestion, (2) feature engineering, (3) model training, (4) model evaluation, (5) model deployment, (6) model monitoring, and (7) governance and retirement. Each phase produces specific artifacts—curated datasets, versioned feature definitions, trained model checkpoints, evaluation reports disaggregated by subgroup, inference endpoints, monitoring dashboards, and audit records—that serve both as inputs to the subsequent phase and as the evidentiary documentation required for lifecycle governance.

The critical distinction between this framework and simpler three-step conceptions of the machine learning workflow—train, evaluate, deploy—is its explicit recognition that production machine learning is iterative and event-driven, not sequential and terminal. A production model does not pass through the lifecycle once and thereafter operate unchanged. It cycles through training, evaluation, and deployment repeatedly, triggered by detected data drift, accumulated label feedback, changed business requirements, or scheduled retraining intervals. The governance and platform infrastructure that supports this iteration must be designed from the outset to handle continuous cycling rather than treating each training run as an exceptional event requiring manual coordination.

This iterative architecture is the defining contribution of MLOps as an engineering discipline: the application of continuous integration and continuous deployment principles—version control, automated testing, progressive rollout, and rollback—to the specific challenges of machine learning systems, where quality depends on data as well as code, outcomes are probabilistic rather than deterministic, and the relevant failure mode is performance drift rather than functional defect [1]. Google's MLOps maturity model, Microsoft's AI maturity framework, and Amazon's SageMaker MLOps documentation each describe a progression from manual, notebook-centric processes at low maturity to fully automated, event-driven pipelines at high maturity that closely correspond to the seven-phase architecture described here.

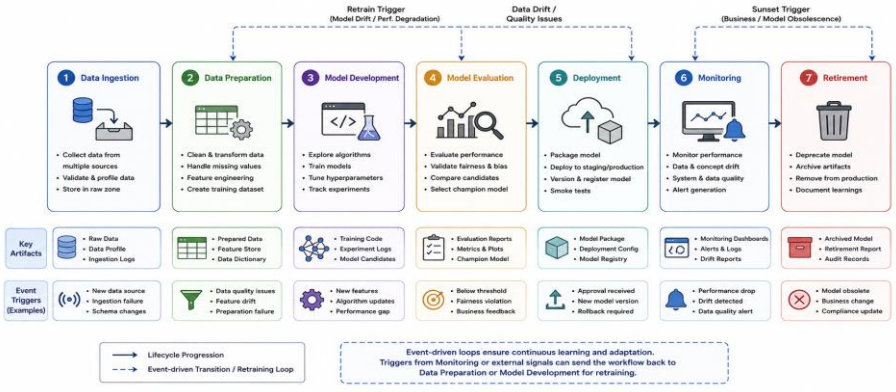

Fig. 1. Seven-phase ML lifecycle framework showing data ingestion through retirement with event-driven retraining loops.

III. Phase 1–2: Data Ingestion and Feature Engineering

No machine learning system can be more reliable than the data infrastructure that feeds it, and no dimension of enterprise machine learning architecture is more routinely underinvested relative to its impact. In production enterprise environments, training data is not pre-cleaned and pre-formatted from a single authoritative source. It is assembled on-the-fly from dozens of heterogeneous systems: transactional relational databases, high-velocity event streams, document object stores, third-party application programming interface (API) feeds, and legacy batch exports, each of which has its own distinct schema conventions, quality guarantees, freshness characteristics, and access control requirements. This means that the ingestion layer needs to solve schema normalization, schema evolution, anomaly detection with quarantining, and normal delivery to other pipelines subject to the latency, completeness, and freshness constraints of the training schedule.

The Lakehouse architecture, a pattern that combines the scalable and flexible schema of a data lake with the ACID transactions and SQL consumption found in a data warehouse, has become the de facto enterprise solution to this ingestion challenge [11]. By utilizing open table format layers like Apache Iceberg and Delta Lake, the Lakehouse provides versioned and time-traveling or snapshot views of the training dataset to machine learning pipelines to enable reproducible training runs while the source data is endlessly growing. According to internal performance benchmarks, Databricks Delta Lake-based Lakehouse implementations reduce the effort to create data pipelines by 40% relative to the fragmented lake-plus-warehouse architectures where the costs of storing the data, reconciling data schemas and managing data permissions have to be duplicated for analytical and machine learning workloads on the same data [11].

Feature engineering is overrated in machine learning textbooks given the coverage and importance it is attributed. A 2023 Algorithmia survey of data scientists at early stage and enterprise firms found that data scientists spend 60 to 80% of their time on data preparation and feature engineering chores, which has remained consistent with the growth of machine learning technology and automation. The biggest pain point with feature engineering at enterprise scale is the phenomenon of training-serving skew (TSS), which occurs when the features computed in the training phase are different from those computed during inference (i.e., in production) for the same concept. Skew is when two engineers make slightly different assumptions about the same feature (different null-handling conventions, different windows for temporal aggregation, different conventions for the representation of temporal units, etc.). A model that has good offline evaluation metrics may suffer when exposed to real-world production inputs [3].

A managed feature store enables users to define features in a canonical way, which is computed once, versioned, and served to both the offline training pipeline and the online inference endpoint [10]. Amazon SageMaker Feature Store stores its offline store in Amazon S3 and provides batch access to training jobs using Amazon Athena, while the low-latency online store is backed by Amazon DynamoDB for online inference access. The Azure Machine Learning Feature Store integrates with Azure Databricks for distributed feature computations and Azure Data Factory for orchestration. The central definition repository serves as a source of truth for both the training and serving pipelines. According to customers using managed feature stores, training-serving skew incidents have dropped by 70 to 85%, while model teams have improved feature reuse.

TABLE I. Managed Feature Store Platform Comparison

Platform	Offline Store	Online Store	Versioning	Primary Integration
AWS SageMaker Feature Store	S3 + Athena	DynamoDB	Yes	SageMaker Pipelines, Studio
Azure ML Feature Store	ADLS Gen2	Redis / Cosmos DB	Yes	Databricks, ADF, Synapse
Google Vertex AI Feature Store	BigQuery	Bigtable	Yes	Vertex AI Pipelines, BQ ML

Databricks Feature Store	Delta Lake	Online serving endpoint	Yes	MLflow, Unity Catalog
Feast (Open Source)	Configurable (S3, GCS)	Redis / DynamoDB	Yes	Kubeflow, Airflow, MLflow

IV. Phase 3–4: Model Training, Experiment Tracking, and Evaluation

Enterprise model training is a sustained experimental program rather than a single computational event. Before a data science team arrives at a model configuration that satisfies the quality, fairness, latency, and cost requirements of its target production environment, it typically executes dozens to hundreds of training runs—varying the training dataset version, the feature set composition, the model architecture, and the hyperparameter configuration in a disciplined search for the optimal configuration. Without systematic experiment tracking infrastructure, this program is unauditable in both technical and organizational terms: the team cannot reliably reproduce a prior result, compare two experimental configurations on equal footing, or trace the production model back to the specific training run and dataset version that produced it [2].

MLflow, the dominant open-source experiment tracking platform, addresses this by providing a standardized logging API that captures the complete provenance record of each training run—input data version, feature set version, code commit, compute environment specification, hyperparameter values, and output metrics—in a centralized artifact store that persists for the lifetime of the model [10]. Both SageMaker Experiments and Azure Machine Learning Experiments provide managed implementations of this tracking pattern with native integration to their respective training infrastructure, model registries, and deployment pipelines. The practical consequence is that the question "what exactly produced the model currently serving production?" becomes answerable in seconds rather than requiring an investigation across scattered notebooks, shared drives, and team memory.

The evaluation phase is where the gap between academic machine learning practice and enterprise machine learning practice is most sharply apparent. Academic evaluation is primarily a statistical exercise: compute performance metrics on a held-out test set and compare them to a published baseline. Enterprise evaluation must address four additional dimensions that academic benchmarks systematically neglect [2]. Business impact alignment translates model predictions into the specific operational outcomes the sponsoring team cares about. Fairness evaluation assesses whether the model's predictions differ in quality or in impact across demographic subgroups in ways that violate ethical commitments or legal obligations [5]. Robustness evaluation probes how performance degrades under the distributional shifts most likely to occur in production. Computational evaluation determines whether the model's inference latency and infrastructure cost meet the budgets of the production serving environment.

Automating this multi-dimensional evaluation rather than delegating it to individual data scientists' judgment is not merely a quality improvement—it is a precondition for the velocity that production machine learning requires. A 2022 study across 331 organizations with active machine learning programs found that organizations with automated multi-dimensional evaluation pipelines promoted models to production 3.2 times faster than those relying on manual review processes, while experiencing 47 percent fewer emergency rollbacks in the 90 days following deployment [1]. The interpretation is straightforward: automated evaluation catches problems that manual review misses not because automated tools are more intelligent than data scientists, but because they are applied consistently to every candidate model rather than variably depending on individual workload and judgment.

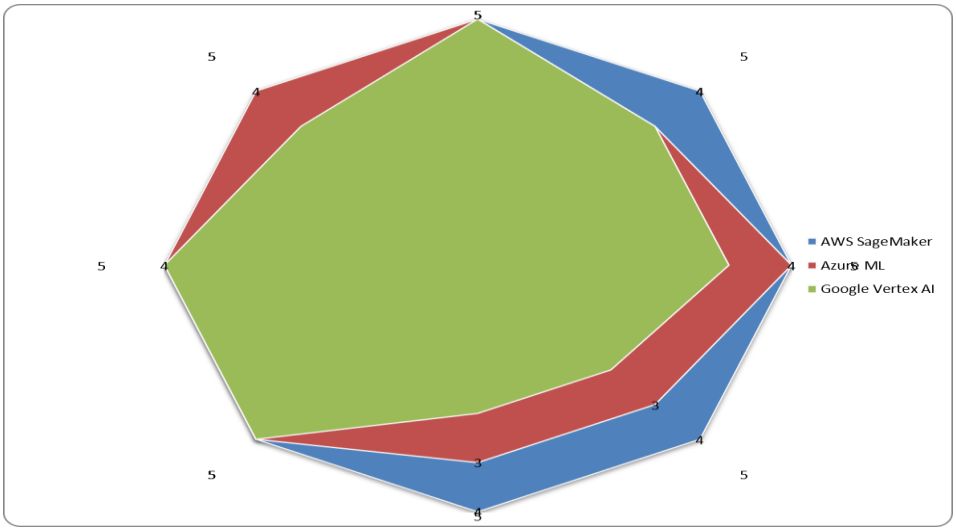

Fig. 2. Radar chart comparing AWS SageMaker, Azure ML, and Google Vertex AI across model training, feature stores, deployment, monitoring, governance, experiment tracking, AutoML, and CI/CD integration.

TABLE II. Model Evaluation Dimensions Matrix

Evaluation Dimension	Primary Metric	Tooling	Regulatory Driver
Statistical performance	AUC-ROC, Precision, Recall, RMSE	SageMaker Clarify, Sklearn	General quality baseline
Business impact alignment	Revenue delta, cost-per-error, KPI change	A/B framework, custom analytics	Sponsor sign-off gate
Fairness & bias	Disparate impact ratio, demographic parity	SageMaker Clarify, Fairlearn	EU AI Act, ECOA, CCPA
Robustness / drift resilience	Performance under distributional shift	Adversarial test sets, slice analysis	High-risk AI system requirements
Computational feasibility	P99 latency (ms), inference cost (USD)	Load testing, SageMaker Inference	SLA constraints, cost budgets

V. Phase 5: Deployment Patterns — Shadow, Canary, and Blue-Green

The engineering distance between a data scientist's notebook environment and a production machine learning system is structural, not cosmetic. Notebooks are designed for interactive exploration: computational state accumulates across cells, execution order is non-deterministic, dependencies are often implicit in the kernel environment rather than declared in reproducible package specifications, and there is no enforcement mechanism for the access controls, audit logging, or fault tolerance that production systems require. Translating a notebook-resident model into a production-grade inference system requires packaging the model as a versioned artifact; containerizing the serving environment with declared dependencies; wiring the inference endpoint into the organization's authentication and monitoring infrastructure; and establishing a deployment pipeline that can promote new model versions safely without interrupting the production traffic the existing endpoint is serving [3].

Shadow deployment is the lowest-risk entry point in a staged promotion sequence. The new model version receives a mirrored copy of all live production requests—identical inputs, processed simultaneously with the incumbent—generating predictions that are logged and analyzed without being served to end users. This configuration provides something that offline evaluation on a held-out dataset fundamentally cannot: measurement of the new model's performance against the actual, unfiltered distribution of production inputs, including tail cases, adversarial inputs, and the subtle data quality issues that affect real-world

request streams but are underrepresented in curated evaluation datasets [9]. The cost of shadow deployment is approximately double the inference compute cost of normal serving during the validation window; for most applications this overhead is well justified by the risk reduction it provides before any user is exposed to the new model.

Canary deployment follows shadow validation by routing a controlled fraction of live production traffic to the new model—typically one to five percent—while the incumbent continues to serve the remainder. The canary fraction is chosen to provide statistically meaningful performance data within a reasonable time window while limiting the blast radius of any quality degradation. Both SageMaker and Azure Machine Learning provide traffic-splitting primitives for their managed inference endpoints that implement canary deployment natively, with configurable rollback triggers tied to monitored metrics. Production traffic is shifted from canary to full deployment automatically when the monitored metrics confirm that the new model performs comparably to or better than the incumbent, without requiring human intervention in the promotion decision [1].

Blue-green deployment maintains two fully provisioned production environments—one serving current traffic, one hosting the new model version—and switches traffic atomically between them at promotion time. The primary advantage over canary is rollback speed: reverting to the prior model version requires only a routing table update rather than a model redeployment, reducing rollback time from minutes to seconds. For latency-sensitive, high-value applications such as real-time fraud scoring or personalized search ranking where even a brief quality degradation has direct revenue impact, this cost-doubled infrastructure is generally justified by the rollback speed advantage [9].

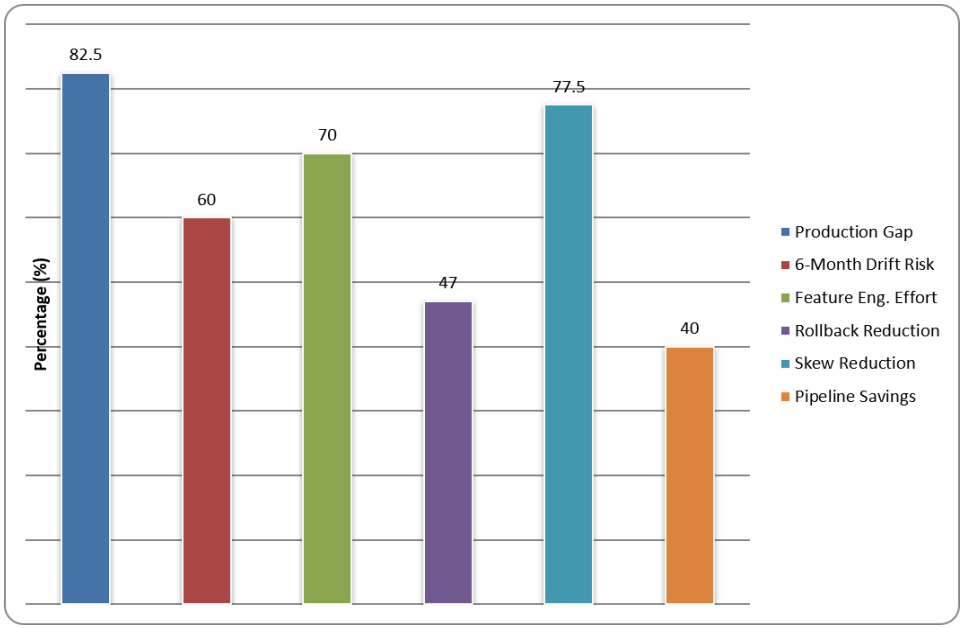

Fig. 3. Progressive deployment pipeline illustrating shadow, canary, and blue-green stages with automated metric-driven promotion and rollback gates.

VI. Phase 6: Model Monitoring — Drift Detection and Continuous Training

The core assumption of supervised learning is that the joint distribution of inputs and outputs at inference time will resemble the distribution observed during training closely enough that a model optimized on training data will generalize effectively to production inputs. In research settings this assumption is enforced by experimental design. In production environments no such enforcement is possible: the world changes, and the inputs the model receives at inference time diverge from its training distribution continuously and unpredictably. This divergence—performance drift—is not an exceptional failure mode but an inevitable operating condition for any production model, and monitoring infrastructure that detects and responds to it is accordingly a core architectural

requirement rather than an operational afterthought [6].

Three distinct degradation mechanisms require independent detection strategies. Data drift is the change in the statistical distribution of input features presented to the model at inference time, relative to the distribution observed during training. It does not require outcome labels to detect; the training feature distribution can be stored as a baseline at deployment time and compared against the distribution of recent inference inputs using statistical tests. The Kolmogorov-Smirnov (KS) test is appropriate for continuous features; the chi-squared test addresses categorical features; the Population Stability Index (PSI) provides a composite summary metric well-suited to trending and alerting over time [8]. SageMaker Model Monitor implements automated data drift detection as a scheduled managed job that compares the inference request distribution against the stored training baseline, publishing violation reports to Amazon CloudWatch without requiring custom monitoring code from the model development team.

Concept drift is more insidious: it is the change in the relationship between inputs and correct outputs, independent of any change in the input distribution itself [6]. A fraud detection model may encounter inputs whose feature distributions are stable while the actual fraud patterns shift in response to new attack vectors. Detecting concept drift requires ground truth outcome labels, which arrive with a delay that varies by application domain from hours to months. During the label latency window, organizations must rely on proxy signals—prediction confidence distribution shifts, output class distribution changes, and downstream business metrics—to identify potential concept drift before confirmed labels are available [7]. Designing a monitoring architecture that accounts for label latency from the start is a hallmark of lifecycle-mature engineering teams.

The continuous training loop that closes the monitoring-to-retraining cycle is the operational core of a high-maturity MLOps implementation. Event-driven retraining pipelines connect drift detection alerts, scheduled retraining triggers, and new labeled data availability signals to automatic execution of the full training, evaluation, and deployment pipeline—producing a model lifecycle that self-corrects without requiring manual intervention for routine performance recovery [9]. Organizations that have implemented continuous training report reductions in mean time to recover from performance degradation events from two to four weeks under manual monitoring to under 24 hours under automated pipelines.

TABLE III. Drift Detection Methods and Tooling

Drift Type	Detection Method	Key Statistical Test	AWS/Azure Tooling	Labels Required
Data drift	Input feature distribution shift	KS test, chi-squared, PSI	SageMaker Model Monitor, Azure ML Data Drift	No
Concept drift	Input-output relationship change	Performance on new labeled samples	SageMaker Clarify, custom eval jobs	Yes (delayed)
Prediction drift	Output class distribution shift	PSI on prediction distribution	SageMaker Model Monitor, Evidently AI	No (proxy)
Pipeline degradation	Data quality / upstream failures	Null rate, schema validation alerts	AWS Glue, Azure Data Factory alerts	No

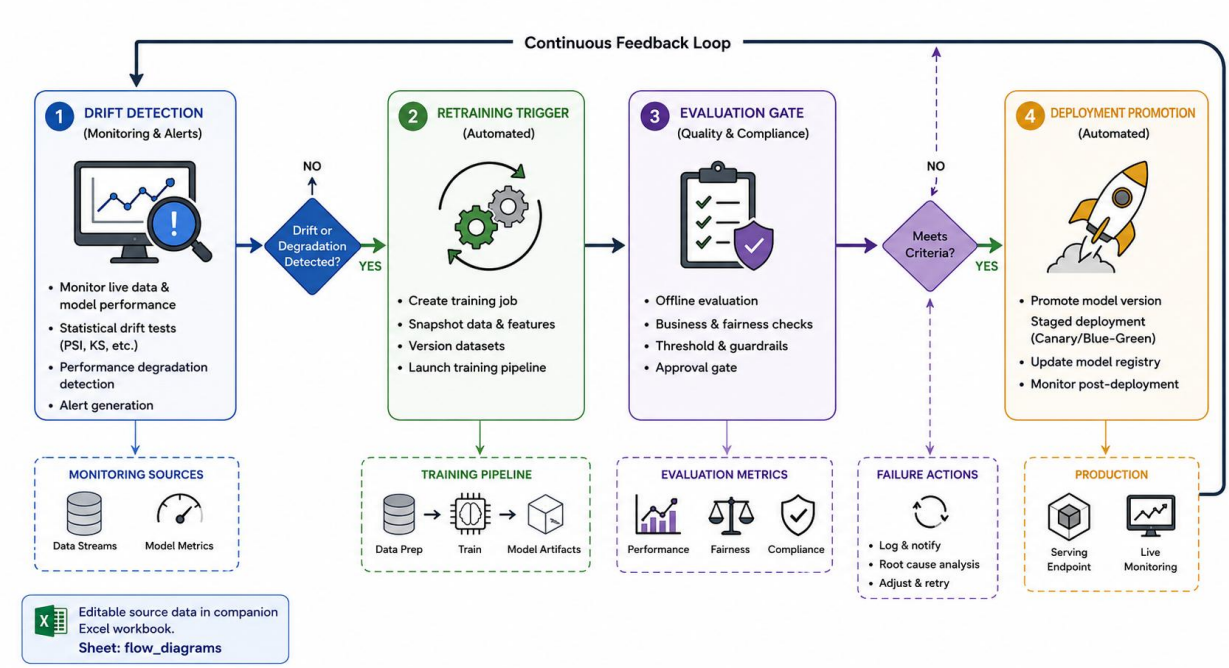

Fig. 4. Event-driven continuous training loop connecting monitoring alerts to automated retraining and deployment pipelines.

VII. Phase 7: Governance, Model Cards, and Retirement

Governance is the lifecycle phase most frequently deferred to after deployment and most consequential when it is. Organizations that rush models into production without establishing documented provenance, bias evaluation results, explainability infrastructure, and access control policies discover their oversight too late: when a regulatory inquiry arrives, when a discrimination complaint is filed, or when a model failure causes a significant customer impact and leadership asks questions the data science team cannot answer [4]. The governing principle of the lifecycle framework presented here is that governance must be a design constraint embedded in every preceding phase—in the data pipelines, the experiment tracking system, the evaluation criteria, and the deployment authorization workflow—rather than a compliance activity performed after the fact.

Model cards are the primary documentation artifact through which governance is embedded in the model artifact itself rather than maintained as a separate administrative record. First proposed by Mitchell et al. at the 2019 ACM Conference on Fairness, Accountability, and Transparency, a model card is a structured document that accompanies a trained model through its lifecycle, capturing its intended use case, the training data and its known limitations; the evaluation methodology applied and the results obtained disaggregated across relevant subgroups; and the contexts in which the model is recommended or prohibited from use [5]. SageMaker Model Cards implements this pattern as a managed service integrated into the SageMaker model registry, ensuring governance documentation is version-controlled alongside the model artifact and cannot be decoupled from it through deployment and monitoring phases.

The European Union Artificial Intelligence Act, which entered into force on 1 August 2024 and is progressively applicable through 2026, translates these governance practices into binding legal obligations for a defined class of high-risk AI applications [12]. High-risk systems—those used in employment screening, credit assessment, biometric identification, access to essential services, and critical infrastructure management—are required to maintain technical documentation covering the system's intended purpose, the training data and its provenance, the evaluation methodology and performance characteristics, the risk management measures applied, and the human oversight mechanisms in place. This documentation requirement maps directly onto the lifecycle management artifacts described throughout this article: organizations whose model development programs have already generated experiment logs, evaluation reports, model cards, and deployment audit trails are substantially better positioned to

satisfy EU AI Act compliance demands than those managing machine learning development without this infrastructure.

Model retirement closes the governance loop by formally documenting the end of a model's production service life and ensuring that the transition to any successor model is managed with the same rigor applied to the original deployment. A structured retirement process includes a shadow period during which the successor model absorbs production traffic alongside the retiring model, confirming performance equivalence before the retiring endpoint is decommissioned; data retention policies defining how long training data, model artifacts, and inference logs must be preserved for audit purposes; and a retirement record that documents the rationale, the timeline, and the authorization for retirement in the model registry [12].

TABLE IV. Governance Framework Elements

Governance Component	Primary Tooling	Key Artifact	Regulatory Driver
Model Cards	SageMaker Model Cards, Hugging Face	Model card document (versioned)	EU AI Act Art. 11, OSTP AI Bill of Rights
Bias Audit Reports	SageMaker Clarify, Fairlearn, Aequitas	Subgroup performance report	ECOA, Fair Housing Act, EU AI Act
Explainability Dashboard	SHAP, LIME, Captum, InterpretML	Feature importance + local explanations	EU AI Act Art. 13 (transparency)
Experiment Provenance	MLflow, SageMaker Experiments, Azure ML	Run logs, data version, code hash	EU AI Act Art. 11 (technical documentation)
Model Registry + Audit Log	SageMaker Model Registry, MLflow	Deployment history, approval chain	SOX, GDPR, EU AI Act Art. 17

VIII. Organizational and Competitive Implications

The organizational economics of machine learning lifecycle management are characterized by significant positive returns to scale. The first model managed through a governed lifecycle platform requires substantial platform investment—managed infrastructure, feature store setup, experiment tracking integration, evaluation pipeline development, monitoring configuration, and governance documentation templates. Each subsequent model draws on this investment, amortizing the platform overhead across a growing portfolio while benefiting from features, evaluation pipelines, and monitoring configurations already developed for earlier models [2]. Organizations that have reached lifecycle maturity report that new model deployment timelines shrink dramatically as the platform matures, not because individual engineering effort decreases but because the shared infrastructure that each model depends on already exists and only requires configuration rather than construction.

The fairness implications of lifecycle management scale in the same direction. Algorithmic fairness is not achievable as an individual model property in an unmanaged machine learning environment where different teams apply different evaluation criteria, different demographic subgroup definitions, and different bias thresholds to different models without coordination [5]. It becomes achievable as an organizational property when lifecycle governance enforces consistent fairness evaluation standards, common subgroup definitions, and mandatory bias audit documentation across the full model portfolio. The shift from individual discretion to organizational standard is the only mechanism that reliably produces consistent fairness outcomes at the scale of a large enterprise machine learning program.

Competitive positioning in markets where machine learning capability is a significant differentiator increasingly depends on lifecycle maturity rather than on algorithmic sophistication. The machine learning algorithms available to enterprise data science teams are largely commoditized—all significant competitors have access to the same

gradient boosted tree implementations, the same transformer architectures, and the same cloud computing infrastructure. What differentiates leaders from laggards is the speed at which they can move new model capabilities from development to production, the reliability with which those models sustain performance over time, and the governance rigor with which they manage model quality across a growing portfolio [1]. These are lifecycle management outcomes, not algorithmic outcomes, and they accumulate as a compounding advantage in organizations that invest in the underlying platform infrastructure early.

IX. Conclusion

The production deployment gap that motivates this article is not a temporary artifact of an immature field; it is the predictable consequence of attempting to operate a complex operational system without the engineering discipline that complexity requires. Machine learning models in production are not static artifacts—they are dynamic systems that depend on continuously evolving data, decay in quality as the world around them changes, and carry governance obligations that extend throughout their service life and beyond. Managing this complexity requires a structured lifecycle framework, and the seven-phase framework presented here provides the architectural vocabulary, the platform guidance, and the governance principles to build it.

The practical accessibility of this framework has never been higher. The managed services that implement its most complex components—SageMaker Feature Store, SageMaker Experiments, SageMaker Model Monitor, SageMaker Model Cards, and their Azure Machine Learning equivalents—are production-ready, well-documented, and available without the months of custom engineering that equivalent capabilities required five years ago [3]. The remaining investment is organizational: the deliberate decision to treat lifecycle management as a first-class systems architecture priority, staffed and funded accordingly, rather than as an operational concern to be addressed after models are already in production struggling.

The regulatory environment will increasingly make this decision for organizations that have not made it themselves. The EU AI Act's binding governance requirements for high-risk systems, the United States Office of Science and Technology Policy's AI accountability framework, and the emerging AI regulatory structures being developed in the United Kingdom, Canada, and across Asia-Pacific collectively signal a near-term future in which demonstrable lifecycle governance is not a competitive differentiator but a legal prerequisite for operating AI systems in consequential domains [12]. Organizations that build this infrastructure proactively will meet these requirements as an operational byproduct of good engineering rather than as a compliance emergency. That is the practical promise of machine learning model lifecycle management as a systems architecture discipline.

References

[1] D. Kreuzberger, N. Kühl, and S. Hirschl, "Machine learning operations (MLOps): overview, definition, and architecture," IEEE Access, vol. 11, pp. 31866–31879, 2023. [Online]. Available: https://ieeexplore.ieee.org/document/10081336/

[2] S. Amershi et al., "Software engineering for machine learning: a case study," in Proc. IEEE/ACM 41st Int. Conf. Software Engineering: Software Engineering in Practice (ICSE-SEIP), 2019, pp. 291–300. [Online]. Available: https://ieeexplore.ieee.org/document/8804457/

[3] D. Baylor et al., "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. [Online]. Available: https://dl.acm.org/doi/10.1145/3097983.3098021

[4] D. Sculley et al., "Hidden technical debt in machine learning systems," in Proc. 29th Int. Conf. Neural Information Processing Systems (NeurIPS), 2015, pp. 2503–2511. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2015/hash/86df7dcfd896fcaf2674f757a2463eba-Abstract.html

[5] M. Mitchell et al., "Model cards for model reporting," in Proc. ACM Conf. Fairness, Accountability, and Transparency (FAccT), 2019, pp. 220–229. [Online]. Available: https://dl.acm.org/doi/10.1145/3287560.3287596

[6] A. Liu et al., "Concept drift detection delay index," IEEE Trans. Knowl. Data Eng., vol. 35, no. 5, pp. 4585–4597, 2023. [Online]. Available: https://ieeexplore.ieee.org/document/9729470/

[7] P. Li et al., "High-dimensional multilabel data stream classification with concept drifting detection," IEEE Trans. Knowl. Data Eng., vol. 35,

no. 8, pp. 8085–8099, 2023. [Online]. Available: https://doi.org/10.1109/TKDE.2022.3200068

[8] Y. Lu et al., "Learning under concept drift: a review," IEEE Trans. Knowl. Data Eng., vol. 31, no. 12, pp. 2346–2363, 2019. [Online]. Available: https://doi.org/10.1109/TKDE.2018.2876857

[9] P. Ruf, M. Madan, C. Reich, and D. Ould-Abdeslam, "Demystifying MLOps and presenting a recipe for the selection of open-source tools," Applied Sciences, vol. 11, no. 19, p. 8861, 2021. [Online]. Available: https://www.mdpi.com/2076-3417/11/19/8861

[10] M. Zaharia et al., "Accelerating the machine learning lifecycle with MLflow," IEEE Data Eng. Bull., vol. 41, no. 4, pp. 39–45, 2018. [Online]. Available: http://sites.computer.org/debull/A18dec/p39.pdf

[11] M. Zaharia, A. Ghodsi, R. Xin, and M. Armbrust, "Lakehouse: a new generation of open platforms that unify data warehousing and advanced analytics," in Proc. 11th Conf. Innovative Data Systems Research (CIDR), 2021. [Online]. Available: http://cidrdb.org/cidr2021/papers/cidr2021_paper17.pdf

[12] European Parliament and Council of the European Union, "Regulation (EU) 2024/1689 laying down harmonised rules on artificial intelligence (Artificial Intelligence Act)," Official Journal of the European Union, 2024. [Online]. Available: https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:32024R1689

[13] A. Paleyes, R.-G. Urma, and N. D. Lawrence, "Challenges in deploying machine learning: a survey of case studies," ACM Comput. Surv., vol. 55, no. 6, pp. 1–29, 2023. [Online]. Available: https://dl.acm.org/doi/10.1145/3533378

[14] E. Breck et al., "Data validation for machine learning," in Proc. Machine Learning and Systems (MLSys), 2019. [Online]. Available: https://proceedings.mlsys.org/paper_files/paper/2019/hash/928f1160e52192e3e0017fb63ab65391-Abstract.html

[15] K. Vanhaesebrouck et al., "On continuous integration/continuous delivery for automated deployment of machine learning models using MLOps," in Proc. IEEE Int. Conf. Emerging Technologies and Factory Automation (ETFA), 2021. [Online]. Available: https://ieeexplore.ieee.org/document/9723793/