Deep Science Publishing, 2026
https://doi.org/10.70593/978-93-7185-416-0_9

DeepScience
Open Access Books

Chapter 9: Automation, MLOps, and Continuous Learning in Production AI Environments

9.1. Introduction

Although MLOps concepts and tooling have been evolving for over a decade, gaps remain and trends are still unfolding. MLOps can be defined as the application of DevOps principles and practices to the machine learning (ML) lifecycle. The original motivation lay in the well-established empirical observation of the "80–20" rule, with relatively little effort being spent on the engineering aspects that enable the operational use of models in production.

A successful end-to-end MLOps capability helps organizations to fully realize the promise of ML. It facilitates reproducibility of results, encourages experimentation with a broad variety of models, allows intelligent systems to operate reliably in production, and improves return on investment by increasing the success rate of model updates. Rather than adopting a one-size-fits-all approach, organizations should consider MLOps a collection of patterns and best practices rather than a prescriptive checklist with binary adherence criteria.

9.1.1. Overview of MLOps Fundamentals

State-of-the-art MLOps concepts, roles, and lifecycle stages are defined; MLOps are distinguished from DevOps, and governance requirements and compliance implications are identified; stakeholders are mapped to responsibilities; a typical tooling ecosystem and data flows are summarized, and success metrics for production AI are enumerated.

The principal core principles are articulated: reproducibility, automation, CI/CD for machine learning, data lineage, monitoring, and governance; architecture patterns for production machine learning systems are described; and risk management, scalability, and security considerations are discussed. The integration of machine learning models within enterprise software pipelines is explained.

9.2. Foundations of MLOps in Production

MLOps operates on core principles that ensure enterprise-grade deployment and monitoring of machine learning components. These principles encompass reproducibility, automation, continuous integration and delivery for machine learning, data lineage capture, monitoring, and governance capabilities. Moreover, architectural patterns provide guidance for structuring production ML systems. Successful operationalization of ML requires careful consideration of risk management, scalability, and security. A holistic approach integrates ML models into enterprise software pipelines, managing the risks introduced by ML and ensuring that higher-risk business-critical models can be scaled effectively.

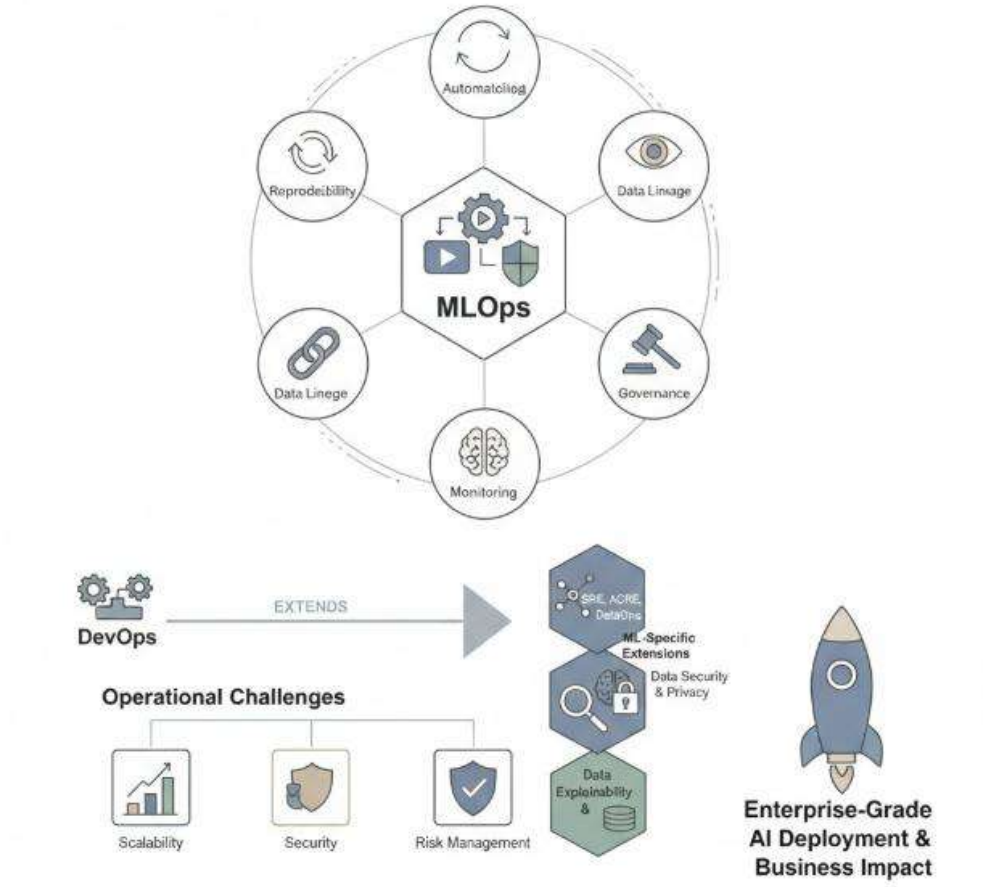

Fig 9.1: From DevOps to Enterprise MLOps: Evolution of Architectural Patterns for Scalable, Risk-Aware Machine Learning Governance

MLOps builds upon traditional DevOps practice. While the core integration and automation practices will be very familiar to anyone with a solid grasp of DevOps, the scale and particular characteristics of ML workloads require some modification of the stack. Consequently, while applying DevOps capabilities to ML workloads is also often

described as MLOps, it typically involves specific extensions or variations to support many of the challenges in operating and deploying ML models supported by SRE, ACRE, or DataOps frameworks. These go above and beyond the very important concerns of data security, audit, explainability, data privacy, and protection to deal with the remaining risks now introduced by ML although they themselves form part of the overall MLOps picture.

9.2.1. Core Principles of MLOps Implementation

Production ML systems must be designed and built in accordance with a set of core principles that collectively support their long-term success and sustainability. Models deliver value only if they remain reliable in varying real-world conditions. They must therefore be subjected to constant scrutiny and monitoring to expose even subtle performance changes. When these changes threaten the value delivered by a model, alerts must be raised for human review or, in certain conditions, a model must be retrained and redeployed automatically.

MLOps is an abbreviation of Machine Learning Operations and derives much of its philosophy from DevOps. Reproducibility, automation, continuous integration and delivery, data lineage, monitoring for data and model quality, and risk management are as important for ML as for software engineering, yet convincing an organization to treat them with suitable seriousness is challenging. Production ML applications differ fundamentally from traditional software—data is central—and the software community lacks an established discipline for MLOps. The creation of satisfactory observations is non-trivial and usually requires business domain knowledge, combined with clear delineation of acceptable and unacceptable model behaviors. Such detailing is the realm of MLOps and it needs to be enshrined within governance tooling (e.g., ServiceNow) as much as coding standards. With increasing scrutiny from external regulators and legal liabilities for biases incorporated into AI systems, governance must also incorporate fairness and bias considerations. Failure to detect and remediate undesirable model events introduces risk and risk is a valid reason for not deploying a model. Properly completed MLOps practices significantly de-risk production deployment.

9.3. Automation in AI Pipelines

Automation increases the speed and reduces the cost of delivering information products. All stages of processing—data collection, model training, monitoring, and deployment—are candidates for automation. Having made the business case for a production AI pipeline, practical considerations drive which aspects to automate and the

extent of that automation. Autonomy of the systems involved is local rather than global; orchestrating workflows across complex systems is the real challenge.

A typical AI application has a set of training data in production, usually far smaller than the dataset on which the model was first trained. Each practice associated with supervised model training—time series data capture, validation, versioning, and quality checks—is implemented as a pipeline in which artifacts can be stored, catalogued, and indexed. The idea is for the output of one step to be the input to the next. Training data should capture not only the input features but also the most important contextual variables to reduce the chance of regressor drift. Channeling real-world data—and the conditions under which they produce a relevant outcome—into the preparation of retraining data is crucial.

9.3.1. Data Ingestion and Validation Automation

Data ingestion pipelines, which automate the collection of fresh data, often have importance in production ML systems. For training and retraining, the data must not only come from the appropriate source, but also be trustworthy. Therefore, data validation checks are essential. In the simplest case, a batch of new data can have a schema check applied against a well-known schema, such as one stored and consumable via an API. When the schema is validated against the rules, the data can continue to the next phase; otherwise, the ingestion process gets aborted. Other checks might involve applying range filters to key columns so that records meeting the filters can pass while invalid entries are marked. Another useful feature is schema evolution handling, such that a new field addition can go through. The absence of a column with a Nullable attribute during insertion must always go through irrespective of the field being present in the schema or not. Monitoring data quality–including completeness, accuracy, freshness, reliability, and relevance–through data quality metrics ensures that such an issue can be rectified before it impacts the pipeline.

Data lineage tracking helps to understand the flow of the data, including any modifications applied across the family of pipelines, thus allowing the team to know what part of the ML system has been affected if any issues crop up. In order to manage these larger flows, it is also useful to set up anomaly detection for the data. Given that the objective is to trust fresh data in a production environment, not only must the old data be validated, but also the logic that generates the density of fresh data and its completeness must go through a round of testing too.

9.3.2. Model Training Orchestration

Training ML models requires orchestrating complex workflows that seek to fulfil higher-level objectives. Orchestration coordinates the execution of a training workflow, determining when the ML model should be retrained and where, how, and with which data. It tracks dependencies, provisions the necessary infrastructure, invokes the right components in the correct order, and handles failures. The use of an orchestration tool simplifies interface management by abstracting them into function calls.

Training workflows are commonly complex systems that comprise pre-processing, training-expression generation, training, and post-processing. Although these steps may be implemented in a single programming language, the orchestration engine can exploit good design practices by breaking these sub-processes into tasks that are handled with a dedicated technology stack. Workflow orchestration tools automatically handle many of these aspects. Dependencies can be declared, facilitating parallel processing, and fault tolerance can also be controlled.

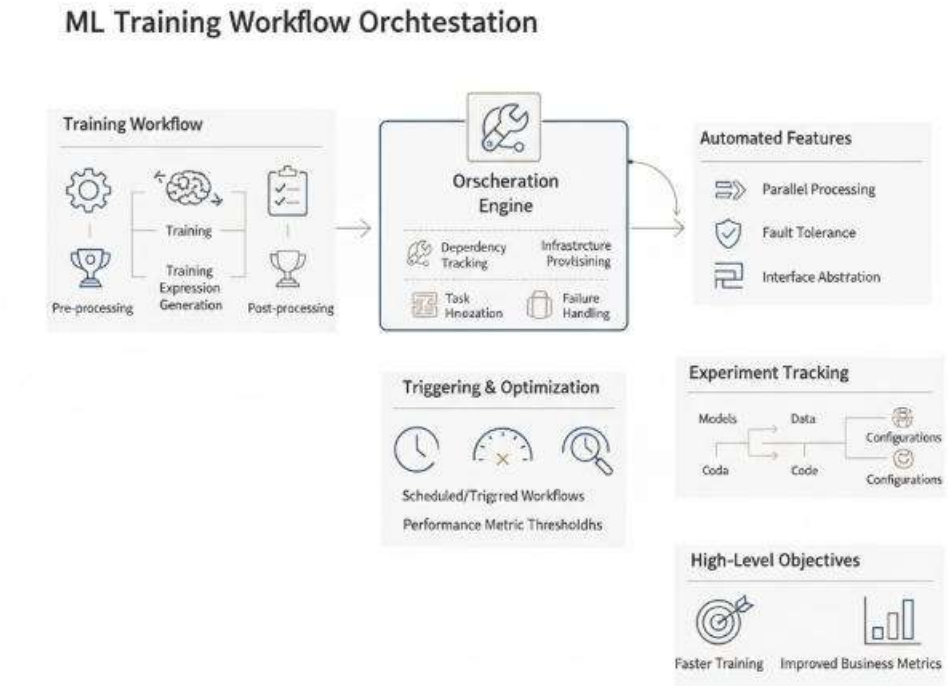

Fig 9.2: Autonomous Machine Learning Orchestration: A Framework for Dynamic Workflow Synthesis, Hardware Optimization, and Performance-Triggered Retraining Systems

To speed up the training process, the framework can utilize scheduled or triggered workflows that exploit hardware at peak load time. Depending on business requirements, training jobs may also be executed when a performance metric goes beyond certain

limits, for example, when a customer-support request system shows signs that it is becoming less precise. Other aspects of orchestration include dependency management (the correct versions of libraries for the execution environment), hyperparameter tuning (searching for the hyperparameter set that results in the best-performing model), and experiment-tracking layers (fororganizing models, data, code, and configurations).

9.4. Continuous Learning Paradigms

In most production scenarios, models are trained periodically in a batch-learning regime using data collected in a feedback loop. The batch frequency is determined by infrastructure constraints and need for fresh data, balancing various factors. Latency can be optimized by speedy retraining and short time to deployment, yet data freshness must not be ignored. Furthermore, updating the model too early may prove hazardous. With relatively little or slowly growing data, retraining a model too frequently may create excessive instability. Brisk labeling ensures timely feedback for classification tasks, while fresh data can help address distribution leaks or recover from a sudden concept shift. In other cases, the data is so voluminous that periodic batch retraining is unfeasible, necessitating a distinct update paradigm whereby the model is continuously retrained on small chunks of data. Ongoing model assessments, preferably through online A/B testing, enable efficient allocation of labeling and monitoring resources. A/B testing can also facilitate the low-risk introduction of feature changes that influence a model's decisions.

The risk of data drift can be decreased through effective probe design. Spike testing helps identify frequency variances, while dimension-reduction methods detect unexpected low-dimensional distributions. The risk of concept drift is recorded in a detection model. A feedback loop can also address evolving user needs, nudging the model toward user satisfaction. Such cases demand cautious monitoring of A/B experiments; after a feature set change, checkpoints across time axes should be examined to confirm that user satisfaction remains predictably aligned with success metrics. Users can also be considered part of the model's input space, leading to the design of models with user information as inputs for explicit personalization. Users may have repeatable preferences for similar products on similar tasks, even if the products or tasks differ substantially.

9.4.1. Online versus Batch Learning in Production

In production AI systems, a choice must be made between online learning, wherein model parameters are updated as new training data becomes available, and batch learning, wherein more substantial updates are periodically carried out. Data freshness, whether real or proxy, is often a core business requirement, since predictions are used

not to anticipate a future state, but to act in the present. Online learning may therefore seem like an obvious choice, but the alternative may actually be more appropriate if inferred or predicted data is used for updates. In such circumstances, the latency and cost of updates is a central consideration; moreover, there is a risk of data and concept drift. Online learning is appropriate for settings with high prediction frequency and low cost per update, but relatively small changes to the prediction distribution. Otherwise, concept drift should be infrequent, and updates more costly than minor retraining of batch systems.

Online learning becomes more attractive, relative to batch learning maintained in production until the next scheduled update is completed. In addition to frequency, freshness of update data for batch systems is relevant. Recent training data may provide a better representation of current conditions than older data; however, if the model is being used to predict, freshness is not naturally supported. To mitigate risk during update, data availability is also a consideration; typically, a stable period is desired, during which that traffic appear similar to past patterns. Regardless of regime, instrumentation should monitor and trigger updates on data and concept drift, and regular evaluation with fresh data is essential; such tests can be conducted against earlier training data, or even an entirely different hold-out set.

9.4.2. Feedback Loops and Monitoring for Model Drift

Robust models for near-real-time predictions can be implemented in practice by formalizing a closed feedback loop for monitoring prediction performance. Data drift, whether from new or periodic re-training, can be automatically detected by continuous monitoring of prediction performance against high-quality inference data. Continuous retraining, using alert thresholds and support from data scientists, keeps the models up to date. Models deployed as part of an A/B testing framework receive their own data comparisons to catch drift early. A well-structured collection of retrospective error analyses serves as an additional advice source. Periodically retraining the model using a larger dataset or signature approach balances the gap between update frequencies and the amount of anomalous data that accumulates. Agility is key: if the last inferred data point confirms that a separate test model has better prediction quality and drift speed, it must be substituted without a cautious wait for a regular interim.

Two failure modes can be detected fairly easily: if anomalous data (unexpected distribution or features) pass the monitoring stage and arrive at the external service, the data-monitoring test can sound an alert (rather than cause the external service to freeze). More challenging are updates when the distribution does not change (hypothesis testing detects no difference) but the model’s quality does. One heuristic approach involves checking a panel of choice predictive system performance indicators: if three systems

consistently give different results with equivalent accuracy, the best signature and all its candidates are checked for the last ten points; if one system has gained a clear edge and prediction performance is still at least as good as before, a switch can be made.

9.5. Deployment Strategies and Automation

Successfully deploying updated models into production requires well-defined experiment design, monitoring criteria, and rollback plans. Moreover, automation and a canary strategy minimize deployment risks, and safety nets protect against botched rollouts.

Model updates are first validated using either canary or A/B testing before being accepted into production. Canary deployment involves initial rollout to a small segment, with careful monitoring for errors before wider promotion. A/B testing measures the chosen metric for both the updated and existing models, promoting the one with superior performance based on established statistical power. Rollback plans specify the symptoms justifying model reversion and implement sequencing rules to avoid reverting to a previous version. Such planning is crucial, given that model pushes often involve multiple alterations—e.g., both the model and its serving environment may have changed.

A canary deployment strategy mitigates the risk of a misbehaving model, ensuring that faults will affect only a small portion of the overall traffic, and doctrinal approval for switching back to the previous version is part of the deployment plan. Canary releases go further, deploying the model to a small fraction of incoming requests but not reassigning all traffic until its performance consistently meets the established acceptance criteria. Such experimentation design involves a trade-off between precision and decision latency—greater precision requires greater traffic diversion. Thus, the greater the potential negative impact of a bug, the larger the canary.

Going beyond HTTPS, applied machine learning offers a rich seam of experimentation opportunities. By tracking ongoing performance, details of retraining events, and the best-performing model, ML models can be treated like software. Bugs detected in production are flagged as known issues, which in turn informs future model designs. Caution is therefore required in discussion of canary procedures—the controls correspond to best practice in engineering and require only appropriate IMLP service.

9.5.1. Canary and A/B Testing in Model Rollouts

Canary and A/B testing frameworks facilitate controlled introduction of new models into production to verify performance or stability before wider deployment. Invocations of

the new model are limited, with promotion criteria defined ahead of time. A rollback plan is specified in case of performance degradation; safeguards ensure that rollbacks can occur even if the new model is incrementally promoted beyond the canary stage.

ML evaluation is inherently complex. Multiple distinct metrics may be necessary to assess a model's performance, and often, the assessment is not merely the comparison of a model to its predecessor, but to different candidate models that may be supplied by other teams. A/B tests help tame this complexity by methodically designing every model introduction (including, if applicable, the re-introduction of a previous model) as a controlled experiment with clear power considerations. However, building different models or even different versions of the same model for A/B testing is not always feasible, and many of the deployment and naming challenges are avoided when one process can support both A/B tests and canary releases. A test strategy is chosen depending on the question being answered and whether the same traffic is used to evaluate the models.

9.5.2. Canary Releases, Rollbacks, and Immutable Deployments

Canary releases, immutable deployments, and blue-green techniques enhance control and safety across the full model rollout cycle. Deployment pipelines coordinate staging and promotion through testing, with rollback plans and precautions. Deployment artifacts are immutable, ensuring provenance for every promotion and rollback, and post-rollout validation detects other problems. The details and considerations for each stage—prepare, promote, roll back—are as follows.

Prepare stage: Whether a canary release or a full rollout, preparation involves defining staging criteria, testing the deployment—including data quality in Shadow mode, if deployed—establishing a rollback plan, and assessing risk. For canaries, a criteria-based promotion plan should also be established, along with tests that determine when to roll back after promotion. Testing requirements are a wider challenge; for example, sensitive user service models may require safety guard rails based on extensive experimentation and simulation. Data-backed evidence is essential for Canary A/B testing.

Promote stage: Blue-green techniques eliminate downtime but do not inherently reduce rollout risk. In practice, system managers need greater flexibility to manage release timing across geographically distributed sites, induce gradual traffic shifts to reveal problems not detected in pre-release testing, manage costs, and introduce changes only when user demand warrants. Canary releases, indeed any kind of staged rollout, require clear criteria for success or failure and plans for confirming and recovering from

detected problems. Canary setup, traffic shifting, post-promote testing, and problem discovery/testing must all be considered.

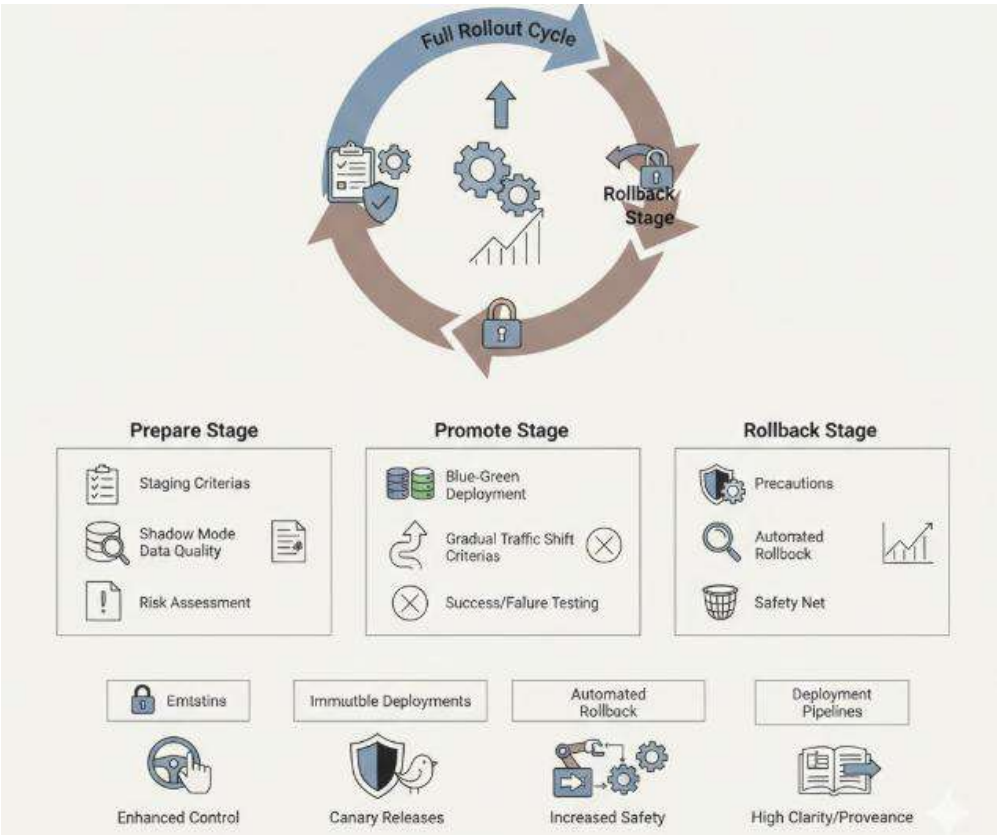

Fig 9.3: Ensuring Model Reliability Through Managed Rollouts: A Tri-Stage Framework for Immutable Deployments, Canary Orchestration, and Risk-Averse Rollback

Rollback stage: Effective rollback requires precautions throughout the prepare and promote stages, applied as appropriate to the kind of deployment at hand. With careful design, rollback can be a mundane, low-stress activity. Relying on speed and automation increases its risk. When any natural rollback capability or safety net is missing, management loses options and flexibility, and the requirement that every new deployment be better—in some sense—becomes especially acute.

9.6. Observability and Metrics for Production AI

Effective observability and insightful metrics play crucial roles in the successful MLOps of production AI. Dependable metrics enable the detection of problems before they impact end users. The spectrum of observability extends from AI model performance,

representing the application's business function, to the underlying infrastructure and system resources, informing operational viability. Adequate support for observability includes a telemetry-collection architecture tailored to the specific logic of machine learning models; structured logging that balances information richness and privacy; end-to-end tracing that connects user requests to data, models, software, and configuration; actionable dashboards for stakeholders; and incident-response playbooks.

A broad array of metrics supports the observability of production AI. Metrics such as model accuracy, calibration, fairness, and robustness indicate model quality; latency and throughput reflect system responsiveness; and cost considerations relate to resource utilization. Anomalous drift in any of these metrics necessitates intervention, whether via model retraining, rollback to a prior model version, or some other action. The specific actions triggered by individual metrics are typically governed by a combination of historical precedent and presets agreed upon by responsible parties.

9.6.1. Metrics for Model Performance and Fairness

Machine learning models deployed in mission-critical applications must undergo routine evaluations to identify problems before they affect stakeholders. Metrics for determining whether a model should be retrained often include accuracy, calibration, fairness, robustness to adversarial examples, prediction latency, throughput, and resource utilization. Performance must be regularly monitored because even accurate models may give rise to negative consequences when they become better than baselines on datasets that are the same distribution as the training dataset. AI governance requirements further motivate performance monitoring, as an organization may seek to obey a regulation or business model that requires fairness despite the experimental results indicating that certain fairness conditions cannot be satisfied by the ML models.

Both model performance and AI governance metrics must be monitored regularly, and acceptable thresholds for each metric are usually specified ahead of time. Being able to show drift beyond those thresholds may trigger an automatic retraining of the model (potentially of an ensemble of models) or the initiation of a formal retraining evaluation process. In addition to retraining, deactivation of the model or a fallback to an earlier version of the model is often a viable action when a model's performance is degrading too much. Because satisfactory model performance on production data is necessary for the decision of whether or not to retrain a model, defining a retraining strategy in terms of "data-backed evidence for model updates" includes defining when to ask such questions and not just when to act on the answer.

9.6.2. System Telemetry, Logging, and Traceability

Telemetry encompasses system-level logging, capturing essential data to inform operations and incident investigation. Captured telemetry informs incident response, enabling efficient and effective diagnosis and remediation of system failures. Structured logging metadata allows aggregation and search across large volumes of logs. Telemetry is analyzed to generate observability dashboards, enabling quick situational awareness when incidents occur. Specific dashboards for key stakeholders guide effective incident response by surfacing relevant information. The telemetry required and the appropriate means of capturing this telemetry must be understood during the design of the system, as introduction of additional telemetry can impose significant overhead.

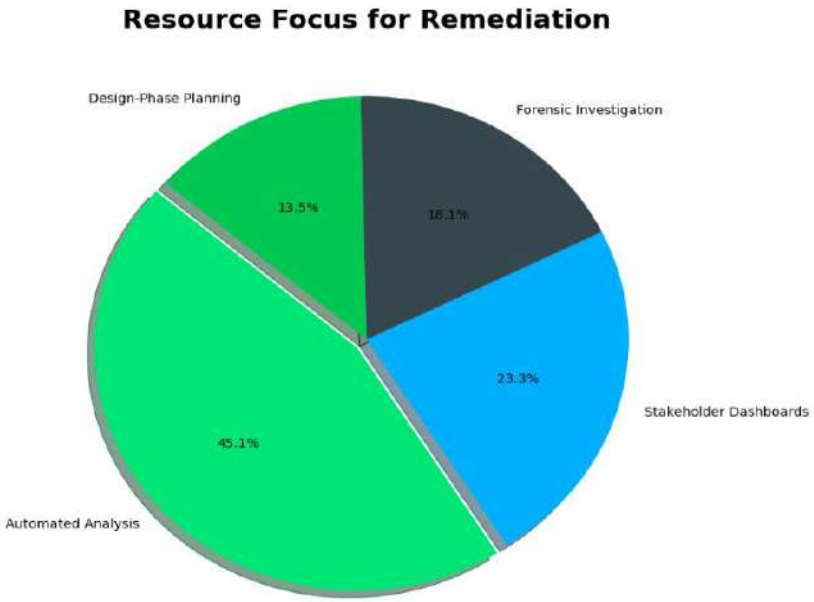

Fig 9.4: Resource Focus for Remediation

Traceability connects data with enabling code and models. Structured logging, layered architecture with distinct role separation and absence of monkey patching, and observability tooling such as OpenTracing/Jaeger and OpenTelemetry support automated incident analysis and auditing, allowing fast forensic investigation of model performance issues or operational incidents. Collecting data about data lineage—how artifacts were created including input datasets, executed code and model versions—provides context when reviewing model performance and lifespan. When privacy constraints permit, logging should record identifying attributes to facilitate post-hoc investigation and improve future modelling efforts.

9.7. Conclusion

Continuous Learning in Production AI Environments—recent tech industry MLOps initiatives recommend emphasizing MLOps as money-driven enterprise risk management. MLOps-governed algorithms direct critical business operations, with associated ML adoption costs, user accessibility, and disenfranchisement risks. Money-driven MLOps expands classic technological focus on automation, including model rollouts, observability, feedback-triggered retraining, and continuous learning, through the lens of risk exposure: managing the risk arising from production AI continually operating with outdated, poorly aligned models.

These real-time, governance-sensitive observations of production drift, bias, and emerging alternative distribution hypotheses, requiring cost-sensitive CI/CD experimentation, are crucial for MLOps success. MLOps considers learning to live in ML/AI products as important as writing ML/AI products in the first place—monitoring deployed models for undesired performance changes and comparing alternatives in CI/CD initiatives. MLOps versioning traceability ensures that model retraining decisions based on a years-long absence of drift are evidence-based, risk decisions guiding product manifests expressing user need, enabling timely development of rare-value capability.

9.7.1. Key Takeaways and Future Directions in MLOps

An earlier version of this article articulated state-of-the-art MLOps concepts, roles, and lifecycle stages; differentiated DevOps from MLOps; identified governance requirements and compliance implications; mapped stakeholders to responsibilities; summarized typical tooling ecosystems and data flows; and enumerated success metrics for production AI. Core principles of MLOps implementation were delineated, encompassing reproducibility, automation, CI/CD for ML, data lineage, monitoring, and governance. Architecture patterns for production ML systems were described, while risk management, scalability, and security considerations received attention. Devising a suitable rollout strategy for ML models was highlighted, detailing canary and A/B testing frameworks, promotion criteria, rollback plans with safety nets, and the sequencing of deployments.

Subsequent exposition specified automation in AI pipelines, focusing on the ingestion of training data: the use of ingestions pipelines; checks for data quality and schema conformance; mechanisms for handling schema evolution; metrics for gauging data quality; tracking data provenance; and detecting anomalous data changes. Moreover, it articulated the orchestration of model-training workflows, covering trigger definition, storage and versioning of artifacts, dependency resolution, resource provisioning, experiment tracking, hyperparameter tuning, and the provision of reproducible

environments; fault-tolerant strategies; techniques for parallelizing training; and the automation of batch jobs.

Discussion then turned to continuous learning paradigms in production AI. The suitability of online versus batch learning was considered, along with the latency implications of updating models and the risks of data and concept drift. Key aspects of embedding a feedback loop into a production system were outlined, encompassing monitoring for data and concept drift, the specification of alerting thresholds, the cadence for evaluating model performance in a production setting, and the criteria for confirming that a model needed updating.

References

Amershi, S., Begel, A., Bird, C., et al. (2021). Software engineering for machine learning: A case study. IEEE Transactions on Software Engineering, 47(12), 2913–2932.

Yandamuri, U. S. An Intelligent Analytics Framework Combining Big Data and Machine Learning for Business Forecasting. Journal of Finance (IJFIN), 36(6), 682-706.

Sculley, D., Holt, G., Golovin, D., et al. (2015). Hidden technical debt in machine learning systems. Advances in Neural Information Processing Systems, 28, 2503–2511.

Guntupalli, R. (2025). Predictive cloud resource management: Developing ml models for accurately predicting workload demands (CPU, memory, network, storage) to enable proactive auto-scaling. AI-driven instance type selection and rightsizing. predicting spot instance interruptions. forecasting cloud costs with higher accuracy. Available at SSRN 5267834.

Zaharia, M., Chen, A., Davidson, A., et al. (2018). Accelerating the machine learning lifecycle with MLflow. IEEE Data Engineering Bulletin, 41(4), 39–45.

Vadisetty, R., Polamarasetti, A., Goyal, M. K., Rongali, S. K., Prajapati, S. K., & Butani, J. B. (2025, March). Smart Sorting Systems: Implementing IoT, Generative AI, and AI for Real-Time Monitoring of Plastic Waste Sorting. In Doctoral Symposium on Computational Intelligence (pp. 101-125). Singapore: Springer Nature Singapore.

Kreps, J., Narkhede, N., & Rao, J. (2019). Kafka: A distributed messaging system for log processing. IEEE Data Engineering Bulletin, 42(2), 28–38.

Amistapuram, K. (2021). Digital Transformation in Insurance: Migrating Enterprise Policy Systems to .NET Core. Universal Journal of Computer Sciences and Communications, 1(1), 1–17.

Bai, T., Zheng, Z., Ren, K., & Shi, S. (2024). Cloud-native machine learning systems: Architecture and optimization. IEEE Software, 41(1), 50–58.

Gama, J., Žliobaitė, I., Bifet, A., Pechenizkiy, M., & Bouchachia, A. (2020). A survey

Kelleher, J. D., & Tierney, B. (2018). Data science. MIT Press.

Villamizar, M., Garcés, O., Ochoa, L., et al. (2016). Infrastructure as a service: A comparative performance analysis of public cloud providers. IEEE Cloud Computing, 3(2), 38–47.

Nagabhyru, K. C., Garapati, R. S., & Aitha, A. R. (2025). UNIFIED INTELLIGENCE FABRIC: AI-DRIVEN DATA ENGINEERING AND DEEP LEARNING FOR CROSS-DOMAIN AUTOMATION AND REAL-TIME GOVERNANCE. Lex Localis, 23(S6), 3512-3532.

Inala, R. Advancing Group Insurance Solutions Through Ai-Enhanced Technology Architectures And Big Data Insights.

Segireddy, A. R. (2021). Containerization and Microservices in Payment Systems: A Study of Kubernetes and Docker in Financial Applications. Universal Journal of Business and Management, 1(1), 1–17.DOI: 10.31586/ujbm.2021.1352

Chen, Y., & Zhang, L. (2022). Data engineering practices for real-time analytics: Challenges and approaches. IEEE Transactions on Services Computing, 15(4), 2288–2302.

Varri, D. B. S. (2020). Automated Vulnerability Detection and Remediation Framework for Enterprise Databases. Available at SSRN 5774865.

Hüttermann, M. (2021). DevOps for developers. Apress.

Nagabhyru, K. C. (2024). Data Engineering in the Age of Large Language Models: Transforming Data Access, Curation, and Enterprise Interpretation. Computer Fraud and Security.

Newman, S. (2021). Building microservices (2nd ed.). O'Reilly Media.

Kim, G., Debois, P., Willis, J., & Humble, J. (2016). The DevOps handbook. IT Revolution Press.

Dean, J., & Ghemawat, S. (2020). MapReduce: Simplified data processing on large clusters. Communications of the ACM, 63(1), 72–81.

Armbrust, M., Xin, R. S., Lian, C., et al. (2020). Delta Lake: High-performance ACID table storage over cloud object stores. Proceedings of the VLDB Endowment, 13(12), 3411–3424.

Wang, S., Cao, J., Yu, P. S., et al. (2022). Deep learning for anomaly detection: A survey. ACM Computing Surveys, 54(2), 1–38.

Radanliev, P., De Roure, D., Walton, R., & Van Kleek, M. (2021). AI systems safety and cybersecurity: A systematic mapping study. Computers & Security, 102, 102192.

Gounaris, A., & Tzortzis, G. (2021). A survey of platforms for scalable data analytics and AI in the cloud. Journal of Cloud Computing: Advances, Systems and Applications, 10(1), 45.

Kolla, S. H. (2024). RETRIEVAL-AUGMENTED GENERATION WITH SMALL LLMS FOR KNOWLEDGE-DRIVEN DECISION AUTOMATION IN ENTERPRISE SERVICE PLATFORMS. Turkish Journal of Computer and Mathematics Education (TURCOMAT), 15(3), 476–486. https://doi.org/10.61841/turcomat.v15i3.15497

Varian, H. R. (2019). Artificial intelligence, economics, and industrial organization. In A. Agrawal, J. Gans, & A. Goldfarb (Eds.), The economics of artificial intelligence: An agenda (pp. 399–419). University of Chicago Press.