WWW.IJRPR.COM

International Journal of Research Publication and Reviews

Journal Homepage: www.ijrpr.com ISSN 2582-7421

Agentic Self-Healing MLOps Platform

Hemalatha S V [a], Nandhini S [a], Indhu Priyan M [a*], Krishnan S A [a], Kisore P [a], Keerthana M R [a], Kathrin J [a]

[a] *Department of Computer Science and Engineering, Sri Shakthi Institute of Engineering and Technology, Coimbatore, Tamil Nadu, India*

A B S T R A C T

Machine learning models rarely remain accurate indefinitely once deployed: shifts in the input data distribution, changing user behaviour and differences between training and production environments gradually erode prediction quality. In conventional MLOps practice the response to such degradation is still largely manual, with engineers inspecting dashboards, diagnosing the cause, retraining, re-evaluating and redeploying the model through loosely connected tools. This paper presents the Agentic Self-Healing MLOps Platform, an integrated system that manages the complete model lifecycle and coordinates recovery from performance degradation with minimal human intervention. The platform combines a Model Registry, a Training Worker, an Evaluation Module, a Deployment Manager, a Monitoring Module, Drift Detection and Root Cause Analysis behind a FastAPI backend with a React frontend, using SQLite for persistent lifecycle records and Evidently AI for distribution analysis. An AI Agent Decision Layer evaluates structured evidence from monitoring, drift and root-cause analysis and selects one of three actions: NO_ACTION, MONITOR or RETRAIN. When retraining is selected, a LangGraph state-based workflow orchestrates retraining, evaluation gating, deployment, verification and, where a previous deployment and its artifacts are available, rollback. An end-to-end demonstration executed the full self-healing pipeline, including retraining, evaluation, deployment and workflow verification, and served predictions through the deployed-model inference API. The platform treats drift as evidence rather than proof of accuracy loss, separates operational metrics from prediction-quality metrics and records every decision for traceability. Limitations, including dependence on accessible training assets and the local-development scope of the current implementation, are discussed together with directions for future work.

Keywords: MLOps; Agentic AI; Self-Healing Systems; Machine Learning Lifecycle; Data Drift Detection; Root Cause Analysis; LangGraph; Automated Model Retraining; Model Monitoring

1. Introduction

Machine learning (ML) has become a core technology in healthcare, finance, education, business analytics and intelligent automation, where models are used to make predictions, classify data and support decisions. Deploying a model, however, is not the end of its lifecycle. Once in production the model must be monitored continuously so that its accuracy, latency and reliability remain acceptable as the real-world environment changes. The engineering effort required to keep such systems healthy was characterised early as a form of hidden technical debt (Sculley et al., 2015), and later work has shown that a large share of the cost of ML systems arises after deployment, in monitoring, maintenance and retraining (Paleyes, Urma, & Lawrence, 2022; Amershi et al., 2019).

Machine Learning Operations (MLOps) has emerged as the discipline that applies software engineering, automation and monitoring practices to the ML lifecycle (Kreuzberger, Kühl, & Hirschl, 2023; Testi et al., 2022). Existing MLOps platforms provide model registries, experiment tracking, training pipelines and deployment management, and monitoring tools report infrastructure and performance metrics after deployment. Yet when a model's accuracy decreases or the input distribution changes, developers typically still have to identify the problem, analyse its root cause, retrain the model, evaluate the new version and redeploy it by hand. These manual processes consume considerable time and may lead to service interruptions, delayed responses and operational difficulties.

1.1 Model degradation, data drift and concept drift

Model performance may deteriorate because the statistical distribution of incoming data changes with respect to the reference data used for training (data drift), or because the relationship between inputs and the target changes over time (concept drift) (Gama et al., 2014; Lu et al., 2019). The two phenomena are related but distinct, and detected data drift does not by itself prove that accuracy has fallen. A sound maintenance system therefore has to combine distributional evidence with prediction-quality evidence and with analysis of which features and conditions are associated with the change before it commits to a costly intervention such as retraining.

1.2 Need for intelligent automated recovery

Conventional pipelines automate individual stages but seldom close the loop between observation and recovery. Training, evaluation, deployment, monitoring and rollback are often implemented as separate components without a unified decision-making workflow, and automated retraining, where it exists, is driven by fixed triggers rather than by an assessment of the available evidence. What is needed is an orchestrated system that collects monitoring evidence, decides whether action is warranted, executes the recovery stages under quality gates, verifies the outcome and records the whole episode for later inspection.

1.3 Motivation, objectives and contributions

The Agentic Self-Healing MLOps Platform was designed to address these gaps. Its objectives, as documented in the project report, are to develop a centralised platform for model registration and version management; to automate training using registered scripts, datasets and dependency configurations; to evaluate trained models with accuracy, precision, recall and F1-score; to deploy models that satisfy configured evaluation requirements and expose prediction APIs; to monitor accuracy, latency, throughput, error rate and other operational metrics; to implement drift detection and root cause analysis; to develop an AI Agent decision mechanism that selects an appropriate action from the collected evidence; to implement a LangGraph-based self-healing workflow that coordinates retraining, evaluation, deployment, verification and rollback; and to maintain persistent records of metadata, job execution, agent decisions and workflow history.

The main contributions of this work are: (i) an integrated open-source platform architecture in which lifecycle management, monitoring and recovery share a single backend and persistent store; (ii) an evidence-based agent decision policy with three explicit outcomes (NO_ACTION, MONITOR, RETRAIN) that avoids retraining on isolated or unconfirmed anomalies; (iii) a LangGraph state-based self-healing workflow with evaluation gating, deployment verification and conditional rollback; and (iv) a working end-to-end demonstration of the retrain-evaluate-deploy-verify sequence. The remainder of the paper reviews related work (Section 2), describes the proposed architecture (Section 3) and its implementation (Section 4), reports the results obtained (Section 5), presents the platform's user interface (Section 6) and concludes with limitations and future work (Section 7).

2. Related Work and Literature Review

2.1 MLOps lifecycle management

Sculley et al. (2015) argued that only a small fraction of a real-world ML system is learning code, the rest being configuration, data collection, serving infrastructure and monitoring, and that this surrounding machinery accumulates technical debt if it is not engineered deliberately. Breck et al. (2017) proposed the ML Test Score, a rubric of tests for data, models, infrastructure and monitoring that operationalises production readiness. Amershi et al. (2019) studied ML engineering practice at a large software company and identified data management, model customisation and the lack of modularity in ML components as the dominant challenges. Kreuzberger et al. (2023) consolidated these threads into a definition of MLOps with nine principles, including CI/CD automation, workflow orchestration, reproducibility, versioning of data, model and code, continuous monitoring and feedback loops, and described the roles and architecture that an MLOps system should provide. Testi et al. (2022) offered a taxonomy and methodology for MLOps that likewise places monitoring and automated retraining at the end of the lifecycle. Platforms such as MLflow (Zaharia et al., 2018) address experiment tracking, packaging and a model registry; the present work adopts the registry concept but extends it with an integrated recovery workflow.

2.2 Machine learning model monitoring

Paleyes et al. (2022) surveyed case studies of ML deployment and reported that monitoring and maintenance are among the least mature stages in practice, with practitioners lacking agreed signals for when a model should be retrained. Monitoring literature distinguishes operational metrics (latency, throughput, error rate, resource utilisation) from prediction-quality metrics (accuracy, precision, recall, F1) that require ground-truth labels, and from data-level indicators such as feature distribution shifts. The platform described here treats these three categories as separate sources of evidence, because infrastructure health alone cannot establish prediction accuracy.

2.3 Data drift detection

Gama et al. (2014) surveyed concept drift adaptation and classified detection strategies into sequential analysis, statistical process control, window-based comparison and contextual approaches. Lu et al. (2019) extended the review to learning under concept drift, covering drift detection, understanding and adaptation. Rabanser, Günnemann and Lipton (2019) empirically compared dataset-shift detection methods and found that comparing reference and current samples with univariate statistical tests on (possibly reduced) feature representations is a strong and practical baseline. Evidently AI (n.d.) packages such reference-versus-current comparisons as reusable reports and tests, and is used by the proposed platform for drift analysis. Consistent with Gama et al. (2014), drift is treated in this work as evidence of a potential data change rather than automatic proof of accuracy degradation.

2.4 Root cause analysis

Monitoring can flag that an anomaly has occurred, but additional analysis is required to identify the affected features and supporting evidence. Feature-level analysis of numerical distribution changes, categorical value differences and missing-value percentages is a direct way to localise the source of a data problem. Feature-attribution methods such as SHAP (Lundberg & Lee, 2017) complement this by ranking the contribution of individual features to model output. In the proposed platform, root cause findings are stored with their severity and evidence and are interpreted as indications of possible contributing factors rather than as proof of causation.

2.5 Automated machine learning retraining

Automated retraining is a recurring recommendation in the MLOps literature (Kreuzberger et al., 2023; Testi et al., 2022), but it depends on access to the original training source, datasets, dependencies and configuration, and on evaluation gates that prevent an inferior candidate from replacing the current model. Breck et al. (2017) explicitly include tests that a new model is validated against a baseline before serving. The present work incorporates both requirements: retraining is only possible for models whose training assets are registered, and a failed evaluation blocks the normal deployment path.

2.6 Agentic AI and workflow orchestration

Recent work on language-model-based agents frames an agent as a component that observes state, reasons about evidence and selects actions (Yao et al., 2023; Wang et al., 2024). Frameworks such as LangGraph (LangChain, n.d.) model such agents as state graphs whose nodes perform work and whose conditional edges route execution according to the accumulated state. The platform described in this paper applies this agentic pattern to MLOps maintenance: the decision layer applies a defined policy to structured evidence, and LangGraph coordinates the subsequent recovery stages as a graph with conditional transitions.

2.7 Summary of the gap

Existing approaches provide the essential building blocks of registration, training, evaluation, deployment and monitoring, but manual investigation, fragmented workflows and unverified redeployment remain important engineering challenges. The Agentic Self-Healing MLOps Platform integrates these capabilities through a centralised model management system, an evidence-based agent decision layer and a LangGraph workflow that connects evidence collection, decision-making, retraining, evaluation, deployment verification and rollback.

3. Proposed System and Architecture

3.1 Overall architecture

The platform is a Python-based (Python Software Foundation, n.d.) MLOps system with a web interface. The frontend is developed in React.js and communicates with a FastAPI backend (FastAPI, n.d.) over REST using Axios; the backend owns all lifecycle operations and persists its state in SQLite through SQLAlchemy (SQLAlchemy, n.d.). Model artifacts, training scripts, datasets and dependency files are stored as files in the platform's storage directories and referenced from the database. Fig. 1 shows the end-to-end flow of the system, from model registration and training, through evaluation gating and deployment, to monitoring, drift detection, root cause analysis, the AI Agent decision and the LangGraph self-healing loop.

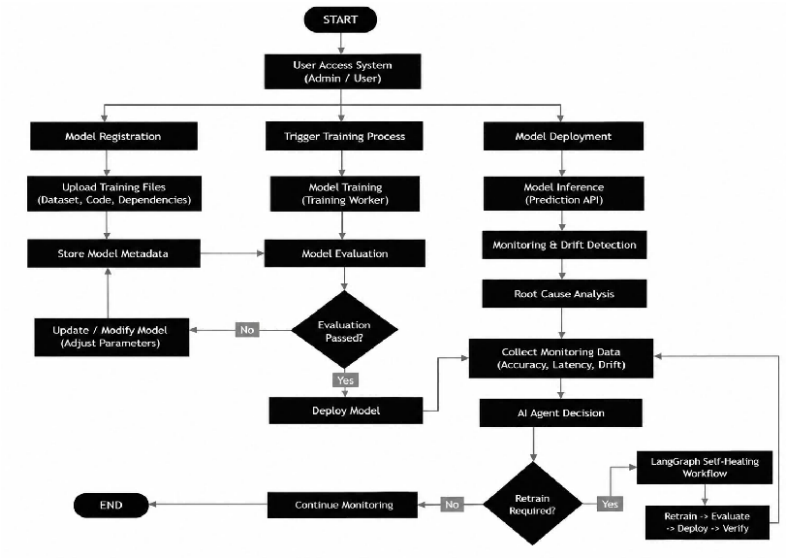


Fig. 1 - System architecture and end-to-end workflow of the Agentic Self-Healing MLOps Platform.

3.2 Frontend and backend architecture

The frontend provides dedicated interfaces for the dashboard, model registry, training and evaluation, deployment and inference, monitoring and drift detection, root cause analysis, the AI agent and the self-healing workflow. Vite is used as the development and build tool. The frontend periodically requests updated metrics from the backend and displays the current values, so that all figures shown are retrieved from backend records rather than computed in the browser.

The backend exposes REST endpoints for model registration, training, evaluation, deployment, prediction, monitoring, telemetry, root cause analysis and self-healing operations, and documents them through Swagger UI. Uvicorn serves the FastAPI application. Training jobs run in the background so that the API can return job information immediately rather than blocking until training completes. The backend therefore acts as the processing and orchestration layer that connects lifecycle management, monitoring, intelligent decisions and automated recovery.

3.3 Lifecycle components

Model Registry. The registry maintains model name, version, framework, training command, training source, dataset, dependency file and relevant integration endpoints. Metadata is stored in SQLite with references to files in the storage directories, so that the Training Worker can retrieve the required assets when a job is initiated. Compatible externally hosted models can also be registered with their prediction endpoint (for example through the Hugging Face Inference API); external inference support is deliberately separate from local retraining.

Training Worker. The worker prepares an isolated job workspace, makes the registered training assets available and runs the training command as a subprocess. Logs, job status, execution details and generated artifact paths are recorded. A successful process exit is not sufficient to mark a job successful; the expected model artifact must also have been generated.

Evaluation Module. After training, the module measures the model with configured metrics (accuracy, precision, recall, F1-score) and compares the result with the configured acceptance threshold. It acts as a quality gate: a failed evaluation blocks the normal deployment path so that an unsuccessful candidate is not accepted as an approved model.

Deployment Manager. The manager deploys evaluated artifacts, maintains deployment status and supports restoration of eligible active deployments after a backend restart. The serving component loads local artifacts and executes inference in the model's expected input format, or forwards requests to a registered external prediction API when appropriate.

Monitoring Module. The module tracks operational and model-related metrics including latency, throughput, error rate, accuracy, CPU utilisation, memory usage and drift information where available. Prediction telemetry records predictions and, when ground-truth labels are available, the corresponding actual values, from which accuracy over a rolling observation window is calculated and compared with a configured threshold.

Drift Detection. Evidently AI is used to analyse differences between the reference dataset and current data, identify drifted features and produce results that the decision workflow can consume.

Root Cause Analysis. The module inspects numerical distribution changes, categorical value differences and changes in missing-value percentages, and records findings, severity, affected features and supporting evidence for retrieval through the API and display in the frontend.

AI Agent Decision Layer. The layer evaluates the structured evidence from monitoring, drift detection and root cause analysis against the implemented decision policy and selects NO_ACTION, MONITOR or RETRAIN. The decision and its supporting information are stored for later review.

LangGraph Workflow Orchestration. LangGraph coordinates the self-healing workflow as a state-based graph whose nodes collect evidence, execute the agent decision, initiate training, perform evaluation, manage deployment, verify the deployment state and invoke rollback handling when verification fails.

3.4 Database and artifact management

SQLite stores model metadata, training jobs, evaluation results, deployment information, telemetry records, agent decisions and self-healing workflow history; SQLAlchemy provides the object-relational mapping layer. Artifacts, uploaded training scripts, datasets and execution logs are kept on the file system and referenced by path. Database migrations maintain compatibility when fields are added, and persistent job records support troubleshooting when training or evaluation fails. Table 1 summarises the technology stack and the responsibility of each component.

Table 1 - Technology stack of the platform and the responsibility of each component.

Layer	Technology	Responsibility in the platform
Frontend	React.js, JavaScript, HTML, CSS	User interface for registry, training, deployment, monitoring, RCA, agent and workflow views
Frontend	Vite	Development server and production build tool

Layer	Technology	Responsibility in the platform
Frontend	Axios	HTTP communication with the backend REST APIs
Backend	Python, FastAPI, Uvicorn	REST endpoints for all lifecycle operations; Swagger UI documentation; application server
Backend	SQLAlchemy, SQLite	Object-relational mapping and persistent storage of lifecycle records
Backend	Scikit-learn, Pandas, NumPy	Model training, prediction and evaluation; dataset processing and numerical operations
Backend	Evidently AI	Reference-versus-current data analysis and drift detection
Backend	LangGraph	State-based orchestration of the self-healing workflow and its conditional transitions
Integration	Hugging Face Inference API (optional)	Prediction through compatible externally hosted models

4. System Implementation and Methodology

4.1 Model registry and lifecycle management

Registration is the entry point of the lifecycle. Users provide model metadata and, for retrainable models, the training source, dataset, dependency file and training configuration. The platform stores the metadata and maintains references to the associated files. Because retraining requires compatible training code and data, the registry distinguishes locally retrainable models from externally hosted models that can only be monitored and used for inference.

4.2 Model training and evaluation

A training job creates an isolated workspace, executes the registered script using the required assets and records logs, status and the generated artifact path. Scikit-learn (Pedregosa et al., 2011) is used for the demonstration models, with Pandas and NumPy for data handling. Evaluation computes the configured metrics and compares them with the acceptance threshold; the result determines whether the candidate can proceed through the normal deployment workflow. Training execution checks also distinguish successful completion from missing artifacts.

4.3 Model deployment and inference

Models that pass evaluation are deployed by the Deployment Manager and exposed through a prediction API. For local models the deployed artifact is loaded and inference is executed in the model's expected input format; for external models the request is forwarded to the registered endpoint. Deployment records are persisted so that eligible active deployments can be restored after a restart, preserving continuity and allowing users to test predictions after the service becomes available again.

4.4 Monitoring and drift detection

Operational metrics are collected through backend services and exposed through APIs that the frontend polls. Prediction telemetry is recorded when compatible integrations submit predictions and actual labels, and accuracy is computed over a rolling window when labels are available. Operational metrics and model-quality metrics are kept as separate categories. Drift detection compares reference and current datasets with Evidently AI and records drifted features and a drift result; the result is treated as evidence, not as proof that accuracy has decreased.

4.5 Root cause analysis

When investigation is required, the Root Cause Analysis module analyses the available data for numerical distribution changes, categorical differences and missing-value changes, and records findings, severity, affected features and supporting evidence. Results are retrievable through the API and are shown in the frontend so that previous investigations can be reviewed.

4.6 AI agent decision making

The AI Agent Decision Layer combines monitoring evidence, accuracy degradation, drift information and RCA severity and applies the implemented decision policy summarised in Table 2. NO_ACTION is selected when the available evidence does not indicate that intervention is required, for example when performance metrics remain healthy. MONITOR is selected when an anomaly is present but the evidence does not sufficiently support immediate

retraining. RETRAIN is selected when the configured conditions are satisfied; in the implemented policy this requires accuracy degradation together with drift and high-severity root cause evidence. Each decision and its supporting information are stored, which provides traceability and prevents retraining from being initiated solely because of an isolated or unconfirmed anomaly.

Table 2 - Decision policy of the AI Agent Decision Layer.

Decision	Condition on the collected evidence	Resulting workflow behaviour
NO_ACTION	Metrics within configured thresholds; no supporting evidence of degradation	Workflow terminates; normal operation continues
MONITOR	An anomaly is present (e.g. drift or a metric change) but accuracy degradation, drift and RCA evidence do not jointly satisfy the retraining conditions	Workflow terminates without retraining; model remains under observation
RETRAIN	Accuracy degradation detected together with drift and high-severity root cause evidence	LangGraph workflow proceeds to retraining, evaluation, deployment and verification

4.7 Self-healing workflow

The self-healing workflow follows the sequence Monitoring Evidence → Drift Detection → Root Cause Analysis → Agent Decision → Retraining → Evaluation → Deployment → Verification → Rollback (when required). LangGraph models this sequence as a state graph. The workflow begins by collecting monitoring evidence and invoking the agent decision node. If the decision is NO_ACTION or MONITOR the graph terminates without retraining. If the decision is RETRAIN, the workflow prepares the required training assets and invokes the Training Worker; the Evaluation Module then checks the new candidate, and a failed evaluation blocks the normal deployment path. A successful evaluation allows the Deployment Manager to deploy the model, after which a verification node checks the resulting deployment state. If verification succeeds the workflow completes; if it fails, rollback handling restores a previous deployment when the necessary deployment information and artifacts are available. The workflow records execution status and history at each stage, and a retry limit prevents repeated retraining attempts within the same execution. Table 3 summarises the stages.

Table 3 - Stages of the LangGraph self-healing workflow and their outcomes.

Stage	Action performed	Outcome / transition
Evidence collection	Gather accuracy, drift and RCA evidence from persistent records	State passed to the agent decision node
Agent decision	Apply the decision policy (Table 2)	NO_ACTION or MONITOR: terminate; RETRAIN: continue
Retraining	Prepare assets and invoke the Training Worker	Job status, logs and artifact path recorded
Evaluation	Compute configured metrics against the acceptance threshold	Fail: deployment path blocked; Pass: continue
Deployment	Deployment Manager deploys the evaluated artifact	Deployment record created
Verification	Check the resulting deployment state	Success: workflow completed; Failure: rollback handling
Rollback	Restore a previous deployment when its record and artifacts are available	Previous deployment reinstated; status recorded

5. Results and Discussion

This section reports what the implementation demonstrated, as documented in the project report, and separates demonstrated behaviour from expected benefits. No benchmark comparisons, latency measurements or accuracy-improvement figures were produced as part of the project, and none are claimed here; the project report notes that testing with representative datasets and documented evaluation results is necessary to establish the actual performance of a deployed model.

5.1 Model registration and version management

The platform was successfully designed and implemented with a centralised model registry that manages model metadata, versions, training sources, datasets and dependencies. Registered models can be viewed with their framework, latest version and lifecycle status, and externally hosted models can be registered with their prediction endpoint for monitoring and inference.

5.2 Training and evaluation outcomes

The Training Worker executed registered training scripts in isolated job workspaces and recorded logs, status and artifact paths. The Evaluation Module computed accuracy, precision, recall and F1-score and compared them with the configured threshold; models that satisfied the criteria proceeded to deployment, while failed evaluations prevented the normal deployment process.

5.3 Deployment and prediction functionality

Evaluated models were deployed and served predictions through the deployed-model inference API, which allowed the system to be tested with actual inference requests. Deployment records were persisted and eligible deployments were restored after a backend restart.

5.4 Monitoring and drift detection

Operational monitoring tracked prediction latency, throughput, error rate, model accuracy (where labels were available) and data drift. The drift detection module identified changes between incoming and reference data, and the Root Cause Analysis module examined changes in feature distributions and data quality to identify possible contributing factors.

5.5 AI Agent decisions

The AI Agent Decision Layer produced NO_ACTION, MONITOR and RETRAIN decisions according to the implemented policy. When metrics remained within configured thresholds and no drift was detected, the agent selected NO_ACTION and the workflow completed without retraining; when the configured conditions were satisfied, RETRAIN was selected. Decisions and their evidence were stored and could be reviewed in the interface.

5.6 Self-healing workflow execution

The implemented end-to-end demonstration successfully executed the self-healing pipeline, including model retraining, evaluation, deployment and final workflow verification, with the LangGraph workflow coordinating the stages and recording status and history. This demonstrates the feasibility of integrating automated MLOps operations with intelligent decision-making to reduce manual intervention.

5.7 Discussion and limitations

The results show that lifecycle management, monitoring and AI-driven decision-making can be integrated into a unified workflow, and that evaluation gates and verification steps can prevent unsuccessful updates from being accepted without checks. Several limitations qualify these results. The effectiveness of self-healing depends on the availability and quality of training datasets, accessible training scripts, suitable evaluation thresholds and reliable monitoring data. Accuracy-based monitoring requires ground-truth labels, detected drift does not necessarily indicate reduced accuracy, and root cause findings identify possible contributing factors rather than proving causation. Externally hosted models can be monitored and used for inference but cannot be retrained automatically unless their training assets and execution access are provided. Finally, the current implementation is intended for local development and demonstration; production-scale deployment would require capacity planning, security controls such as authentication, authorisation and secrets management, and further infrastructure configuration. Expected benefits such as reduced maintenance effort and faster recovery therefore remain to be quantified in future evaluations.

6. User Interface and Platform Modules

The React frontend exposes the platform's modules through dedicated screens. Figs. 2 to 7 show the implemented interface for a demonstration workspace in which a fraud-detection classifier and several other models are registered. The values visible in the screenshots are the records displayed for that demonstration workspace and are included to illustrate the interface; they are not presented as experimental results.

6.1 Main dashboard

Fig. 2 shows the centralised dashboard. Summary tiles report the total number of registered models, active deployments, training jobs and overall system health. A performance overview plots accuracy, precision, recall and F1-score over a selectable window, and a drift status panel summarises the monitored features classified as normal, moderate or high drift, with a highlighted alert for any feature in high drift. Tables list recent training jobs with their status and duration, active deployments with inference volume and latency, recent alerts and events, and a timeline of recent AI agent activity (analysis, decision, execution and completion events).

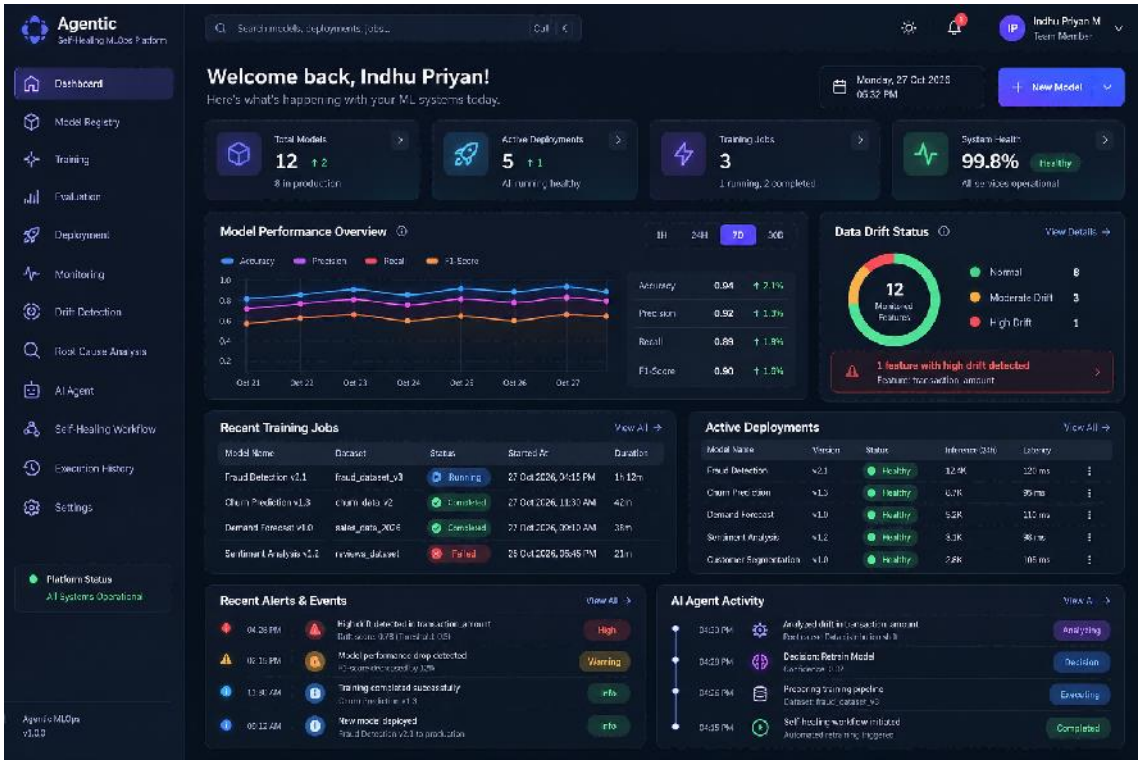


Fig. 2 - Main dashboard of the platform showing model statistics, performance overview, drift status, recent training jobs, active deployments, alerts and AI agent activity.

6.2 Model registry

Fig. 3 shows the Model Registry. The header reports total models, total versions, storage used by artifacts and the number of production models. The registry table lists each model with its framework, latest version, lifecycle status (development, testing, staging, production or archived) and creation and update dates, and can be filtered by framework and status. Selecting a model opens a detail panel with its description, framework, latest version, status, total versions and artifact size, together with tabs for versions, artifacts and metadata and actions to view details, test in the playground, deploy or archive the model.

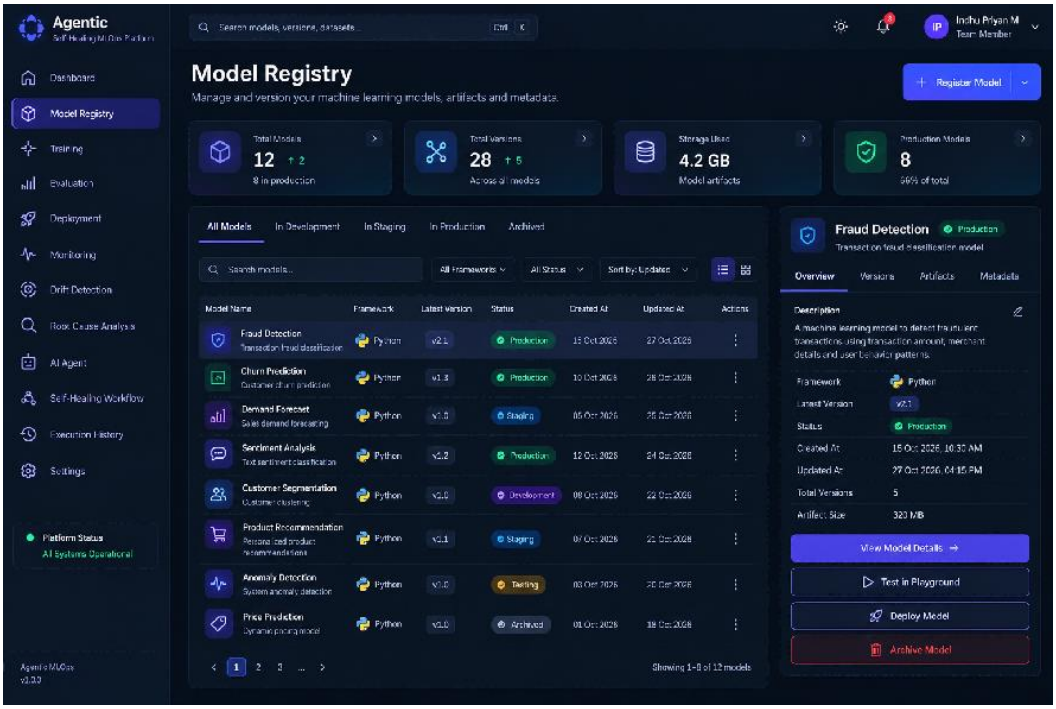


Fig. 3 - Model Registry interface showing registered models, framework information, versions, lifecycle status and model metadata.

6.3 Model training and evaluation

Fig. 4 shows the training and evaluation screen. A four-step indicator (setup, training, evaluation, register) tracks the pipeline. The configuration panel captures the model name, framework, dataset and training parameters, while the progress panel reports elapsed time, training and validation loss and accuracy for a running job. Training metrics are plotted per epoch, live training logs are streamed from the job, and the evaluation results panel reports accuracy, precision, recall and F1-score with the evaluation status against the configured threshold. A comparison table lists recent models with their datasets and metrics.



Fig. 4 - Model Training and Evaluation interface showing training configuration, dataset selection, training progress, logs and evaluation metrics.

6.4 Deployment and inference

Fig. 5 shows the deployment and inference screen. Summary tiles report active deployments, total predictions, average latency and uptime. The deployed-models table lists each deployment with its health, version, environment and last update, and the detail panel shows the endpoint URL, framework, environment, instance type, autoscaling settings, last deployment time and status. A prediction playground accepts JSON or form input and returns the prediction, its confidence and the class probability distribution, and a log of recent inference requests shows prediction, confidence and latency per request.

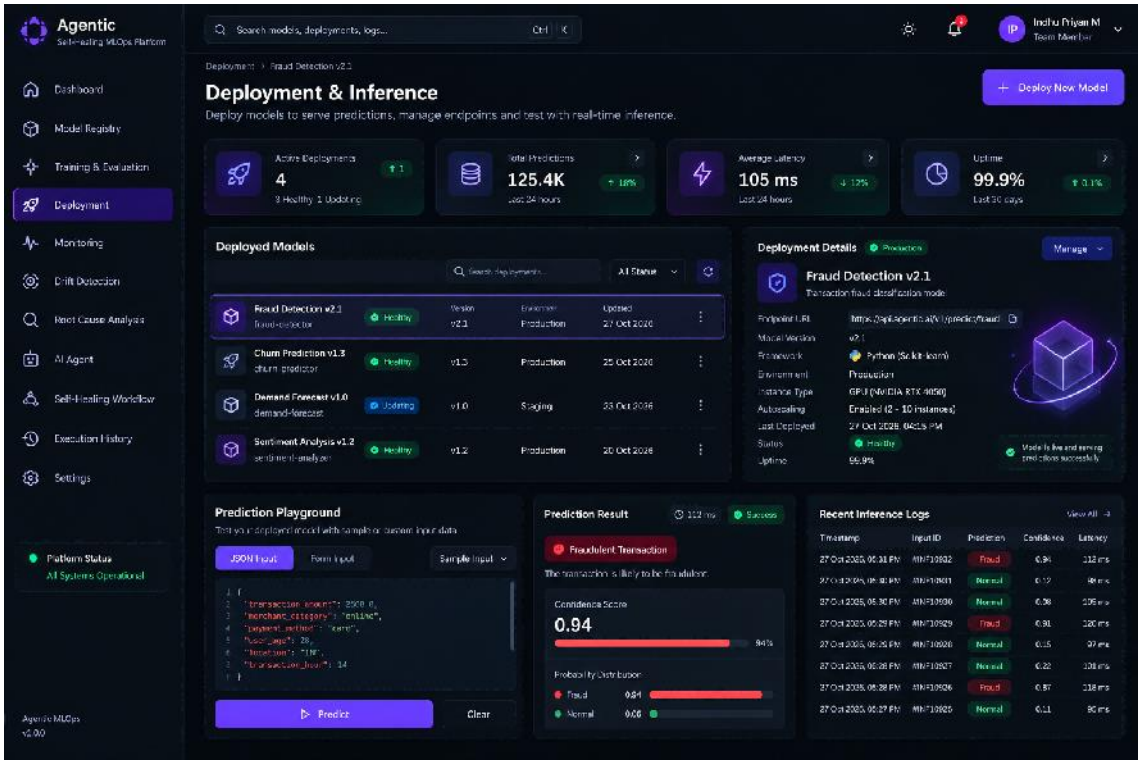


Fig. 5 - Deployment and Inference interface showing deployment management, endpoint configuration, the prediction playground and inference results.

6.5 Monitoring, drift detection and root cause analysis

Fig. 6 shows the monitoring screen. Operational tiles report prediction latency, throughput, error rate and model accuracy; a live performance chart plots accuracy, precision, recall, F1-score and latency; and resource utilisation gauges show CPU, GPU, memory and disk usage. The drift detection panel lists drifted features with their drift score, status (high, moderate or normal), a current-versus-reference distribution sparkline and an investigate or monitor action. The root cause analysis panel ranks the top contributing features (labelled as SHAP contributions in the interface) and plots reference against current data for a selected feature. Recent alerts and a model health summary with a self-healing recommendation complete the view.



Fig. 6 - Monitoring, Drift Detection and Root Cause Analysis interface showing operational metrics, drifted features, feature impact and root cause investigation.

6.6 AI agent and self-healing workflow

Fig. 7 shows the AI agent and self-healing workflow screen. Header cards report the agent decision, the trigger reason (for example drift detected with its score and threshold), the decision confidence, the estimated remediation time and the expected outcome. The pipeline visualisation shows the six workflow stages (detect issue, root cause analysis, agent decision, retrain model, evaluate, deploy) with their sub-tasks and status. The decision details panel lists the current issue, an analysis summary, the recommended action and the evidence supporting it, with controls to approve and execute or to view the detailed analysis. The workflow execution timeline shows each step with its timestamp and status (completed, in progress or pending), including deployment with health checks and the final verify-and-monitor step that precedes rollback handling when verification fails. A preview compares the current and retrained model metrics, and a history table lists recent agent actions with decision, confidence, status and outcome.

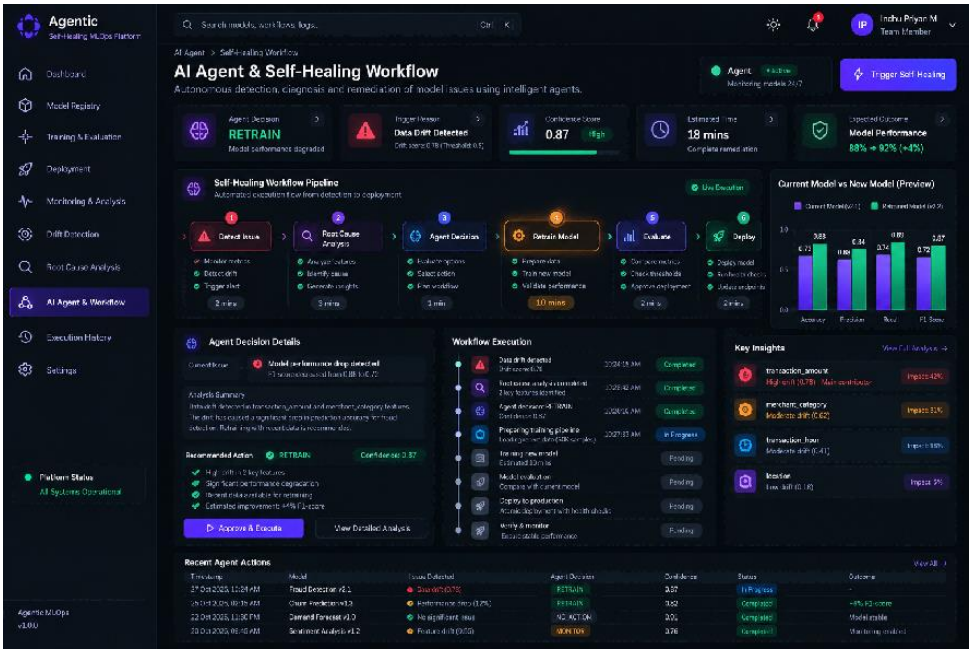


Fig. 7 - AI Agent and Self-Healing Workflow interface showing autonomous decisions, workflow stages, execution status, retraining, evaluation, deployment, verification and rollback handling.

7. Conclusion and Future Enhancements

This paper presented the Agentic Self-Healing MLOps Platform, an integrated system for managing machine learning models throughout their lifecycle and for responding systematically to performance degradation. The platform combines a centralised model registry, a Training Worker, an Evaluation Module, a Deployment Manager, monitoring, drift detection and root cause analysis behind a FastAPI backend with a React frontend and persistent SQLite records. An AI Agent Decision Layer applies an explicit policy to structured evidence to select NO_ACTION, MONITOR or RETRAIN, and a LangGraph workflow coordinates retraining, evaluation gating, deployment, verification and conditional rollback. The end-to-end demonstration executed these stages and verified completion of the workflow, and the prediction API served deployed models. The main outcomes are reduced dependence on manual lifecycle operations, evidence-based rather than trigger-based retraining, and traceability of versions, jobs, decisions and recovery attempts.

The work has clear limitations. The implementation targets local development and demonstration rather than production scale; self-healing is only possible for models whose training assets are registered; accuracy monitoring depends on ground-truth labels; and no quantitative evaluation of maintenance effort or recovery time has yet been carried out.

Several enhancements are proposed as future work and are not part of the current implementation. Distributed training workers and containerised training environments (for example with Docker and Kubernetes) would allow larger and concurrent workloads. Cloud-based object storage for datasets, artifacts and historical records would support multiple models and larger datasets. Advanced drift detection could monitor prediction confidence, data quality, feature-level drift and performance across data segments, with configurable alerts. More sophisticated AI-based root cause analysis could analyse historical incidents, compare recovery strategies and recommend corrective actions while retaining human oversight for critical operations. Additional model-serving integrations and external MLOps tools, experiment tracking, scheduled and event-triggered retraining policies, and validation methods such as comparison with the deployed model, canary deployment and controlled rollout would reduce deployment risk. Production-scale infrastructure would further require role-based access control, secure authentication, encrypted communication, secrets management, audit logs, actual health checks against deployed endpoints and human approval workflows for high-impact actions. With these enhancements the platform can evolve into a more intelligent, scalable and production-ready solution for maintaining machine learning systems.

Acknowledgements

The authors express their sincere gratitude to the project guide, Nandhini S, Department of Computer Science and Engineering, for her valuable technical suggestions and continuous guidance throughout this work, and to Sri Shakthi Institute of Engineering and Technology, Coimbatore, for providing the facilities and support required to carry out the project.

References

Amershi, S., Begel, A., Bird, C., DeLine, R., Gall, H., Kamar, E., Nagappan, N., Nushi, B., & Zimmermann, T. (2019). Software engineering for machine learning: A case study. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)* (pp. 291–300). IEEE.

Breck, E., Cai, S., Nielsen, E., Salib, M., & Sculley, D. (2017). The ML test score: A rubric for ML production readiness and technical debt reduction. In *2017 IEEE International Conference on Big Data (Big Data)* (pp. 1123–1132). IEEE.

Evidently AI. (n.d.). *Evidently documentation: ML model monitoring and data drift detection.* Retrieved October 30, 2026, from https://docs.evidentlyai.com/

FastAPI. (n.d.). *FastAPI documentation.* Retrieved October 30, 2026, from https://fastapi.tiangolo.com/

Gama, J., Žliobaitė, I., Bifet, A., Pechenizkiy, M., & Bouchachia, A. (2014). A survey on concept drift adaptation. *ACM Computing Surveys, 46*(4), Article 44.

Kreuzberger, D., Kühl, N., & Hirschl, S. (2023). Machine learning operations (MLOps): Overview, definition, and architecture. *IEEE Access, 11*, 31866–31879.

LangChain. (n.d.). *LangGraph documentation.* Retrieved October 30, 2026, from https://docs.langchain.com/oss/python/langgraph/overview

Lu, J., Liu, A., Dong, F., Gu, F., Gama, J., & Zhang, G. (2019). Learning under concept drift: A review. *IEEE Transactions on Knowledge and Data Engineering, 31*(12), 2346–2363.

Lundberg, S. M., & Lee, S.-I. (2017). A unified approach to interpreting model predictions. In *Advances in Neural Information Processing Systems 30 (NeurIPS 2017)* (pp. 4765–4774).

Paleyes, A., Urma, R.-G., & Lawrence, N. D. (2022). Challenges in deploying machine learning: A survey of case studies. *ACM Computing Surveys, 55*(6), Article 114.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, É. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research, 12*, 2825–2830.

Python Software Foundation. (n.d.). *Python 3 documentation.* Retrieved October 30, 2026, from https://docs.python.org/3/

Rabanser, S., Günnemann, S., & Lipton, Z. C. (2019). Failing loudly: An empirical study of methods for detecting dataset shift. In *Advances in Neural Information Processing Systems 32 (NeurIPS 2019)*.

Sculley, D., Holt, G., Golovin, D., Davydov, E., Phillips, T., Ebner, D., Chaudhary, V., Young, M., Crespo, J.-F., & Dennison, D. (2015). Hidden technical debt in machine learning systems. In *Advances in Neural Information Processing Systems 28 (NeurIPS 2015)* (pp. 2503–2511).

SQLAlchemy. (n.d.). *SQLAlchemy documentation.* Retrieved October 30, 2026, from https://docs.sqlalchemy.org/

Testi, M., Ballabio, M., Frontoni, E., Iannello, G., Moccia, S., Soda, P., & Vessio, G. (2022). MLOps: A taxonomy and a methodology. *IEEE Access, 10*, 63606–63618.

Wang, L., Ma, C., Feng, X., Zhang, Z., Yang, H., Zhang, J., Chen, Z., Tang, J., Chen, X., Lin, Y., Zhao, W. X., Wei, Z., & Wen, J.-R. (2024). A survey on large language model based autonomous agents. *Frontiers of Computer Science, 18*(6), Article 186345.

Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., & Cao, Y. (2023). ReAct: Synergizing reasoning and acting in language models. In *Proceedings of the Eleventh International Conference on Learning Representations (ICLR 2023)*.

Zaharia, M., Chen, A., Davidson, A., Ghodsi, A., Hong, S. A., Konwinski, A., Murching, S., Nykodym, T., Ogilvie, P., Parkhe, M., Xie, F., & Zumar, C. (2018). Accelerating the machine learning lifecycle with MLflow. *IEEE Data Engineering Bulletin, 41*(4), 39–45.