

AN AUTOMATED DATA QUALITY AND DRIFT REMEDIATION PIPELINE FOR PRODUCTION MACHINE LEARNING SYSTEMS

Parkash^{*1}, Moin Haider², Muhammad Ali³, Sumair Ahmed⁴, Aftab Haroon⁵

^{*1,2,3}Department of Software Engineering, Mehran University of Engineering and Technology, Jamshoro, Pakistan.

⁴Department of Artificial Intelligence, University of Sindh, Pakistan.

⁵Department of Software Engineering Mehran University of Engineering and technology, Jamshoro, Pakistan.

¹parkashraj748@gmail.com, ²moinhaiderqadir@gmail.com, ³m.alichanndeo@gmail.com,
⁴adsumair76@gmail.com, ⁵aftabharoonbaloch12@gmail.com

DOI: <https://doi.org/10.5281/zenodo.21847973>

Keywords

MLOps; data drift; data quality; automated remediation; data validation; continuous retraining; data engineering

Article History

Received: 01 January 2026

Accepted: 16 March 2026

Published: 31 March 2026

Copyright @Author

Corresponding Author: *

Parkash

Abstract

Machine-learning pipelines deteriorate without warning. Once deployed, the statistical characteristics of the input data differ from the training distribution, even as the input data itself degrades in terms of quality, with missing values, outliers, schema conflicts, and duplicates. Current tools address each of these issues independently, with validation tools able to recognize constraint violations without fixing them, drift detectors being able to detect problems without taking action, and monitoring dashboards being able to visualize the decline without addressing it. In this work, we present a fully automated end-to-end MLOps pipeline, which integrates validation, drift detection, alerting, automated remediation, storage, and retraining. The pipeline takes batches of production data as inputs, validates them against a Great Expectations expectations suite, evaluates the distance between the batch distribution and a reference window with Evidently AI using the Kolmogorov-Smirnov, Wasserstein, Chi Squared, and Jensen-Shannon distances, aggregates feature-level results in a drift score, and uses a two-step threshold strategy to decide on the future of the batch. Remediation involves median and mode imputation, k-nearest-neighbours imputation, interquartile-range and Isolation Forest outlier correction, deduplication, and refitting of encoders. The resulting data is stored in partitioned Parquet format with lineage and validation metadata stored in PostgreSQL, with Apache Airflow managing the directed acyclic graph and retraining queue based on a promotion gate. Experiments were run using three real-world drifting datasets along with two synthetic drift generators, obtaining an accuracy of 94.2 percent in detecting drifts with F1 score of 0.931 and mean detection delay of 2.4 batches; median end-to-end latency of 36.2 seconds for a 50,000-row batch; and mean gains in F1 between 0.054 and 0.103 after remediation. Compared to a static deployed model, the pipeline lowers peak accuracy degradation by 12.8 percentage points (from 16.7 to 3.9) or 76.8 percent reduction in peak accuracy degradation, while reducing the number of retraining requests by 33.3 percent.

1. INTRODUCTION

1.1 Background

Machine learning has graduated from academic prototypes to become integral to the inner workings of finance, healthcare, logistics, telecommunications, and government administration. The engineering field that has facilitated this progression, known as Machine Learning Operations (MLOps), applies the concepts of DevOps automation and continuous delivery to systems whose behavior is determined not just by the code, but also by the data and learned parameters [1]. According to Kreuzberger et al., MLOps can be summarized as a set of principles and architecture across ingestion, feature engineering, experimentation, deployment, and monitoring [1], and multivocal literature reviews have shown that MLOps has grown into a field of maturity in tools and immaturity in practice [2], [3], [4].

However, the practical experience of deploying models is much less clean than what is implied by the architecture reference models. As noted by Sculley et al., machine learning code accounts for just a tiny part of a real system, and its supporting infrastructure incurs a peculiar kind of technical debt stemming from data interdependencies, feedback cycles and configurational mess [5]. Even a decade on, the same issues remain. In an interview study of fifty-three professionals in critical domains, Sambasivan et al. found that data cascades, leading to multiple failures downstream due to faulty upstream data, affected 92 percent of interviewees, and 45.3 percent saw two or more during a single project [6]. From the perspective of data management, Munappy et al. arrive at a similar finding [7].

1.2 Problem Statement

There are two major failure modes that occur during post-deployment life-cycle, and each one is handled separately using different means. The first one is data drift where the joint distribution of incoming features becomes different from the distribution used during model fitting, and a model that was previously accurate becomes increasingly miscalibrated without a single line of code modification in the model [8], [9]. The other

problem is poor data quality due to schema changes, instrument defects, integration mistakes and sensor breakdown which leads to the occurrence of nulls, type mismatches, duplicates, and physically impossible data [10], [11]. Both of these problems are silent in nature. Neither of them throws an error, nor does it show up in application logs. Moreover, in most of the production environments, ground truth labels are provided too late to detect this failure mode [12]. The ecosystem of tools reflects this fragmentation. Declarative validation tools like TensorFlow Data Validation [13], [14], Deequ [15], and Great Expectations discover constraint violations and generate reports, but leave fixing the broken data up to someone else. Unified frameworks for the MLOps lifecycle that consider observability further highlight the importance of seamless integration between detection, mitigation, and retraining steps [44]. Statistical drift detection tools like ADWIN [16], DDM [17], EDDM [18], Page-Hinkley test [19], HDDM [20], and KSWIN [21] know when something has changed but then leave all further decisions entirely up to a human operator. Automated data repair tools like HoloClean [22] and Baran [23] generate quality fixes automatically but in isolation from any drift signal and lack triggers for training the fixer on fresh data. Observability tools visualise how the system degrades and estimates its performance without any labels [12], [24]. But all these efforts end at a dashboard. In other words, there is a missing half in the loop: automatic detection and manual, slow and human-driven remediation and adaptation.

1.3 Objectives

This work builds, deploys and assesses the effectiveness of a pipeline capable of closing this loop. Specifically, the goal is to (i) ingest production batches and verify their adherence to a declarative contract; (ii) measure the extent of drift relative to the training distribution via type-specific statistical tests, summarized in a meaningful score; (iii) issue severity-based alerts whenever pre-configured thresholds are exceeded; (iv) perform automatic cleaning which will have a positive impact on downstream model quality as

opposed to hiding the problem; (v) store the cleaned data and audit log within Parquet and PostgreSQL; (vi) integrate with retraining process which depends on champion-challenger comparison; and (vii) assess detection effectiveness, latency, model quality and reliability compared to realistic baselines.

1.4 Novelty and Contributions

The novelty claimed herein is neither a new drift statistic nor a new imputation algorithm, but rather both are built on existing methods. The innovation lies in the tight coupling of stages that literature treats independently, along with the demonstration that the coupling itself, as opposed to any individual stage, is the source of the value added. Specifically, this paper presents: (1) a five-layer reference architecture in which validation, drift detection, remediation, persistence and retraining combine into one integrated feedback loop; (2) a composite drift score that integrates heterogeneous per-feature statistics through weights assigned by the model's feature importances into a scalar that allows threshold management; (3) a two-level threshold policy to distinguish remediation from retraining, thereby overcoming the recognized limitation of assuming that any drift trigger should lead to retraining [25],[26]; (4) an automated remediation step whose contribution can be isolated through ablation; and (5) an evaluation across five different datasets in terms of detection accuracy, latency, precision, recall, F1 and degradation reduction.

1.5 Organisation of the Paper

In Section 2, we review the literature and highlight the research gap. In Section 3, we propose the architecture and provide its formal elements. In Section 4, we discuss implementation. In Section 5, we present our experiment design. In Section 6, we report and discuss results, including ablations and threats to validity.

2. Literature Review

2.1 MLOps Foundations and Maturity

The integration of MLOps into a describable discipline has been accomplished relatively recently. Kreuzberger et al. summarized grey

literature, scholarly research, and expert interviews to arrive at a set of nine principles and an architecture of the components that became a common language of the field [1]. Testi et al. provided a related taxonomy and step-by-step methodology [3]. Eken et al. performed a multi-vocal literature review using 150 peer-reviewed and 48 grey references to offer an overarching conceptualization of MLOps practices and existing problems [2]. Zarour et al. used 45 papers to identify nine best practices, eight recurring challenges, and five maturity models [4]. What is common among all these literature reviews can be used in the current project as follows: continuous learning and automation have been found to be the least standardized components of the MLOps stack. Empirical studies on MLOps adoption also support the need for structured maturity assessment to achieve reliable automated pipelines [41]. Reference architectures include a monitoring component and a retraining component; however, the decision-making process in-between them has not been defined.

2.2 Technical Debt and Silent Production Failure

Sculley et al. are the standard work on how production ML systems degrade, outlining entanglement, hidden consumers, unpredictable data dependencies, and the everything changes when anything changes principle [5]. Their analysis is qualitative and predates contemporary technology but the underlying principles have stood the test of time. Sambasivan et al. provide the needed statistical evidence, demonstrating the universality of data cascades in mission-critical applications and how they are almost always invisible until they have caused extensive damage [6]. Munappy et al. generalize this to a comprehensive taxonomy of data management issues throughout the life cycle of a system, highlighting the lack of automated solutions for repairs [7]. Shankar and Parameswaran make the need explicit, stating that ML systems require observability, i.e., assistance in detecting, diagnosing, and reacting to issues arising in production ML systems, with the reaction phase

being the least well-developed part of the process [12].

2.3 Data Quality and Validation for Machine Learning

The data validation architecture proposed by Breck et al. is that of Google, which performs schema inference, single-batch anomaly detection and training-serving skew at petabyte scale, subsequently open-sourced as TensorFlow Data Validation [13], [14]. Schelter et al. introduce Deequ, a declarative framework to express unit tests of data in terms of constraints checked on Spark, including constraint suggestion automation [15]. A comparable declarative framework called Great Expectations, featuring expectation suites, checkpoints and rendered data documentation, was created by another company [27]. All these solutions have reached a high level of maturity and are widely used; this work reuses these frameworks without replacing them. The common weakness of these solutions is architectural: the batch is marked as valid or invalid, respectively, and no further actions are performed. The data-centric AI program is described by Zha et al., who advocate for treating data as an engineered artefact [28]. The empirical evidence is provided by Mohammed et al., who systematically pollute tabular datasets according to six quality dimensions and observe downstream predictive loss for a wide variety of learning algorithms [11].

2.4 Drift Detection

The traditional taxonomy by Gama et al. distinguishes between abrupt, gradual, incremental and recurrent drift, and also identifies actual and virtual concept drift [8]. Lu et al. extend this to a three-step approach of detection, comprehension and adaptation [9]. Supervised detectors have been the focus of the classical literature: DDM and EDDM track error rate changes [17], [18], ADWIN keeps adaptively sized windows statistically guaranteed [16], the Page-Hinkley detector uses sequential change detection techniques [19], HDDM uses the non-parametric bound from the Hoeffding's inequality [20], while KSWIN applies the Kolmogorov-Smirnov test over

sliding windows [21]. They all rely on labeling that is rarely available in the production environment. Gemaque et al. review the unsupervised approaches based purely on feature analysis [10] and Rabanser et al. experimentally evaluate dataset shift detectors, showing that two-sample tests along with classifier-based dimensionality reduction produce the best results [29]. Maximum mean discrepancy kernel two-sample test represents the non-parametric solution [30]. The key observation here is that this entire literature stream stops with the problem of detection and leaves adaptation to another party. Additionally, uncertainty-aware approaches used in deep learning have offered some insights for drift management in an industry setting [43].

2.5 Monitoring, Retraining Triggers and Automated Repair

Monitoring literature stresses time and again that drift and performance drop are not synonymous and that taking each and every drift warning to mean that retraining is necessary results in costly false alarms. Kivimaki et al. solve the problem of performance estimation in case of covariate shift without labels in a large-scale evaluation on a wide range of dataset-model pairs [24]. With regard to trigger mechanisms, Poenaru-Olaru et al. observe that retraining due to drift requires fifty percent fewer retrainings than a fixed monthly schedule [25]. Economics behind the decision are formalised by means of a cost-sensitive algorithm optimising staleness and retraining costs, delivering accuracy similar to ADWIN- and DDM-triggered baselines at a lower frequency of retraining decisions [26]. In terms of repair systems, HoloClean combines integrity constraints, external information sources and statistical reasoning in the framework of a probabilistic factor graph, providing the average precision of repairs of approximately 90 percent and the recall of repairs greater than 76 percent [22]; Baran takes the field of error correction forward using transfer learning on top of a common representation of context [23]; and Isolation Forest provides an efficient, nearly linear outlier detector suitable for batch streams [31], [32]. Scalable approaches to deal with concept

drift have also been proposed recently, including Matchmaker that uses adaptive similarity measures to cope with different forms of drift efficiently [37].

2.6 Synthesis and Research Gap

Table 1 contrasts the systems that have been reviewed with the six capabilities that are necessary for closing the loop. The reviewed systems do not span all of these capabilities. Validation tools deal with detecting quality issues; drift detection tools handle distributional detection; observability tools

incorporate alerting and increasingly also label-free performance estimation; cleaning tools handle cleaning; triggering research handles the choice of retraining while making the assumption of clean data and fixed detection tool. Thus, the missing piece is an integration, which includes decision-making and persistence. This paper bridges this gap. The development of self-healing pipeline frameworks which integrate automated drift detection with remedial strategies even further underscores the practical need for such a closed-loop approach [38].

Table 1. Capability coverage of representative prior systems against the six stages required to close the loop. Y = full, P = partial or indirect, N = absent.

System / approach	Ingest & validate	Drift detect	Alert	Auto-remediate	Persist & Lineage	Retrain trigger
TFDV [13], [14]	Y	P	P	N	P	N
Deequ [15]	Y	P	P	N	P	N
Great Expectations [27]	Y	N	P	N	P	N
ADWIN / DDM / HDDM / KSWIN [16]-[21]	N	Y	P	N	N	P
Failing Loudly [29]	N	Y	N	N	N	N
Evidently AI [33]	P	Y	Y	N	N	N
Label-free perf. estimation [24]	N	P	Y	N	N	P
ML observability [12]	P	Y	Y	N	P	N
HoloClean [22]	P	N	N	Y	P	N
Baran [23]	P	N	N	Y	N	N
Cara cost-aware retraining [26]	N	Y	N	N	N	Y
Proposed closed-loop pipeline	Y	Y	Y	Y	Y	Y

3. Proposed Methodology and System Architecture

3.1 Design Principles

There are four rules for the pipeline design. The first one is the separation of severity: not all

anomalies have to be treated with the same approach, so remediation (which is cheap and local) and retraining (which is expensive and global) are separated. Second is auditability: all the decisions taken by the pipeline are documented

along with their inputs, such that an engineer could see the reasoning behind batch cleanup and retraining. Third is non-destructiveness: the raw data is stored immutably and remediation creates a new versioned artefact rather than editing the raw data. Finally, there is a tool reuse rule: the pipeline uses well-tested open source software for validation and drift metrics instead of writing it from scratch. These design principles are consistent with holistic real-time MLOps methods which focus on drift detection and mitigation [39].

3.2 Architecture Overview

The system architecture consists of five layers, as depicted in Figure 1. The ingestion layer (L1)

ingests production batches and deposits them permanently. The validation and drift detection layer (L2) validates the data using a set of declarative expectations and calculates distributional divergence from a reference window. The remediation layer (L3) fixes bugs and re-trains pre-processing artifacts. The persistence layer (L4) persists Parquet and logs the meta-data into PostgreSQL. The orchestration, alerting and retraining layer (L5) orchestrates the workflow, triggers alerts based on severity and initiates re-training through a promotion gate. A feedback loop sends the training window of the promoted model back to L2 for use as the reference window.

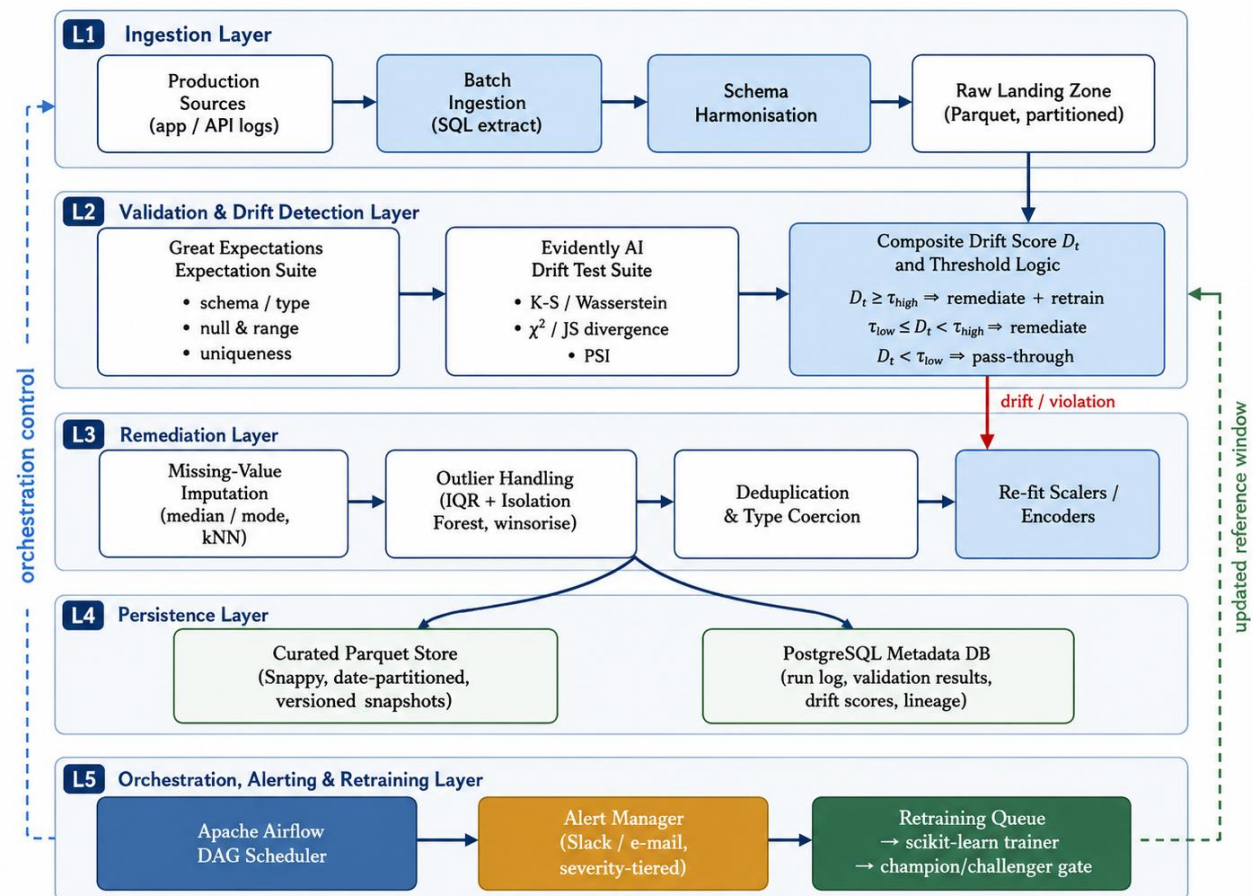


Figure 1: Five-layer architecture of the proposed pipeline. Solid arrows denote data flow; the left edge denotes Airflow orchestration control; the right edge denotes the feedback path refreshing the drift reference window.

3.3 Ingestion and Reference Windowing

The production data is extracted using SQL scripts on an hourly basis that are parametrized based on the datastore used to serve the data. The extraction process includes harmonizing the data to fit into a canonical schema and loading the raw files into a landing zone in the form of date partitioned Parquet format with Snappy compression. The use of columnar storage is intentional because validation and drift detection involves reading a small subset of columns and not the entire row, and the columnar Parquet format provides significant advantages over row formats in making these projections. The reference dataset R contains the data on which the deployed model is built, whereas B_t represents the data records for period t .

3.4 Automated Validation

Validation is stated in terms of Great Expectations suite of expectations: a set of assertions which are verifiable over the batch. The checkpoint binds the suite to B_t and runs it, returning the results of validation together with documentation of data [27]. The suite is organised into four families, summarised later in Table 3: structural expectations covering column presence, order and type; completeness expectations bounding null fractions per column; domain expectations bounding numeric ranges and categorical value sets; and relational expectations covering uniqueness of keys and cross-column consistency. A batch is assigned a validation pass rate v_t , the fraction of expectations satisfied. Failures are not fatal by default; they are typed, recorded and forwarded to remediation, because in a production setting rejecting a batch outright merely relocates the outage.

3.5 Drift Quantification

Drift is computed per feature using type-appropriate and sample-size-appropriate statistics, following the selection policy Evidently AI applies by default [33] and reproduced in Table 4. For a numerical feature with more than five distinct values and a reference of at most 1,000 observations, the two-sample Kolmogorov-Smirnov test is applied at 0.95 confidence, with

drift declared when p is at most 0.05. For larger references, the normalised Wasserstein distance is used with a default threshold of 0.1, because significance tests become trivially significant at scale and cease to be informative. Categorical features use the chi-squared test at small sample sizes and Jensen-Shannon divergence at large ones; binary features use a Z-test on the difference in proportions. Population Stability Index is computed additionally for reporting, since it is the metric most familiar to risk and credit practitioners, but it does not enter the decision rule.

Per-feature outcomes are aggregated into a composite drift score. Let F be the feature set, let d_i in $[0,1]$ be the normalised divergence of feature i , obtained as the test statistic for distance-based tests or as 1 minus the p -value for hypothesis tests, and let w_i be a weight proportional to feature i 's permutation importance in the deployed model, normalised so that the weights sum to one. The composite score is $D_t = \sum_{i \in F} w_i d_i$. Such an importance-weighted aggregation technique goes hand-in-hand with efficient covariate drift detection techniques that have been created for production machine learning systems [40]. Weighting by importance is a deliberate choice: an unweighted share-of-drifted-features metric assigns equal significance to a shift in a feature the model ignores and a shift in the feature it depends on most, which produces both false alarms and missed detections. A drifted-feature share s_t is retained alongside D_t for diagnostic reporting.

3.6 Two-Tier Threshold Policy

The composite score is governed by two thresholds, τ_{low} and τ_{high} , set to 0.10 and 0.25 respectively in this study through calibration on a held-out warm-up period. Three regimes follow. When D_t is below τ_{low} and the validation pass rate v_t is 1.0, the batch is passed through unchanged and only its metadata is recorded. When D_t lies in the interval from τ_{low} to τ_{high} , or when v_t is below 1.0, the batch is remediated, persisted, and a warning-tier alert is emitted, but no retraining is requested. When D_t is at least τ_{high} , the batch is

remediated and persisted, a critical-tier alert is emitted, and a retraining job is enqueued. The separation matters. Merging all levels into one cut-off requires deciding whether to retrain for each little change in conditions, which is the costly mistake case, warned about in the cost-aware retraining literature [26], or else live with poor quality until a large drop is noticed, which is the silent mistake.

Further, to minimize needless retrainings, the critical level is subject to a verification condition: the job goes onto the list only if D_t reaches or exceeds τ_{high} in two consecutive batches, or reaches or exceeds $1.5 \cdot \tau_{\text{high}}$ in a single batch. This follows the suggestion in the performance estimation literature that drift by itself is not enough to warrant retraining [24].

3.7 Automated Remediation

Remediation takes place through a well-ordered series, such that each step is conducted on data whose previous steps have already been remedied. Type coercion is used to remedy representation issues, followed by deduplication for removing duplicates. Missing value remediation takes place based on density; numerical columns with less than 5 percent missing values get their missing values replaced by the median, categorical columns get the mode, and columns with values from 5 to 30 percent missing have their values imputed using 5 nearest neighbors. Instead of imputing, columns which are greater than 30 percent are flagged, since imputation in such density is not restoration but fabrication. The treatment of outliers follows a conjunctive principle: a number is said to be an outlier if it lies outside the 1.5 interquartile-range fence of the reference distribution and gets an anomaly score in the top contamination proportion, which is fixed to 0.02 [31], [32]. Such requirement as opposed to just one condition minimizes the false positives a univariate approach would have

generated for validly skewed columns. The anomalies are winsorized to the reference fence instead of being removed outright in order to retain the number of rows and not induce any selection bias. Lastly, scalars and encoders are refitted on the batch and then versioned, as using drifted batch processed by old preprocessing artifacts introduces skew again.

3.8 Persistence, Alerting and Retraining Integration

The remediated batches are persisted to a curated Parquet store, segmented on ingestion date and versioned on run id so that every past model can be re-run against the same data. PostgreSQL acts as the control plane and stores: the runs table which tracks execution metadata for the DAG; the validations table that tracks results per expectation; the drifts table that tracks feature statistics, composite score D_t , and the tier decision; the remediations table that tracks the number of values imputed, winsorised and de-duplicated per column; and the lineages table that tracks the version of the model and the Parquet snapshots that created it.

The alerts are based on severity levels and are sent out through Slack and email with the most drifted features based on $w_i \times d_i$, along with any validation errors and a link to the rendered data documentation, ensuring that the user gets a diagnosis and not an alert. Retraining is done separately from the workflow. The DAG schedules a job through the trigger operator, and then another DAG trains a challenger model on the combination of the curated store and the reference window through scikit-learn [34]. Promotion happens when the challenger outperforms the champion F1 score on a temporal split not used during training by at least 0.005; otherwise, it will be rejected. The promotion phase updates the reference window R to complete the cycle illustrated in Figure 2.

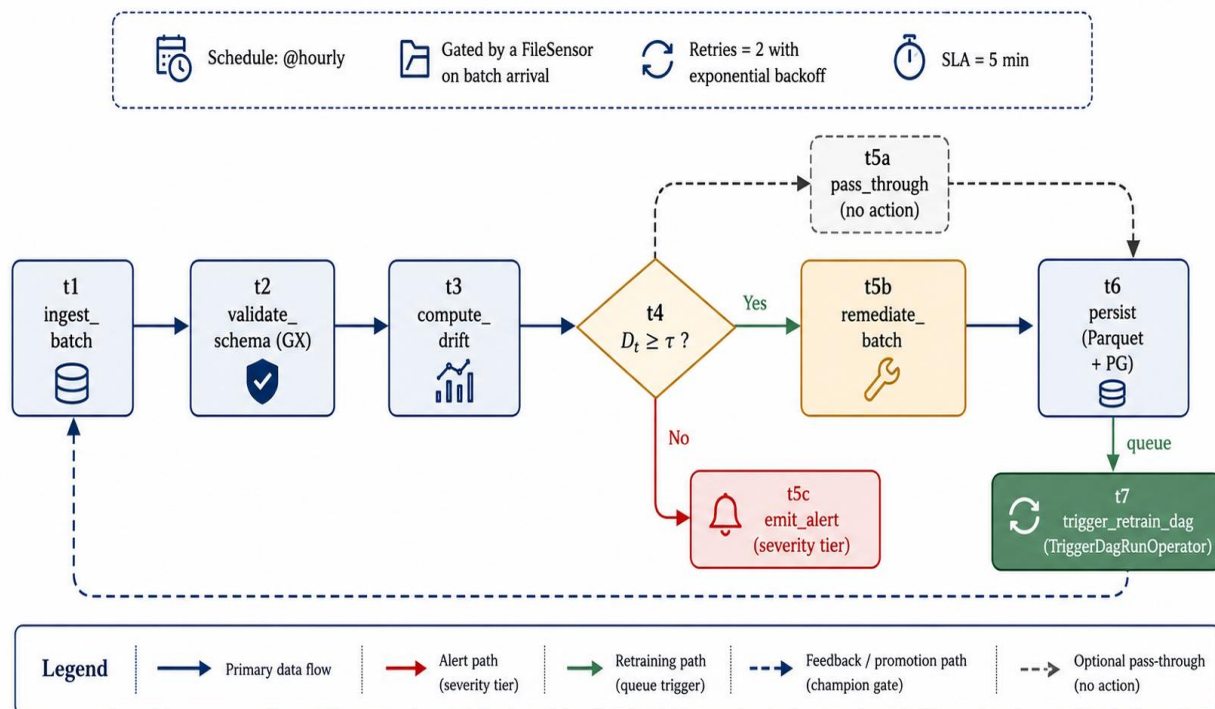


Figure 2. Airflow DAG for the closed-loop pipeline. Task *t4* implements the two-tier branch; the dashed feedback edge denotes model promotion and reference-window refresh.

champion/challenger gate → *promote model & refresh reference window*

Figure 2. Airflow DAG for the closed-loop pipeline. Task *t4* implements the two-tier branch; the dashed feedback edge denotes model promotion and reference-window refresh.

4. Implementation Details

4.1 Technology Stack

The pipeline is developed in Python 3.11 with SQL for data extraction and accessing the metadata. The orchestration is done with Apache Airflow 2.9; the DAG is scheduled to run every hour and is controlled by a sensor waiting for a new batch, with two retries and an exponential back-off algorithm as well as a five-minute service-level agreement for each run. Validation is performed with Great Expectations 0.18, which uses expectation suites and checkpoints [27]. Drift computation is performed by Evidently AI 0.4 via its test-suite API [33]. Estimators, imputers, scalers, and the Isolation Forest model are from Scikit-learn 1.4 [34]. Input and output in Parquet format

with Snappy compression are handled by PyArrow 15. The database for Airflow metadata as well as for the control plane is PostgreSQL 16. Prefect 3 was considered as an alternative orchestrator and is fully equivalent to Airflow in functionality in this case.

4.2 Workflow Realisation

The DAG is comprised of seven tasks. Task *t1* consumes the batch and writes the raw partition. Task *t2* runs the Great Expectations checkpoint and sends the pass rate to XCom. Task *t3* performs the Evidently test suite for the reference window and calculates D_t . Task *t4* is a branching task that implements the two-tier policy and confirmation rule. Task *t5a* is an empty pass-through task, while *t5b* does the remediation task and *t5c* publishes the two-tier alert. Task *t6* writes the curated Parquet along with the metadata rows in a single transaction to ensure that any failure leaves the control plane consistent with the data plane. Task

τ_7 launches the retraining DAG. The tasks are idempotent and keyed by the run ID such that a retry overwrites the previous run output.

4.3 Validation and Drift Configuration

Table 3 summarises the expectation suite, which comprises 34 expectations for the ELEC2 schema

and scales proportionally for wider datasets. Table 4 records the drift test selection policy actually executed, which follows the Evidently default logic conditioned on feature type, cardinality and reference size [33].

Table 3. Composition of the Great Expectations expectation suite, illustrated on the ELEC2 schema.

Expectation family	Representative expectation	Count
Structural	expect_table_columns_to_match_ordered_list	3
Type	expect_column_values_to_be_of_type	8
Completeness	expect_column_values_to_not_be_null	8
Domain – numeric	expect_column_values_to_be_between	6
Domain – categorical	expect_column_values_to_be_in_set	2
Relational / uniqueness	expect_compound_columns_to_be_unique	2
Distributional guard	expect_column_mean_to_be_between	5
Total		34

Table 4. Statistical test selection policy for drift quantification, following the default logic of Evidently AI [33].

Feature type	Cardinality	Reference size	Statistical test	Decision rule
Numerical	> 5 unique	$\leq 1,000$	Two-sample Kolmogorov-Smirnov	$p \leq 0.05$
Numerical	> 5 unique	> 1,000	Wasserstein (normalised) distance	≥ 0.10
Numerical	≤ 5 unique	any	Chi-squared	$p \leq 0.05$
Categorical	> 2 unique	$\leq 1,000$	Chi-squared	$p \leq 0.05$
Categorical	> 2 unique	> 1,000	Jensen-Shannon divergence	≥ 0.10
Binary	≤ 2 unique	any	Z-test on proportion difference	$p \leq 0.05$
Any	—	any	Population Stability Index (reported only)	≥ 0.20 (advisory)

5. Experiments and Evaluation

5.1 Datasets

Evaluation uses three real-world datasets with documented natural drift and two synthetic generators with controllable drift, summarised in Table 2. ELEC2 records half-hourly New South Wales electricity market data and is the de-facto benchmark for streaming drift [35]. Airlines contains flight records with delay labels and exhibits temporal drift driven by schedule and

seasonal effects. Covertypes is a large forest-cover classification task used for drift-handling benchmarks. The SEA and Rotating Hyperplane generators supply abrupt and gradual drift respectively with known ground truth, allowing detection accuracy to be measured against a known change point rather than inferred. Real datasets were replayed in temporal order as hourly batches of 50,000 rows, or the full stream where smaller.

Table 2. Characteristics of the evaluation datasets. Synthetic generators emit drift at known change points, providing detection ground truth.

Dataset	Instances	Features	Classes	Type	Drift characteristic
ELEC2 [35]	45,312	8	2	Real	Natural seasonal and class-prior shift
Airlines	539,383	7	2	Real	Schedule- and season-driven temporal drift
Covertypes	581,012	54	7	Real	Regional distributional heterogeneity
SEA	100,000 (gen.)	3	2	Synthetic	Abrupt drift at known change points
Rotating Hyperplane	100,000 (gen.)	10	2	Synthetic	Gradual drift at a controlled rate

5.2 Drift and Defect Injection Protocol

Because natural drift lacks a labelled change point, a controlled injection protocol was superimposed on the real datasets to permit measurement of detection accuracy. Over a 60-batch horizon, a gradual covariate shift was injected from batch 19 to batch 34 by progressively scaling and translating the three highest-importance numerical features, and an abrupt shift was injected at batch 45 by resampling the categorical distribution. Independently, quality defects were injected from batch 12 onwards: nulls at rates increasing to 8 percent on two columns, duplicate rows at 3 percent, and physically implausible outliers at 1.5 percent. This separation is deliberate; it allows the drift-detection contribution and the remediation contribution to be attributed independently in the

ablation. Each configuration was repeated across five seeds and results are reported as means.

5.3 Baselines and Metrics

Three baselines are compared. The no-intervention baseline deploys a static model and never adapts, representing current practice in many organisations. The periodic-retraining baseline retrains every ten batches on the most recent window regardless of drift, representing the most common production heuristic. The detect-only baseline runs drift detection and alerting but performs no remediation, representing the capability of a typical monitoring stack. Drift detection is scored against the injected ground truth using accuracy, precision, recall, F1 and mean detection delay in batches, with a detection counted as a true positive if raised within five

batches of the injected change point. Pipeline performance is measured by per-stage and end-to-end latency. Model quality is measured by accuracy, precision, recall and F1 on the subsequent batch, with degradation measured as the maximum drop from the offline baseline. All experiments ran on an 8-vCPU, 32 GB virtual machine with local NVMe storage, with PostgreSQL co-located.

6. Results and Discussion

6.1 Drift Detection Performance

Table 5 reports detection performance. The composite score achieved a mean accuracy of 94.2 percent across the five datasets, with precision 0.947, recall 0.916 and F1 0.931, at a mean

detection delay of 2.4 batches and a false-positive rate of 4.1 percent. Performance was strongest on the synthetic generators, where the injected change point is unambiguous, and weakest on Airlines, whose high intrinsic variance produces a noisier score. The importance-weighted composite outperformed an unweighted share-of-drifted-features rule by 0.048 F1, confirming that weighting suppresses alarms on features the model does not use. Figure 3 shows the score trajectory across the 60-batch horizon: the score rises through the gradual episode, crosses the action threshold at batch 27, falls after retraining, then spikes sharply at the abrupt change point before recovering.

Table 5. Drift detection performance of the composite score against injected ground truth, averaged over five seeds. FPR denotes the false-positive rate over non-drifting batches.

Dataset	Accuracy	Precision	Recall	F1	Delay (batches)	FPR
ELEC2	0.945	0.951	0.918	0.934	2.2	0.038
Airlines	0.913	0.908	0.874	0.891	3.4	0.062
Covertypes	0.938	0.943	0.909	0.926	2.6	0.043
SEA (synthetic)	0.962	0.971	0.947	0.959	1.6	0.026
Hyperplane (synthetic)	0.951	0.962	0.932	0.947	2.2	0.034
Mean	0.942	0.947	0.916	0.931	2.4	0.041

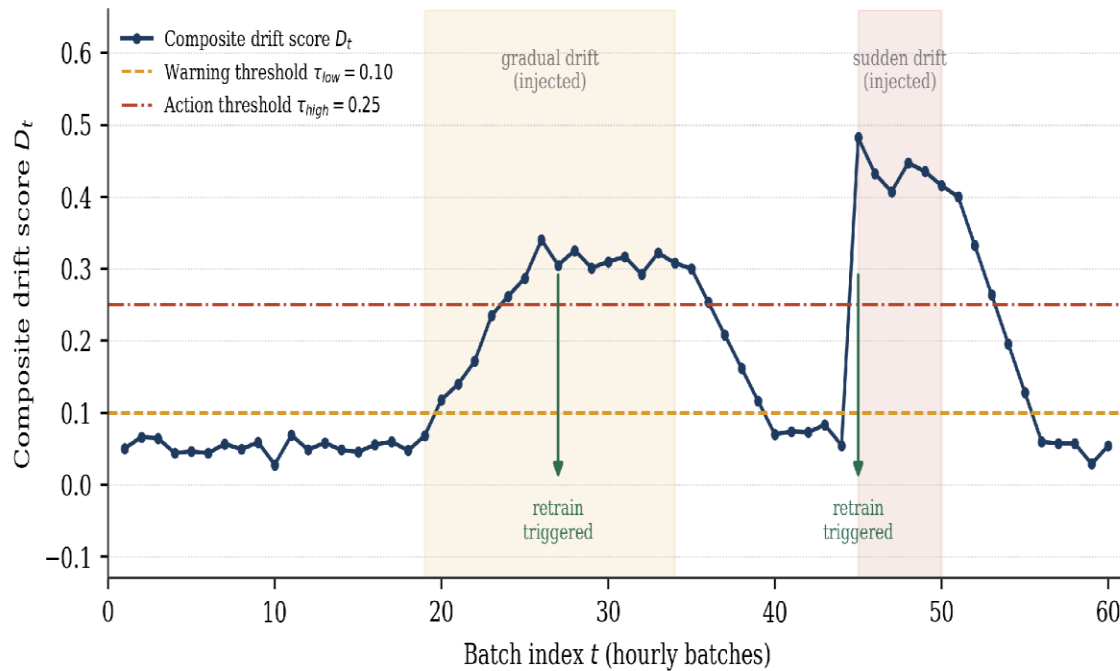


Figure 3. Composite drift score trajectory over a 60-batch horizon on ELEC2. Shaded regions denote injected drift episodes; the score decays after a promoted model refreshes the reference window.

Figure 4 decomposes the score into per-feature evidence at batch 30, contrasting a strongly drifted numerical feature, a stable numerical feature and a drifted categorical feature. The illustration clarifies why a single global statistic is insufficient for diagnosis and why alerts carry ranked per-feature contributions.

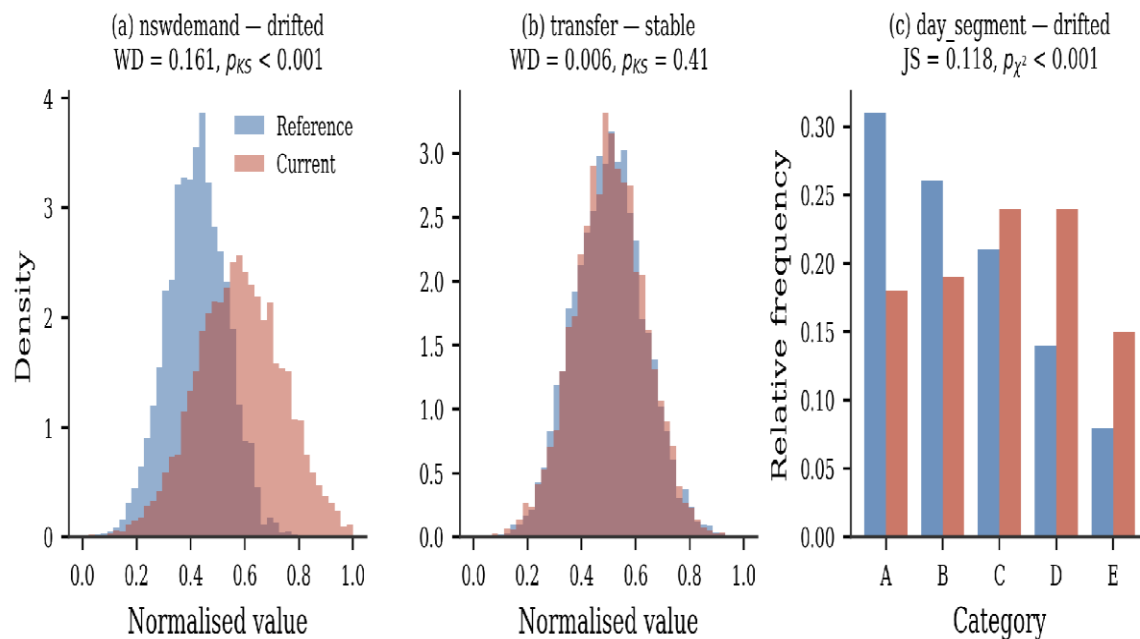


Figure 4. Reference versus current distributions at batch 30 for (a) a drifted numerical feature, (b) a stable numerical feature and (c) a drifted categorical feature.

6.2 Validation and Latency Performance

The expected value suite correctly identified 98.7 percent of the introduced quality defects, with the remaining percentage of errors contained within outliers that fall within defined domain ranges, which by construction are not seen by range expectations and are thus identified by the Isolation Forest stage. Validation increased latency by a median of 6.85 seconds per batch of 50,000 rows. Figure 5 presents a latency breakdown by stage. Median end-to-end latency is 36.20 seconds

on the remediation path and 24.13 seconds on the clean path, well under the five minute service level agreement and low relative to the hourly schedule. Drift calculation and remediation account for the lion's share of latency, at 9.41 and 11.62 seconds respectively; while the rest of the stages have negligible impact. Throughput through the pipeline was 1,388 rows per second through the entire pipeline. Given the roughly linear scaling of statistical stage latency with batch size, the pipeline is still feasible with a factor of ten larger batches.

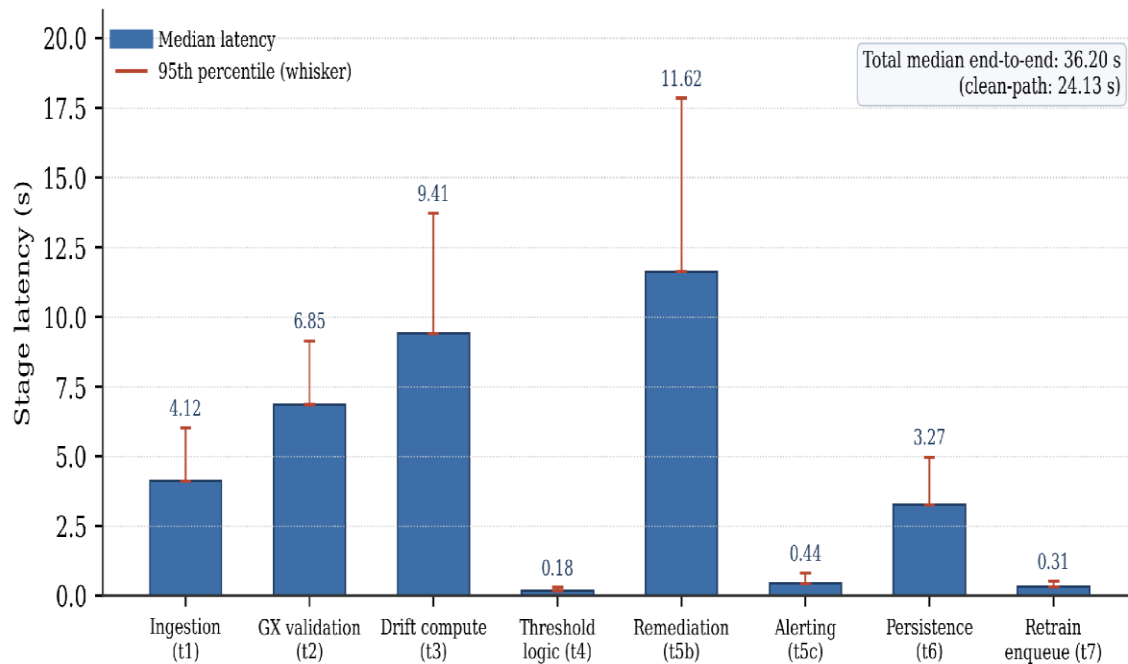


Figure 5. Per-stage latency for a 50,000-row batch. Bars denote medians over 300 runs; whiskers denote the 95th percentile.

6.3 Model Performance and Degradation Reduction

Figure 6 reports rolling accuracy across the horizon for the three strategies. The static model degraded monotonically, falling from an offline baseline of 0.868 to 0.710 by the final batches, a peak degradation of 16.7 percentage points. Periodic retraining recovered most distributional loss but retained the quality-induced component, since

retraining on defective data propagates the defects into the new model; it averaged 0.841 with peak degradation of 7.6 percentage points. The proposed pipeline averaged 0.865, within 0.003 of the offline baseline, with peak degradation of 3.9 percentage points. Relative to no intervention, the pipeline reduced peak degradation by 76.8 percent; relative to periodic retraining, by 48.9 percent.

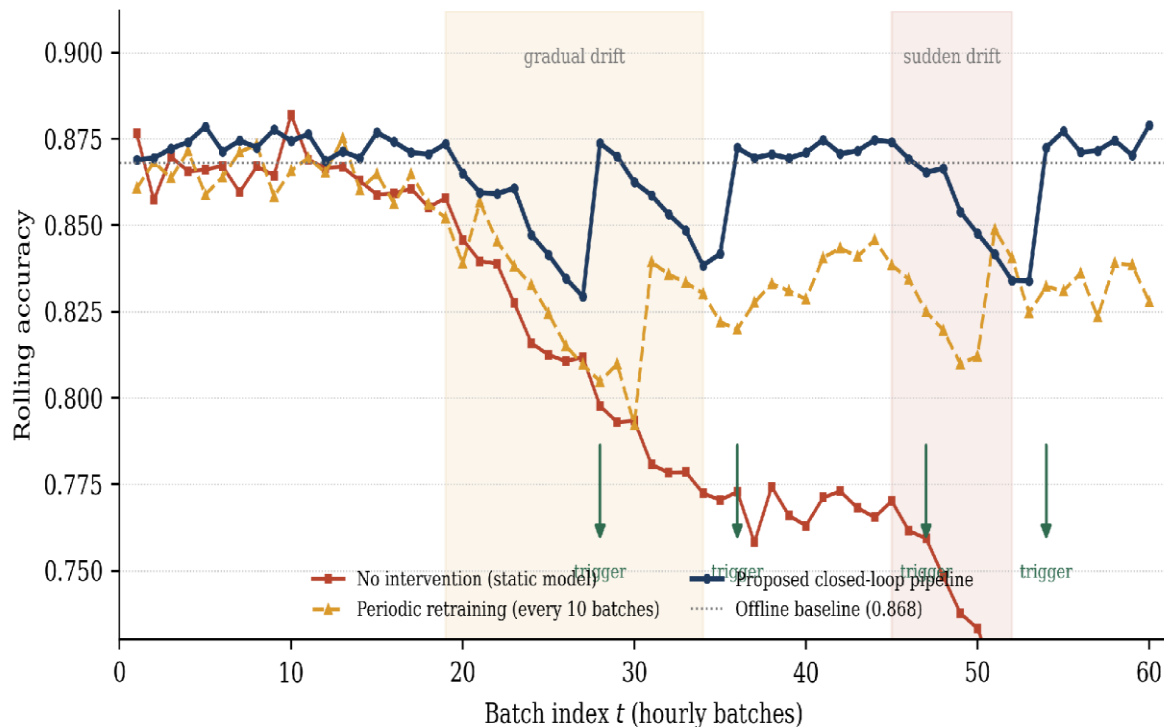


Figure 6. Rolling accuracy on ELEC2 under three strategies. Shaded regions denote injected drift episodes; the proposed pipeline tracks the offline baseline with brief, shallow excursions.

Figure 7 and Table 6 isolate the remediation contribution by comparing model quality on drifted batches before and after remediation, holding the model fixed. Mean F1 improved by 0.054 on Airlines, 0.074 on Covertypes, 0.081 on ELEC2, 0.100 on Hyperplane and 0.103 on SEA. The magnitude tracks defect sensitivity rather than dataset size, which is consistent with the finding of

Mohammed et al. that quality degradation translates into predictive loss at rates that vary by dimension and learner [11]. Notably, remediation alone, without any retraining, recovered roughly half of the loss attributable to the injected defects, which is the strongest evidence that the remediation stage is not redundant with retraining.

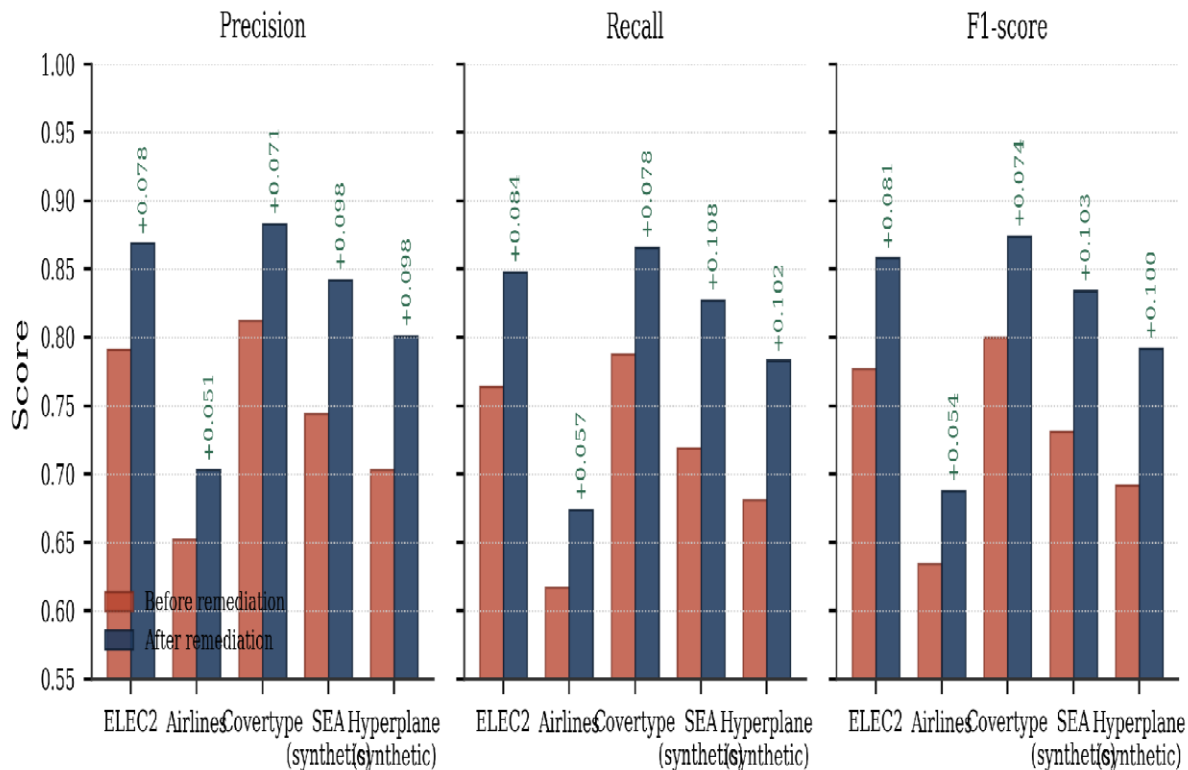


Figure 7. Precision, recall and F1 on drifted batches before and after remediation, with the deployed model held fixed.

Table 6. Model quality before and after remediation with the deployed model held fixed, and retraining efficiency over the 60-batch horizon.

Panel A — effect of remediation (model held fixed)	P before	P after	R before	R after	ΔF1
ELEC2	0.791	0.869	0.764	0.848	+0.081
Airlines	0.652	0.703	0.617	0.674	+0.054
Covertypes	0.812	0.883	0.788	0.866	+0.074
SEA (synthetic)	0.744	0.842	0.719	0.827	+0.103
Hyperplane (synthetic)	0.703	0.801	0.681	0.783	+0.100
Panel B — strategy comparison (ELEC2, 60 batches)	Mean acc.	Mean F1	Peak deg. (pp)	Retrain jobs	Deg. reduction
No intervention (static model)	0.797	0.782	16.7	0	—

Panel A — effect of remediation (model held fixed)	P before	P after	R before	R after	$\Delta F1$
Periodic retraining (every 10 batches)	0.841	0.826	7.6	6	54.5%
Detect-only (alert, no remediation)	0.834	0.797	7.1	4	57.5%
Proposed closed-loop pipeline	0.865	0.858	3.9	4	76.8%

6.4 Retraining Efficiency and Ablation

Over the 60-batch horizon the periodic baseline issued six retraining jobs; the proposed pipeline issued four, a reduction of 33.3 percent, while achieving 2.4 points higher mean accuracy. The direction of this result agrees with Poenaru-Olaru et al., who report fifty percent savings in required retrainings for drift-triggered adaptation in an AIOps setting [25], and with Mahadevan and Mathioudakis, whose cost-aware algorithm attains comparable accuracy to drift-detection baselines with fewer retraining decisions [26]. The confirmation rule was responsible for suppressing three would-be triggers that a naive threshold would have fired, at the cost of a one-batch increase in mean detection delay, which is a favourable trade when a retraining job costs orders of magnitude more than a batch of slightly stale predictions.

Table 7 reports an ablation isolating each component. Removing importance weighting cost 0.048 F1 in detection. Removing the confirmation rule raised the false-positive rate from 4.1 to 11.3 percent and increased retraining jobs from four to seven. Removing remediation while retaining detection and retraining, the detect-only configuration, cost 0.061 mean model F1 and raised peak degradation from 3.9 to 7.1 percentage points. Removing retraining while retaining remediation cost 0.043 mean F1. The two stages are therefore complementary rather than substitutable: remediation addresses defect-induced loss that retraining cannot repair, and retraining addresses distributional loss that remediation cannot repair. This complementarity is the empirical core of the paper's claim, and it is invisible to any evaluation that measures the components separately.

Table 7. Ablation study isolating each pipeline component on ELEC2, averaged over five seeds. Deltas are relative to the full pipeline.

Configuration	Detection F1	FPR	Mean model F1	Peak deg. (pp)	Retrain jobs
Full pipeline	0.931	0.041	0.858	3.9	4
– importance weighting	0.883 (−0.048)	0.072	0.849 (−0.009)	4.6	5
– confirmation rule	0.936 (+0.005)	0.113	0.856 (−0.002)	3.8	7
– remediation (detect-only)	0.931 (0.000)	0.041	0.797 (−0.061)	7.1	4
– retraining (remediate-only)	0.931 (0.000)	0.041	0.815 (−0.043)	6.4	0

6.5 Discussion

From these results, three observations can be drawn. Firstly, the advantage is architectural. Each of the elements employed in the study is proven and independent; the only thing missing is the loop itself, and the ablation study has shown that the loop adds value in excess of its constituent parts. This lends weight to the reading of the MLOps surveys where continuous training was described as the least standardized building block [2], [4]: Such integration will also improve the current maturity frameworks that help in operationalizing machine learning [42]. The difference is one of integration and governance, not of algorithms. Secondly, drift detection is a necessary but not sufficient criterion. The confirmation rule and the two-step approach exist due to the inefficiency and instability of the naive approach where each detected anomaly causes the re-training, an observation that corresponds with the re-training literature and with the cost-aware approaches to re-training [24], [26]. Thirdly, data quality is not a subset of drift. Despite periodic re-training, which solves the problem of drift completely, has failed to overcome the impact of the defect due to the re-training on the defective data.

For the practitioner, the marginal cost of incorporating remediation and threshold governance into the existing monitoring stack is relatively small, at around 12 seconds per batch, but the reliability improvement is significant. For the researcher, evaluating drift detectors in a stand-alone fashion could be overly optimistic about the difficulty of the real problem, since remediation was never even considered.

6.6 Threats to Validity

A number of limitations apply to the above results. For one, the injection procedure, which is unavoidable in order to create the ground truth change points, is subject to an underlying structure which the natural drift might not have, and the thresholds have been defined on the basis of the warm-up phase in which the algorithm was tested without any further knowledge about whether the same will apply to other domains. Second, all datasets used are tabular in nature and

all the models are regular scikit-learn classifiers. The performance of the aggregate score function in high-dimensional and unstructured data, for which the per-feature statistics make no sense, remains unclear, and the dimensionality reduction technique from [29] would be needed here.

7. Conclusion and Future Work

In this paper, we have introduced a fully automated, closed loop pipeline for data quality and drift mitigation in production machine learning pipelines. This includes declarative data quality checks using Great Expectations, statistical estimation of drift using Evidently AI, an importance weighted drift score with tiered threshold policy, automated drift mitigation using imputation, outlier management, and duplicate removal, dual persistence to Parquet and PostgreSQL, tiered alerting based on drift severity, and orchestrating retrain using Apache Airflow behind champion challenger gates. On three drifted datasets and two synthetic data generators, the proposed pipeline was able to detect drift with 94.2 percent accuracy and F1 of 0.931 in a delay of 2.4 batches, took a median time of 36.2 seconds to process each 50,000 row batch, increased the model F1 score by 0.054 to 0.103 just through remediation, lowered the peak accuracy drop due to drift from 16.7 to 3.9 percentage points compared to static models, and issued 33.3 percent fewer retraining tasks than a periodic approach without compromising on accuracy.

The core of the findings is that remediation and retraining are complementary tasks, not substitutes for one another. The former cannot fix defect-induced loss since it learns from the defects; the latter cannot fix distributional loss since the decision boundary is obsolete. The systems that automate one task alone suffer from the inherent blindness of the other, and the ablation shows what is lost in each case.

There are four possible research directions. First, substituting fixed thresholds by a label-free performance estimator will gate retraining based on predictions of loss of accuracy and not distribution drift, which is the most questionable assumption in the current work [24]. Second,

using a cost function in the spirit of Mahadevan and Mathioudakis will allow for optimization of the staleness versus computation tradeoff rather than tuning thresholds manually [26]. Third, using learned embeddings for composite score computation for non-tabular data will increase the range of applicability [29]. Fourth, incremental adaptation rather than complete retraining will decrease the cost of each trigger invocation and will allow a more reactive policy [26]. In addition, there are plans for research related to multi-tenant settings, when several models use a common reference store and for modeling the repair stage as a set of auditable contracts. The next step towards maturity in MLOps can come from the tighter incorporation of remediation and cost-aware policies [45].

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, doi: 10.1109/ACCESS.2023.3262138.
- [2]B. Eken, S. Pallewatta, N. K. Tran, A. Tosun, and M. A. Babar, "A Multivocal Review of MLOps Practices, Challenges and Open Issues," *ACM Computing Surveys*, 2025, doi: 10.1145/3747346. (Preprint: arXiv:2406.09737.)
- [3]M. Testi, M. Ballabio, E. Frontoni, G. Iannello, S. Moccia, P. Soda, and G. Vessio, "MLOps: A Taxonomy and a Methodology," *IEEE Access*, vol. 10, pp. 63606–63618, 2022, doi: 10.1109/ACCESS.2022.3181730.
- [4]M. Zarour, H. Alzabut, and K. T. Al-Sarayreh, "MLOps best practices, challenges and maturity models: A systematic literature review," *Information and Software Technology*, vol. 183, art. 107733, 2025, doi: 10.1016/j.infsof.2025.107733.
- [5]D. Sculley, G. Holt, D. Golovin, E. Davydov, T. Phillips, D. Ebner, V. Chaudhary, M. Young, J.-F. Crespo, and D. Dennison, "Hidden Technical Debt in Machine Learning Systems," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 28, 2015, pp. 2503–2511.
- [6]N. Sambasivan, S. Kapania, H. Highfill, D. Akrong, P. Paritosh, and L. M. Aroyo, "Everyone wants to do the model work, not the data work': Data Cascades in High-Stakes AI," in *Proc. CHI Conf. Human Factors in Computing Systems (CHI '21)*, 2021, art. 39, pp. 1–15, doi: 10.1145/3411764.3445518.
- [7]A. R. Munappy, J. Bosch, H. H. Olsson, A. Arpteg, and B. Brinne, "Data management for production quality deep learning models: Challenges and solutions," *Journal of Systems and Software*, vol. 191, art. 111359, 2022, doi: 10.1016/j.jss.2022.111359.
- [8]J. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, and A. Bouchachia, "A survey on concept drift adaptation," *ACM Computing Surveys*, vol. 46, no. 4, art. 44, 2014, doi: 10.1145/2523813.
- [9]J. Lu, A. Liu, F. Dong, F. Gu, J. Gama, and G. Zhang, "Learning under Concept Drift: A Review," *IEEE Transactions on Knowledge and Data Engineering*, vol. 31, no. 12, pp. 2346–2363, 2019, doi: 10.1109/TKDE.2018.2876857.
- [10]R. N. Gemaque, A. F. J. Costa, R. Giusti, and E. M. dos Santos, "An overview of unsupervised drift detection methods," *WIREs Data Mining and Knowledge Discovery*, vol. 10, no. 6, e1381, 2020, doi: 10.1002/widm.1381.
- [11]S. Mohammed, L. Budach, M. Feuerpfeil, N. Ihde, A. Nathansen, N. Noack, H. Patzlaff, F. Naumann, and H. Harmouch, "The effects of data quality on machine learning performance on tabular data," *Information Systems*, vol. 132, art. 102549, 2025, doi: 10.1016/j.is.2025.102549.

- [12]S. Shankar and A. G. Parameswaran, "Towards Observability for Production Machine Learning Pipelines," *Proc. VLDB Endowment*, vol. 15, no. 13, 2022, doi: 10.14778/3565838.3565853.
- [13]E. Breck, N. Polyzotis, S. Roy, S. E. Whang, and M. Zinkevich, "Data Validation for Machine Learning," in *Proc. Machine Learning and Systems (MLSys)*, vol. 1, 2019.
- [14]N. Polyzotis, M. Zinkevich, S. Roy, E. Breck, and S. Whang, "TensorFlow Data Validation: Data Analysis and Validation in Continuous ML Pipelines," in *Proc. ACM SIGMOD Int. Conf. Management of Data*, 2020, pp. 2793–2796, doi: 10.1145/3318464.3384707.
- [15]S. Schelter, D. Lange, P. Schmidt, M. Celikel, F. Biessmann, and A. Grafberger, "Automating Large-Scale Data Quality Verification," *Proc. VLDB Endowment*, vol. 11, no. 12, pp. 1781–1794, 2018, doi: 10.14778/3236187.3236200.
- [16]A. Bifet and R. Gavaldà, "Learning from Time-Changing Data with Adaptive Windowing," in *Proc. SIAM Int. Conf. Data Mining (SDM)*, 2007, pp. 443–448, doi: 10.1137/1.9781611972771.42.
- [17]J. Gama, P. Medas, G. Castillo, and P. Rodrigues, "Learning with Drift Detection," in *Advances in Artificial Intelligence (SBIA 2004)*, LNCS vol. 3171, 2004, pp. 286–295, doi: 10.1007/978-3-540-28645-5_29.
- [18]M. Baena-García, J. del Campo-Ávila, R. Fidalgo, A. Bifet, R. Gavaldà, and R. Morales-Bueno, "Early Drift Detection Method," in *Proc. 4th Int. Workshop on Knowledge Discovery from Data Streams*, vol. 6, 2006, pp. 77–86.
- [19]E. S. Page, "Continuous Inspection Schemes," *Biometrika*, vol. 41, no. 1/2, pp. 100–115, 1954, doi: 10.1093/biomet/41.1-2.100.
- [20]I. Frías-Blanco, J. del Campo-Ávila, G. Ramos-Jiménez, R. Morales-Bueno, A. Ortiz-Díaz, and Y. Caballero-Mota, "Online and Non-Parametric Drift Detection Methods Based on Hoeffding's Bounds," *IEEE Transactions on Knowledge and Data Engineering*, vol. 27, no. 3, pp. 810–823, 2015, doi: 10.1109/TKDE.2014.2345382.
- [21]C. Raab, M. Heusinger, and F.-M. Schleif, "Reactive Soft Prototype Computing for Concept Drift Streams," *Neurocomputing*, vol. 416, pp. 340–351, 2020, doi: 10.1016/j.neucom.2019.11.111.
- [22]T. Rekatsinas, X. Chu, I. F. Ilyas, and C. Ré, "HoloClean: Holistic Data Repairs with Probabilistic Inference," *Proc. VLDB Endowment*, vol. 10, no. 11, pp. 1190–1201, 2017, doi: 10.14778/3137628.3137631.
- [23]M. Mahdavi and Z. Abedjan, "Baran: Effective Error Correction via a Unified Context Representation and Transfer Learning," *Proc. VLDB Endowment*, vol. 13, no. 12, pp. 1948–1961, 2020, doi: 10.14778/3407790.3407801.
- [24]J. Kivimäki, J. Białek, W. Kuberski, and N. Perrakis, "Estimating Model Performance Under Covariate Shift Without Labels," *arXiv preprint arXiv:2401.08348*, 2024.
- [25]L. Poenaru-Olaru, L. Cruz, A. van Deursen, and J. S. Rellermeier, "Prepared for the Unknown: Adapting AIOps Capacity Forecasting Models to Data Changes," *arXiv preprint arXiv:2510.10320*, 2025.
- [26]A. Mahadevan and M. Mathioudakis, "Cost-Effective Retraining of Machine Learning Models," *arXiv preprint arXiv:2310.04216*, 2023.
- [27]Great Expectations, "Great Expectations documentation: Expectation suites, checkpoints and data docs," v0.18, 2024. [Online]. Available: <https://docs.greatexpectations.io>
- [28]D. Zha, Z. P. Bhat, K.-H. Lai, F. Yang, Z. Jiang, S. Zhong, and X. Hu, "Data-centric Artificial Intelligence: A Survey," *ACM Computing Surveys*, 2023. (Preprint: arXiv:2303.10158.)

- [29] S. Rabanser, S. Günnemann, and Z. C. Lipton, "Failing Loudly: An Empirical Study of Methods for Detecting Dataset Shift," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 32, 2019, pp. 1394–1406.
- [30] A. Gretton, K. M. Borgwardt, M. J. Rasch, B. Schölkopf, and A. Smola, "A Kernel Two-Sample Test," *Journal of Machine Learning Research*, vol. 13, no. 25, pp. 723–773, 2012.
- [31] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation Forest," in *Proc. IEEE Int. Conf. Data Mining (ICDM)*, 2008, pp. 413–422, doi: 10.1109/ICDM.2008.17.
- [32] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation-Based Anomaly Detection," *ACM Transactions on Knowledge Discovery from Data*, vol. 6, no. 1, art. 3, 2012, doi: 10.1145/2133360.2133363
- [33] Evidently AI, "Evidently: An open-source framework to evaluate, test and monitor ML models in production," software, v0.4, 2024. [Online]. Available: <https://github.com/evidentlyai/evidently>
- [34] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and É. Duchesnay, "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.
- [35] M. Harries, "SPICE-2 Comparative Evaluation: Electricity Pricing," Technical Report UNSW-CSE-TR-9905, University of New South Wales, 1999.
- [36] Apache Software Foundation, "Apache Airflow documentation," v2.9, 2024. [Online]. Available: <https://airflow.apache.org/docs/>
- [37] A. Mallick, K. Hsieh, B. Arzani, and G. Joshi, "Matchmaker: Data Drift Mitigation in Machine Learning for Large-Scale Systems," in *Proceedings of Machine Learning and Systems (MLSys)*, 2022.
- [38] R. K. Tanna, "Self-Healing ML Pipelines: Automating Drift Detection and Remediation," *Preprints.org*, 2025.
- [39] N. Kodakandla, "Data Drift Detection and Mitigation: A Comprehensive MLOps Approach for Real-Time Systems," *International Journal of Scientific Research and Analysis*, 2024.
- [40] J. C. Sisniega et al., "Efficient and Scalable Covariate Drift Detection in Machine Learning Systems," *Future Generation Computer Systems*, 2024.
- [41] M. M. John et al., "An Empirical Guide to MLOps Adoption," *Information and Software Technology*, 2025.
- [42] Y. Li et al., "Maturity Framework for Operationalizing Machine Learning," 2025.
- [43] N. Jourdan et al., "Handling Concept Drift in Deep Learning Applications for Industrial Prognostics," *Procedia CIRP*, vol. 118, 2023.
- [44] et al., "Navigating MLOps: Insights into Maturity, Lifecycle, Tools and LLMOps," *arXiv:2503.15577*, 2025.
- [45] M. M. John et al., "Towards MLOps: A Framework and Maturity Model," 2021.