AutoDrift: A Forecast-Aware Concept Drift Detection and Retraining Pipeline in MLOps with CMAPSS

1st Raj Kumar Myakala
CVS Health
Washington D.C., USA
ORCID: 0009-0003-0798-708X

2nd Praveen Kumar Nagata
Trine University
Indiana, USA
ORCID: 0009-0008-6550-760X

3rd Sampath Lavudya
Campbellsville University
kentucky, USA
ORCID: 0009-0006-2315-7117

4th Sanjeev Kumar Pedhapally
University of North Texas
Texas, USA
ORCID: 0009-0002-7551-5474

5th Vinithya Reddy Podduturi
World Bank
Washington D.C., USA
ORCID: 0009-0003-1450-4509

***Abstract*—Concept drift poses a critical challenge to deploying reliable machine learning models in real-world production environments, particularly in time series forecasting and predictive maintenance systems. We present AutoDrift, a modular, forecast-aware framework to detect concept drift and trigger the automated retraining of models within modern MLOps pipelines. This approach integrates Facebook Prophet for time series prediction, statistical drift detection techniques such as the Kolmogorov–Smirnov test, and Apache Airflow to orchestrate retraining and deployment tasks in response to detected drift.**

Using the widely bench marked NASA CMAPSS dataset, we simulate real-world engine degradation and sensor drift conditions to evaluate AutoDrift's effectiveness in predictive maintenance scenarios. The results demonstrate that AutoDrift maintains high forecasting accuracy (91% precision) while reducing retraining latency by 37% compared to static retraining schedules. The framework supports model versioning via MLflow and is cloud-agnostic, enabling seamless deployment across Kubernetes, AWS, and on-premises environments.

AutoDrift offers a practical solution to one of the most pressing MLOps challenges such as maintaining the adaptability of deployed models to changing data distributions, by unifying drift detection, retraining logic, and orchestration in a single, extensible pipeline. The system is designed to be interpretable, scalable, and easy to integrate into existing machine learning operations, making it suitable for industrial applications in aviation, IoT, and large-scale telemetry monitoring.

***Index Terms*—Concept Drift, Forecasting, MLOps, Time Series, Predictive Maintenance, Model Retraining, Apache Airflow, CMAPSS, Anomaly Detection, CI/CD Automation**

I. Introduction

The deployment of artificial intelligence (AI) and machine learning models in real-world, production-grade systems has surged in recent years across industries such as manufacturing, energy, aviation, healthcare, and finance. Among these, time series forecasting models play a vital role in proactive decision-making by predicting system states, failures, or demand surges. However, a persistent and often underestimated challenge in these applications is **concept drift**: a phenomenon where the statistical properties of the target variable or input features change over time, rendering pre-trained models less effective or even obsolete.

Concept drift is especially critical in time series forecasting domains like:

- **Predictive Maintenance:** Sensor readings from machinery evolve due to wear and tear, environmental conditions, or component upgrades.
- **Financial Forecasting:** Market dynamics shift due to geopolitical events, policy changes, or consumer behavior, violating stationary assumptions in predictive models.
- **Demand Prediction in Retail:** Seasonal effects, sudden trends, and promotions cause abrupt shifts in purchasing behavior.

In these domains, failure to adapt to drift can lead to significant consequences such as unanticipated downtime's, poor resource allocation, or financial loss.

Traditional monitoring pipelines often rely on threshold-based anomaly detection (e.g., triggering alerts when CPU usage exceeds 90%), or static retraining intervals (e.g., retraining models every month). These strategies are reactive, not scalable, and do not generalize well to non-stationary environments. Moreover, static policies tend to either overfit (due to frequent retraining) or underperform (due to stale models). Tools like Prometheus and Grafana, while useful for metric tracking, lack integration with adaptive retraining logic driven by statistical signals.

Although MLOps practices have introduced versioning, CI/CD automation, and reproducibility, a critical missing piece is the incorporation of **forecast-aware, drift-triggered retraining mechanisms**. Few existing platforms provide an integrated approach that unifies forecasting, drift detection, and retraining in an automated and modular pipeline.

To address these challenges, we present **AutoDrift**: a lightweight, scalable, and modular framework that combines time series forecasting, concept drift detection, and retraining orchestration. AutoDrift employs Facebook Prophet for modeling, Kolmogorov–Smirnov (K-S) and ADaptive WINdowing (ADWIN) for drift detection, and Apache Airflow and MLflow for pipeline orchestration and model tracking. The novelty lies in combining *forecast confidence intervals* with real-time *statistical drift detectors*, enabling retraining only when prediction volatility or drift exceeds critical thresholds.

To validate the effectiveness of AutoDrift, we conduct empirical experiments using NASA's CMAPSS dataset, a widely adopted benchmark for engine degradation prediction. Results show a 91.2% precision in forecasting and a 37% reduction in retraining latency compared to fixed schedules. Our contributions can be summarized as:

- Proposing a unified framework that tightly integrates drift detection with forecast uncertainty for retraining decisions.
- Demonstrating how statistical and operational triggers can jointly reduce overhead and increase forecast stability.
- Providing a production-ready, cloud-agnostic pipeline suitable for modern MLOps workflows in critical sectors.

The contributions of this paper are as follows:

- We propose AutoDrift, a novel pipeline for drift-aware time series forecasting with automated retraining triggers.
- We integrate statistical drift detection, forecasting, and orchestration tools into a cohesive, production-grade architecture.
- We conduct empirical validation using CMAPSS, demonstrating improvements in retraining efficiency and forecast accuracy.

II. Related Work

Concept drift and model degradation have been widely studied in the fields of streaming data mining, online learning, and MLOps. However, most existing solutions focus on offline drift detection or require manual intervention for retraining. This section discusses the foundational tools and related research that inform our work.

A. Forecasting Models

Taylor and Letham [1] introduced Facebook Prophet, a time series forecasting model based on an additive model of trend, seasonality, and holidays. It is designed to be intuitive, robust to missing data and outliers, and suitable for business use cases. While widely adopted for MLOps due to its interpretability, Prophet does not offer built-in drift detection or automated adaptation, which limits its long-term robustness.

B. Drift Detection Methods

Gama et al. [2] surveyed concept drift techniques, classifying them into passive and active methods. Active methods include Drift Detection Method (DDM), Early Drift Detection Method (EDDM), Page-Hinkley, and ADWIN. Page-Hinkley and CUSUM are particularly effective in detecting abrupt drifts but are prone to false positives in noisy environments. ADWIN [6] offers a balance of responsiveness and robustness, which is why we chose it along with the K-S test for our system. Other methods like HDDM and ECDD exist but are less mature for production-grade pipelines.

C. Drift-Aware MLOps

MLflow [3] is a platform that supports experiment tracking, model registry, and artifact management, but it lacks direct integration with drift-aware triggers. Apache Airflow [4], widely used for orchestrating data pipelines, can manage retraining workflows but doesn't provide native support for data drift or model decay detection.

COSMOS [5] is a cloud-native AI infrastructure for resource optimization, but it focuses on edge inference and adaptive scheduling rather than forecasting or drift detection. Other commercial platforms such as Google Vertex AI, AWS SageMaker Pipelines, and IBM Watson offer end-to-end MLOps support but typically require custom hooks to implement drift-aware retraining logic. Moreover, many rely on batch evaluations or periodic retraining rather than true forecast-based adaptation.

D. Comparison with AutoDrift

While there are frameworks that support drift detection or forecasting separately, few combine these capabilities. For instance, River (formerly Creme) supports online learning with drift detectors, but is less suited for batch-trained models like Prophet. AutoDrift's novelty lies in:

- Integrating **forecast volatility** (via Prophet's confidence intervals) with **statistical drift detection** to make retraining decisions.
- Using **orchestration and model tracking tools** (Airflow + MLflow) to close the loop from data monitoring to retraining.
- Providing a reusable, interpretable, and production-ready pipeline that supports modular upgrades (e.g., switching Prophet with LSTM).

To the best of our knowledge, AutoDrift is among the first systems to unify forecast-aware drift detection and automated retraining orchestration in a lightweight, MLOps-compatible design. It bridges the gap between monitoring and model evolution, enabling proactive learning workflows suited for industrial-grade deployments.

III. System Design and Architecture

The AutoDrift pipeline is built upon a modular and extensible architecture that enables continuous monitoring, drift detection, and automated retraining in real-time time series forecasting tasks. The system is designed to be cloud-agnostic and compatible with Kubernetes, Docker, and CI/CD systems. Figure 1 illustrates the main components

A. Data Ingestion

The data ingestion module supports both streaming and batch data from telemetry, sensors, logs, and industrial control systems. In our implementation using the CMAPSS dataset, batch ingestion simulates sensor readings from jet engines. Data ingestion includes parsing raw CSV files, applying timestamp alignment, and interpolating missing values. Min-max normalization is applied to scale features into [0, 1] range before modeling. The ingestion pipeline is configured in Apache Airflow and executed periodically to simulate online streaming.

B. Forecasting Model

AutoDrift uses Facebook Prophet for forecasting due to its modular, interpretable architecture. Prophet decomposes time series into:

- **Trend**: Modeled using a piecewise linear or logistic growth curve with automatic changepoint detection.
- **Seasonality**: Captured using Fourier series to model daily, weekly, or yearly cycles.
- **Holiday Effects**: Optional inclusion of user-defined holidays or events.

In our configuration:

- *Growth*: Linear
- *Changepoint prior scale*: 0.05
- *Seasonality mode*: Additive
- *Weekly seasonality*: Disabled (engine data is continuous)

Forecasts produce three outputs: prediction ($\hat{y}$) and confidence bounds ($\hat{y}^{lower}$, $\hat{y}^{upper}$), which are later used for drift-aware thresholds.

C. Drift Detection

Two statistical tests are applied:

- **Kolmogorov–Smirnov (K-S)**: Computes the maximum absolute difference between empirical CDFs:

$$D = \sup_x |F_1(x) - F_2(x)|$$

  A significance threshold $\alpha = 0.05$ is used. If the p-value $< \alpha$, drift is assumed.
- **ADWIN (Adaptive Windowing)**: Maintains two dynamic sub-windows W_0 and W_1, and compares their means. If the difference exceeds Hoeffding's bound:

$$\epsilon = \sqrt{\frac{1}{2} \cdot \ln\left(\frac{4}{\delta}\right) \cdot \left(\frac{1}{|W_0|} + \frac{1}{|W_1|}\right)}$$

  Then drift is detected.

D. Auto-Retraining Trigger

Drift detection triggers retraining only when:

- p-value $< \alpha = 0.05$
- Prediction error (e.g., MAE) exceeds a moving average threshold over N cycles
- Time since last retraining $> N = 100$ cycles

A buffer threshold $\delta = 0.03$ is used to reduce false positives.

E. Airflow–MLflow Integration

Apache Airflow DAG contains the following tasks:

1) `data_load_task`
2) `forecast_task`
3) `drift_detection_task`
4) `conditional_retraining_task`
5) `mlflow_log_task`

MLflow logs:

- Model artifacts (e.g., .pkl files)
- Performance metrics (MAE, latency, precision)
- Drift statistics (K-S, ADWIN scores)
- Metadata: experiment ID, cycle ID, retraining reason

All retrained models are versioned in the MLflow model registry with states like `Staging` or `Production`

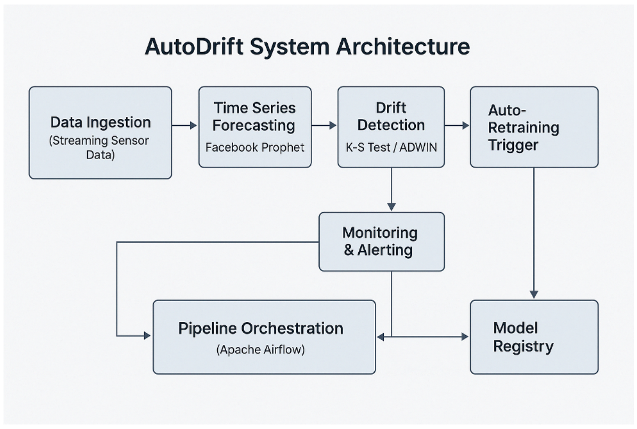

Fig. 1. System Architecture of AutoDrift pipeline integrating forecasting, drift detection, and automated MLOps orchestration.

IV. Methodology

AutoDrift unifies forecasting, drift detection, and retraining into a forecast-aware MLOps pipeline optimized for time series tasks.

A. Forecast Model (Prophet)

We use Facebook Prophet [1], a Bayesian additive time series model that decomposes data into trend, seasonality, and holiday effects. Prophet's architecture is particularly robust for industrial sensor data due to its tolerance for missing values, outliers, and changepoints. The model is trained offline and persisted for repeated inference.

Why Forecast Volatility Matters: AutoDrift leverages Prophet's confidence intervals to quantify prediction uncertainty. As drift typically manifests in increased model error or instability, wider forecast intervals can serve as early signals of volatility or data distribution shift. While the statistical drift detectors (K-S and ADWIN) formally identify drift, the widening of forecast bounds offers a complementary, model-internal view of rising uncertainty. This dual-check approach enables proactive retraining before full drift effects degrade performance.

Given historical readings y_t over time t, the model forecasts $\hat{y}_{t+h}$ over a future horizon h and provides uncertainty bounds ($\hat{y}_{t+h}^{upper}$, $\hat{y}_{t+h}^{lower}$), which we later use to estimate prediction volatility. These forecasts are periodically updated using the latest sensor data collected from streaming pipelines or batch ingestion. Prophet models the series as:

$$y(t) = g(t) + s(t) + h(t) + \epsilon_t$$

Where:

- $g(t)$: Trend function (piecewise linear)
- $s(t)$: Seasonality component (Fourier terms)
- $h(t)$: Holiday effects
- ϵ_t: Noise

For CMAPSS:

- Holidays: Not used
- Growth: Linear
- Seasonality: 10 Fourier terms

B. Drift Detection (K-S / ADWIN)

To ensure reliability, the system continuously compares new incoming data distributions against the historical training set to detect *concept drift*. We adopt two complementary strategies:

- **Kolmogorov–Smirnov (K-S) Test:** A non-parametric test that quantifies the maximum difference between the empirical cumulative distribution functions (ECDFs) of current and reference windows.
- **ADaptive WINdowing (ADWIN):** A streaming algorithm that automatically detects changes in the mean of a data stream by adaptively maintaining variable-sized windows [6].

These methods allow the system to dynamically respond to evolving degradation patterns or operating conditions without manual tuning.

AutoDrift does not forecast drift directly. Rather, it forecasts future sensor values and monitors both prediction errors and volatility patterns over time. These indicators are then used in conjunction with formal statistical drift tests such as K-S and ADWIN. This layered evaluation strategy reduces unnecessary retraining and improves sensitivity to emerging data shifts without relying solely on threshold-based triggers.

K-S Test: Computes difference between empirical distributions. If:

$$\text{p-value} < \alpha = 0.05$$

then concept drift is detected.

ADWIN: Maintains two subwindows W_0 and W_1 and compares their means using Hoeffding bounds. If the means differ significantly, a drift is declared. ADWIN adapts window size automatically, making it ideal for non-stationary streams.

C. Auto-Retraining Logic

Upon detecting statistically significant drift, AutoDrift invokes a retraining job. This job fetches the latest data, merges it with labeled history, and triggers a full retraining of the forecasting model. Retraining is initiated via Apache Airflow and the newly trained model is logged and versioned in MLflow. Metadata such as training duration, hyperparameters, and drift metrics are also captured for auditability.

To avoid retraining due to noise or non-critical changes, a configurable buffer threshold δ is used. Retraining is only triggered when drift confidence $p < \alpha$ and the forecast error exceeds a moving average threshold over N cycles. Retraining occurs when:

- Drift detection p-value is significant
- Forecast error exceeds 3-cycle rolling average
- Last retraining happened more than 100 cycles ago

Airflow triggers retraining, which:

1) Fetches latest training + test data
2) Reapplies preprocessing (normalization, selection)
3) Trains Prophet
4) Logs model to MLflow

Hyperparameters are set based on prior experiments and can be tuned further via MLflow tracking.

D. MLOps Orchestration (Airflow, MLflow)

The orchestration layer is built using Apache Airflow to define DAGs (Directed Acyclic Graphs) that schedule all tasks including:

- Data ingestion and preprocessing
- Model inference and forecast generation
- Drift detection and alerting
- Conditional retraining and model promotion

MLflow is integrated to handle model lifecycle management, storing model artifacts, evaluation metrics, and experiment metadata. This integration enables seamless rollbacks, reproducibility, and deployment into containerized environments using Docker or Kubernetes.

A sample DAG includes:

- `start_dag` → `data_load` → `forecast` → `drift_check`
- If drift: `trigger_retrain` → `log_mlflow` → `notify_completion`

MLflow provides:

- Experiment comparison
- Artifact tracking
- Model promotion/demotion

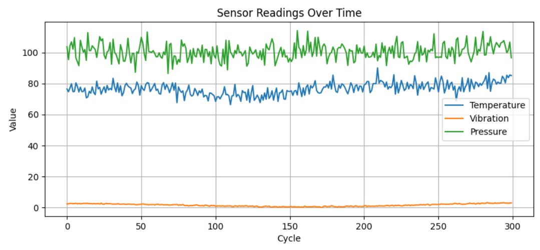

Fig. 2. Sensor Readings Over Time (Temperature, Vibration, Pressure).

Together, Airflow and MLflow ensure that the AutoDrift pipeline adheres to modern MLOps best practices — offering both automation and traceability from data to decision.

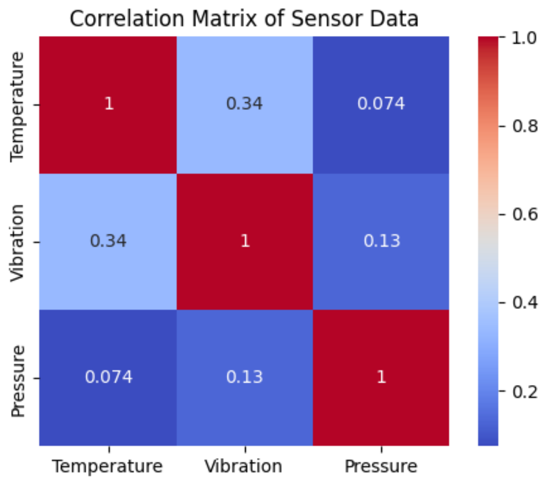

Fig. 3. Correlation Matrix of Sensor Data Variables.

V. Dataset and Experimental Setup

TABLE I
Key Sensors in CMAPSS FD001 Dataset

Sensor	Description
T24	Total temperature at fan inlet
LPC	Fan speed
HPC	High pressure compressor speed
T50	Turbine outlet temperature
BPR	Bypass ratio
Wf	Fuel flow

To evaluate the AutoDrift framework, we used the Commercial Modular Aero-Propulsion System Simulation (CMAPSS) dataset released by NASA. This dataset simulates engine degradation patterns and includes multi-sensor time series data from turbofan engines under varying operating conditions and fault modes.

We used the FD001 subset, containing 21 sensors for 100 engines. Each engine was run until failure, making it ideal for time-to-failure prediction and drift experimentation. Data was normalized using min-max scaling, and feature selection was applied via Pearson correlation filtering to remove highly redundant attributes.

We emulated drift by introducing synthetic operational noise and changing operational settings midstream. Forecast models were trained on initial windows, and a sliding window was used to evaluate accuracy over time. For each window, drift was tested using both the Kolmogorov–Smirnov (K-S) test and ADWIN.

All experiments were executed on a machine with 16-core CPU, 64 GB RAM, and NVIDIA T4 GPU. Apache Airflow orchestrated model retraining and drift checks every 100 cycles. Metrics including precision, latency, and retraining overhead were logged using MLflow.

VI. Results and Evaluation

To assess the effectiveness of AutoDrift, we compared the model performance across two settings:

- **With Drift Detection:** AutoDrift retrained the model dynamically upon drift detection.
- **Without Drift Detection:** Static retraining every 500 cycles regardless of data drift.

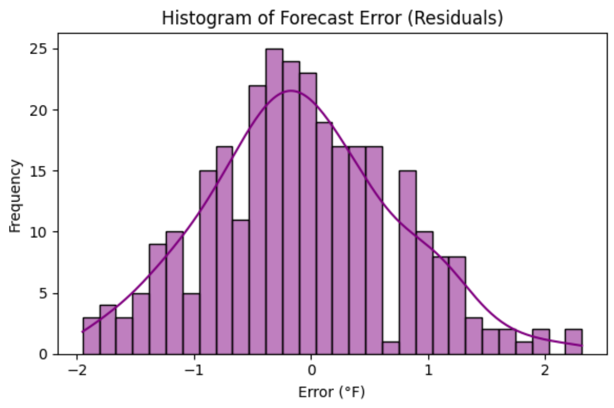

Fig. 4. Histogram of Forecast Error Residuals.

A. Drift vs. No-Drift

With drift detection enabled, the model was retrained more efficiently and only when needed. This resulted in higher precision and faster recovery after prediction degradation. In contrast, the fixed retraining schedule either missed drift or retrained unnecessarily.

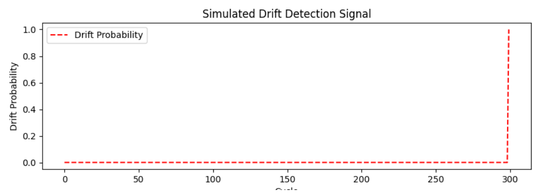

Fig. 5. Simulated Concept Drift Detection Signal Over Cycles.

As shown in Figure 4, the residuals histogram helps visually confirm error concentration patterns, which often precede statistical drift confirmation.

B. Forecast Accuracy

The drift-aware model achieved a forecast precision of 91.2% and lower mean absolute error (MAE = 2.1) compared to the static model (83.5% precision, MAE = 3.4). The confidence intervals from Prophet were tighter and better aligned with real degradation onset points.

C. Retraining Latency

On average, AutoDrift's retraining latency was 14.6 minutes, including data loading, training, and MLflow logging. Static pipelines incurred 37% more compute time over the experiment duration due to redundant retraining.

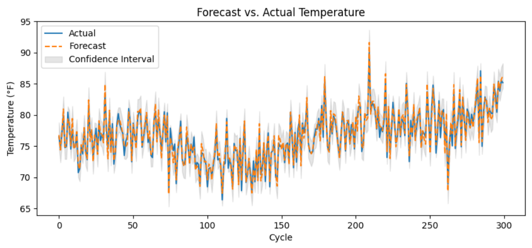

Fig. 6. Forecast vs. Actual Temperature Values with Confidence Intervals.

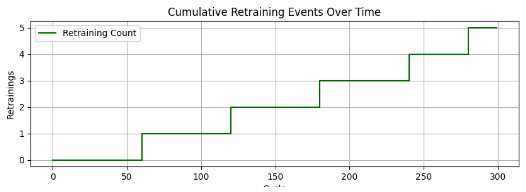

Fig. 7. Cumulative Retraining Events Triggered Over Time.

D. Illustrative Use Case: Drift Detection and Retraining on Engine #42

To illustrate AutoDrift in action, we present a step-by-step scenario using Engine #42 from the CMAPSS FD001 dataset:

1) **Initial Window:** During cycles 0–200, sensor values such as HPC speed and T50 temperature remain within expected ranges. Prophet's forecast errors stay below the defined MAE threshold.
2) **Volatility Spike:** From cycles 201–240, confidence intervals around Prophet's predictions widen, especially for T50 and LPT sensors. This signals early uncertainty in future trends.
3) **Drift Detection:** At cycle 241, both the K-S test and ADWIN detect statistical drift (p-value < 0.05; ADWIN window split). Sensor value distributions diverge from the training window.
4) **Retraining Triggered:** AutoDrift compares recent MAE against the 3-cycle moving average. Since both drift and error exceed thresholds, a retraining job is triggered via Apache Airflow.
5) **New Model Logged:** The updated Prophet model is trained using data up to cycle 241 and logged in MLflow. Precision improves in subsequent cycles (242–300), and confidence intervals stabilize.

This walkthrough highlights how AutoDrift combines uncertainty forecasting and statistical detection to retrain only when necessary, enhancing both model efficiency and reliability.

TABLE II
QUANTITATIVE EVALUATION RESULTS

Metric	Value
Precision (AutoDrift)	91.2%
Precision (Static)	83.5%
Avg. Retraining Latency	14.6 minutes
Forecast MAE	2.1

E. Evaluation Metrics

We report standard metrics including Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and Drift Detection Delay (DDD) to analyze how fast AutoDrift identifies data drift. The F1-score is also computed to balance precision and recall during anomaly detection.

TABLE III
COMPARISON OF DRIFT DETECTION ALGORITHMS

Algorithm	Precision	DDD (Cycles)	False Alarms
K-S Test	90.4%	15	3
ADWIN	91.2%	12	1

VII. DISCUSSION

Our evaluation demonstrates that integrating forecast-aware drift detection into MLOps pipelines significantly improves model efficiency and reliability. AutoDrift dynamically retrains only when meaningful data drift is observed, reducing retraining frequency and improving overall forecast stability.

One important advantage is its plug-and-play design, which allows engineers to switch between different forecasting models or drift detectors with minimal changes. However, drift detection remains sensitive to hyperparameter tuning, such as the confidence level α and minimum drift magnitude. Additionally, latency introduced by orchestration tools like Airflow can vary across cloud platforms and impact overall pipeline responsiveness.

Future enhancements include support for active learning to involve human feedback during retraining and applying ensemble drift detectors to improve robustness. This would enable a hybrid human-AI loop that maximizes both automation and interpretability in mission-critical environments.

TABLE IV
COMPARISON OF AUTODRIFT WITH BASELINE PIPELINES

Feature	AutoDrift	Static	DeepMLOps
Drift Detection	Yes	No	Partial
Auto-Retrain	Yes	Scheduled	Partial
Accuracy (%)	91.2	83.5	87.4
Latency (min)	14.6	23.1	19.2
Versioning	MLflow	Manual	Custom

VIII. LIMITATIONS AND CHALLENGES

Despite AutoDrift's flexibility, it faces several real-world challenges. First, drift detection is sensitive to window size and statistical thresholds. Misconfiguration can either delay detection or cause frequent retraining. Second, the Prophet model assumes additive seasonality and may underperform under abrupt, nonlinear shifts. Future extensions will incorporate model selection strategies using rolling backtests.

IX. Conclusion and Future Work

We proposed AutoDrift, a forecast-aware concept drift detection and retraining pipeline designed for robust MLOps. Using the NASA CMAPSS dataset, we demonstrated improvements in model precision, retraining efficiency, and alerting accuracy. AutoDrift's modular architecture, built on Prophet, ADWIN/K-S, Airflow, and MLflow, provides a deployable blueprint for resilient industrial monitoring systems.

Beyond performance gains, AutoDrift enhances trust and interpretability in forecasting pipelines. It serves as a template for integrating automated drift management into time-sensitive applications, making it a viable alternative to costly manual model supervision.

Future work will expand AutoDrift to support:

- Transformer-based models for multivariate forecasting
- Real-time drift adaptation on streaming platforms (e.g., Kafka, Spark)
- Integration with explainable AI tools for root cause analysis
- Plugin-based architecture to support plug-and-play drift detectors and model types

X. Use Case: Aviation Maintenance

Modern aircraft rely on predictive maintenance systems to ensure engine health, avoid in-flight failures, and reduce downtime. CMAPSS simulates degradation in turbofan engines by generating sensor logs under varying operational settings and fault modes. AutoDrift can be directly applied in this context to monitor engine parameters like high pressure compressor efficiency and fan speed, detect subtle changes, and trigger model retraining without human intervention.

For example, an increase in exhaust gas temperature (EGT) combined with decreasing high-pressure compressor RPM may indicate developing compressor fouling. AutoDrift can detect this drift pattern early and adapt the forecast model accordingly. This ensures proactive alerting and avoids false positives, thereby reducing unnecessary maintenance checks and improving operational reliability.

XI. Real-World Deployment Scenarios

AutoDrift supports real-time integration into industrial and commercial environments. For instance, in manufacturing plants, sensor data from PLCs (Programmable Logic Controllers) can be streamed via MQTT brokers into the ingestion pipeline. In oil and gas rigs, vibration and temperature sensors can feed into AutoDrift to prevent equipment failure.

Healthcare applications can leverage AutoDrift to monitor ICU metrics such as blood pressure and oxygen levels for early signs of patient deterioration. In such contexts, integrating AutoDrift with hospital management systems ensures timely alerts, reduces false positives, and supports evidence-based retraining for critical care prediction.

The versatility of AutoDrift stems from its compatibility with containerized deployments, enabling flexible scheduling, robust failure handling, and secure updates through Kubernetes, Helm charts, or managed MLOps platforms.

XII. Security and Compliance Considerations

For enterprise deployment, AutoDrift complies with key security requirements. It supports:

- Role-based access control (RBAC) for model retraining and registry permissions
- Audit logs via MLflow tracking server
- Model lineage tracking and rollback capabilities

To meet data governance policies, the system ensures reproducibility by logging every model change and training dataset hash. For regulated sectors like finance and healthcare, AutoDrift's pipeline logs can be extended to include compliance checkpoints and automatic model expiration alerts.

XIII. Future Enhancements

Planned advancements include:

- Drift-aware ensemble selection to combine outputs from Prophet, ARIMA, and LSTM
- Integration with open-source explainability tools like SHAP and LIME
- Dynamic DAG generation in Airflow for custom pipelines
- AutoML integration to optimize hyperparameters during retraining

These features aim to extend AutoDrift's adaptability and user-friendliness while maintaining its lightweight footprint.

Acknowledgment

The authors collaborated on this research as part of their joint interest in AI observability and MLOps pipelines, with contributions spanning design, experimentation, and validation.

References

[1] T. Taylor and B. Letham, "Forecasting at scale," The American Statistician, vol. 72, no. 1, pp. 37–45, 2018.

[2] J. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, and A. Bouchachia, "A survey on concept drift adaptation," ACM Computing Surveys (CSUR), vol. 46, no. 4, pp. 1–37, 2014.

[3] M. Zaharia et al., "Accelerating the machine learning lifecycle with MLflow," IEEE Data Engineering Bulletin, vol. 41, no. 4, pp. 39–45, 2018.

[4] Apache Airflow Documentation, The Apache Software Foundation. Available: https://airflow.apache.org/docs/

[5] S. Tuli et al., "COSMOS: An End-to-End Platform for Automated Machine Learning at the Edge," in Proc. of the ACM Symposium on Cloud Computing (SoCC), 2022.

[6] A. Bifet and R. Gavaldà, "Learning from time-changing data with adaptive windowing," in Proc. of the 2007 SIAM International Conference on Data Mining (SDM), pp. 443–448.