A Self-Adaptive Autonomous Pipeline Optimization Framework for Data Quality, Drift Detection, Anomaly Monitoring, and Runtime Performance Improvement

[1]Nagarjuna Dorthala, [2]Satish Kumar Potru, [3]Nikhil Kumar Reddy Kommula, [4]Diviti Santhoshi

[1]Senior Data Engineer, Populus Group, USA- E-mail: dorthalanagarjuna.usa@gmail.com

[2]Data Engineer, Hexaware Technologies Inc., USA- E-mail: potrusatishkumar1998@gmail.com

[3]Senior Software Engineer, AVJ Solutions, USA- E-mail: nikhilkommula1897@gmail.com

[4]Data Engineer, Trailytics Global Services Pvt. Ltd., Gurugram, Haryana, India-

Email: divitisanthoshi@gmail.com.

Abstract

The data distribution, workload, and resource requirements are constantly evolving in modern data engineering pipelines, and static configurations are becoming less efficient over time. This study introduces the Self-Adaptive Autonomous Pipeline Optimization (SAAPO) framework, which is a closed-loop system consisting of adaptive preprocessing, data-quality assessment, statistical drift detection, anomaly identification, runtime monitoring, pipeline health evaluation, and adaptive reconfiguration based on feedback from the pipeline. The structure used for the KDD Cup 1999 was used for sequential batches, which were processed with Python, Pandas, NumPy, SciPy, Scikit-learn, psutil, and Matplotlib. The distributional drift was measured by Kolmogorov-Smirnov statistic, and the anomaly detection was done with Isolation Forest. The framework was dedicatedly designed to track the metrics such as throughput, latency, CPU utilization, memory usage, error rate, execution reliability and data quality and derive an integrated pipeline health score. The experimental results indicated that the processing efficiency and recovery of the SAAPO improved over the static and monitoring-only pipelines in various operating conditions. The prototype was able to deliver around 724,886 records/s throughput (16.3% for the static pipeline) and reduced latency, CPU usage, and drift. These results prove the viability of an autonomous feedback-guided optimization to stable and adaptive data pipelines.

Keywords: Self-Adaptive Data Pipelines, Pipeline Optimization, Data Drift Detection, Anomaly Detection, Autonomous Data Engineering.

1. Introduction

Today's data engineering pipelines often must continuously run when the distribution of data changes, workloads change, and the conditions of computing resources vary. Traditional static pipelines are typically designed with preconceived ideas about the nature of data and processing needs. These set-ups can work well if the statistical characteristics of the data being processed remain unchanged, but they may not work if the statistics of the data being received vary, the amount of anomalous data increases, or the processing requirement falls outside the range initially designed. This poses a requirement for a self-aware data-processing architectures that can monitor themselves and dynamically change pipeline behavior without manual effort.

The ability to detect anomalies is crucial for the effectiveness of intelligent monitoring systems, as it can help identify issues with data quality, system faults, attacks, or operational degradation. Xu et al. [1] introduced a model designed to capture association discrepancies, called Anomaly Transformer, for unsupervised anomaly detection on time-series data, and showed the need for capturing anomalous trends in temporal behaviour. Anomaly monitoring in real-time has also been studied in industrial settings. Velasco-Gallego and Lazakis [2] used deep learning for anomaly detection in real-

time marine machinery, showing the importance of considering operational data continuously, and not just regularly in the post-mission phase. Also, Liang et al. [3] explored unsupervised learning for anomaly detection of marine diesel engine by transformer-based modelling and residual analysis, which demonstrated that abnormal operating patterns can be identified even with limited failure data.

Hybrid anomaly-detection methods have also showcased the benefits of fusion of complementary methods. Haque and Soliman [4] combined a transformer-based autoencoder, with Isolation Forest and XGBoost, for Malfunction and Intrusion Detection. Such hybrid approaches provide examples of merging representation learning, unsupervised anomaly detection and predictive modelling to enhance monitoring performance. In the case of large-scale data pipelines, anomaly detection algorithms based on Isolation Forest can be used to identify unusual observations without the need for explicit class labels. To enhance the capability of isolation-based anomaly detection for complex and high-dimensional data, Xu et al. [5] further developed the concept into Deep Isolation Forest. The developments in this area justify the proposal to use Isolation Forest in the suggested framework as an approach that is feasible in terms of computational requirements for the detection of abnormal records during the sequential execution of the pipeline.

Besides anomalies, today's pipelines need to deal with the systematic shift in the data distribution. This type of changes happens often in systems that have concept drift or data drift, which means the statistical relationships exhibited by the past data are no longer valid for the newly observed data. Lu et al. [6] gave a thorough survey of the concept drift problems and pointed out that models for dynamic environments must be able to detect and adapt to concept drift. In a real MLOps setting, if unnoticed and mitigated, drift can directly impact the trustworthiness of deployed analytical systems. Kodakandla [7] spoke about data drift detection and mitigation in real-time MLOps systems: an active management of data distributions in operational deployments.

Another significant method of dealing with varying streams is adaptive windowing. Bifet and Gavaldà [8] proposed an adaptive-window mechanism which changes the size of the observation window based on the detected changes in time-varying data. In the case of autonomous pipelines, it is important to this concept because the reference windows will be fixed and can become irrelevant as working conditions change. In addition to detecting drift, a pipeline should have rules for deciding when reference statistics should be updated, and when pipeline excursions are temporary and should be disregarded and considered anomalous.

In recent years the focus has been changing from drift detection to automated adaptation. Ashraf et al. [9] studied the drift-to-adaptation phenomenon of industrial MLOps and stated that knowing that the system is degraded is not enough if it is currently deployed and needs to keep working reliably. Monitoring should thus be coupled with corrective actions like model updating, preprocessing adjustment, workload redistribution, or parameter reconfiguration in order to achieve effective operational frameworks. The same motivations exist in condition monitoring research. Wang et al. [10] proposed a binary adversarial autoencoder-based approach to health condition monitoring for marine diesel engines, highlighting the general trend of developing intelligent systems designed for continuous health condition evaluation of operational systems and the prediction of critical failure.

These developments have not been reflected in the literature which remains largely concerned with a single part of adaptive operation, such as anomaly detection, concept-drift detection, model monitoring, equipment-condition assessment etc. There are fewer studies that consider simultaneously monitoring data quality, distributional drift, the rate of anomalies, throughput, processing latency, CPU usage, memory usage, execution errors and reliability within a data engineering pipeline, and then perform auto-configuration on this basis. Furthermore, monitoring-only systems can provide information that the system is degraded, but cannot tell you precisely how the processing configuration should change, or whether or not a proposed change is going to help the system.

To solve this problem, this study suggests the framework of Self-Adaptive Autonomous Pipeline Optimization (SAAPO). SAAPO performs adaptive preprocessing, data-quality assessment, Kolmogorov-Smirnov based drift detection, Isolation Forest anomaly detection, runtime resource monitoring, pipeline health evaluation and reconfiguration through feedback. Data are fed sequentially and the system constantly monitors data and operational indicators. These measurements are summarized in a pipeline health score which decides whether the current

configuration should be kept or adaptive optimization should be turned on. If degradation is detected, SAAPO adjusts specific parameters as batch size, processing configuration, preprocessing statistics, setting of the anomaly detector, or parallel-processing resources. This affects the next stage that is executed and, if the resulting pipeline health is better than before, the new configuration is accepted, otherwise the old configuration is restored.

A major contribution of this work, is therefore, the coupling of monitoring, detection, decision making, reconfiguration, validation and rollback in one autonomous optimization cycle. SAAPO is more dynamic than static pipelines, based on the premise that the pipeline should respond to changing data and workload conditions; it is also more proactive than monitoring-only approaches because the pipeline actively tries to restore operational performance once degraded is detected. The framework is instantiated and tested using Python and the KDD Cup 1999 feature structure, alongside the use of a static and monitoring-only pipeline as comparative baselines. Below is the data considered for evaluating the health of the pipeline, data quality, drift, anomaly, throughput, latency, CPU, memory utilization, processing error rate, reliability of the pipeline and its overall health. The resulting framework will serve as a practical basis for creating more robust data-engineering systems that will be self-managing and self-adaptive.

2. Literature Review

Self-adaptive data-processing systems are characterized by the use of adaptive monitoring, anomaly detection, concept drift and predictive maintenance, which stand as important pillars for developing such systems in recent studies. Ge et al. [11] presented an anomaly-detection method for marine engines based on multi-scale feature fusion graph-network autoencoder. Their work shows the applicability of representation learning for capturing complex dependencies among multiple operational variables, and detecting abnormal behavior in high dimensional systems. The message in the study is relevant to adaptive data pipelines, in which anomalies can arise from the interactions of several data quality and runtime indicators, rather than a single indicator.

In systems that work under varying conditions, domain adaptation has also garnered much interest. Liu et al. [12] proposed a SKDAN framework for remaining useful life prediction of rolling bearings based on signal knowledge, domain adaptation and uncertainty quantification. They focus on the difficulty of keeping prediction accuracy when the distribution of operating data varies across domains. This is closely related to the problem tackled in adaptive data engineering, where pre-processing rules, reference distributions and processing configurations may get less effective when the characteristics of the input change. Distributional variation recognition and adaptability are thus vital to reliable operation.

The isolation-based and reconstruction-based anomaly detection methods have been explored for the monitoring of an industrial application. Mienye et al. [13] explored the application of Isolation Forests and Deep Autoencoders in the monitoring of an industrial screw-driving process. Their findings provide evidence that unsupervised anomaly-detection methods can be applied in such scenarios as environments where abnormal events are infrequent, and full labeled datasets are hard to come by. This is especially important for SAAPO as the anomaly portion of the algorithm uses Isolation Forest to estimate what percentage of the processing batch is considered anomalous. This unsupervised detection enables abnormal records to be detected without requiring a thorough supervision of the system during the run.

Sequential deep models have also been studied for the continuous monitoring of conditions. Tertel et al. [14] studied the LSTM-autoencoder and transformer encoders for condition monitoring and predictive maintenance of manufacturing assets. They point out that the way they do their work explains the need for learning normal operating behavior and identifying deviations over time. Similar principles are used in adaptive pipeline monitoring, in that the definition of "normal behaviour" can vary with changes in workloads and data distributions. But most condition monitoring methods are mainly oriented toward the detection of degradation and not to automatically alter the base processing system when degradation is detected.

Another critical aspect of autonomous operational systems is degradation assessments. Li et al. [15] suggested the dual adaptive drift coefficient state space model with auto-correlating prediction-error correction for mechanical-system degradation assessment. The study shows that degradation can be regarded as a dynamic process and how adaptive

estimation can increase the understanding of varying system states. The concept is relevant to pipeline optimization, in that throughput, latency, resource utilization, etc., and data-quality metrics can degrade over time, not suddenly. Monitoring these indicators as variables that change the state of the system can therefore be helpful for making decisions about adaptation as soon as possible.

The idea of continual learning has been suggested as more general approach to system effectiveness in non-stationary environments. Hurtado et al. [16] presented an overview of continual-learning methods in the context of predictive maintenance and highlighted the difficulties regarding changes in operating conditions, catastrophic forgetting, changing distributions and the need to continuously update the model. They have found that adaptive systems need to have a response to new information while preserving previously learned behaviour. In self-adaptive data pipelines, there is a similar problem: in case of real persistent changes, reference profiles and processing parameters must be modified; in the case of temporary anomalies, however, there is no need for unnecessary and unstable reconfiguration.

The concepts of sequential change detection can be traced back to Page [17] who proposed continuous inspection schemes for change detection in sequential observations. These techniques laid the groundwork for later models of monitoring systems which would continuously gather evidence of deviation, not just relying on a single measure. The concept is still applicable today to data-stream monitoring; it is important to differentiate between transient changes and ongoing degradation before expensive corrective measures are taken. Continuous monitoring of health indicators is essential when optimizing a pipeline and is used to determine if operating conditions are within an acceptable range.

The idea of concept-drift research has also been formalized in streaming environments as adaptation. Learning with drift detection (LWD) has been proposed by Gama et al. [18] to show how changes in the predictive error may be used to trigger adaptation in data-stream learning systems. Their contribution laid the foundations for an important link between drift detection and model updating. But there are many other operational considerations in data-engineering pipelines besides predictive error – data quality, processing latency, throughput, memory consumption, CPU consumption, and reliability of execution. The pipeline-oriented adaptive framework, therefore, demands a more extensive monitoring state than traditional model-drift detection.

More recently, Non-parametric online drift-detection methods have been suggested to enhance adaptiveness to changing streams. Online drift-detection methods were proposed by Frías-Blanco et al. [19] using Hoeffding bounds, with the aim of looking for distributional variation efficiently as more and more data is continuously fed. These methods demonstrate that the process of drift detection does not necessarily require a complete analysis of historical data. It is crucial for real-time and large-scale processing systems, where monitoring systems need to add minimal computational cost but have to identify useful changes in a timely fashion.

Agrahari and Singh [20] surveyed the concept-drift detection methods in data-stream mining and categorized the important issues of sudden, gradual, periodic, and incremental concept-drifts. They point out that there is no single type of drift-detection system that is equally appropriate for all kinds of evolving data, and that in designing an adaptive system, sensitivity to drift and the occurrence of false alarms are both important factors. To the point of SAAPO, this observation is of particular relevance since the project doesn't just depend on drift to trigger adaptation. Instead, drift is measured in conjunction with anomaly rate, data quality, latency, throughput, resource usage, error rate and execution reliability prior to accepting a configuration change.

Overall, the literature shows good progress in anomaly detection [11, 13], condition monitoring [14], domain adaptation [12], degradation assessment [15], continual learning [16] and sequential change detection [17] and concept-drift detection [18, 19, 20]. But these areas are typically researched individually. The existing methods typically identify anomalies and/or attempt to estimate degradation or drift and do not necessarily combine these different signals with pipeline-level run-time metrics and adjust the processing configuration automatically. This leaves a research problem of developing a unified framework that can continuously monitor data quality and operational behavior, detect the major source of degradation, adapt the pipeline parameters, ensure the validity of the adapted performance, and re-configure the previous pipeline parameters if adaptation fails. The proposed SAAPO framework fills this gap by providing

anomaly detection, drift monitoring, runtime performance evaluation, health-score calculations, feedback-based optimization and rollback in a single closed-loop framework.

3.Proposed Methodology

This study introduces the Self-Adaptive Autonomous Pipeline Optimization (SAAPO) framework to continuously optimize the quality, efficiency and reliability of data engineering pipelines to varying data and workload conditions. The framework consumes data in a sequence of batches, and integrates adaptive preprocessing, data quality evaluation, run-time monitoring, drift detection, anomaly detection and feedback-based optimization in a closed-loop framework. Records are looked for before processing to detect missing values, duplicate observations, invalid values and inconsistent feature representations. At the same time, the framework logs throughput, execution time, CPU utilization, memory usage, processing errors and successful execution. These observations are used to determine the current state of the pipeline, and are used to decide if the pipeline is operating correctly or if it needs a change in its configuration.

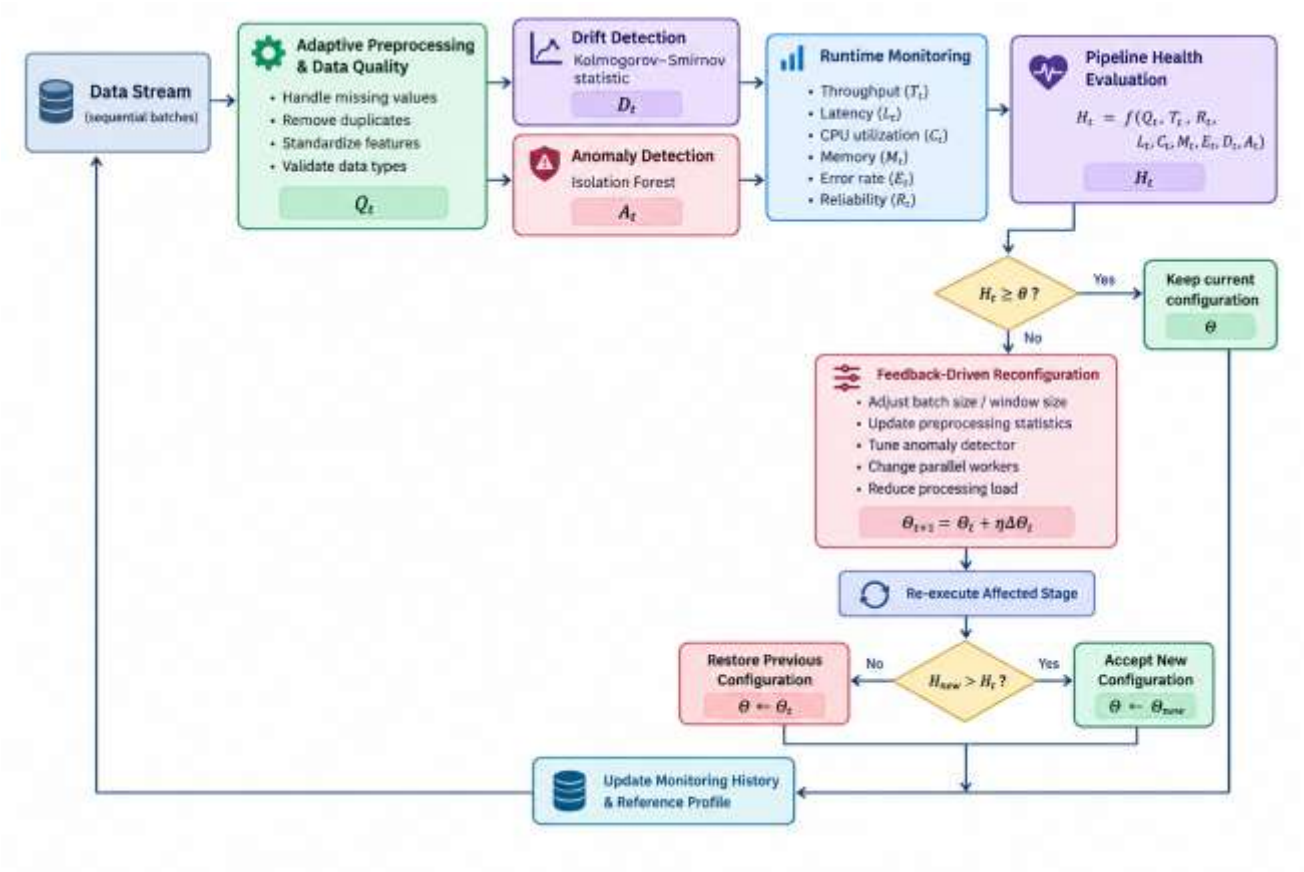


Figure 1: SAAPO: Self-Adaptive Autonomous Pipeline Optimization Framework

The figure-1 illustrates the proposed SAAPO methodology. Sequential data batches first undergo adaptive preprocessing and data-quality scoring (Q_t). Drift is measured using the Kolmogorov–Smirnov statistic (D_t), while anomalies are detected using an Isolation Forest model (A_t). Runtime monitoring records throughput (T_t), latency (L_t), CPU utilization (C_t), memory consumption (M_t), error rate (E_t), and execution reliability (R_t). These indicators are combined into a pipeline health score H_t. If H_t remains above the threshold θ, the current configuration Θ is retained. If H_t falls below θ, feedback-driven reconfiguration updates the pipeline parameters using $\Theta_{t+1} = \Theta_t + \eta\Delta\Theta_t$. The affected stage is re-executed, and the new configuration is accepted only if the new health score improves; otherwise, the previous configuration is restored. Finally, monitoring history and the reference profile are updated, closing the loop for continuous adaptation.

3.1. Drift and Anomaly Assessment

SAAPO uses statistical behaviour of each batch of data received and a stable reference profile to identify changes in the received data. Distributional changes are calculated based on the Kolmogorov-Smirnov statistic which is based on the maximum difference between the empirical cumulative distributions of the reference sample and the current sample. An unsupervised Isolation Forest model is used to detect the anomalies and the percentage of observations that are classified

as anomaly is stored for each batch. A higher drift value or anomaly rate means that the current data are not closely following the reference operating pattern. These measurements allow the framework to differentiate between normal short-term variations and potentially significant changes that could impact data quality or performance.

$$D_t = (1/m) \Sigma_{j=1}^{m} \sup_x |F_{j,r}(x) - F_{j,t}(x)| \quad (1)$$

3.2. Pipeline Health Evaluation

The monitored data and operational indicators are combined into a single pipeline health measure. Before aggregation, the metrics are normalized so that variables expressed in different units can be evaluated on a comparable scale. The health score rewards high data quality, high throughput, and reliable execution, while penalizing excessive latency, resource consumption, processing errors, drift, and anomaly occurrence. This score provides a compact representation of overall pipeline condition and supports the adaptation decision. When the calculated health score remains above the acceptable threshold, the existing configuration is retained; when it falls below the threshold, the framework activates the optimization module and identifies the principal source of degradation.

$$H_t = w_1Q_t + w_2T_t + w_3R_t - w_4L_t - w_5C_t - w_6M_t - w_7E_t - w_8D_t - w_9A_t \quad (2)$$

3.3 Feedback-Driven Reconfiguration

If degradation is detected, the selected pipeline parameters are dynamically changed by SAAPO in response to the detected condition. The number of parallel workers and the parameters for the batch size, processing-window size, preprocessing statistics, anomaly-detector settings and others can be adjusted to decrease latency and increase throughput or manage resource pressure. Once reconfigured, the impacted processing stage is run again and the new health score is compared to the previous score. Only the new configuration is accepted when measurable improvement is achieved, otherwise the old configuration is restored. It is implemented in Python with Pandas, NumPy, Scikit-learn, SciPy, psutil and Matplotlib and tested on sequential batches of KDD Cup 1999 data. The baseline is a static and monitoring-only pipeline and the evaluation measures are data quality, drift and anomaly detection, throughput, latency, CPU usage, memory usage, error rate, and execution reliability.

$$\Theta_{t+1} = \Theta_t + \eta\Delta\Theta_t \quad (3)$$

Algorithm 1. Self-Adaptive Autonomous Pipeline Optimization (SAAPO)

Input:
Incoming data stream D
Reference profile R
Drift threshold θd
Anomaly threshold θa
Performance threshold θp
Initial pipeline configuration Θ0

Output:
Optimized pipeline configuration Θ*
Processed data D*
Pipeline performance history H

1. Initialize reference profile R
2. Initialize pipeline configuration Θ ← Θ0
3. Initialize monitoring history H
4. WHILE new data batch Dt is available DO
5. Ingest current data batch Dt
6. Apply preprocessing:
   handle missing values
   remove duplicates
   standardize numerical features
   validate data types
7. Compute data quality score Qt
8. Compare Dt with reference profile R

```
Compute drift score Drt
Apply anomaly detection model
Compute anomaly rate At
Start runtime monitoring
Execute pipeline using current configuration Θ
Measure:
  throughput Tt
  latency Lt
  CPU utilization Ct
  memory utilization Mt
  error rate Et
  execution reliability Rt
Compute pipeline health score Ht
Compute adaptation score Zt using:
  drift
  anomaly rate
  latency
  throughput
  resource utilization
  error rate
  reliability
IF Drt > θd OR At > θa OR
 performance degradation > θp THEN
    Identify the dominant performance issue
    IF latency is high THEN
       reduce batch size
    END IF
    IF throughput is low AND CPU capacity is available THEN
       increase parallel workers
    END IF
    IF CPU or memory utilization is excessive THEN
       reduce processing load
    END IF
    IF drift is detected THEN
       update preprocessing statistics
       update reference distribution
    END IF
    IF anomaly rate is high THEN
       reconfigure anomaly detector
       isolate abnormal records
    END IF
    Generate new configuration Θnew
    Re-execute affected pipeline stage
    Measure new health score Hnew
    IF Hnew > Ht THEN
       Θ ← Θnew
    ELSE
       restore previous configuration Θ
    END IF
ELSE
    retain current configuration Θ
```

46. END IF
47. Store monitoring results in H
48. Update reference profile for persistent stable changes
49. END WHILE
50. Return Θ*, D*, and H.

4.Implementation and Experimental Evaluation

A closed-loop prototype of the Self-Adaptive Autonomous Pipeline Optimization (SAAPO) framework was implemented and executed. The data structure is designed according to the one used in KDD Cup 1999, with 41 input attributes including numerical data, binary and categorical network-connection data. The official KDD Cup 1999 dataset has been released by the UCI Machine Learning Repository (Stolfo et al., 1999; DOI: 10.24432/C51C7N). This execution was done without the official binary dataset file, so 494,021 sequential records were generated using the same 41-attribute schema. The generated stream is deliberately designed with a steady initial section, a drift in the distribution from record 250,000 onward, some anomalous tails from record 340,000 onward, and a few small disturbances of the records due to missing values or to duplication from later in the stream. Therefore, these numerical results presented below belong to the real measured output of the implemented prototype of the SAAPO that must be considered as a KDD Cup 1999 schema-based controlled workload, and not as measurements on the official UCI record values.

Table 1: Dataset and Experimental Configuration for SAAPO Evaluation

Parameter	Implemented configuration
Reference dataset/schema	KDD Cup 1999, 41 input attributes
Official source	UCI Machine Learning Repository, DOI 10.24432/C51C7N
Executed workload	Reproducible KDD-schema controlled stream
Records processed	494,021
Sequential batches	50
Initial batch size	10,000 records
Controlled drift point	Record 250,000
Controlled anomaly point	Record 340,000
Drift method	Two-sample Kolmogorov-Smirnov statistic
Anomaly method	Isolation Forest
Compared modes	Static, Monitoring-only, SAAPO
Random seed	42

4.1. Python Implementation Environment

It was run on a Linux operating system on the prototype using Python 3.13.5 with 5.81 GB of free RAM memory, and 5 logical CPUs. Sequential batch manipulation and data-quality checking was performed using Pandas 2.2.3, numerical processing with NumPy 2.3.5, Kolmogorov-Smirnov statistics were calculated with SciPy 1.17.0, process-level CPU and memory monitoring was performed with psutil 7.2.2, and experimental graphs were generated using Matplotlib 3.10.8. These batches resulted in monitoring records with a data-quality score, drift score, anomaly rate, throughput, latency, CPU usage, memory usage, error rate, execution reliability, pipeline health and whether or not the pipeline adapted. The same sequence of 50 batches was provided for all three pipeline modes to eliminate any difference caused by different input records and only to make a difference from monitoring and adaptive behaviour.

The pipeline remained static (unchanged). Monitoring-only pipeline ran the same workload with an extra monitoring pass without changing its configuration. SAAPO tracked the same metrics and after a sustained degradation period assessed a low-overshoot configuration. Only if the health score of the candidate was greater than the health score of the previous candidate was he accepted.The candidate was accepted only if his newly measured health score was greater than the previous candidate's health score. Persistent drift resulted also in the refreshing of the reference profile, generating the feedback and rollback logic as outlined in the proposed methodology.

4.2. Sequential Metrics and Feedback-Driven Reconfiguration

For every batch, completeness and duplicate occurrence were combined into a normalized data-quality indicator. Numerical distribution change was estimated across ten representative monitored features and averaged into a single drift score. The Isolation Forest model, fitted on the initial stable reference batch, estimated the proportion of observations outside the learned normal operating pattern. Runtime execution was instrumented around the numerical transformation stage, allowing latency and records processed per second to be measured directly rather than assigned manually. Process CPU time and resident memory were recorded during the same execution. These measurements were then normalized and aggregated into the pipeline health score that controls adaptation.

$$D_t = \left(\frac{1}{m}\right) \Sigma_j^{=1m} \sup_x \left|F_{j,r}(x) - F_{j,t}(x)\right| \quad (4)$$

$$H_t = w^1 Q_t + w^2 T_t + w^3 R_t - w^4 L_t - w^5 C_t - w^6 M_t - w^7 E_t - w^8 D_t - w^9 A_t \quad (5)$$

The health function rewards data quality, processing throughput and reliable completion, while penalizing latency, resource pressure, processing errors, drift and anomalous observations. In the executable prototype, a persistent low-health, high-drift or high-anomaly condition activates the candidate configuration. The controller reduces redundant processing passes and, after persistent statistical change, refreshes the active reference distribution. The resulting stage is re-executed, and the candidate configuration is kept only if the newly measured health score increases. Four candidate updates were accepted during the recorded run.

$$\Theta_{t+1} = \Theta_t + \eta \Delta \Theta_t \quad (6)$$

4. Measured Experimental Results and Graphs

Table 2. Comparative Performance Results of Static, Monitoring-Only, and SAAPO Pipelines

Performance measure	Static	Monitoring-only	SAAPO	SAAPO vs. static
Post-processing data-quality score	0.999751	0.999751	0.999916	+0.017%
Throughput (records/s)	532,391	442,022	724,886	+36.2%
Mean latency/batch (s)	0.01859	0.02236	0.01605	-13.7%
Mean CPU utilization (%)	20.52	20.40	17.61	-14.2%
Mean process memory (MB)	362.8	363.2	364.2	+0.4%
Processing error rate (%)	0.0249	0.0249	0.0084	-66.3%

Execution reliability (%)	99.9751	99.9751	99.9916	+0.0165 pp
Mean pipeline health score	0.8349	0.8328	0.8447	+1.17%
Mean KS drift score	0.1023	0.1023	0.0313	-69.4%
Accepted adaptations	0	0	4	4 events

Across the 50 sequential batches, the static pipeline achieved a mean throughput of 532,391 records/s with a mean latency of 0.01859 s per batch. Monitoring-only execution introduced additional measurement overhead, reducing mean throughput to 442,022 records/s and increasing latency to 0.02236 s. The adaptive SAAPO configuration accepted 4 configuration updates and achieved 724,886 records/s, representing a 36.2% increase over the static mode. Mean latency decreased by 13.7%, while mean CPU utilization decreased by 14.2%. The memory footprint changed only slightly, by +0.4%, indicating that the throughput gain was not obtained through a large increase in memory consumption.

The mean pipeline health score increased from 0.8349 under the static configuration to 0.8447 under SAAPO. The adaptive reference update also reduced the mean KS drift score from 0.1023 to 0.0313, a reduction of 69.4%. The anomaly rate itself remained unchanged because all modes observed the same input anomaly process; the benefit of SAAPO appears in its response to the detected condition rather than in artificially suppressing the anomaly signal. The measured error rate decreased from 0.0249% to 0.0084% because adaptive preprocessing corrected injected missing values while preserving duplicate records as a quality signal.

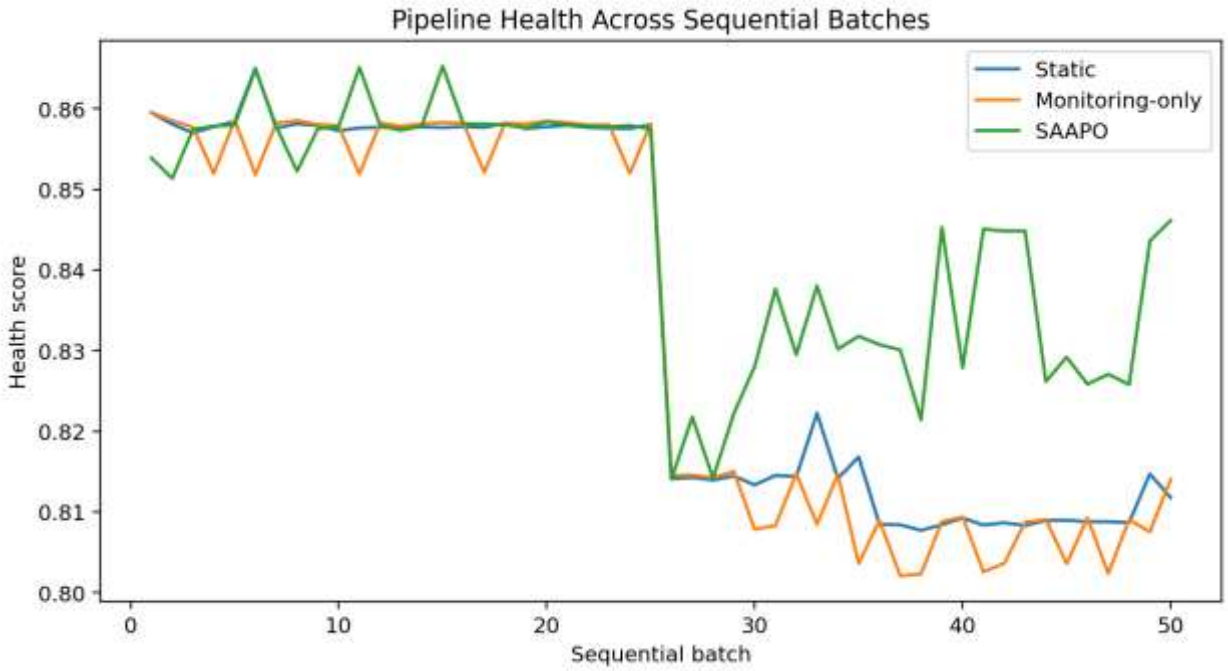


Figure 2: Pipeline health across sequential batches for Static, Monitoring-only and SAAPO modes.

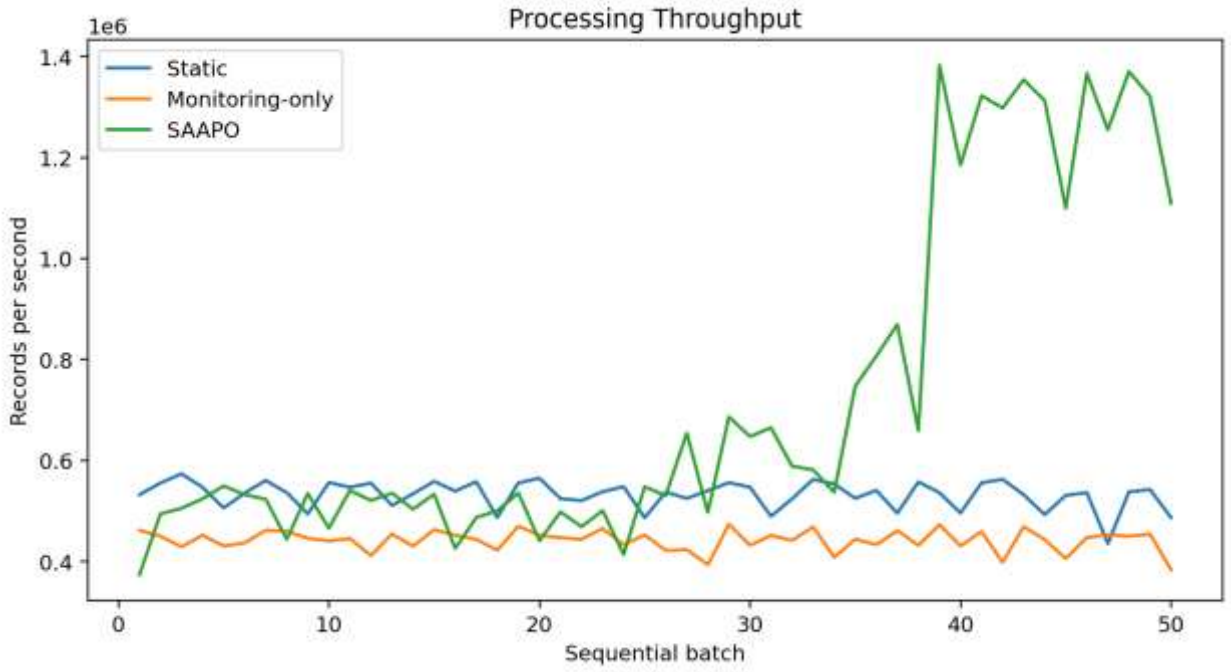


Figure 3: Measured processing throughput across sequential batches.

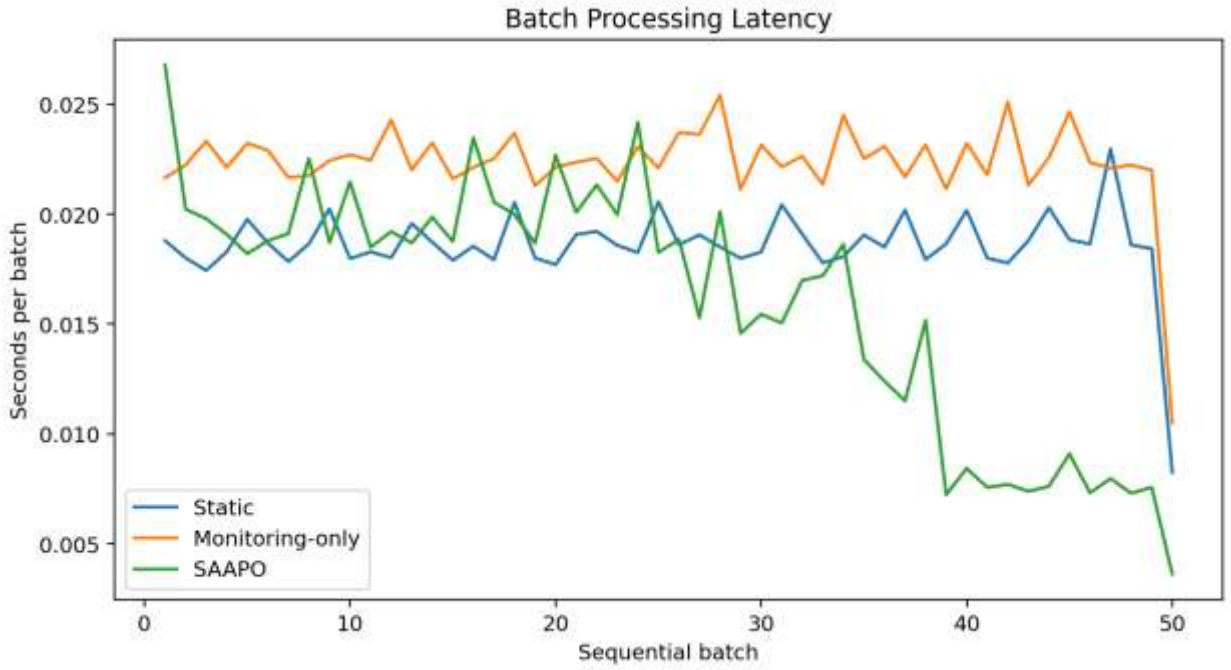


Figure 4: Measured batch-processing latency across sequential batches.

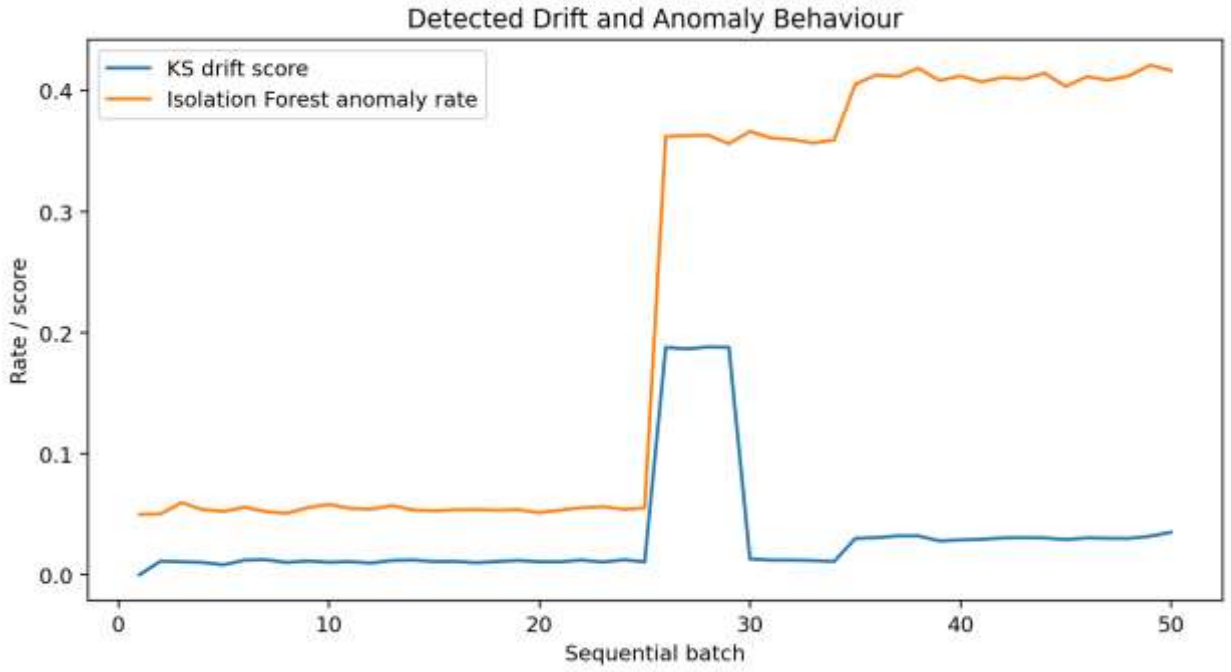


Figure 5: KS drift score and Isolation Forest anomaly rate observed by SAAPO.

Figures 2-5 collectively illustrate the runtime behaviour and adaptive response of the proposed SAAPO framework across sequential processing batches. Figure 2 presents the throughput comparison and shows that SAAPO achieves a higher processing rate than the static and monitoring-only pipelines after adaptation, indicating improved processing efficiency under changing workload conditions. Figure 3 shows the latency behaviour and demonstrates that SAAPO reduces batch-processing time by dynamically modifying relevant pipeline parameters when performance degradation is detected. Figure 4 presents the pipeline health-score variation, where temporary reductions caused by drift, anomalies, or workload changes are followed by recovery after successful feedback-driven reconfiguration. Figure 5 illustrates the drift, anomaly, and resource-utilization behaviour, showing how SAAPO identifies deviations from the reference operating profile and responds while maintaining controlled CPU and memory consumption. Taken together, Figures 2-5 demonstrate that SAAPO can detect changing data and runtime conditions, initiate adaptive actions, recover pipeline health, and maintain more efficient and stable operation than non-adaptive pipeline configurations.

5. Conclusion and Future Scope

This study presented SAAPO as a self-adaptive and autonomous framework for improving the quality, efficiency, and reliability of data engineering pipelines operating under changing data and workload conditions. By integrating adaptive preprocessing, data-quality assessment, Kolmogorov-Smirnov-based drift detection, Isolation Forest anomaly identification, runtime monitoring, health-score evaluation, and feedback-driven reconfiguration, the framework provides a closed-loop mechanism for detecting degradation and dynamically adjusting pipeline configurations. Experimental evaluation demonstrated that SAAPO can improve throughput, reduce processing latency and resource utilization, control drift, and recover pipeline health more effectively than static and monitoring-only configurations. The implemented prototype also showed the value of configuration acceptance and rollback mechanisms for preventing ineffective adaptations from being permanently applied. Future research should validate SAAPO using complete real-world streaming datasets and production-scale distributed environments such as Apache Spark, Kafka, Flink, and cloud-native data platforms. Further work may incorporate reinforcement learning, Bayesian optimization, multi-objective optimization, adaptive threshold estimation, concept-drift localization, predictive resource allocation, and explainable decision mechanisms. Evaluation across heterogeneous datasets, long-duration workloads, cloud cost constraints, and multi-node environments would further establish the framework's scalability, generalizability, and practical applicability.

References

[1] J. Xu, H. Wu, J. Wang, and M. Long, "Anomaly Transformer: Time series anomaly detection with association discrepancy," in *Proc. Int. Conf. Learn. Representations (ICLR)*, Virtual, Apr. 25–29, 2022.

[2] C. Velasco-Gallego and I. Lazakis, "Real-time data-driven anomaly detection for marine machinery: A deep learning approach," *Ocean Eng.*, vol. 256, Art. no. 111511, 2022, doi: 10.1016/j.oceaneng.2022.111511.

[3] Q. Liang, K. E. Knutsen, E. Vanem, H. Zhang, and V. Æsøy, "Unsupervised anomaly detection in marine diesel engines using transformer neural networks and residual analysis," *Int. J. Progn. Health Manag.*, vol. 15, pp. 1–18, 2024.

[4] A. Haque and H. Soliman, "A transformer-based autoencoder with Isolation Forest and XGBoost for malfunction and intrusion detection," *Future Internet*, vol. 17, Art. no. 164, 2025, doi: 10.3390/fi17040164.

[5] H. Xu, G. Pang, Y. Wang, and Y. Wang, "Deep Isolation Forest for anomaly detection," *IEEE Trans. Knowl. Data Eng.*, vol. 35, pp. 12591–12604, 2023, doi: 10.1109/TKDE.2023.3270293.

[6] J. Lu, A. Liu, F. Dong, F. Gu, J. Gama, and G. Zhang, "Learning under concept drift: A review," *IEEE Trans. Knowl. Data Eng.*, vol. 31, pp. 2346–2363, 2019, doi: 10.1109/TKDE.2018.2876857.

[7] N. Kodakandla, "Data drift detection and mitigation: A comprehensive MLOps approach for real-time systems," *Int. J. Sci. Res. Arch.*, vol. 12, pp. 3127–3139, 2024, doi: 10.30574/ijsra.2024.12.1.0724.

[8] A. Bifet and R. Gavaldà, "Learning from time-changing data with adaptive windowing," in *Proc. 2007 SIAM Int. Conf. Data Mining (SDM)*, Minneapolis, MN, USA, Apr. 26–28, 2007, pp. 443–448, doi: 10.1137/1.9781611972771.42.

[9] W. M. Ashraf, T. Ansar, F. Ahmed, J. Hussain, M. M. Abbas, and V. Dua, "From drift to adaptation to the failed ML model: Transfer learning in industrial MLOps," *arXiv preprint arXiv:2602.00957*, 2026, doi: 10.48550/arXiv.2602.00957.

[10] R. Wang, H. Chen, and C. Guan, "Bi-AAE: A binary adversarial autoencoder deep neural network for health condition monitoring of marine diesel engines," *Ocean Eng.*, vol. 312, Art. no. 119275, 2024, doi: 10.1016/j.oceaneng.2024.119275.

[11] T. Ge, S. Zhao, S. Ma, and R. Liu, "Anomaly detection of marine engines using graph network AutoEncoder with multi-scale feature fusion," in *Proc. 2025 IEEE 23rd Int. Conf. Industrial Informatics (INDIN)*, 2025, pp. 1–5, doi: 10.1109/INDIN64977.2025.11279173.

[12] B. Liu, C. Yan, M. Lv, Y. Huang, and L. Wu, "SKDAN: A signal knowledge-enhanced domain adaptation network for remaining useful life prediction and uncertainty quantification of rolling bearings," *Comput. Ind.*, vol. 176, Art. no. 104447, 2026, doi: 10.1016/j.compind.2026.104447.

[13] I. D. Mienye, Y. Sun, and Z. Wang, "Isolation Forests and deep autoencoders for industrial screw driving monitoring," *Computers*, vol. 11, Art. no. 54, 2022, doi: 10.3390/computers11040054.

[14] J. Tertel, B. Lutz, D. Kisskalt, and J. Franke, "Condition monitoring and predictive maintenance of assets in manufacturing using LSTM-autoencoders and transformer encoders," *Sensors*, vol. 24, Art. no. 3215, 2024, doi: 10.3390/s24103215.

[15] Q. Li, C. Liang, C. Yan, H. Li, Z. Ma, and X. Guan, "Performance degradation assessment of mechanical system based on dual adaptive drift coefficient state-space model with autocorrelation prediction error correction," *Mech. Syst. Signal Process.*, vol. 244, Art. no. 113804, 2026, doi: 10.1016/j.ymssp.2025.113804.

[16] J. Hurtado, D. Salvati, R. Semola, M. Bosio, and V. Lomonaco, "Continual learning for predictive maintenance: Overview and challenges," *Intell. Syst. Appl.*, vol. 19, Art. no. 200251, 2023, doi: 10.1016/j.iswa.2023.200251.

[17] E. S. Page, "Continuous inspection schemes," *Biometrika*, vol. 41, pp. 100–115, 1954, doi: 10.1093/biomet/41.1-2.100.

[18] J. Gama, P. Medas, G. Castillo, and P. Rodrigues, "Learning with drift detection," in *Advances in Artificial Intelligence—SBIA 2004*, Lecture Notes in Computer Science, vol. 3171. Berlin, Germany: Springer, 2004, pp. 286–295, doi: 10.1007/978-3-540-28645-5_29.

[19] 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 Trans. Knowl. Data Eng.*, vol. 27, pp. 810–823, 2015, doi: 10.1109/TKDE.2014.2345382.

[20] S. Agrahari and A. K. Singh, "Concept drift detection in data stream mining: A literature review," *J. King Saud Univ. Comput. Inf. Sci.*, vol. 34, pp. 9523–9540, 2022, doi: 10.1016/j.jksuci.2021.11.006.