JIRBDAI Journal of Institutional Research, Big Data Analytics and Innovation

Development of a Lightweight Framework for Runtime Monitoring of Data Drift in Edge-Deployed IoT Machine Learning Pipelines

[1*]Aisha Ibrahim Bobbo; [2]Oluwatobi Noah Akande, [3]Bilkisu Larai Muhammad-Bello, [4]Ibrahim Anka Salihu

[1,3,4] Software Engineering Department, Nile University of Nigeria, Abuja, Nigeria
[2]Computer Science Department, Nile University of Nigeria, Abuja, Nigeria
*Corresponding Author: bobboaishaibrahim@gmail.com

ABSTRACT

As machine learning models increasingly enter production, especially on resource-limited edge devices in Internet of Things (IoT) systems, concerns around their long-term reliability due to data drift also grow. Data drift can lead to severe performance degradation and inaccurate predictions, wrong decisions, and even system failure without the awareness of the operators. This study designs, implements, and evaluates a lightweight runtime monitoring framework using a combination of statistical and distance-based approaches as a modular and framework-agnostic monitoring solution, measuring accuracy, latency, and memory. The proposed framework design comprises a data interceptor, preprocessor and statistical summarizer, baseline storage with fixed-size reservoir storage, drift detection engine, and alerting and reporting interface. It makes use of three statistical and distance-based techniques (the Kolmogorov-Smirnov test, the Population Stability Index, and the Wasserstein Distance) combined using the majority vote to detect drifts in a model-agnostic and label-free manner. Performance was assessed on 20 synthetic scenarios and the London Bike Sharing dataset for real-world evaluation over five independent random seeds. A mean detection accuracy of 92.63% (95% CI: 86.79%–98.48%) and a constant memory footprint of 0.039 MB and stable latency (±1.59 ms) were achieved. Competitive benchmarking against Frouros, Evidently AI, and Alibi-Detect under identical hardware conditions demonstrated higher detection accuracy than Frouros and Evidently AI, the most stable latency among all evaluated tools, and three times lower memory usage than Evidently AI. These results demonstrate that lightweight statistical ensemble monitoring can provide practical and reliable drift detection for real-time IoT deployments without heavy infrastructure overhead.

KEYWORDS: Runtime Monitoring, Data Drift, Machine Learning, Internet of Things (IoT)

I. INTRODUCTION

The deployment of machine learning (ML) models in practice is widespread, and decision systems powered by ML have been at the forefront of many application domains like healthcare, finance, transport, and autonomous systems. Konno et al. (2022) defined data drift as being “the statistical changes of the input data, which eventually result in the invalidity of the assumptions made by ML models regarding the training dataset distribution". (Kumar et al., 2022). This is the core of this failure mode of ML operations (MLOps) problem, which leads to wrong predictions and loss of trust in ML without any crash or failure evidence from the system, thus it having a significant impact on both the robustness and stability of deployed production ML systems. While heavy and to certain extents open-source drift monitoring tools like Amazon SageMaker Model Monitor, Evidently AI, Alibi-Detect, and Frouros have been the subject of development, those are majorly for heavy infrastructure-dependent use cases

Volume 2, Issue 1, 2026
https://doi.org/10.5281/zenodo.19730405

with some path or offline analysis ways. Therefore, lightweight runtime monitoring techniques that are easy to integrate into ML pipelines with minimal computational or integration cost are in demand.

Continuous monitoring and adaptation of deployed ML, i.e., MLOps, is an important part of reliable system maintenance, being one of the software engineering aspects in higher need for research (Amershi et al., 2019). As organizations rely increasingly on Artificial Intelligence (AI) to run, the paradigm has switched from model development to longer-term performance management, automation, and observability (Garg et al., 2021). Though frameworks focus more on scalability than accessibility, this could limit their effectiveness in smaller environments such as academic institutions and startups that cannot cope with limited access. The development of an efficient, open-source runtime monitoring framework will promote responsible deployment of AI by ensuring that these systems are transparent, trusted, and sustainable. The objectives of this study are to design a lightweight runtime monitoring framework using a combination of statistical and distance-based approaches, implement the framework as a modular and framework-agnostic monitoring solution, and evaluate its performance using accuracy, latency, and memory across synthetic and real-world drift scenarios. This study proposes a memory-efficient, flexible, model-agnostic, and label-free drift monitoring framework based on the Kolmogorov–Smirnov test, Regularized Population Stability Index, and Wasserstein Distance using a majority voting scheme and provides an extensive benchmark comparison against existing tools made under equal hardware conditions.

II. RELATED WORKS

Over the last couple of years, the body of research on data drift and concept drift in machine learning pipelines has grown together with interest in real-time, adaptive, scalable monitoring solutions. Statistical hypothesis testing, distribution distance measures, or adaptive streaming detectors are often employed to respond to drift monitoring, although research is usually computationally expensive and used in environments where a volume of data streams are produced, having inputs that continue to evolve over time. Eck et al.'s (2022) supply chain case studies were used to propose a near real-time end-to-end fully automated monitoring framework for deployed ML models. They used statistical drift detection techniques such as KS test, Bhattacharyya distance, etc. and showed that large drifts could be identified without deteriorating the performance of models. Similarly, Conde et al. (2022) proposed a new lightweight real-time monitoring framework for Feature Monitoring (FM) of streaming systems using Exponential Moving Histograms to implement multivariate monitoring with minimal computation and memory requirements. Pathak Mavromatis et al. (2023) propose various metrics for IoT. They have based it on a compact drift detection model, LE3D, that can be used for IoT sensor environments and achieve high accuracy in detecting drift at little resource cost. Yu et al. (2024) proposed Type-LDD: a knowledge distillation-based framework that identifies which type of drift has occurred with negligible performance drop over synthetic and real datasets to facilitate lightweight drift monitoring.

Aside from academic approaches, several open-source tools have been created to facilitate drift detection and monitoring in a production setting. Evidently AI offers very rich dashboards and a variety of statistical checks to monitor data & model quality but tends to be more heavily weighted in resource-constrained deployment environments. While Alibi-Detect provides state-of-the-art drift, outlier, and adversarial detection algorithms that offer high performance in terms of detection capabilities, the requirements, both in terms of CPU power and time, can be considerable. Frouros offers a single Python library that incorporates several stream-based as well as batch drift detection methods in the same package (but some of those will not allow you to run online) and an easy way to implement your models by integrating with the selected

Volume 2, Issue 1, 2026
https://doi.org/10.5281/zenodo.19730405

detectors, but practical integration gets complicated and should be evaluated for each use case in detail when using different available techniques. Continuing with this idea, WhyLabs

announced Whylogs, which provides free and lightweight profiling and logging for AI observability at an extremely low cost of implementation; however, drift detection is not part of the profile pipeline yet.

However, many gaps still exist in light of these tools and frameworks. Most such solutions prioritize scalability and dashboard monitoring over minimal deployment overhead, which disqualifies them from being turned into edge-IoT environments where memory and computation are heavily constrained. Finally, current tools need a good deal of initial integration effort and do not cater to lightweight runtime monitoring with a memory upper bound. Additionally, most existing drift detection libraries are developed specifically for method-specific detection, which may lead to less robustness against various representation drifts in the real streaming environments. Such gaps indicate the demand for a lightweight framework for drift monitoring, which is model agnostic and label free and able to consistently identify drifts while occupying limited memory and navigates with a low-latency protocol on edge-based IoT ML pipelines. Table 1 summarizes key related works and monitoring tools, highlighting their methodologies, contributions, and reported outcomes.

Table 1: Summary of Related Works

Author (Year)	**Methodology**	**Contributions**	**Results**
Eck et al., (2022)	Statistical tests (KS, Bhattacharyya)	Real-time, automated monitoring framework	Detected significant drifts without affecting model performance
Conde et al., (2022)	Multi-variate tests with Exponential Moving Histograms	Lightweight, real-time framework for streaming systems	Detects multi-feature drift efficiently with low latency
Mavromatis et al., (2023)	ADWIN, Page-Hinkley Test, KSWIN ensemble	Lightweight IoT drift detection framework	High detection accuracy (up to 97%), low CPU/RAM usage
Yu et al., (2024)	Type-LDD with knowledge distillation	Lightweight runtime framework identifying drift type	Improved adaptation accuracy in synthetic and real datasets
Poenaru-Olaru et al., (2023)	Error Rate-Based and Data Distribution-Based detectors	Automated AIOps maintenance pipeline	Real-time adaptive retraining based on drift
Evidently AI (2021)	Comprehensive data and model quality monitoring.	Wide range checks, excellent visualizations, and dashboards.	Heavyweight
(WhyLabs, 2021)	Lightweight data profiling and logging	Extremely low overhead for creating statistical data profiles	Drift detection is not integrated.

Volume 2, Issue 1, 2026
https://doi.org/10.5281/zenodo.19730405

Van Looveren, A. et al. (2021)	Advanced outlier, adversarial, and drift detection.	State-of-the-art detection algorithms	Computationally intensive
Céspedes Sisniega & López García (2024)	Unified library for online/offline drift detection	Real-time streams	Performance is algorithm-dependent.

While recent approaches for monitoring (drifting) in data streams have proposed ways to either create statistical monitoring frameworks or lightweight streaming detectors, practical drift monitoring tools exist while being limited in terms of their accuracy, budget constraints, and ease of deployment inside production ML pipelines on resource-constrained edge IoT devices. To fill this gap, the current study proposes a memory-efficient drift monitoring framework integrating output from the Kolmogorov–Smirnov test, Population Stability Index, and Wasserstein Distance via a majority voting mechanism, demonstrating through synthetic and semi-synthetic drifting datasets its effectiveness compared to state-of-the-art drift detection tools.

III. METHODOLOGY

This framework tackles the continuous and incessant real-time streaming of inference requests through stateless models, where incoming batches are treated as independent segments for capturing temporal shifts through sliding windows. Figure 1 depicts the entire architecture of the proposed framework, from data stream through interceptor, summarizer, detection engine, and alerting interface, which is visualized as a sequential flow. The Data Interceptor is a middleware layer that connects directly to the input pipeline that allows for a shallow copy of input features, which means it does not change the core logic or response times. The Data Summarizer takes these raw feature matrices and converts them into small statistical vectors and only stores the statistical fingerprints in memory to ensure efficiency.

The use of the London Bike Sharing dataset as a means of real-world validation thereby simulating realistic IoT-style streaming settings by virtue of sensor-driven demand patterns that vary over time (e.g., during different weather conditions, seasonality, and human mobility behavior). The dataset includes historical counts of bike rentals, along with contextual input variables (e.g., temperature, humidity, wind speed, and season) that make it perfect for testing distribution shifts in tabular ML pipeline inputs. The dataset was considered a continuous stream with the initial segment used to build the baseline reservoir distribution and additional segments processed live in batches using sliding windows to mimic runtime monitoring. This allowed the framework to be tested against realistic drift settings beyond synthetic generation and enabled us to validate the proposed methods of monitoring both in controlled synthetic drift situations as well as in nonstationary data from real-world-emphasized applications.

Drift Detection Engine: This is the analysis engine that detects changes of statistical significance between the real-time stream data and a fixed-size sliding window of 500 samples. It integrates three complementary detection methods, including the Kolmogorov-Smirnov (KS) test to account for statistical significance, the Population Stability Index (PSI) with equal-frequency binning to identify structural change, and the Wasserstein Distance for geometric interpretability. The engine consists of a two-level majority vote, based on the fact that feature-level voting is done in such a way that at boundary conditions at least 2 out of 3 methods need

Volume 2, Issue 1, 2026
https://doi.org/10.5281/zenodo.19730405

to agree with each other from any segment and there are also certain boundaries set for dataset voting wherein it should deviate (drift) more than 10 % of features before throwing an alert.

A. Detection Criteria

(a). Kolmogorov–Smirnov (KS) Test: A non-parametric hypothesis test that determines the maximum deviation between empirical CDFs of the live and baseline samples. The p-value is used to decide if the two samples likely came from the same distribution (drift) at the significance level $\alpha = 0.05$. It yields significance testing with no assumption about the shape of distribution.

(b). Population Stability Index (PSI): A bin-based stability measure that calculates the proportional differences between baseline and live feature distributions. Results are therefore defined exactly according to baseline quantiles by means of 10 equal-frequency bins, preserving the static reference frame for monitoring. Drift is flagged if PSI exceeds 0.20, the standard threshold for distributional shift in industry. Equal-frequency binning was chosen over equal-width because unlike the latter, it guarantees that each bin contains enough baseline samples to make the proportion estimation stable (thus preventing instability in skew distributions). PSI is defined as:

$$PSI = \sum_{i=1}^{n} \left(p_{live,i} - p_{base,i}\right) \times \ln\left(\frac{p_{live,i} + \varepsilon}{p_{base,i} + \varepsilon}\right)$$

where $p_{live,i}$ and $p_{base,i}$ are the proportions of live and baseline samples in bin *i,* and $\varepsilon = 1x10^{-10}$ prevents division by zero.

(c). Wasserstein Distance (Earth Mover's Distance): A distance-based metric that calculates the least-costly conversion of the live distribution into the baseline distribution. In contrast to purely significance-based tests, the Wasserstein Distance offers a direct geometric interpretation of drift distances. The detection threshold is determined empirically per feature through bootstrapping the baseline distribution: 1,000 resamples are obtained with replacement, and the Wasserstein distance between each of these resamples and the complete baseline is calculated. The threshold is set at mean + 2 x standard deviation of these bootstrap distances to represent an upper bound on expected distance under no-drift conditions with approximately the 95th percentile. This generates a principled, data-driven threshold that is dependent upon the scale and spread of each individual feature rather than applying an arbitrary fixed value.

B. Majority Vote

A feature is flagged as a drift when at least 2 of 3 methods agree. An overall pipeline drift alert is raised only when more than 10% of monitored features are flagged. This two-level mechanism was empirically validated: using a single feature flag produced a dataset-level false positive rate of 0.17, and the 10% fraction threshold reduced it to 0.03. The baseline storage module maintains a fixed-size reservoir of 500 samples drawn from the baseline dataset, irrespective of the original dataset size. For a 10-feature dataset using 64-bit floating point values, this amounts to exactly 500 × 10 × 8 = 40,000 bytes = 0.039 MB of persistent storage, a bound that does not grow with baseline size. This O(1) memory property was empirically confirmed in scalability experiments where latency plateaued at approximately 65–68 ms across baseline sizes from 1,000 to 50,000 samples, confirming that detection cost does not scale with baseline size. The alerting and reporting interface converts detection outputs into structured diagnostic reports containing per-feature method votes, metric scores, and drift fractions.

Volume 2, Issue 1, 2026
https://doi.org/10.5281/zenodo.19730405

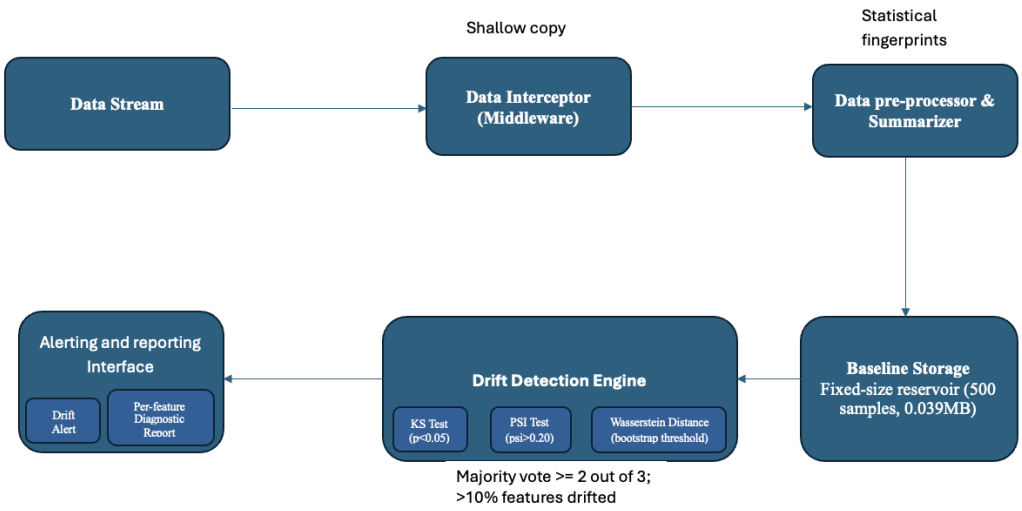


Figure 1: Lightweight Data Drift Monitoring Framework Architecture

The proposed framework processes IoT requests through five main components. The data stream feeds into the data interceptor, which creates a shallow copy of input features. These are then passed to the preprocessor & summarizer to generate the statistical fingerprints (mean, variance, quantile). The drift engine compares the live data with the baseline storage, which has a fixed-size reservoir of 500 samples (0.039 MB). Using three complementary methods: KS test ($p<0.05$), PSI test ($psi>0.20$), and Wasserstein distance (bootstrap threshold). A two-level majority vote is applied if 2 or more of the methods at the feature level show drifts and more than 10% of features are flagged at the dataset level. When drift is detected, the Alerting and Reporting interface generates a drift alert and per-feature diagnostic report.

C. Algorithmic Workflow of Lightweight Drift Detection Framework

Input: Baseline Dataset, Live Dataset, Thresholds
Output: Drift alert report
.fit() Phase (executed once)

(a). Generate a fixed-size reservoir of 500 samples from the baseline dataset.
(b). Compute per-feature statistical summaries.
(c). Calculate Wasserstein thresholds via 1,000 bootstrap resamples.

.check() Phase (run for every new batch)

(d). Capture a shallow copy of the incoming live dataset via the data interceptor.
(e). Preprocess and summarize raw batch into statistical fingerprints (mean, variance, quantile).
(f). For each feature j:
  i. Compute KS-test by flagging if p-value < 0.05.
  ii. Compute PSI score by using 10 equal-frequency bin and flagging if score > 0.20.
  iii. Compute Wasserstein by flagging if distance > threshold.
  iv. Mark feature j as drift if >= 2 out of 3 tests are flagged.
(g). Compute drift ratio = (number of drifted features / total features) * 100.
(h). If drift ratio > 10%, trigger alert and generate drift report.

Volume 2, Issue 1, 2026
https://doi.org/10.5281/zenodo.19730405

D. Performance Evaluation Metrics

Three main metrics were used to perform a statistical and operational analysis of framework performance.

(a). Detection Accuracy: Shows the percentage of all cases where the framework correctly recognized drift presence or absence:

$$\text{Accuracy} = \frac{\text{True Positives} + \text{True Negatives}}{\text{Total Scenarios}} \times 100\%$$

Accuracy was calculated for each seed and averaged via mean accuracy with a 95% confidence interval over five seeds. The accuracy computation excludes the boundary scenario due to its intentionally ambiguous ground truth. Separate independent no-drift false positive trials (100) were measured.

(b). Detection Latency: Time taken during one .check() call on 1000 samples with 10 features using time.perf_counter():

$$Latency = T_{end} - T_{start}$$

Where T_{end} and T_{start} are recorded immediately before and after the check() call, respectively. Latency was measured across 100 consecutive runs and reported as mean, standard deviation, and 95% confidence interval.

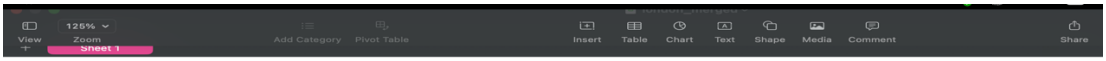
(c). Memory Consumption: Reported as two separate metrics for full transparency. Persistent storage memory is the real memory size of framework objects kept between calls—reservoir array, statistical summaries, and Wasserstein thresholds—determined by numpy.ndarray.nbytes and sys.getsizeof(). Peak detection memory is the highest Python heap allocation reached during a call to check() measured with tracemalloc, which counts all arrays that are allocated temporarily and freed afterwards during computation. In both cases, the figures are reported separately in order to distinguish between the framework's standing memory demand and its transient computation cost.

IV. RESULTS AND DISCUSSION

The developed lightweight drift detection model has targeted resource-constrained environments like edge-IoT pipelines. It separates the data interceptor, reservoir-based storage, statistical summarization, and the detection engine to ensure low latency. The framework is developed to process structured tabular data, which are stored and transmitted in Comma-Separated Values (CSV) format as shown in Figure 2 and Figure 3. It depends fully on lightweight libraries of Python such as NumPy 1.26.4, SciPy 1.13.1, and Pandas 2.3.3, therefore requiring only 3 to 5 lines of code for full installation and integration, which greatly reduces the risk of deployment mistakes compared to the 10–50 lines required by competing tools.

Figure 2 presents a raw dataset sample in tabular form from the real-world London Bike Sharing dataset, while Figure 3 illustrates a raw CSV sample from a synthetic dataset scenario. These figures demonstrate the type of structured feature inputs supported by the proposed framework and confirm its applicability to both real-world and synthetic monitoring conditions.

Volume 2, Issue 1, 2026
https://doi.org/10.5281/zenodo.19730405



london_merged

timestamp	cnt	t1	t2	hum	wind_speed	weather_code	is_holiday	is_weekend	season
2015-01-04 00:00:00	182	3.0	2.0	93.0	6.0	3.0	0.0	1.0	3.0
2015-01-04 01:00:00	138	3.0	2.5	93.0	5.0	1.0	0.0	1.0	3.0
2015-01-04 02:00:00	134	2.5	2.5	96.5	0.0	1.0	0.0	1.0	3.0
2015-01-04 03:00:00	72	2.0	2.0	100.0	0.0	1.0	0.0	1.0	3.0
2015-01-04 04:00:00	47	2.0	0.0	93.0	6.5	1.0	0.0	1.0	3.0
2015-01-04 05:00:00	46	2.0	2.0	93.0	4.0	1.0	0.0	1.0	3.0
2015-01-04 06:00:00	51	1.0	-1.0	100.0	7.0	4.0	0.0	1.0	3.0
2015-01-04 07:00:00	75	1.0	-1.0	100.0	7.0	4.0	0.0	1.0	3.0
2015-01-04 08:00:00	131	1.5	-1.0	96.5	8.0	4.0	0.0	1.0	3.0
2015-01-04 09:00:00	301	2.0	-0.5	100.0	9.0	3.0	0.0	1.0	3.0
2015-01-04 10:00:00	528	3.0	-0.5	93.0	12.0	3.0	0.0	1.0	3.0
2015-01-04 11:00:00	727	2.0	-1.5	100.0	12.0	3.0	0.0	1.0	3.0
2015-01-04 12:00:00	862	2.0	-1.5	96.5	13.0	4.0	0.0	1.0	3.0
2015-01-04 13:00:00	916	3.0	-0.5	87.0	15.0	3.0	0.0	1.0	3.0
2015-01-04 14:00:00	1039	2.5	0.0	90.0	8.0	3.0	0.0	1.0	3.0
2015-01-04 15:00:00	869	2.0	-1.5	93.0	11.0	3.0	0.0	1.0	3.0
2015-01-04 16:00:00	737	3.0	0.0	93.0	12.0	3.0	0.0	1.0	3.0
2015-01-04 17:00:00	594	3.0	0.0	93.0	11.0	3.0	0.0	1.0	3.0
2015-01-04 18:00:00	522	3.0	1.5	93.0	6.5	3.0	0.0	1.0	3.0
2015-01-04 19:00:00	379	3.0	1.0	93.0	7.0	3.0	0.0	1.0	3.0
2015-01-04 20:00:00	328	3.0	3.0	93.0	4.0	3.0	0.0	1.0	3.0
2015-01-04 21:00:00	221	3.0	2.5	93.0	5.0	4.0	0.0	1.0	3.0
2015-01-04 22:00:00	178	3.0	2.0	93.0	6.0	4.0	0.0	1.0	3.0
2015-01-04 23:00:00	157	4.0	3.5	87.0	5.0	4.0	0.0	1.0	3.0
2015-01-05 00:00:00	83	4.0	3.0	93.0	6.0	4.0	0.0	0.0	3.0
2015-01-05 01:00:00	67	4.0	3.5	93.0	5.0	4.0	0.0	0.0	3.0
2015-01-05 02:00:00	32	5.0	4.0	87.0	6.0	4.0	0.0	0.0	3.0
2015-01-05 03:00:00	22	6.0	4.5	84.0	7.5	4.0	0.0	0.0	3.0
2015-01-05 04:00:00	38	6.5	5.0	84.0	8.0	4.0	0.0	0.0	3.0

Figure 2: Raw Dataset Sample in tabular format from real-world dataset.

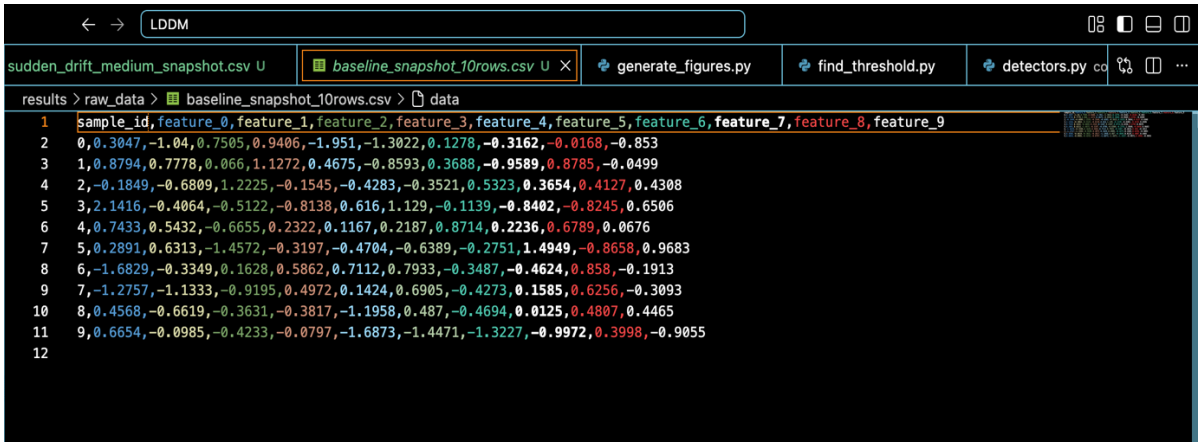


Figure 3: Raw Dataset Sample in CSV Format from Synthetic Dataset.

This framework achieved a mean accuracy of 92.63% across all synthetic and real-world scenarios from five independent random seeds. It encompassed the strong detection of weak (0.15σ), medium, and strong gradual drift signals along with sudden cases, seasonality-periodic drifts, and dense covariate distribution drifts. The most noteworthy benefit case for memory efficiency was that the footprint in persistent storage showed a mere 0.039 MB. It has been empirically confirmed that O(1) memory and detection cost for the reservoir design, where data remain in persistent storage at a fixed 0.039 MB regardless of baseline dataset size (baselines of only 1,000 to 50,000 samples returned a steady-state plateau in detection latency at ~65–68 ms). In terms of responsiveness, the mean detection latency was 22.50 ms per 1,000 samples (±1.59 ms [CI], representing the most consistent latency amongst tools).

The efficiency of the framework was further confirmed through scalability experiments. Despite the increase in baseline dataset size from 1,000 to 50,000 samples, detection latency leveled off at around 65–68 ms, confirming that the detection cost is not based on baseline size. This result corroborates the O(1) memory and detection cost statement, because the reservoir sampling mechanism guarantees that a system keeps at most some fixed number of baseline samples no matter how many examples there are in the original dataset. And those operational efficiencies translate into cost savings in real-world deployment scenarios. For example, all

Volume 2, Issue 1, 2026
https://doi.org/10.5281/zenodo.19730405

100 concurrent models monitored using the same configuration of the framework would consume only 0.10% in case of a 4GB RAM instance, which makes the framework operationally affordable to operate at IoT scale.

Table 2: Comprehensive Comparison Summary at Fixed Seed = 42

Tool	Accuracy (%)	Latency Mean (ms)	Latency CI (±ms)	Memory (MB)	Source	Composite score
Lightweight Monitor (Ours)	89.5	23.09	±1.59	0.288	Experimentally measured	0.5783
Frouros 0.9.0	84.2	6.65	±0.19	0.118	Experimentally measured	0.9241
Evidently AI 0.7.20	84.2	51.86	±5.40	0.861	Experimentally measured	0.4312
Alibi-Detect 0.13.0	94.7	5.95	±0.11	0.159	Experimentally measured	0.9226

Table 2 shows a benchmarking comparison between the proposed framework and three well-known drift monitoring libraries, Frouros, Evidently AI, and Alibi-Detect. Under seed 42, the proposed framework achieved a single-seed performance of 89.5% and its multi-seed performance averaged 92.63% across five independent seeds (42, 123, 256, 512, and 999). With fixed-seed accuracy comparison, Frouros and Evidently AI returned accuracies of 84.2% each under the same condition, further demonstrating that the proposed framework provides a clear advantage in terms of accuracy over these tools. Alibi-Detect obtained the highest single-seed accuracy of 94.7% with a multivariate KS detector, thus confirming that in lower computational resource situations, it was meant to achieve strong drift detection performance. Frouros and Alibi-Detect achieve higher composite scores when using peak memory, but the proposed framework maintains the lowest persistent memory footprint of 0.039 MB making it most suitable for long-running edge IoT where memory is limited. Despite this, the proposed framework achieved comparable accuracy at a much lower persistent memory overhead.

Regarding latency, the proposed framework reported a mean runtime of 23.09 ms, greater than Frouros (6.65 ms) and Alibi-Detect (5.95 ms). This is expected due to the fact that the yielded solution applies three complementary drift detection methods per feature instead of just one detector. The extra computation resilience to a larger patch of drift patterns, which is what makes it more accurate. While Evidently AI showed the maximum latency (51.86 ms) among all, it also had the highest variance in latencies (±5.40 ms). With a confidence interval in the order of ±1.59 ms, it produced the most stable latency of all evaluated tools (at design time), which points to trustworthy runtime behavior and consistent monitoring performance compared to other contenders.

Results from memory show that the edge deployment scenario is even more suitable for our proposed framework. The maximum memory consumption needed by peak detection of the new framework is 0.288 MB, about three times less than Evidently AI (0.861 MB). The persistent baseline storage memory, with a score of just 0.039 MB, is also the lowest as well for all evaluated tools. This result validates that the reservoir design offers a significant benefit

Volume 2, Issue 1, 2026
https://doi.org/10.5281/zenodo.19730405

for long-running monitoring processes in environments with limited resources that substantially restricts memory overhead.

The findings are consistent with relevant literature where statistical monitoring methods have shown cost-effective drift detection. Eck et al. These studies found statistical tests (e.g., KS) do detect large drift without harmful effect on the performance of the model, which is observed here as well as using KS along with a host of other complementary measures to corroborate. Conde et al. (2022) highlighted lightweight feature monitoring for computing-at-the-edge tasks with the least possible computational overhead, and store only statistical fingerprints, and a baseline reservoir has a fixed size in order to cover this footprint. It also finds support for the results of Mavromatis et al. (2023), where ensemble-based IoT drift detection delivers high accuracy with low resource consumption. In this work, it presents a model-friendly framework that employs three complementary drift detection methods in a simple majority voting mechanism while preserving memory efficiency in order to be deployed in edge-IoT settings, contrary to many existing approaches.

V. CONCLUSION

In this work, proposed a lightweight runtime monitoring framework to detect data drift in IoT machine learning pipelines that are deployed on the edge. Its goals are to develop and deploy a memory-limited drift monitoring framework with statistical and distance-based detection methods while measuring the detection performance in terms of accuracy, latency, and memory usage on simulated as well as real-world drift cases. Results showed that the proposed framework can detect a diverse range of drift types. The framework yielded a mean detection accuracy of 92.63% across 20 synthetic scenarios and over the London Bike Sharing dataset evaluated on five independent random seeds with only 0.039 MB of persistent baseline storage memory usage (a static system state) along with stable run time latency. When conducting competitive benchmarking of the proposed framework versus Frouros, Evidently AI, and Alibi-Detect across identical hardware configurations, it was demonstrated that detection accuracy was superior to both Frouros (with a 49% improvement) and Evidently AI packages. This study contributes to the literature in three ways: (i) designed a model-agnostic and label-free drift monitoring framework for edge environments; (ii) proposed an ensemble of a two-level majority voting mechanism that integrates three complementary drift detection techniques, namely, the Kolmogorov–Smirnov test, Population Stability Index, and Wasserstein Distance; and (iii) achieved O(1) persistent baseline memory usage through a fixed-size reservoir sampling strategy. Moreover, the study delivered empirical benchmarking of its observed results against standard drift monitoring tools which showed real-time trade-offs between accuracy and latency as well as memory efficiency. The framework can be embedded into production ML pipelines for continuous observability and early drift warnings with marginal integration effort and low infrastructure cost. Further work will validate the framework on even more comprehensive real-world IoT datasets from diverse domains, broaden the scope of data modalities covered in unstructured forms for anomaly detection, and explore adaptive tuning strategies to automatically adjust detection sensitivity based on context. Subsequent work can also explore closed-loop drift response mechanisms (fully automated model retraining triggers) to proactively contribute to the long-term reliability of models in online edge deployments.

REFERENCES

Abdel Wahab, O. (2022). Intrusion detection in the IoT under data and concept drifts: Online deep learning approach. *IEEE Internet of Things Journal, 9*(20), 19706–19716. https://doi.org/10.1109/JIOT.2022.3167005

\

Amershi, S., Begel, A., Bird, C., DeLine, R., Gall, H., Kamar, E., Nagappan, N., Nushi, B., & Zimmermann, T. (2019). Software engineering for machine learning: A case study. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)* (pp. 291–300). https://doi.org/10.1109/ICSE-SEIP.2019.00042

Céspedes Sisniega, J., & López García, Á. (2024). Frouros: An open-source Python library for drift detection in machine learning systems. *SoftwareX, 26*, 101733. https://doi.org/10.1016/j.softx.2024.101733

Conde, J., Moreira, R., Torres, J., Cardoso, P., Ferreira, H. R. C., Sampaio, M. O. P., Ascensão, J. T., & Bizarro, P. (2022). Lightweight automated feature monitoring for data streams. *arXiv*. https://doi.org/10.48550/arXiv.2207.08640

Dong, S., Wang, Q., Sahri, S., Palpanas, T., & Srivastava, D. (2024). Efficiently mitigating the impact of data drift on machine learning pipelines. *Proceedings of the VLDB Endowment, 17*(11), 3072–3081. https://doi.org/10.14778/3681954.3681984

Eck, B., Kabakci-Zorlu, D., Chen, Y., Savard, F., & Bao, X. (2022). A monitoring framework for deployed machine learning models with supply chain examples. In *2022 IEEE International Conference on Big Data (Big Data)* (pp. 2231–2238). https://doi.org/10.1109/BigData55660.2022.10020394

Evidently AI. (2021). *Comprehensive data and model quality monitoring: Open-source Python library*. Evidently AI. https://www.evidentlyai.com

Garg, S., Pundir, P., Rathee, G., Gupta, P. K., Garg, S., & Ahlawat, S. (2021). On continuous integration/continuous delivery for automated deployment of machine learning models using MLOps. In *2021 IEEE Fourth International Conference on Artificial Intelligence and Knowledge Engineering (AIKE)* (pp. 25–28). https://doi.org/10.1109/AIKE52691.2021.00010

Konno, Y., Nakano, M., & Oguchi, M. (2022). Efficient data selection indicators for updating models under data drifted environment. In *2022 IEEE International Conference on Big Data (Big Data)* (pp. 6724–6726). https://doi.org/10.1109/BigData55660.2022.10020630

Mavromatis, I., & Khan, A. (2023). Demo: LE3D: A privacy-preserving lightweight data drift detection framework. In *2023 IEEE 20th Consumer Communications & Networking Conference (CCNC)* (pp. 917–918). https://doi.org/10.1109/CCNC51644.2023.10060554

Nigenda, D., Karnin, Z., Zafar, M. B., Ramesha, R., Tan, A., Donini, M., & Kenthapadi, K. (2022). Amazon SageMaker Model Monitor: A system for real-time insights into deployed machine learning models. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining* (pp. 3671–3681). https://doi.org/10.1145/3534678.3539145

Poenaru-Olaru, L., Cruz, L., Rellermeyer, J. S., & Van Deursen, A. (2023). Maintaining and monitoring AIOps models against concept drift. In *2023 IEEE/ACM 2nd International Conference on AI Engineering – Software Engineering for AI (CAIN)* (pp. 98–99). https://doi.org/10.1109/CAIN58948.2023.00024

Van Looveren, A., Klaise, J., Knatsko, A., & Vacanti, G. (2021). *Alibi Detect: Algorithms for outlier, adversarial and drift detection* [Software]. Seldon Technologies. https://github.com/SeldonIO/alibi-detect

WhyLabs. (2021). *whylogs: The open standard for data logging and AI observability* [Software]. WhyLabs. https://whylabs.ai/whylogs

Yu, H., Li, J., Lu, J., Song, Y., Xie, S., & Zhang, G. (2024). Type-LDD: A type-driven lite concept drift detector for data streams. *IEEE Transactions on Knowledge and Data Engineering, 36*(12), 9476–9489. https://doi.org/10.1109/TKDE.2023.3344602