International Journal of
INTELLIGENT SYSTEMS AND APPLICATIONS IN ENGINEERING
ISSN:2147-6799 www.ijisae.org Original Research Paper

Mitigating Data Drift in Distributed Streaming Pipelines for Real-Time Financial Fraud Detection

Ravindra Motiram Gurnani

Abstract: Fraud detection systems built on real-time data streams must contend with a persistent challenge: transaction data does not stay stationary. As attackers adapt, merchant patterns shift, and economic conditions change, the statistical profile of both legitimate and fraudulent activity evolves continuously—a phenomenon called concept drift that erodes classifier accuracy and, absent remediation, renders deployed models unreliable within weeks. The challenge compounds in distributed streaming environments, where high-velocity data arrives across heterogeneous partitions and centralized monitoring becomes computationally impractical. This paper introduces a multi-layer drift detection and adaptation framework tailored to distributed financial fraud pipelines. Three detection mechanisms work in tandem: Adaptive Windowing (ADWIN) handles memory-efficient stream segmentation, the Drift Detection Method (DDM) tracks error rates against statistical control thresholds, and the Population Stability Index (PSI) monitors individual feature distributions. Each mechanism targets a different temporal scale and sensitivity level, so the system can raise early warnings before downstream classification quality visibly degrades. A selective ensemble retraining strategy activates on confirmed drift events while leaving stable feature subsets undisturbed. Experiments on synthetic and publicly available financial stream benchmarks confirm that the layered approach shortens detection delay and lowers false alarm rates compared to single-detector configurations. The framework targets Apache Kafka and Apache Flink deployments; operator-level computational overhead is characterized throughout.

Keywords: *deployments, detection, Population, sensitivity*

1. Introduction

Real-time fraud detection is among the most operationally demanding applications of machine learning in the financial sector. Card networks, digital payment processors, and banking institutions collectively process billions of transactions per day, each requiring a risk assessment within milliseconds. The economic stakes are significant: global payment card fraud losses exceeded $32 billion annually in recent years, with projections indicating continued growth as digital commerce expands. Detection systems must therefore achieve high precision — minimizing false positives that block legitimate transactions — while maintaining high recall against an adversarially evolving threat landscape.

At the core of this difficulty lies the non-stationarity of fraudulent transaction data. When institutions deploy countermeasures, attackers modify their tactics; when new payment channels emerge, their transaction signatures differ from historical norms; and when spending seasons shift, the baseline legitimate-activity distribution moves with them. Zliobaite (2010) termed this phenomenon concept drift—changes in the joint distribution P(X, y) that steadily accumulate prediction error in models trained on historical snapshots. Lu et al. (2018) offer a detailed taxonomy distinguishing four drift varieties: gradual, sudden, recurring, and incremental, each carrying different implications for how detection and adaptation strategies should be designed.

The complexity deepens inside distributed streaming architectures of the kind represented by Apache Kafka for transport and Apache Flink for stateful processing. Events arrive as an unbounded, ordered sequence partitioned across multiple compute nodes. A distributional shift may first surface within a single partition, only becoming visible at the aggregate level after it has already spread, or it may only be apparent when all partitions are viewed together. Sculley et al. (2015) documented how monitoring and drift correction quietly consume a disproportionate share of

Independent Researcher, USA
ORCID ID: 0009-0005-5122-5554

operational budgets in production ML systems, characterizing this burden as hidden technical debt. Despite that, most published detection methods were developed for single-stream, single-node settings and do not account for the synchronization and communication constraints that distributed deployments impose.

Three contributions follow from this work. A multi-layer detection pipeline is introduced that unifies ADWIN (Bifet & Gavaldà, 2007), DDM (Gama et al., 2004), and PSI (Bayram et al., 2022) under an escalation protocol with well-defined transition rules. An ensemble adaptation mechanism is described that reacts differently to warning and confirmed drift signals, thereby avoiding disruptive full retraining on transient fluctuations. Finally, the framework is mapped onto concrete Kafka and Flink operator topologies, and processing overhead is quantified at the operator level. The paper proceeds as follows: Section 2 surveys concept drift and streaming fraud detection; Section 3 presents the framework; Section 4 reports experimental results; Section 5 addresses deployment implications; and Section 6 concludes.

2. Background and Related Work

2.1 Concept Drift: Taxonomy and Characterization

Concept drift refers to changes in the underlying data distribution that degrade predictive model performance over time. Webb et al. (2016) distinguish between feature drift — changes in the marginal distribution P(X) — and concept drift in the narrow sense — changes in the posterior P(y|X). In fraud detection, both forms occur simultaneously: the distribution of transaction amounts and merchant categories evolves independently of fraud labels, while the conditional probability of fraud given a feature vector also shifts as attack strategies change.

Drift onset can be sharp—as when a new attack vector appears in the wake of a data breach—or slow-moving, as seasonal spending shifts accumulate over several weeks. Recurring drift poses a particular modeling challenge: an attacker may withdraw a technique once detection rates rise, then reintroduce it months later when monitoring attention has moved on. Ditzler et al. (2015) reviewed algorithms for non-stationary learning and grouped adaptation strategies into retraining-based, ensemble-based, and hybrid categories. Lu et al. (2018) reached a consistent conclusion in their broader review: detection latency and false alarm rate are the primary benchmarks, with computational overhead becoming a constraining factor in resource-limited settings.

2.2 Drift Detection Methods

The Drift Detection Method (DDM) proposed by Gama et al. (2004) monitors the error rate of a base classifier and applies statistical control chart logic. A warning level is triggered when the error rate exceeds the baseline by 2 standard deviations; a drift level is triggered at 3 standard deviations above baseline. This dual-threshold design enables a graduated response: upon reaching the warning level, the system starts building a fresh training window while retaining the active model; once the drift threshold is crossed, the accumulated window replaces the prior training set for classifier retraining. The approach is computationally lightweight but sensitive to the choice of base classifier and to gradual drift that does not produce detectable error spikes.

ADWIN (Bifet & Gavaldà, 2007) takes a different approach: it tracks an adaptive window over the data stream and continually checks whether distributional differences between any two sub-windows exceed what statistical theory predicts under stationarity. When the Hoeffding bound is violated, the stale sub-window is dropped and the window contracts. Per-element complexity stays at O(log W), where W is the window length, so the detector does not become a bottleneck even under sustained high traffic. Assis and Souza (2025) have since extended this idea to work without labels—an important capability when fraud review queues are slow and ground-truth outcomes take hours or days to materialize.

Credit-risk modeling gave rise to PSI long before its adoption in machine learning pipelines, and Bayram et al. (2022) provided the formal bridge between the two domains. The index captures how far the distribution of a score or feature in a current window has moved from an established reference period, computed by binning observations into equal-frequency buckets and summing a symmetric KL-divergence term over those bins. The alert levels Bayram et al. (2022) documented have become industry defaults: readings above 0.2 call for investigation, and readings above 0.25 point toward model review or outright retraining. A key practical advantage is that PSI works at the feature level, so it

can surface covariate shift while prediction accuracy still looks acceptable—giving practitioners an earlier opportunity to respond.

Gonçalves et al. (2014) conduct a comparative study of drift detectors and find that no single detector dominates across all drift types and stream characteristics. Single detectors optimized for abrupt drift tend to generate excessive false alarms on gradual drift, while detectors tuned for gradual drift exhibit high latency on sudden changes. This motivates multi-layer approaches that combine detectors with complementary sensitivity profiles.

2.3 Ensemble Methods for Evolving Streams

Ensemble methods have demonstrated sustained advantages for streaming classification because they can respond to drift by reweighting or replacing individual members without abandoning all accumulated knowledge. Street and Kim (2001) introduced the Streaming Ensemble Algorithm (SEA), which maintains a fixed-size ensemble and periodically retires its weakest-performing member in favor of a freshly trained replacement. Wang et al. (2003) showed that assigning votes in proportion to each member's recent accuracy lets the ensemble downweight components that have fallen behind the current distribution, a simple mechanism that carries surprisingly broad applicability. Adaptive Random Forests (ARF), introduced by Gomes et al. (2017), extend the random forest framework to streaming settings by embedding a per-tree drift detector and replacing drifted trees with background learners. Krawczyk et al. (2017) survey ensemble methods for data streams and conclude that adaptive ensembles consistently outperform single-model approaches on non-stationary benchmarks, though the advantage depends critically on the drift detector's ability to distinguish genuine drift from noise.

Brzezinski and Stefanowski (2014) introduced AUE2, an ensemble design that reweights components based on chunk-level accuracy rather than purely on age, producing better resistance to both abrupt and gradual distributional change. Cassales et al. (2019) tackled a different bottleneck—the time required to update large ensembles after drift—by showing that parallelizing the update step yields substantial wall-clock savings. What this body of work has not addressed, however, is how ensemble update work should be distributed across the nodes of a streaming topology, which is often the binding constraint in production financial deployments.

2.4 Gap Analysis

Three specific gaps motivate the present work. Existing multi-detector approaches—including those reviewed by Gonçalves et al. (2014) and Gomes et al. (2019)—operate at the prediction level and therefore miss covariate shift captured by feature-level statistics like PSI. Adaptation strategies in the literature are also typically designed and validated on single-stream benchmarks, providing little guidance on handling drift heterogeneity across multiple partitions in a distributed setting. Third, computational overhead is almost never characterized at the individual operator level, making it hard for practitioners to judge deployment feasibility under latency budgets. Section 3 addresses each gap directly.

Table 1: Drift Detection Algorithm Comparison

Algorithm	Drift Type Detected	Computational Complexity	Memory Footprint	Detection Latency	Recommended Stream Velocity
ADWIN	Concept drift (error stream)	O(log n) amortized	Variable, adaptive	Sub-second on error stream	All tiers (scalar application)
PSI	Feature distribution shift	O(B) per feature (B=bins)	O(BxF) reference histograms	1-15 min (window-dependent)	Low to Medium
KL Divergence	Continuous feature marginal drift	O(n log n) with KDE	O(n) per window	Per window slide (30s-5min)	Medium
Jensen-Shannon Div.	Symmetric distributional drift	O(n log n)	O(n) per window	Per window slide	Low to Medium

Algorithm	Drift Type Detected	Computational Complexity	Memory Footprint	Detection Latency	Recommended Stream Velocity
EDDM	Gradual concept drift (labeled)	O(1) per sample	O(1)	Requires label arrival	Low (label-available context)

3. Proposed Framework

3.1 Architecture Overview

The proposed framework is organized as a three-layer monitoring pipeline that sits alongside the primary classification pipeline in a Flink topology. Each layer operates at a different temporal granularity and targets a different drift signal. Layer 1 (feature monitoring) applies PSI to a rolling reference window computed over the preceding 24-hour period. Layer 2 (window management) applies ADWIN to the model's raw prediction scores to detect changes in score distribution. Layer 3 (error monitoring) applies DDM to the binary classification error signal once labels become available — typically within minutes for fraud decisions resolved by rule-based review queues.

Signals from the three layers are combined through a deterministic escalation protocol. A PSI alert on any feature triggers a monitoring escalation that increases ADWIN's sensitivity by reducing its confidence parameter. A DDM warning signal causes the adaptation controller to begin training a background ensemble on the current data window without decommissioning the active model. A DDM drift signal, or a PSI reading above 0.25 sustained for two consecutive monitoring intervals, triggers a model swap to the background ensemble. This escalation structure is designed to decouple the high-frequency feature monitoring from the lower-frequency and higher-cost model retraining operation.

3.2 Layer 1: PSI-Based Feature Monitoring

Within the framework, PSI is recalculated for every input feature on a configurable schedule—the default interval is 15 minutes—by comparing the current observation window against a rolling 24-hour reference. Bins are derived from the reference window using equal-frequency partitioning into K buckets, and the index is computed as the sum, over all bins, of the difference between current and reference proportions multiplied by the log ratio of those proportions. Bayram et al. (2022) codified the alert thresholds that practitioners now treat as industry defaults. A score below 0.1 leaves no cause for concern; one between 0.1 and 0.2 warrants closer attention; a score exceeding 0.2 points to a shift material enough to investigate; and once a score crosses 0.25, the model itself is typically up for review. This computation lives in a Flink ProcessFunction that maintains the reference histogram in managed state and emits alert events downstream whenever the score crosses a configured threshold.

3.3 Layer 2: ADWIN Window Management

Layer 2 applies ADWIN to the sequence of raw model prediction scores, targeting distributional changes in the score stream that tend to appear before the binary error rate moves appreciably. The algorithm holds an adaptive window, periodically testing whether the score means in the window's sub-regions have diverged significantly; when they have, it discards the stale portion and alerts the adaptation controller. Bifet & Gavaldà (2007) showed that this procedure carries O(log W) per-element complexity, where W is the current window length, meaning latency stays bounded regardless of how long a stable period lasts. The delta confidence parameter defaults to 0.002 but is tightened to 0.0002 when a concurrent PSI alert is active, sharpening the detector's sensitivity during periods of known feature instability.

3.4 Layer 3: DDM Error-Rate Monitoring

DDM (Gama et al., 2004) is applied to the binary error signal generated when delayed labels become available. The method tracks the running mean error rate and standard deviation and applies the following decision rules: a warning is issued when the error rate statistic exceeds the minimum observed value by 2 standard deviations; drift is declared when it exceeds the minimum by 3 standard deviations. These thresholds correspond to the 2-sigma and 3-sigma criteria described by Gama et al. (2004). In the distributed setting, DDM is run independently on each Flink partition, with partition-level drift signals aggregated by a coordinator operator. Drift is declared globally when a configurable fraction of partitions — default majority — signal drift

simultaneously, reducing susceptibility to false positives from localized partition anomalies.

3.5 Ensemble Adaptation Strategy

The adaptation layer maintains two ensemble pools: an active pool used for inference and a background pool trained incrementally on recent data. Each pool consists of ten gradient-boosted trees with incremental update capability via warm-start retraining on the most recent observations. When a DDM warning signal is received, the background pool begins training from the current state of the active pool, allowing it to accumulate recent distributional information without committing to a model swap. When drift is confirmed, the active and background pools are swapped, and the former active pool is reinitialized for the next background training cycle.

Any feature whose PSI stays below 0.1 throughout the monitoring window is treated as stable and excluded from the background training pass, so the ensemble retains what it has learned about that part of the input space. In financial fraud, attacks often concentrate on a narrow set of channels or merchant categories while leaving the rest of the transaction profile largely unchanged; selective retraining aligns naturally with that pattern by limiting update cost to the regions that have actually moved. Losing et al. (2018) surveyed incremental learning approaches and concluded that targeted update policies are among the most reliable methods for preserving accumulated knowledge on non-drifted feature regions.

Table 3: DAP Pipeline Stage Configuration Matrix

Stream Velocity Tier	Events/sec	Recommended Detector	PSI Threshold (Alert/Retrain)	Retraining Trigger	Checkpoint Interval
Low	< 100K eps	PSI + EDDM	0.20 / 0.25	Sustained ALERT (>2 windows)	60 seconds
Medium	100K-1M eps	PSI + KL Divergence + ADWIN	0.20 / 0.25	Single TRIGGER or sustained ALERT	30 seconds
High	> 1M eps	ADWIN (error stream) + PSI (sampled)	0.22 / 0.28	Single TRIGGER	10 seconds

4. Experimental Evaluation

4.1 Experimental Setup

Evaluation was conducted on two data streams. The first is a synthetic stream generated using the MOA framework (Bifet et al., 2010) with the RandomRBF generator configured to produce abrupt drift events at intervals drawn uniformly from between 5,000 and 20,000 instances, interleaved with gradual drift segments. The synthetic stream contains 500,000 instances with a 5% base fraud rate and was designed to include recurring drift segments to stress-test adaptation memory. The second is the PaySim financial transactions dataset, a publicly available synthetic mobile money simulation that reproduces distributional characteristics of real transaction data with known injected fraud patterns.

All experiments were implemented in Python 3.11 using the River streaming ML library for drift detectors and scikit-learn for the ensemble components. The distributed simulation partitioned the stream into eight parallel partitions processed by independent worker processes, with a coordinator process aggregating drift signals. Evaluation metrics include classification accuracy, G-mean (geometric mean of sensitivity and specificity, appropriate for imbalanced fraud data), detection delay measured in number of instances processed after true drift onset before detection, and false alarm rate. Baseline comparators are single-detector configurations: ADWIN-only, DDM-only, and a no-adaptation static model.

4.2 Results

On the synthetic stream, the layered framework consistently shortened detection time relative to the DDM-only configuration, most notably during gradual-drift segments where PSI monitoring surfaced distributional movement before the error

rate reached DDM's statistical thresholds. The ADWIN-only setup detected abrupt drift quickly but produced far more false positives during stable intervals—a sensitivity-specificity tradeoff that Gonçalves et al. (2014) document at length. The unadapted static model accumulated classification error progressively across every drift event and had no recovery mechanism.

Against the PaySim stream, the multi-layer configuration kept G-mean within an acceptable band across all evaluated segments, including a simulated peak-activity period where transaction volume doubled and the distributional profile of fraudulent transactions shifted sharply. Features that crossed PSI 0.2 were reliably detected before the corresponding classifier error reached DDM's warning threshold, giving the adaptation controller a usable lead window. False alarm rates were lower for the layered configuration than for ADWIN-only, which reflects the practical benefit of requiring agreement across multiple detection layers before committing to a model swap.

When drift was confined to a minority of features, the selective retraining strategy measurably cut adaptation overhead. In runs where fewer than half the input features crossed the PSI 0.1 boundary, background ensemble training completed proportionally faster than full retraining, which shortened the swap window and reduced the period of elevated inference latency. Souza et al. (2020) observed that benchmarking stream learning systems on real-world data is inherently difficult to standardize; the PaySim evaluation here, while grounded in realistic transaction distributions, does not replicate every constraint of a live production pipeline.

4.3 Sensitivity Analysis

Two configuration parameters are most consequential for tuning the framework to a given stream: the PSI monitoring interval and the fraction of partitions required to declare a global drift event. Shortening the interval improves responsiveness to sudden feature drift at the cost of additional load on the Flink PSI operator; raising the aggregation threshold cuts false positives but slows confirmation of genuine drift. For streams where distributional changes propagate uniformly across partitions, a majority-vote threshold balances these concerns well; when partition-local events are the dominant concern, a lower threshold around 25% is more appropriate. Both parameters are exposed as Flink job configuration values rather than hardcoded constants, so operators can calibrate them empirically for each deployment.

Table 2: Model Performance Degradation Without Drift Detection (Illustrative Pattern from Literature)

Time Interval	False Negative Rate	False Positive Rate	Precision	Recall	Operational Status
T=0 (baseline)	Nominal baseline	Nominal baseline	Nominal baseline	Nominal baseline	Nominal
T+30 days	Increasing (per Webb et al. 2016)	Shifting	Declining	Declining	Warning (undetected)
T+60 days	Continued degradation	Elevated	Further decline	Further decline	Degraded (undetected)
T+90 days	Severe, near random baseline	Elevated	Low	Low	Critical (undetected)
DAP-enabled	Maintained near baseline	Near baseline	Near baseline	Near baseline	Actively managed

5. Discussion

5.1 Integration with Kafka and Flink Pipelines

The framework maps naturally onto the operator model of Apache Flink. The PSI monitoring operator is implemented as a KeyedProcessFunction keyed on feature identifier, maintaining reference histograms in Flink managed state with RocksDB backend for fault tolerance. The ADWIN operator is a RichFlatMapFunction that holds window state per partition. The DDM operator aggregates error signals from a downstream labeling stream joined

against the primary transaction stream using a temporal join with configurable label arrival latency. The adaptation controller is implemented as a separate Flink job that subscribes to a dedicated Kafka drift-alerts topic, isolating the retraining overhead from the primary inference pipeline.

This separation of concerns means that a model swap is executed by updating a Kafka configuration topic consumed by the inference operators, which reload their model artifacts from a shared model registry. The approach avoids in-band serialization of large model objects through the Flink operator network and allows the retraining job to be scaled independently of the inference job. Gomes et al. (2019) identify operator throughput and state size as primary scalability constraints for streaming ML systems, and the present architecture addresses both by minimizing stateful computation in the hot inference path.

5.2 Computational Overhead

The dominant overhead in the proposed framework is the PSI computation, which requires maintaining a reference histogram for each input feature in Flink state. For a feature space of D features with K bins each, the state footprint per partition is proportional to D times K. With 50 features and 20 bins, this amounts to approximately 1,000 floating-point values per partition — well within the capacity of RocksDB-backed state. The ADWIN operator maintains O(log W) state per partition, as guaranteed by the algorithm's complexity analysis (Bifet & Gavaldà, 2007). DDM maintains constant state through four running statistics. The background ensemble training is the most resource-intensive operation but runs asynchronously and does not affect inference latency.

Cassales et al. (2019) demonstrate that parallelizing ensemble update steps reduces wall-clock training time substantially for large ensembles. The background training job in the proposed framework exploits data parallelism by distributing training data across worker nodes, with gradient aggregation following the parameter server pattern. Practitioners should be aware that increased Kafka partition counts improve throughput but proportionally increase the aggregate state footprint of the PSI and ADWIN operators, requiring careful capacity planning for very high-partition deployments.

5.3 Label Availability and Unsupervised Extensions

DDM needs ground-truth labels to calculate the error signal, which creates a dependency on how quickly the fraud review process resolves cases. When resolutions are delayed by hours or days, DDM shifts from an early-warning tool to a late-confirming signal. Label latency is therefore a practical constraint that Assis and Souza (2025) tackled head-on: their ADWIN-U variant sidesteps the need for outcome labels entirely by watching how feature distributions shift over time rather than waiting for prediction errors to accumulate. Swapping this in for DDM in slow-label deployments requires minimal changes to the rest of the pipeline. Gomes et al. (2022) addressed a related scenario in which labels arrive for only a fraction of records, showing that partial annotation is still sufficient to sustain semi-supervised adaptation — a result directly applicable to fraud systems that receive immediate rule-based dispositions on some transactions but rely on delayed human review for the remainder.

Table 4: Comparison with Prior Fraud Detection Architectures

Architecture	Drift Awareness	Detection Latency	Retraining Automation	Production Deployment Complexity	Reference
DAP (This Work)	Embedded (PSI + KL + ADWIN)	Sub-minute (medium tier)	Automated trigger	Moderate (Flink stateful operators)	This work
Neburi (2025)	None reported	N/A	Scheduled	Low-Moderate	Neburi (2025)
Cui et al. (2025)	None reported	N/A	Not specified	High (GNN + RL)	Cui et al. (2025)
Lu et al. (2022)	None reported	N/A	Not specified	High (graph-based real-time)	Lu et al. (2022)
Rahmati (2025)	None reported	N/A	Scheduled (federated)	High (federated coordination)	Rahmati (2025)

6. Conclusion

This paper introduced a multi-layer drift detection and adaptation framework for real-time financial fraud detection within distributed streaming systems. PSI-based feature monitoring, ADWIN window management, and DDM error tracking are combined into a sequential detection pipeline with deterministic escalation rules. An ensemble adaptation strategy differentiates between warning and confirmed drift signals so that the live model can continue operating without unnecessary disruption during transient fluctuations. Evaluation across synthetic and PaySim-derived financial streams showed shorter detection delays and lower false alarm rates than single-detector baselines, with selective retraining cutting overhead when drift was confined to a subset of the feature space.

Several constraints of this work should be noted. The evaluation draws on synthetic data and a simulation-derived financial benchmark rather than proprietary production streams; how far the results generalize depends on stream characteristics that public benchmarks do not fully capture, as Souza et al. (2020) discuss. The 24-hour PSI reference window is a practical default that may need shortening for streams with pronounced intraday seasonality. The aggregation threshold for declaring global drift presumes that shifts propagate across partitions on roughly similar timescales—an assumption worth testing empirically in heterogeneous data center topologies. The framework also does not yet address adversarial drift, where an attacker deliberately perturbs transaction features to confuse both the fraud model and the drift detector; that remains an open problem.

Future work will pursue three directions. First, integration of ADWIN-U and semi-supervised adaptation strategies to address high-label-latency deployments. Second, automated hyperparameter tuning for the PSI interval, the ADWIN confidence parameter, and the aggregation threshold using a meta-learning approach trained on stream statistics. Third, extension of the framework to federated settings where transaction data cannot be centralized due to privacy or regulatory constraints, requiring drift detection and model adaptation to occur on-device or within institutional silos before gradient aggregation.

References

1. Bayram, F., Ahmed, B. S., & Kassler, A. (2022). From concept drift to model degradation: An overview on performance-aware drift detectors. *ACM Computing Surveys, 55*(4), 1–43. https://doi.org/10.1145/3523152

2. Bifet, A., & Gavaldà, R. (2007). Learning from time-changing data with adaptive windowing. *Proceedings of the 2007 SIAM International Conference on Data Mining*, 443–448. https://doi.org/10.1137/1.9781611972771.42

3. Gama, J., Medas, P., Castillo, G., & Rodrigues, P. (2004). Learning with drift detection. In *Advances in Artificial Intelligence – SBIA 2004* (pp. 286–295). Springer. https://doi.org/10.1007/978-3-540-28645-5_29

4. Zliobaite, I. (2010). Learning under concept drift: An overview. *arXiv preprint arXiv:1010.4784*. https://arxiv.org/abs/1010.4784

5. Lu, J., Liu, A., Dong, F., Gu, F., Gama, J., & Zhang, G. (2018). Learning under concept drift: A review. *IEEE Transactions on Knowledge and Data Engineering, 31*(12), 2346–2363. https://doi.org/10.1109/TKDE.2018.2876857

6. Gomes, H. M., Bifet, A., Read, J., Barddal, J. P., Enembreck, F., Pfharinger, B., Holmes, G., & Abdessalem, T. (2017). Adaptive random forests for evolving data stream classification. *Machine Learning, 106*(9), 1469–1495. https://doi.org/10.1007/s10994-017-5642-8

7. Cassales, G. W., Gomes, H. M., Bifet, A., Pfahringer, B., & Senger, H. (2019). Improving the performance of bagging ensembles for data streams through parallelism. *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*, 1–8. https://doi.org/10.1109/IJCNN.2019.8852241

8. Sculley, D., Holt, G., Golovin, D., Davydov, E., Phillips, T., Ebner, D., Chaudhary, V.,

Young, M., Crespo, J.-F., & Dennison, D. (2015). Hidden technical debt in machine learning systems. *Advances in Neural Information Processing Systems, 28*, 2503–2511. https://proceedings.neurips.cc/paper/2015/hash/86df7dcfd896fcaf2674f757a2463eba-Abstract.html

9. Ditzler, G., Roveri, M., Alippi, C., & Polikar, R. (2015). Learning in nonstationary environments: A survey. *IEEE Computational Intelligence Magazine, 10*(4), 12–25. https://doi.org/10.1109/MCI.2015.2471196

10. Wang, H., Fan, W., Yu, P. S., & Han, J. (2003). Mining concept-drifting data streams using ensemble classifiers. *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 226–235. https://doi.org/10.1145/956750.956778

11. Assis, L. P., & Souza, V. M. A. (2025). ADWIN-U: An unsupervised adaptive windowing method for streaming data. *Expert Systems with Applications, 263*, 125780. https://doi.org/10.1016/j.eswa.2024.125780

12. Webb, G. I., Hyde, R., Cao, H., Nguyen, H. L., & Petitjean, F. (2016). Characterizing concept drift. *Data Mining and Knowledge Discovery, 30*(4), 964–994. https://doi.org/10.1007/s10618-015-0448-4

13. Gomes, H. M., Read, J., Bifet, A., Barddal, J. P., & Gama, J. (2019). Machine learning for streaming data: State of the art, challenges, and opportunities. *ACM SIGKDD Explorations Newsletter, 21*(2), 6–22. https://doi.org/10.1145/3373464.3373470

14. Street, W. N., & Kim, Y. (2001). A streaming ensemble algorithm (SEA) for large-scale classification. *Proceedings of the 7th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 377–382. https://doi.org/10.1145/502512.502568

15. Bifet, A., Holmes, G., Kirkby, R., & Pfahringer, B. (2010). MOA: Massive online analysis. *Journal of Machine Learning Research, 11*, 1601–1604. https://jmlr.org/papers/v11/bifet10a.html

16. Gonçalves, P. M., de Carvalho Santos, S. G. T., Barros, R. S. M., & Vieira, D. C. L. (2014). A comparative study on concept drift detectors. *Expert Systems with Applications, 41*(18), 8144–8156. https://doi.org/10.1016/j.eswa.2014.07.017

17. Souza, V. M. A., dos Reis, D. M., Maletzke, A. G., & Batista, G. E. A. P. A. (2020). Challenges in benchmarking stream learning algorithms with real-world data. *Data Mining and Knowledge Discovery, 34*(6), 1805–1858. https://doi.org/10.1007/s10618-020-00698-5

18. Brzezinski, D., & Stefanowski, J. (2014). Reacting to different types of concept drift: The accuracy updated ensemble algorithm. *IEEE Transactions on Neural Networks and Learning Systems, 25*(1), 81–94. https://doi.org/10.1109/TNNLS.2013.2251352

19. Gomes, H. M., Bifet, A., & Pfahringer, B. (2022). Semi-supervised learning over streaming data. *Machine Learning, 111*(11), 3971–4010. https://doi.org/10.1007/s10994-022-06186-9

20. Krawczyk, B., Minku, L. L., Gama, J., Stefanowski, J., & Wozniak, M. (2017). Ensemble learning for data stream analysis: A survey. *Information Fusion, 37*, 132–156. https://doi.org/10.1016/j.inffus.2017.02.004

21. Losing, V., Hammer, B., & Wersing, H. (2018). Incremental on-line learning: A review and comparison of state of the art algorithms. *Neurocomputing, 275*, 1261–1274. https://doi.org/10.1016/j.neucom.2017.06.084