Published by :
https://www.ijert.org/
An International Peer-Reviewed Journal

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 11 , November - 2026

A Self-Adaptive Framework for Concept Drift in Streaming Classification: Shadow-Model Pre-Training, Reliability-Weighted Detection and Scored Strategy Selection

Ananya Patel G. P.
Department of Data Engineering
GM University
Davanagere, India

Pooja M. V.
Department of Information Science and Engineering GM Institute of Technology
Davanagere, India

***Abstract*—In a data stream the relationship between inputs and labels can change without warning, which is known as concept drift. The usual remedy is to wait for a change detector to fire and then retrain the model from scratch, so accuracy drops first and recovers later. We describe a streaming framework that tries to avoid part of that cost. It starts training a shadow model as soon as a detector enters its warning state, combines the alarms of ADWIN, DDM and Page–Hinkley with weights learned online, describes each confirmed drift by its speed, severity, recurrence and the features that changed, and chooses an adaptation strategy by scoring candidate rules. The system reads from Apache Kafka and has a FastAPI dashboard. We compared it with seven baselines on four synthetic and two real streams (480 runs, of which the 160 runs on the real streams are replays of fixed data). The framework reached a mean accuracy of 90.82%, against 90.15% for a plain Hoeffding tree and 92.07% for Adaptive Random Forest (ARF), whose per-sample latency was 2.6 times higher in our setup. Its mean rank (2.83) was second only to ARF. A Friedman test found differences among the methods (p = 0.002), but a Nemenyi test separated only ARF from the plain and ADWIN-based trees. The framework is almost identical in accuracy to a proactive variant without weighted fusion, profiling or scored selection, and we have not yet run ablations, so our results do not show that those components improve accuracy. What the framework adds is a stored record of the rule scores behind every adaptation.**

***Index Terms*—concept drift, data streams, online learning, drift detection, adaptive learning, Apache Kafka**

I. Introduction

Models that run on streaming data, for example in transaction monitoring, sensor networks or electricity markets, rarely see a stationary world. When the relationship between inputs X and labels Y changes, a model that was accurate yesterday loses accuracy without any visible failure. This is concept drift, and it has been studied for many years [1]–[3].

Most systems handle it in the same way: a detector such as ADWIN [4], DDM [5] or Page–Hinkley [6] watches the error rate, and when it fires the model is replaced or retrained. We see four problems with this. The new model starts cold, because nothing was prepared before the alarm. Detectors differ in how well they work on a given stream, yet a plain vote treats them alike. The same reset is applied to a sudden, severe change and to a slow, mild one. And an operator who sees a reset learns very little about why it happened.

We built a framework that addresses these four points together. Its main elements are as follows.

1) A background (shadow) model is created when a detector reaches its warning level and is trained alongside the active model. If the warning turns out to be false, the shadow is simply dropped.
2) The alarms of three detectors are combined with weights that are updated multiplicatively, depending on whether earlier alarms were confirmed by a later fall in accuracy.
3) Each confirmed drift is described by its speed, severity, recurrence and the features that shifted (PSI, KS and χ^2 statistics).
4) Candidate adaptation rules are scored against that description, and the scores of the best rules and the chosen strategy are stored with each adaptation. A tuner adjusts window size, sensitivity and cooldown.
5) The pipeline reads from Apache Kafka, has a web dashboard, and comes with a benchmark harness that compares the framework and seven baselines on six streams with 10 seeds.

An honest summary of our findings is that the framework is competitive with the baselines and cheaper than ARF, but that our experiments, which have no ablation, do not isolate which of its components matter. Section II covers related work, Section III states the problem, Section IV describes the framework, and Section V the implementation. Sections VI and VII give the experiments and results, Section VIII discusses their limits, and Section IX concludes.

II. Related Work

Drift detectors watch either the error stream or the input distribution. DDM [5] follows the error rate and its deviation and has a warning and a drift level. ADWIN [4] keeps an adaptive window and cuts it when two parts of it differ significantly. Page–Hinkley [6] is a cumulative-sum test. Comparisons of detectors show that none is best on every kind of drift [7], which is why we use three and learn how much to trust each.

Hoeffding trees [8] are the standard incremental learner, and CVFDT [9] and the Hoeffding Adaptive Tree (HAT) [10] add drift handling to them. Ensembles such as Learn++.NSE [11], AUE [12], Leveraging Bagging [13], Adaptive Random

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 11 , November - 2026

Forest (ARF) [14], DDD [15] and Streaming Random Patches (SRP) [16] keep several learners and combine or replace them. Our shadow model is related to the diversity idea in DDD, but it exists only between a warning and its resolution, so it costs nothing in stable periods. We compare only against ARF among these ensembles; the others are discussed but not evaluated here.

The weighted majority algorithm [17] lowers the weight of experts that make mistakes. We use the same multiplicative update, but on detectors instead of classifiers. Webb *et al.* [18] define properties of drift such as abruptness, magnitude and recurrence, and [19] surveys applications. We use speed, severity and recurrence to choose a response, and add per-feature statistics based on the Population Stability Index [20] and two-sample tests [21].

Recent work has moved in several directions. Unsupervised detectors such as STUDD [22] monitor how well a student model mimics the main model and so need no labels, which matters because our confirmation step needs labels (Section IV-C). The two-part survey by Hinder *et al.* reviews drift detection [23] and the localisation and explanation of drift [24]. Our feature attribution is a simple distributional version of the latter and has not been compared with the methods it reviews.

Kafka [25] is a common transport for stream processing. River [26] and MOA [27] provide streaming learners and detectors, but they leave the pipeline and the transport metrics to the user. Our framework uses River-style components and supplies a pipeline around them.

III. Problem Formulation

Let (x_t, y_t) be the instance at time t. Concept drift occurs between t_0 and t_1 if

$$\exists\, t_0 < t_1 :\ P_{t_0}(X, Y) \neq P_{t_1}(X, Y) \tag{1}$$

We evaluate prequentially (test, then train) [28]: the model predicts $\hat{y}_t = M_t(x_t)$, the error $e_t = 1[y_t \neq \hat{y}_t]$ is stored, and only then does M_t learn from (x_t, y_t). Rolling accuracy uses a window of $W = 500$ samples. The aim is to choose an adaptation action online, without being told when drift happens, so that errors stay small while per-sample latency stays bounded and unnecessary adaptations are rare.

IV. Proposed Framework

A. Overview

Fig. 1 shows the components. Every instance is predicted, scored for error and passed to the detectors. When a drift is confirmed, it is profiled and a strategy is chosen. Algorithm 1 gives the loop.

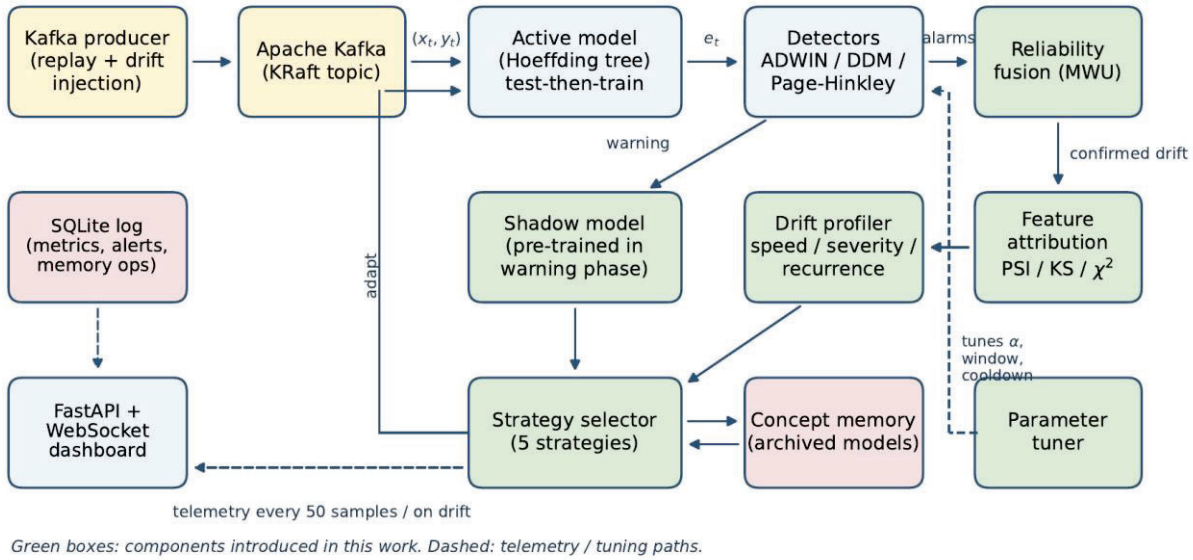


Fig. 1. Architecture of the framework. Green boxes are the parts developed in this work. The active model, the detectors and the Kafka transport are standard components.

Algorithm 1. Self-adaptive prequential loop

for each (x_t, y_t) from the stream **do**
  $\hat{y}_t \leftarrow M$.Predict(x_t); $e_t \leftarrow 1[y_t \neq \hat{y}_t]$; update metrics
  $(s_t, a_t) \leftarrow$ Detectors(e_t) ▷ state, alarms
  if s_t = warning **and** $B = \emptyset$ **then** $B \leftarrow$ SpawnShadow()
  else if s_t = normal **and** $B \neq \emptyset$ **then** Discard(B) ▷ false warning
  M.Learn(x_t, y_t); **if** $B \neq \emptyset$ **then** B.Learn(x_t, y_t)
  if Fuse$(a_t, \mathbf{w}) \geq \tau$ **and** cooldown elapsed **then**
    $\pi \leftarrow$ Profile(speed, severity, recurrence)
    $F \leftarrow$ Attribute(ref, recent)
    $M \leftarrow$ Adapt(arg max$_r$ $\sigma_r(\pi)$)
    start confirmation window; archive concept
  at window end: UpdateWeights(**w**); from time to time Tune()

B. Detectors and the Warning State

The engine runs ADWIN, DDM and Page–Hinkley. Each has a warning level and a drift level. As soon as one detector reaches warning, an independent shadow model B is created. From then on every instance updates both M and B, but only M answers predictions. If the detectors settle without confirming a drift, B is deleted. A false warning therefore costs some computation but never accuracy. The shadow model can only help if the warning comes well before the

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 11 , November - 2026

confirmed drift; when both fire within a few samples, as on a sudden change, the shadow has almost nothing to learn from (Section VII-D).

C. Reliability-Weighted Fusion

Let $a_d^t \in \{0, 1\}$ be the alarm of detector d and $w_d > 0$ its weight. The fused score is

$$S_t = \frac{\sum_{d \in D} w_d \, a_d^t}{\sum_{d \in D} w_d} \tag{2}$$

and drift is declared when $S_t \geq \tau$, with $\tau = 0.50$ by default. Because S_t is a ratio, only the relative weights matter, but this also means that the same τ corresponds to different detector coalitions as the weights change; that is the intended effect of learning the weights, and the plain-voting switch in the dashboard restores equal weights. After an alarm we watch a confirmation window of $W_c = 100$ samples. The alarm counts as confirmed if accuracy in that window falls clearly below its level before the alarm. The weights are then updated as

$$w_d^{(t+1)} = w_d^{(t)} \, \beta^{\pi_d} \gamma^{\rho_d} \tag{3}$$

where $\pi_d = 1$ if detector d raised an unconfirmed alarm and $\rho_d = 1$ if it raised a confirmed one (both are 0 otherwise). The defaults are $\beta = 0.90$ and $\gamma = 1.05$. The rule as written does not bound the weights, so a detector that is repeatedly confirmed can keep growing (DDM reached 1.936 in the run of Fig. 4). Clipping the weights to a fixed interval would prevent one detector from dominating, and we list the missing bound as a limitation.

This step needs the true label of recent samples. Weights are updated only once the confirmation window of 100 labelled samples has passed, so the method assumes that labels arrive within about that many samples. Where labels are delayed or missing, the update would be postponed or impossible; we did not test delayed labels.

D. Drift Profile

Speed is the delay $\Delta t = t_{\text{drift}} - t_{\text{warn}}$ between the first warning and the confirmed drift. A drift is called abrupt when $\Delta t \leq \theta_a = 250$ samples, and gradual or incremental otherwise. *Severity* is the accuracy drop $\Delta\text{Acc} = \text{Acc}_{\text{base}} - \text{Acc}_{\text{cur}}$, graded as minor, medium or severe; a fall from 100.0% to 89.6% (10.4 points) is graded medium. **State the limits for minor, medium and severe from config.yaml.** *Recurrence* compares the feature-moment vector $\mathbf{m}'$ of the new concept with those stored in the concept memory, and the concept is treated as recurrent if

$$\cos(\mathbf{m}, \mathbf{m}') = \frac{\mathbf{m} \cdot \mathbf{m}'}{\|\mathbf{m}\| \, \|\mathbf{m}'\|} \geq 0.85 \tag{4}$$

If the memory is empty there is nothing to compare, and the similarity is reported as 0.0%.

E. Feature-Level Attribution

For each feature f the attributor keeps a reference window (behaviour after the last adaptation) and a recent window. For numeric features it computes

$$\text{PSI}_f = \sum_{i=1}^{k} (p_i - q_i) \ln \frac{p_i}{q_i} \tag{5}$$

$$D_f = \sup_x \left| F_f^{\text{ref}}(x) - F_f^{\text{rec}}(x) \right| \tag{6}$$

where p_i and q_i are the reference and recent shares in k quantile bins and D_f is the two-sample Kolmogorov–Smirnov statistic. Categorical features use a χ^2 test and a categorical PSI. Features are ranked by

$$\text{Score}_f = 0.65 \, \text{PSI}_f + 0.35 \, \text{Stat}_f \tag{7}$$

where Stat_f is the normalised KS or χ^2 statistic, and the top five are reported. A drift is labelled localised if at most two features changed significantly and global otherwise. We did not validate the ranking against drifts whose changed features are known.

F. Scored Strategy Selection

The selector does not follow a fixed decision tree. Each candidate rule r receives a score $\sigma_r(\pi) \in [0, 1]$ from the drift profile π, and the highest score wins:

$$r^\star = \arg\max_r \sigma_r(\pi), \qquad \Delta_\sigma = \sigma_{r^\star} - \max_{r \neq r^\star} \sigma_r \tag{8}$$

The framework stores the three best rules, the choice, the alternative and the margin Δ_σ, together with a short sentence (Fig. 3). Table I lists the strategies and the profiles that favour them. **Give the score of each rule as a function of speed, severity, recurrence and predictability, taken from the selector code.** A cold reset is kept as a fallback.

TABLE I
ADAPTATION STRATEGIES AND THE PROFILES THAT FAVOUR THEM

Favoured when	Strategy	Mechanism
Recurrent concept (cos ≥ 0.85)	historical_switch	Restore archived model and fine-tune on recent data
Abrupt drift, medium or high severity	recent_window (window reset)	Retrain on buffered recent samples with boosted weights
Gradual, predictable drift	background_model (proactive training)	Promote the pre-trained shadow model
Gradual or incremental transition	proactive_ensemble	Blend active and shadow predictions
Uncertain or borderline metrics	reactive_reset (conservative fallback)	Cold restart

G. Recovery and Tuning

After an adaptation the recovery target is 0.95 Acc_{base}, and the recovery time is

$$R_t = t_{\text{rec}} - t_{\text{drift}}, \qquad t_{\text{rec}} = \min \{ t : \text{Acc}(t) \geq 0.95 \, \text{Acc}_{\text{base}} \} \tag{9}$$

The tuner looks at recovery time and rolling accuracy from time to time and changes the window size, the detector sensitivity and the cooldown. Sensitivity is raised when recovery is slow and lowered when false alarms dominate.

H. Hand-Set Constants

Several constants were set by hand: $\theta_a = 250$ samples, the similarity threshold 0.85, the severity limits, $W = 500$, $W_c = 100$, β, γ, τ and the 0.65 / 0.35 weights in Eq. (7). We did not

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 11 , November - 2026

run a sensitivity analysis for any of them, so the results hold only for these settings.

I. Cost

In stable periods the cost is one incremental learner plus constant-time detector updates. During a warning the learning cost roughly doubles, since the shadow trains in parallel. Attribution sorts two windows of size W per feature, which takes $O(|F|\ W \log W)$ per evaluation. Measured latencies are in Section VII.

V. Implementation

The backend is written in Python 3.11 with FastAPI 0.111 and River 0.21. It binds the adaptive model, the detector manager and the profiler, and exposes a WebSocket endpoint for live charts. Metrics, drift alerts and memory operations are stored in SQLite, and the dashboard lists earlier runs with their accuracy, macro-F1 and the number of drifts and adaptations.

The dashboard accepts three kinds of source (a synthetic generator, a pre-loaded dataset or a live Kafka topic) and has switches for parameter tuning, concept memory and the fusion rule, as well as an alarm threshold and a sensitivity multiplier. A drift-injection panel inserts a drift of chosen pattern, mechanism (for example a concept swap), magnitude and duration, and a label-noise test checks robustness. Other panels show rolling accuracy against true and detected drifts, detector trust over time, detection metrics measured against ground truth, and the timeline of adaptations with their scores.

Kafka 3.7.0 runs as a single node with a single partition in KRaft mode. A producer replays a dataset at a set rate and can inject drift at a chosen index, so the true drift positions are known. The consumer is designed to record lag, throughput and end-to-end latency. The stack is started with Docker Compose, which also offers a demo producer that replays Elec2 at 50 records per second with a concept swap at sample 500. We used this path to check that the pipeline works, but we did not run a load test, and all accuracy results in this paper come from the benchmark harness, not from Kafka. The code base has 63 unit and integration tests covering the Kafka path, attribution, benchmarking, statistics and the adaptive pipeline.

VI. Experimental Setup

A. Streams

We used six streams of at most 3000 samples each. Four are synthetic with known drift points: abrupt (SEA-style [29]), gradual (sigmoid transition), incremental (staircase) and recurring (oscillating concepts). Two are real: electricity prices (Elec2) [30] and phishing websites [31]. The real streams have no labelled drift points, so detection metrics are not defined for them. Elec2 is also strongly autocorrelated, which can favour simple baselines [32]. The streams are short and few, and only two are real; this limits what the experiments can show (Section VIII).

Each method was run on each stream with 10 seeds, 480 runs in total. The seeds change the generated data only for the four synthetic streams, so those 320 runs are independent repetitions. Elec2 and Phishing are replayed in a fixed order, which is why most methods show a standard deviation of zero on them; the 160 runs on these two streams are repetitions of the same data, not independent samples, and only HAT shows any variation between seeds. We therefore count only the synthetic seeds as independent evidence, and the statistical tests below use one value per stream.

B. Methods

We compared the proposed framework with seven baselines: a plain Hoeffding tree (VFDT); VFDT with ADWIN; VFDT with DDM; a reactive adaptation that resets on a detector alarm and uses no shadow model; a proactive variant with shadow-model training (PHT-S) that has no weighted fusion, profiling or scored selection; HAT [10]; and ARF [14]. ARF was run with **state ARF ensemble size and settings**. Leveraging Bagging, AUE, DDD and SRP were not run.

C. Metrics

We report prequential accuracy and macro-F1 [28], post-drift error (PDE, the error rate over the 100 samples after a true drift) and mean latency per sample in milliseconds. The benchmark also records recovery time and process memory. We do not report them: recovery times for some methods contradict their post-drift error (for example VFDT + DDM, 0.0 samples on the abrupt stream, and reactive adaptation, 3.2 samples on the incremental one), and we could not find the cause in the summary file; process memory is dominated by the Python process and the stream buffers (about 43 to 160 MB depending on the stream, almost equal across methods) and does not measure model size. Detection quality against ground truth (±250 samples) is available in the dashboard but is not reported here.

D. Statistical Analysis

Following Demšar [33], we ranked the methods by accuracy on each of the $N = 6$ streams, with ties given average ranks, and computed the Friedman statistic $\chi^2_F = 12N / [k(k+1)] \times [\Sigma\ R_j^2 - k(k+1)^2/4]$ for $k = 8$ methods with average ranks R_j [34], and the Iman–Davenport correction $F_F = (N-1)\chi^2_F / [N(k-1) - \chi^2_F]$. The Nemenyi test [35] uses the critical difference $CD = q_{0.05}\sqrt{[k(k+1)/(6N)]} = 3.031\sqrt{2} \approx 4.29$. We also compared the proposed framework with each baseline using the Wilcoxon signed-rank test [36] with Holm correction [37]. Only stream-level mean accuracies were used, so each test has six paired values.

E. Environment

The experiments ran from the project dashboard on a Windows machine (**CPU model and RAM**) with Python 3.11, River 0.21.2 and FastAPI 0.111.0.

VII. Results

A. Accuracy

Table II gives accuracy for every method and stream. ARF is the most accurate method on four of the six streams, the exceptions being the incremental stream, where VFDT + DDM is best (91.64%), and Elec2, where plain VFDT is best

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 11 , November - 2026

(88.47%). The proposed framework is second or third on five of the six streams and mid-table on the incremental one. It has the same accuracy as the proactive variant on three streams and differs from it by at most 0.67 points elsewhere (87.90% against 87.23% on Elec2). On the abrupt stream it reaches 93.07%, 1.71 points above VFDT, although ARF is higher at 95.08%.

TABLE II
PREQUENTIAL ACCURACY (%) PER STREAM, MEAN ± STD OVER 10 SEEDS (3000 SAMPLES PER RUN). BEST VALUE IN EACH COLUMN IN BOLD. ON ELEC2 AND PHISHING THE RUNS REPEAT FIXED DATA.

Method	Abrupt	Gradual	Incremental	Recurring	Elec2	Phishing	Mean
VFDT	91.36 ± 0.47	91.41 ± 0.46	89.96 ± 0.88	92.25 ± 0.61	**88.47 ± 0.00**	87.44 ± 0.00	90.15
VFDT + ADWIN	91.79 ± 0.89	91.49 ± 0.61	90.48 ± 0.86	92.25 ± 0.61	85.77 ± 0.00	87.44 ± 0.00	89.87
VFDT + DDM	93.02 ± 0.59	92.12 ± 1.03	**91.64 ± 0.49**	92.56 ± 0.85	86.03 ± 0.00	87.44 ± 0.00	90.47
Reactive adaptation	91.99 ± 1.03	91.57 ± 0.71	90.82 ± 0.50	92.25 ± 0.61	85.70 ± 0.00	87.44 ± 0.00	89.96
Proactive (PHT-S)	93.07 ± 0.41	92.91 ± 0.44	90.48 ± 0.56	92.98 ± 0.52	87.23 ± 0.00	87.60 ± 0.00	90.71
HAT	91.73 ± 1.04	91.67 ± 0.49	90.43 ± 0.34	92.36 ± 0.74	87.77 ± 1.18	87.60 ± 0.64	90.26
ARF	**95.08 ± 0.32**	**94.84 ± 0.35**	91.18 ± 0.52	**94.24 ± 0.43**	87.63 ± 0.00	**89.44 ± 0.00**	**92.07**
Proposed framework	93.07 ± 0.41	92.87 ± 0.44	90.48 ± 0.56	93.02 ± 0.51	87.90 ± 0.00	87.60 ± 0.00	90.82

Averaged over the six streams (Table III), the framework scores 90.82% accuracy and 60.12% macro-F1. That is about 0.7 points above VFDT, 0.96 above VFDT + ADWIN and 0.56 above HAT, and 1.25 points below ARF (92.07%). The framework and the proactive variant differ by only 0.11 points on average, so these results give no sign that weighted fusion, profiling or scored selection raise accuracy. Elec2 is the one stream where the detector-based methods lose accuracy: VFDT + ADWIN, reactive adaptation and VFDT + DDM fall 2.4 to 2.8 points below plain VFDT, and the proposed framework falls 0.57 points.

TABLE III
MEAN OVER THE SIX STREAMS. PDE: POST-DRIFT ERROR (LOWER IS BETTER). LATENCY IS THE MEAN TIME PER SAMPLE.

Method	Acc. (%)	F1 (%)	PDE	Lat. (ms)
VFDT	90.15	59.47	0.106	**0.44**
VFDT + ADWIN	89.87	59.31	0.105	0.47
VFDT + DDM	90.47	59.82	0.116	0.51
Reactive adaptation	89.96	59.39	0.095	0.52
Proactive (PHT-S)	90.71	60.04	0.089	0.92
HAT	90.26	59.65	0.108	0.74
ARF	**92.07**	**61.03**	**0.086**	2.80
Proposed framework	90.82	60.12	0.094	1.07

B. Drift Handling and Cost

Post-drift error averages 0.094 for the proposed framework, against 0.106 for VFDT (11% lower) and 0.086 for ARF. The proactive variant is slightly lower at 0.089. As explained in Section VI-C, we do not report recovery times.

The cost of the extra components shows in latency. The framework needs 1.07 ms per sample on average, which is 2.5 times VFDT (0.44 ms) and about 0.16 ms more than the proactive variant, but 2.6 times less than ARF (2.80 ms). This comparison depends on the implementations and settings used for each method, and ARF is an ensemble of several trees, so it should be read as a measurement of these particular runs and not as a general statement about cost.

C. Statistical Significance

The average ranks (Fig. 2) are 1.67 for ARF, 2.83 for the proposed framework, 3.33 for the proactive variant, 4.25 for VFDT + DDM, 5.00 for HAT, 5.92 for reactive adaptation, 6.42 for VFDT and 6.58 for VFDT + ADWIN; they sum to 36, as they should for 8 methods. The Friedman statistic is $\chi^2_F = 22.5$ (7 degrees of freedom, $p = 0.002$) and the Iman–Davenport statistic is $F_F = 5.77$ (7 and 35 degrees of freedom, $p = 0.0002$); a tie-corrected Friedman test gives 23.4 ($p = 0.0014$). All three reject equal performance.

The Nemenyi test is less decisive. With CD = 4.29, ARF is significantly better than VFDT (rank difference 4.75) and VFDT + ADWIN (4.91), but the proposed framework does not differ significantly from any method, the largest gap being 3.75 ranks from VFDT + ADWIN.

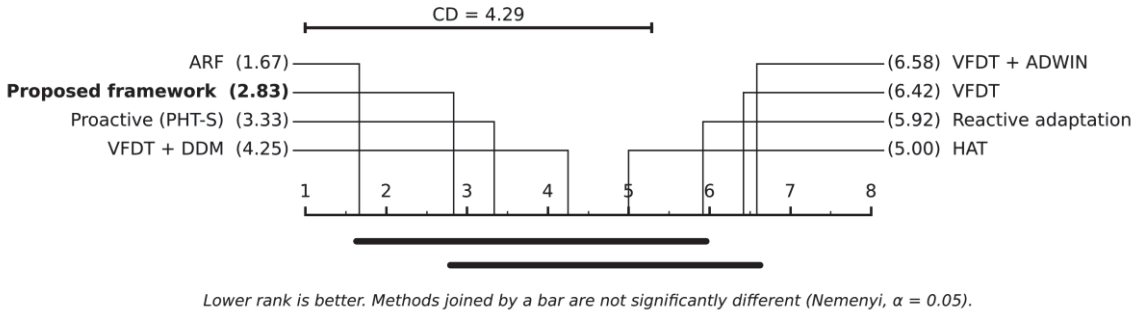


Fig. 2. Critical-difference diagram on accuracy (Nemenyi, α = 0.05, CD = 4.29) over six streams, drawn from the average ranks. Lower rank is better.

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 11 , November - 2026

TABLE IV
PROPOSED FRAMEWORK AGAINST EACH BASELINE: PAIRED WILCOXON SIGNED-RANK TEST OVER THE SIX STREAM-LEVEL MEAN ACCURACIES, HOLM-CORRECTED (7 COMPARISONS). W/T/L = STREAMS WON, TIED, LOST.

Baseline	ΔAcc (pp)	W/T/L	p	Holm p
VFDT + ADWIN	+0.96	6/0/0	0.031	0.219
HAT	+0.56	5/1/0	0.062	0.375
ARF	-1.25	1/0/5	0.062	0.375
Reactive adaptation	+0.86	5/0/1	0.094	0.375
VFDT	+0.68	5/0/1	0.156	0.469
VFDT + DDM	+0.36	5/0/1	0.312	0.625
Proactive (PHT-S)	+0.11	2/3/1	0.500	0.625

Table IV shows the pairwise tests. The proposed framework beat VFDT + ADWIN on all six streams (raw p = 0.031), which is the smallest value possible with six pairs, but after Holm correction no comparison is significant (smallest adjusted p = 0.219). With only six streams these tests have little power, so the honest summary is that the framework is competitive with the baselines and cheaper than ARF, without a statistically established accuracy advantage.

D. An Adaptation Example

Fig. 3 shows one adaptation from an interactive run. DDM raised a warning at sample 385 and confirmed the drift at sample 386 with a fused confidence of 90%. The delay of one sample is below θ_a = 250, so the drift was labelled abrupt. Accuracy fell from 100.0% to 89.6%, which is medium severity. The similarity to stored concepts was 0.0% because the concept memory held no earlier concept, so the drift counted as a first occurrence, and the scope was localised. Three rules were scored: window reset 0.80, proactive training 0.50 and conservative fallback 0.40. The window reset was chosen with a margin of 0.30 and the model was ready again after 0.045 s.

This example also shows a limit of the shadow model. The warning came one sample before the confirmation, so the shadow had at most one or two samples to learn from and could not have been a useful replacement; the selector chose the window reset instead. We did not record how many samples shadow models trained on before use, how often they were promoted, or how often each of the five strategies was chosen across the 480 runs, so we cannot say whether the scored selector uses more than a few of its strategies.

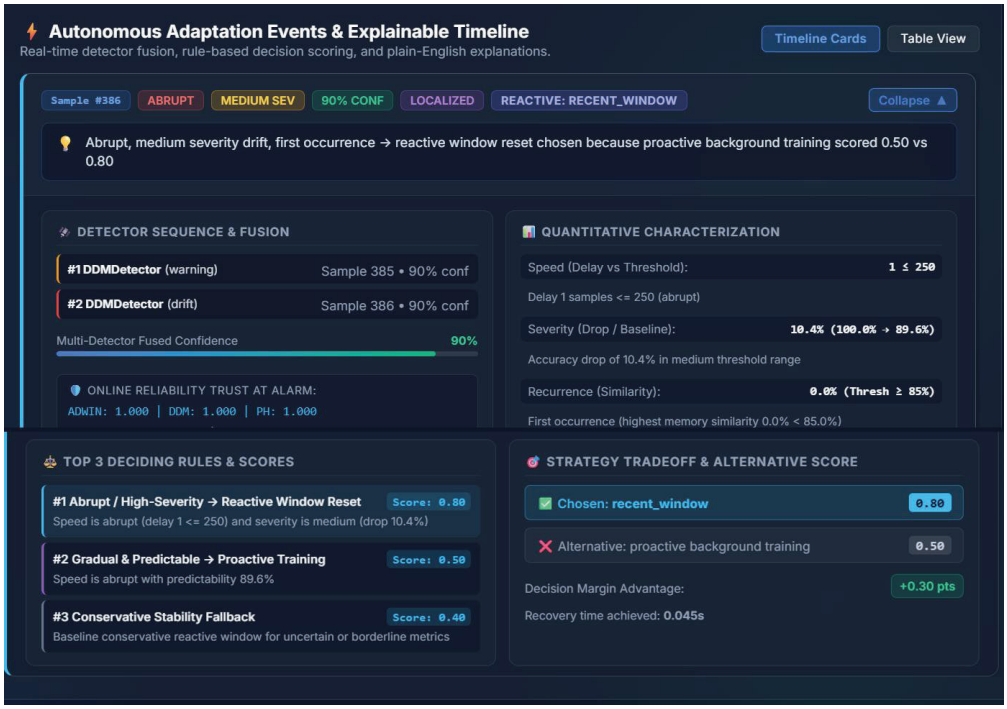


Fig. 3. An adaptation event as shown on the dashboard: detector sequence and trust, drift profile, the three best-scoring rules and the trade-off against the alternative.

E. Live Run and Detector Trust

Fig. 4 shows a live run on the mixed-drift synthetic stream, paused after 5,050 of 20,000 samples. Abrupt drifts are marked at samples 318 and 5000. After the first one, rolling accuracy falls to about 0.4 and returns to about 0.9 by sample 1100. At the pause the accuracy was 84.76% and macro-F1 55.14%, and prediction, learning and detection took 0.05, 0.10 and 0.03 ms. The detector weights had moved away from their starting value of 1.0: DDM stood at 1.936 and ADWIN and Page–Hinkley at 1.239, so the fusion rule had come to trust DDM most on this stream. This is one illustrative run and not a controlled comparison.

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 11 , November - 2026



Fig. 4. Live dashboard run: rolling accuracy with ground-truth drifts (orange) and detector trust scores (ADWIN 1.239, DDM 1.936, Page–Hinkley 1.239).

VIII. Discussion

The results support a modest claim. The framework gives accuracy close to the best baselines at less than half of ARF's per-sample cost in our setup, and it lowers post-drift error compared with VFDT and the reactive baselines. It does not beat ARF on accuracy, and its advantage over the other methods is not statistically significant on six streams. The framework also stores the scores behind each adaptation, which may help someone who has to justify why a model was changed, but we did not evaluate whether these records are correct or useful to an operator.

The most important gap is the missing ablation. The framework is almost identical in accuracy to the proactive variant, so our results do not show that weighted fusion, drift profiling, scored selection or the tuner add anything beyond the shadow model. Running the framework with each component removed (no shadow model, plain voting, a single fixed strategy, no tuner) is the first experiment we would add. Closely related gaps are the frequency with which each strategy is chosen, the training time and promotion rate of the shadow model, and the detection delay, false-alarm and missed-drift rates, all of which the dashboard can record but which we have not reported.

Other limits follow from the design of the experiments. Six streams of 3000 samples, only two of them real and those two replayed in fixed order, give little statistical power, and larger benchmarks such as Covertype, Airlines or Insects and longer runs would be more convincing. Leveraging Bagging, AUE, DDD and SRP were not compared. Many constants are hand-set and were not tested for sensitivity. The weights of the fusion rule are not bounded, and the confirmation step needs labels within about 100 samples; delayed labels were not tested. Feature attribution was not validated against drifts with known changed features, although the drift-injection panel could be used for that. The Kafka path was checked for function only, with a single node and partition and no load test. The learner is a Hoeffding tree throughout, and Elec2 is known to be a weak benchmark [32].

IX. Conclusion

We built a streaming framework that trains a shadow model during the warning phase, weights its detectors by how reliable they have been, profiles each drift and chooses a strategy by scored rules, with the scores stored for later inspection. Over 480 benchmark runs it ranked second behind ARF, reached 90.82% mean accuracy, and needed 1.07 ms per sample against 2.80 ms for ARF. Its accuracy gains over simpler baselines were small and not significant on six streams, and without an ablation we cannot attribute them to any one component. Next steps are the ablation, strategy and shadow-model statistics, detection metrics, delayed-label and sensitivity tests, a load test of the Kafka pipeline, and a larger benchmark with more real streams.

Acknowledgment

Thank your guide, institution and any funding source, or delete this section.

Reproducibility

Code, Docker Compose files, configuration and the benchmark script are at **repository URL**. The full benchmark is run with "python benchmark.py --seeds 10 --max-samples 3000 --output-dir results".

References

[1] J. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, and A. Bouchachia, "A survey on concept drift adaptation," *ACM Comput. Surv.*, vol. 46, no. 4, pp. 44:1–44:37, 2014.

[2] 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, no. 12, pp. 2346–2363, 2019.

[3] G. Ditzler, M. Roveri, C. Alippi, and R. Polikar, "Learning in nonstationary environments: A survey," *IEEE Comput. Intell. Mag.*, vol. 10, no. 4, pp. 12–25, 2015.

[4] A. Bifet and R. Gavaldà, "Learning from time-changing data with adaptive windowing," in *Proc. SIAM Int. Conf. Data Mining (SDM)*, 2007, pp. 443–448.

[5] J. Gama, P. Medas, G. Castillo, and P. Rodrigues, "Learning with drift detection," in *Advances in Artificial Intelligence – SBIA 2004*, ser. LNCS, vol. 3171. Springer, 2004, pp. 286–295.

[6] E. S. Page, "Continuous inspection schemes," *Biometrika*, vol. 41, no. 1/2, pp. 100–115, 1954.

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 11 , November - 2026

[7] P. M. Gonçalves Jr., S. G. T. C. Santos, R. S. M. Barros, and D. C. L. Vieira, "A comparative study on concept drift detectors," *Expert Syst. Appl.*, vol. 41, no. 18, pp. 8144–8156, 2014.

[8] P. Domingos and G. Hulten, "Mining high-speed data streams," in *Proc. ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2000, pp. 71–80.

[9] G. Hulten, L. Spencer, and P. Domingos, "Mining time-changing data streams," in *Proc. ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2001, pp. 97–106.

[10] A. Bifet and R. Gavaldà, "Adaptive learning from evolving data streams," in *Advances in Intelligent Data Analysis VIII (IDA)*, ser. LNCS, vol. 5772. Springer, 2009, pp. 249–260.

[11] R. Elwell and R. Polikar, "Incremental learning of concept drift in nonstationary environments," *IEEE Trans. Neural Netw.*, vol. 22, no. 10, pp. 1517–1531, 2011.

[12] D. Brzeziński and J. Stefanowski, "Reacting to different types of concept drift: The accuracy updated ensemble algorithm," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 25, no. 1, pp. 81–94, 2014.

[13] A. Bifet, G. Holmes, and B. Pfahringer, "Leveraging bagging for evolving data streams," in *Machine Learning and Knowledge Discovery in Databases (ECML PKDD)*, ser. LNCS, vol. 6321. Springer, 2010, pp. 135–150.

[14] H. M. Gomes *et al.*, "Adaptive random forests for evolving data stream classification," *Mach. Learn.*, vol. 106, no. 9–10, pp. 1469–1495, 2017, doi: 10.1007/s10994-017-5642-8.

[15] L. L. Minku and X. Yao, "DDD: A new ensemble approach for dealing with concept drift," *IEEE Trans. Knowl. Data Eng.*, vol. 24, no. 4, pp. 619–633, 2012.

[16] H. M. Gomes, J. Read, and A. Bifet, "Streaming random patches for evolving data stream classification," in *Proc. IEEE Int. Conf. Data Mining (ICDM)*, 2019, pp. 240–249, doi: 10.1109/ICDM.2019.00034.

[17] N. Littlestone and M. K. Warmuth, "The weighted majority algorithm," *Inf. Comput.*, vol. 108, no. 2, pp. 212–261, 1994.

[18] G. I. Webb, R. Hyde, H. Cao, H. L. Nguyen, and F. Petitjean, "Characterizing concept drift," *Data Min. Knowl. Discov.*, vol. 30, no. 4, pp. 964–994, 2016.

[19] I. Žliobaitė, M. Pechenizkiy, and J. Gama, "An overview of concept drift applications," in *Big Data Analysis: New Algorithms for a New Society*. Springer, 2016, pp. 91–114.

[20] N. Siddiqi, *Credit Risk Scorecards: Developing and Implementing Intelligent Credit Scoring*. Hoboken, NJ, USA: Wiley, 2006.

[21] F. J. Massey, Jr., "The Kolmogorov–Smirnov test for goodness of fit," *J. Amer. Statist. Assoc.*, vol. 46, no. 253, pp. 68–78, 1951.

[22] V. Cerqueira, H. M. Gomes, A. Bifet, and L. Torgo, "STUDD: A student–teacher method for unsupervised concept drift detection," *Mach. Learn.*, vol. 112, no. 11, pp. 4351–4378, 2023, doi: 10.1007/s10994-022-06188-7.

[23] F. Hinder, V. Vaquet, and B. Hammer, "One or two things we know about concept drift—a survey on monitoring in evolving environments. Part A: Detecting concept drift," *Front. Artif. Intell.*, vol. 7, Art. no. 1330257, 2024, doi: 10.3389/frai.2024.1330257.

[24] F. Hinder, V. Vaquet, and B. Hammer, "One or two things we know about concept drift—a survey on monitoring in evolving environments. Part B: Locating and explaining concept drift," *Front. Artif. Intell.*, vol. 7, Art. no. 1330258, 2024, doi: 10.3389/frai.2024.1330258.

[25] J. Kreps, N. Narkhede, and J. Rao, "Kafka: A distributed messaging system for log processing," in *Proc. NetDB Workshop*, 2011.

[26] J. Montiel *et al.*, "River: Machine learning for streaming data in Python," *J. Mach. Learn. Res.*, vol. 22, no. 110, pp. 1–8, 2021.

[27] A. Bifet, G. Holmes, R. Kirkby, and B. Pfahringer, "MOA: Massive online analysis," *J. Mach. Learn. Res.*, vol. 11, pp. 1601–1604, 2010.

[28] J. Gama, R. Sebastião, and P. P. Rodrigues, "On evaluating stream learning algorithms," *Mach. Learn.*, vol. 90, no. 3, pp. 317–346, 2013.

[29] W. N. Street and Y. Kim, "A streaming ensemble algorithm (SEA) for large-scale classification," in *Proc. ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2001, pp. 377–382.

[30] M. Harries, "SPLICE-2 comparative evaluation: Electricity pricing," Univ. New South Wales, Sydney, Australia, Tech. Rep. UNSW-CSE-TR-9905, 1999.

[31] R. M. Mohammad, F. Thabtah, and L. McCluskey, "Intelligent rule-based phishing websites classification," *IET Inf. Secur.*, vol. 8, no. 3, pp. 153–160, 2014.

[32] I. Žliobaitė, "How good is the Electricity benchmark for evaluating concept drift adaptation," arXiv:1301.3524, 2013.

[33] J. Demšar, "Statistical comparisons of classifiers over multiple data sets," *J. Mach. Learn. Res.*, vol. 7, pp. 1–30, 2006.

[34] M. Friedman, "The use of ranks to avoid the assumption of normality implicit in the analysis of variance," *J. Amer. Statist. Assoc.*, vol. 32, no. 200, pp. 675–701, 1937.

[35] P. B. Nemenyi, "Distribution-free multiple comparisons," Ph.D. dissertation, Princeton Univ., Princeton, NJ, USA, 1963.

[36] F. Wilcoxon, "Individual comparisons by ranking methods," *Biometrics Bull.*, vol. 1, no. 6, pp. 80–83, 1945.

[37] S. Holm, "A simple sequentially rejective multiple test procedure," *Scand. J. Statist.*, vol. 6, no. 2, pp. 65–70, 1979.

(This work is licensed under a Creative Commons Attribution 4.0 International License.)