Soft Computing Fusion with Applications

www.scfa.reapress.com

Soft. Comput. Fusion. Appl. Vol. 3, No. 3 (2026) 180–193.

Paper Type: Original Article

Adaptive Federated Stream Itemset Mining under Drift and Unequal Client Data Quality

Pritiprava Mishra[1], Ridwan Boya Marqas[2], Awaz Ahmed Shaban[3,*]

[1] Department of Computer Science and Engineering, GIFT Autonomous, Bhubaneswar, Odisha, India; snigdharanip@gift.edu.in.
[2] Department of Computer Science, College of Science, Knowledge University, Erbil, Iraq; ridwanmarqas@gmail.com.
[3] Department of Information Technology, Technical College of Informatics–Akre, Akre University for Applied Sciences, Duhok, Iraq; Awaz.Amedy1@gmail.com.

Citation:

Received: 11 September 2025 *Revised: 22 December 2025* *Accepted: 17 February 2026*	Mishra, P., Boya Marqas, R., & Ahmed Shaban, A. (2026). Adaptive federated stream itemset mining under drift and unequal client data quality. *Soft Computing Fusion with Applications, 3*(3), 180-193.

Abstract

Distributed transaction streams can change over time while client data quality and arrival volumes differ. We present AdaFed-TPM, a federated stream-itemset design with windowed counting, local change alarms, digest suppression, and consensus-based weighting. In the evaluated realization, clients mine unordered itemsets in transaction windows; temporal order and episode constraints are not modelled. A reported 10,000-transaction UCI Adult-derived stream contains six planted three-itemset groups, common drift times at transactions 3333 and 6666, and six clients with item-corruption probabilities between 2 and 20%. Across ten seeds, mean active-pattern recall is 0.988 versus 0.869 for periodic retraining (reported paired Wilcoxon W=0,p=0.0020), with 30.0% fewer uplink bytes than the fixed-window federation. The fixed-window recall of 0.983 is not significantly different; no equivalence test was performed. False re-discovery is higher, at 0.115 versus 0.033 for periodic retraining, and first-drift adaptation is slower than fixed-window refresh. Nine of ten ablation seeds have identical checkpoint outputs, leaving no reliable component-specific benefit established for quality weighting or event synchronization. Geometric decay, asynchronous site-specific drift, a stability floor, and privacy are untested design extensions.

Keywords: Federated stream itemset mining, Concept drift, Client heterogeneity, Adaptation delay, Uplink payload.

1 | Introduction

Distributed monitoring produces streams whose co-occurrence frequencies and sometimes event order change. Sequential and episode mining define ordered temporal objects [1], while drift research studies changing distributions [2], [3]. This campaign evaluates unordered transaction itemsets over time, rather than

Corresponding Author: Awaz.Amedy1@gmail.com
https://doi.org/10.22105/scfa.v3i3.109
Licensee System Analytics. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (http://creativecommons.org/licenses/by/4.0).

episodes or sequential patterns. Federated learning motivates exchanging derived statistics instead of pooled records [4], with communication as a design constraint [5].

Local changes and unequal data quality can complicate distributed stream counting. Fixed schedules may trade communication for stale estimates, while corrupted and legitimately heterogeneous clients may disagree for different reasons. A time-indexed pattern set therefore needs explicit support normalization, refresh rules, and retraction semantics. The campaign has shared global change times and unequal client arrival volumes; it does not independently vary asynchronous site drift.

The proposed design combines change-triggered refresh, consensus-based weighting, and forgetting. Reviewed federated methods motivate its components [6], [7], but the literature comparison does not prove that this combination is absent from every prior protocol. The reported campaign tests a window-based realization; its measured claims are distinguished from the general decayed-support design.

The contribution is a protocol proposal evaluated in one controlled, ten-seed campaign. Design hypotheses and measured comparisons are distinguished below so that the observed trade-offs and the remaining tests can be assessed separately.

A design formulation couples time-indexed support fidelity, stability, and communication. Exponential decay and a hard stability constraint are proposed options, not features validated by the windowed campaign.

The proposed event-triggered refresh combines local alarms with scheduled checkpoints and heartbeats. Its measured effect is limited to the itemset simulation in Section 5; message traffic does not cease between drift events.

The coordinator forms heuristic consensus-agreement scores and quadratic mixture weights. Agreement is not an independently measured probability of data correctness, particularly under legitimate population heterogeneity.

A message-accounting expression includes scheduled digests, heartbeats, and intra-epoch event responses. It bounds the declared uplink payload model rather than total network traffic or sublinear stream-length complexity.

The reported empirical campaign uses a 10,000-transaction UCI Adult-derived stream, common planted changes in six three-itemset groups, six unequal clients, and ten seeds. Fixed-window, periodic-retraining, and local-adaptive baselines and two component ablations allow descriptive contrasts in tracking recall, adaptation delay, re-discovery, and modeled uplink payload. The ablations do not establish an independent benefit from either added component.

Section 2 compares pattern mining, drift detection, and federated coordination. Sections 3 and 4 specify the design and its assumptions. Section 5 reports the windowed itemset campaign; Sections 6 and 7 state its limitations and development requirements.

2 | Related Work

2.1 | Temporal and Stream Pattern Mining

Ordered sequential mining [8], PrefixSpan [9], SPADE [10] and episode mining [1] define temporal pattern semantics. Apriori [11], FP-growth [12], adaptive stream model maintenance [13] and count-min summaries [14] motivate distinct counting and adaptation choices. The present campaign uses unordered itemsets in transaction windows, so its results do not evaluate ordered episodes or sequences.

The proposed forgetting design defines exponential decay, while the evaluated implementation uses fixed and adaptive sliding windows. These memory choices have distinct support semantics and neither guarantees candidate coverage or cross-client quality.

2.2 | Concept-Drift Detection and Adaptation

Concept drift and adaptation have been studied across changing streams [2], [3], [15]. DDM monitors online classification error for evidence of change [16], ADWIN tests differences between subwindows and adjusts its window [17], and change-detection methods for streams treat the sampling assumptions explicitly [18]. Page's CUSUM is a sequential-change method [19]. These approaches supply possible local alarm mechanisms; their nominal error control does not automatically transfer to the repeatedly inspected, temporally dependent itemset summaries evaluated here.

A detected change can motivate a window reset, candidate refresh or model update. Existing drift detectors and stream models [13] provide local mechanisms; this study asks how their reported outputs support distributed windowed itemsets. It compares event responses with scheduled refresh at the stated settings, without asserting absence of every prior federated adaptation method.

2.3 | Federated Learning under Heterogeneity and Reliability

Federated learning studies coordination [4], communication [5], heterogeneity [6], [7], [20], proximal local work [21], and adaptive server optimization [22]. Resource-based client selection [23] has a different purpose from data-driven change alarms. Label-noise studies [24] illustrate corrupted supervision; their results do not establish the validity of a consensus-agreement weight for event-count quality.

2.4 | Gap Synthesis

Table 1 positions the reviewed method families. AdaFed-TPM proposes event-triggered refresh and consensus weighting for time-indexed patterns. The measured realization uses unordered itemsets, window replacement, and carry-forward supports; ordered episodes, geometric forgetting, and a formal stability floor remain outside the evaluated evidence.

Table 1. Where AdaFed-TPM sits among the approach families.

Approach family	Representative Works	Drift Handling	Multi-Client	Output Semantics
Sequential/episode mining	[1], [8], [9], [10]	Static data	No	Pattern sets
Stream adaptation and sketches	[13], [14]	Adaptive models or approximate counts	No	Models or approximate frequency estimates
Drift detection & adaptation	[16-19]	Local detectors	Detector side only	Reset signals
Federated learning (models)	[4], [20-23]	Fixed schedules	Yes	Model parameters
AdaFed-TPM (this work)	Sections 3-4	Event-triggered, per client	Yes, quality-weighted	Windowed itemsets evaluated; stability floor untested

3 | Problem Formulation and Research Questions

3.1 | Research Questions and Hypotheses

RQ1 asks whether the reported event-response mechanism reduces modeled uplink payload relative to scheduled fixed-window refresh while tracking the same planted itemsets. Adaptation delay and re-discovery must be read alongside recall and bytes. The study includes no centralized drift-aware oracle and no complete

network-traffic accounting; its results answer this question only for the stated stream generator, baselines, and checkpoint budget.

RQ2 asks whether agreement-based client weights add measurable value when corruption and arrival volume differ. The two ablations compare variants at the tested settings; they neither independently identify true client quality nor separate corruption from volume. A minority client may disagree because its population is different, so the weights also require a bias analysis before use as a trust score.

RQ3 concerns whether an explicitly specified forgetting or expiry rule improves adaptation without excessive pattern churn. The executed campaign uses two fixed window lengths rather than the decayed-support design in *Eq. (1)*, and it does not report Jaccard-stability trajectories. Thus the decay-rate and stability questions remain prospective.

H1 asks whether event-responsive refresh can reduce uplink payload at acceptable adaptation delay relative to scheduled synchronization. Since heartbeats occur every checkpoint, sublinear total communication in stream length is not implied.

H2 asks whether consensus-based weighting improves discovery under quality degradation without suppressing genuinely different populations. This is an empirical hypothesis, not a variance-optimality or correctness guarantee.

H3 is a prospective hypothesis: an experimentally varied expiry or decay schedule may change the stability-delay trade-off. This campaign supplies no decay-rate sweep or evidence for the direction or existence of an optimum.

3.2 | System Model and Notation

Client i observes a timestamped stream over a shared item alphabet, and the coordinator maintains pattern set P_t at checkpoints. Data quality may change, and an agreement score in [0,1] serves as a heuristic rather than a ground-truth corruption probability. The design assumes reliable communication. The campaign uses six clients, a common global stream, common planted drift times, and a fixed volume-corruption gradient; it does not implement a privacy mechanism.

Table 2. Notation used throughout the manuscript.

Symbol	Meaning	Domain/Unit
m, i	Number of clients, client index	Integer
E_i	Local event stream of client i	Events
P_t	Global pattern set at epoch t	Patterns
P_i^loc(t)	Local pattern set of client i	Patterns
s_lambda(X, t)	Decayed support of pattern X at t	[0, 1]
lambda	Decay rate of support estimates	Per event-time
W, h_i	Detector window, per-client trigger threshold	Events / -
g_i(t)	Drift statistic of client i at t	-
Qhat_i(t)	Online reliability estimate of client i	[0, 1]
w_i(t)	Aggregation weight of client i	[0, 1]
R_t	Set of clients refreshing at epoch t	Subset of clients
J_t	Pattern-set stability index at t	[0, 1]
V(t)	Uplink payload in interval; excludes downlinks	bytes
b_pat, b_hb, b_hdr	Payload sizes: Pattern, heartbeat, header	bytes

3.3 | Formal Problem Statement

For a transaction window at epoch τ, let N_τ be its transaction count and $n_\tau(X)$ the number of transactions containing itemset X, with $0 \le n_\tau(X) \le N_\tau$. *Eq. (1)* weights occurrence counts and transaction exposure by the same recency factor, so its rate has transaction-level support units and is defined only for positive weighted exposure. For clients with different window sizes, aggregate the weighted occurrence and exposure masses before forming a pooled rate; averaging client rates alone changes the population target. An idle epoch with

$N_\tau = 0$ adds neither mass; in the absence of fresh exposure the normalized rate stays constant, even as both old masses decay. Retraction therefore needs an explicit zero-occurrence exposure window or a separate expiry rule. The reported campaign implements fixed and adaptive transaction windows, not this proposed exponential-decay design.

$$s_\lambda(X,t) = \frac{\sum_{\tau \le t} e^{-\lambda(t-\tau)}\, n_\tau(X)}{\sum_{\tau \le t} e^{-\lambda(t-\tau)}\, N_\tau}. \quad (1)$$

The design objective in *Eq. (2)* uses a hypothetical pooled drift-aware reference P^*_t and a stability floor. Such a reference would need explicit construction and authorization in a future evaluation; it was not computed in this campaign. The implemented windowed heuristic does not establish optimality or satisfaction of the complete objective.

$$\max \sum_{w=1}^{T} F_1(P_w, P_w^\star), \quad \text{subject to } \sum_t V(t) \le B_{total}, \quad J_t \ge J_{min}. \quad (2)$$

4 | Proposed Methodology: The AdaFed-TPM Protocol

4.1 | Architecture Overview

At checkpoints and local alarms, clients can refresh window summaries and the coordinator can update a carried-forward ledger. The general design below uses decayed support; Section 5 reports a windowed alternative with a long scheduled window and a short event window.

In the proposed decayed-support variant, a client would update candidate counts and exposure mass through *Eq. (1)*. In the evaluated variant, itemsets are mined from fixed or adaptive transaction windows.

The general design permits the statistic of *Eq. (3)*, subject to calibration. The reported implementation instead monitors vector summaries and pattern supports with its specified CUSUM alarms.

The proposed refresh rule sends digests after eligible alarms and heartbeats in inactive checkpoint rounds; Section 5 describes the implemented top-candidate event messages. The actual message trace is needed to audit charged bytes and carried-forward values.

The coordinator combines available or carried-forward summaries under the declared weighting rule. The reported campaign does not identify these weights with independent corruption probabilities, pooled population rates, or a privacy guarantee.

4.2 | Local Decayed Counting and Drift Detection

A bounded working set limits storage and update cost; it can miss patterns absent from local and global candidate routes. Maintaining k arbitrary itemsets requires at least their relevant membership tests per event unless an index or shared structure supplies a cheaper update. O(1) is meaningful only with a fixed k and fixed pattern length; it is not independent of candidate-set size. Count-min approximates queried counts [14] but does not itself enumerate unknown patterns.

The standardized two-window mean difference in *Eq. (3)* is one proposed change statistic, meaningful when its denominator is positive and its summary is scalar. If both variances vanish, define g=0 for equal means and g=+infinity for unequal means. A normal quantile alone does not control repeated sequential false alarms. The campaign uses a different vector-summary and per-pattern CUSUM with configured allowance and threshold [19], rather than a calibration of *Eq. (3)*.

$$g_i(t) = \frac{|\hat{\mu}_i(t) - \hat{\mu}_i(t-W)|}{\sqrt{\big(v_i(t) + v_i(t-W)\big)/W}}. \quad (3)$$

4.3 | Quality-Weighted Aggregation and Online Reliability

Agreement with a leave-one-out consensus is a heuristic proxy for consistency, not an independent quality measurement. Under Non-IID streams a correct minority client may legitimately disagree. *Eq. (4)* sharpens score differences with a quadratic weight, but no noise model shows this is inverse-variance or optimal. A population support estimate uses pooled occurrence and exposure masses; precision-mixture weighting intentionally changes that target.

$$w_i(t) = \frac{\widehat{Q}_i(t)^2}{\sum_j \widehat{Q}_j(t)^2}. \quad (4)$$

If all agreement scores are zero, use a declared fallback such as uniform weights among available clients; the implementation convention must be confirmed. Updating a score only when a digest arrives leaves a silent client's score stale; an EWMA does not automatically age it. The campaign compares top-15 probe supports with other received or carried-forward supports. These are already communicated values, but unchanged bytes do not constitute privacy protection, and carrying stale peer values can affect the agreement signal.

4.4 | Forgetting, Stability, and the Communication Bound

Decay of raw occurrence mass alone does not imply decay of normalized support when its denominator decays equally. An idle client requires an explicit zero-occurrence exposure window or support-expiry rule before stale patterns are retracted. *Eq. (5)* reports Jaccard stability, with J_t=1 when both sets are empty. Deferring removals does not guarantee J_t>=J_min when new additions enlarge the union; a hard constraint would need to validate the entire proposed set. The campaign does not log J_t or enforce a reported stability floor.

$$J_t = \frac{|P_t \cap P_{t-1}|}{|P_t \cup P_{t-1}|}. \quad (5)$$

Eq. (6) accounts for uplink payload in interval t, including scheduled refresh clients R_t, idle heartbeats, and all intra-interval event responses A_t. Each event response e has a responding client set C_e and itemset payload count k_ie. The expression excludes downlink broadcasts, transport and cryptographic headers, retries, and encoding overhead unless those are charged into the payload constants. Regular heartbeats alone require communication linear in the number of checkpoints, and arbitrary alarms may occur throughout a stream. Event-sensitive digest savings therefore do not establish a sublinear total-communication theorem. The event term sums each responding client's reported candidate payload and header over all alarms in the interval.

$$V_{up}(t) \le \sum_{i \in R_t} \left(k_{it} b_{pat} + b_{hdr}\right) + (m - |R_t|) b_{hb} + V_{event}(t). \quad (6)$$

Algorithm 1. AdaFed-TPM epoch at checkpoint t.

Input: streams E_i, decay lambda, window W, thresholds h_i,
payload sizes b_pat, b_hb, b_hdr, stability floor J_min
1: for every client i = 1..m in parallel do
2: update decayed local supports s_i,lambda(X, t) via Eq. (1)
3: maintain local pattern set P_i^loc(t) by decayed-support ranking
4: compute drift statistic g_i(t) via Eq. (3) on windowed summaries
5: if g_i(t) > h_i then compile digest (top patterns + evidence);
6: send digest to server and join refresh set R_t
7: else send heartbeat only
8: end for
9: server: update reliability Qhat_i(t) from digest agreement
10: compute weights w_i(t) via Eq. (4); merge decayed supports
11: prune global set by decayed-support floor; compute J_t via Eq. (5)
12: if a hard stability floor is required, validate the entire proposed set; otherwise report J_t
13: broadcast delta (P_t vs P_{t-1}) to all clients
Output: P_t, weights w_i(t), stability J_t, communication V(t)

4.5 | Complexity, Assumptions, and Failure Conditions

For a fixed working set, decayed updates use bounded work per event; candidate expansion and counting depend on working-set size and pattern length. Count-min update cost depends on sketch depth, not generically on log k [14]. A refresh includes itemset identifiers, support values, and evidence, while silent clients send heartbeats. Server ranking and reliability comparisons add work beyond summation. These are design costs; actual campaign tables report a simulated uplink byte model.

Several failure modes follow from the proposed mechanisms. Section 5 reports adaptation delay and dead-pattern re-discovery on the simulated planted changes. The other mechanisms below require targeted tests or a replay of individual alarms and digests before their frequency and causal role can be estimated.

A change invisible to the monitored summaries need not trigger a local alarm. The evaluated short and long windows can then continue to carry stale supports until refreshed; decay would require the separate implementation specified above.

Repeated false alarms can increase the event-response term in *Eq. (6)*. The source reports aggregate alarm counts, but without per-client timestamps and messages it does not isolate false-alarm payload.

Agreement-based weighting can favour clients already near the carried-forward consensus and suppress legitimate minority patterns. The source does not provide an independent probe of this feedback or show that the moving average prevents it.

A new item absent from current candidates requires a documented discovery or expansion path before its pattern can enter the global set. No bound on this delay follows from the reported experiment.

5 | Experimental Results and Analysis

5.1 | Setup

The reported experiment mines unordered itemsets from a synthetic transaction stream derived from the UCI Adult census corpus [25]. All 45,222 complete records are discretised into 23 binary items covering age band, education band, weekly-hours band, work class, sex, high income, capital gains and losses, US nativity, marital status, and occupation group. The background stream of 10,000 transactions resamples Adult baskets non-uniformly over three education strata whose mixture weights vary sinusoidally with stream position (amplitude 0.15, period 10,000), so a slow background drift rides on top of the planted events. Six disjoint three-itemset pattern groups are planted on this background: each group joins three mutually exclusive items of one categorical block (for example the three age bands), so its natural Adult support is below 0.05%, and

injected occurrences give each group a base support of 0.06 before drift. Two abrupt drift events change the planted supports: at t = 3,333 pattern P1 is damped by a factor of 0.3, P2 is amplified by 3.0, and P3 dies; at t = 6,666 the same triple of changes is applied to P4, P5, and P6, with P6 newly appearing. The stream is routed to six clients with arrival weights proportional to 0.5, 0.8, 1.1, 1.4, 1.7, and 2.0 (realised shares 6.7% to 26.7%), and data quality is inversely paired with volume: the highest-volume client corrupts 20% of its items and the lowest-volume client 2%, with each present item dropped with probability q and each absent item inserted with probability q/23.

Four federations are compared on identical seeded streams. The global fixed-window federation mines a shared sliding window of 1,000 transactions and merges client digests by pooled volume. The federated periodic-retrain baseline re-mines every checkpoint on the 500-transaction retrain interval and merges identically. The local-adaptive baseline gives each client a CUSUM-shrunk window (250 transactions after an alarm, regrowth by 250 per checkpoint up to 1,000) and merges unweighted. AdaFed-TPM runs the same per-client batch CUSUM as the local-adaptive baseline (local batches of 25 transactions, L1 deviation of the 23-dimensional item-frequency vector against a reference estimated from the first 200 local transactions and re-estimated after each alarm, allowance k = 1.75, threshold h = 2.0, cooldown 100 local transactions, plus a per-pattern support CUSUM over the client's 25 monitored itemsets) and implements the reported combination of: quality-weighted aggregation with weights equal to the square of a reliability EWMA (blending 0.7/0.3) of the agreement between each digest's top-15 probe itemsets and the other clients' reported supports; a two-timescale window (a scheduled long window of 1,000 transactions for stability and an event window of 250 for candidates that moved near a drift); and drift-triggered immediate synchronisation of the top-50 candidates by relative support move, with 20-byte heartbeats in place of full digests whenever a client's pattern set overlaps its last sent digest by Jaccard at least 0.75 and no alarm fired, the server carrying silent clients' last supports forward. Two ablations remove the quality weighting (uniform weights) and the drift-triggered synchronisation. All mining uses a shared compact bitmask Apriori [11] over itemsets of at most three items, local minimum count 3, and a global minimum support rate of 0.008; the coordinator checkpoint interval is 500 transactions, giving 20 checkpoints per stream.

The source reports tracking recall, drift-response delay in checkpoint units, dead-pattern re-discovery, and a declared uplink accounting of 32 bytes per itemset and 20 per heartbeat, including event-sync responses. Mean and standard deviation summarize ten seeded streams. The source reports Python 3.12, NumPy [26], SciPy [27], statsmodels [28], Matplotlib [29], and 55.5 s runtime. Analysis timing, exact payload records, scripts, raw trajectories and artifacts are required to confirm these reports; preregistration is not documented.

5.2|Main Comparison (RQ1): Tracking, Delay, and Bytes

Table 3 reports mean recall 0.988 for AdaFed-TPM versus 0.983 for fixed-window, 0.929 for local adaptive windows, and 0.869 for periodic retraining. The reported paired Wilcoxon contrast with retraining is W=0,p=0.0020 over ten seeds. The fixed-window contrast has Holm p=0.112, which supplies no evidence of a difference but does not establish equivalence. Uplink payload is 1066974 versus 1525188 bytes for fixed-window, a 30.0% reduction, and about 1.6% higher than retraining's 1050272 bytes. These are simulation payload comparisons, not total network-traffic measurements.

Table 3. Main comparison over 10 seeds (mean ± std): Tracking recall of active planted patterns, false re-discovery rate, adaptation delay after each drift event, and uplink bytes.

Method	Tracking Recall	False Re-Discovery	Delay, Drift 1 (Windows)	Delay, Drift 2 (Windows)	Uplink Bytes
Global fixed window (w = 1,000)	0.983 ± 0.017	0.078 ± 0.026	2.90 ± 0.30	1.00 ± 0.00	1,525,188 ± 13,091
Federated periodic retrain (w = 500)	0.869 ± 0.017	0.033 ± 0.011	3.80 ± 2.04	1.20 ± 0.60	1,050,272 ± 9,967
Local adaptive windows (CUSUM)	0.929 ± 0.023	0.052 ± 0.018	4.00 ± 1.90	1.20 ± 0.60	1,233,408 ± 32,583
AdaFed-TPM (proposed)	0.988 ± 0.013	0.115 ± 0.048	4.00 ± 1.10	1.00 ± 0.00	1,066,974 ± 74,985

Fig. 1 reports per-checkpoint trajectories. Periodic retraining mean recall falls to 0.825 at the second checkpoint after the first drift and reaches a minimum 0.720. The damped planted support 0.018 is 2.25 times the 0.008 pruning threshold; short-window sampling variability is a possible explanation, not a demonstrated causal decomposition. Local adaptation reaches minimum 0.775, while AdaFed-TPM and fixed-window remain at least 0.950 and 0.900 in the reported means.

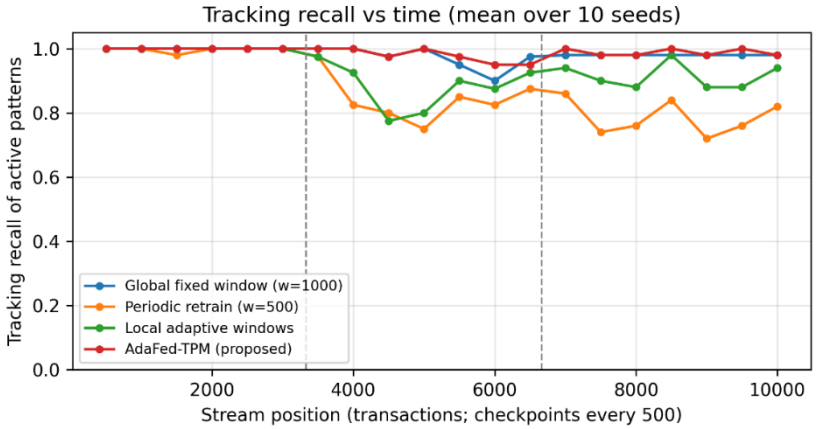

Fig. 1. Tracking recall of active planted patterns per checkpoint (mean over 10 seeds) for the four federated methods.

Dashed vertical lines mark the two drift events (t = 3,333 and t = 6,666); checkpoints occur every 500 transactions (*Fig. 1*).

After the first drift, fixed-window has mean adaptation delay 2.90 checkpoint windows, retraining 3.80, local adaptive windows 4.00 with SD 1.90, and AdaFed-TPM 4.00 with SD 1.10. Retraction of P3 is part of this metric, so carried-forward supports can prolong delay. After the second drift, both AdaFed-TPM and fixed-window have delay 1.00 with SD 0.00, while retraining and local adaptive windows have 1.20 with SD 0.60. The reported table therefore shows a tie, not a uniquely fastest proposed method.

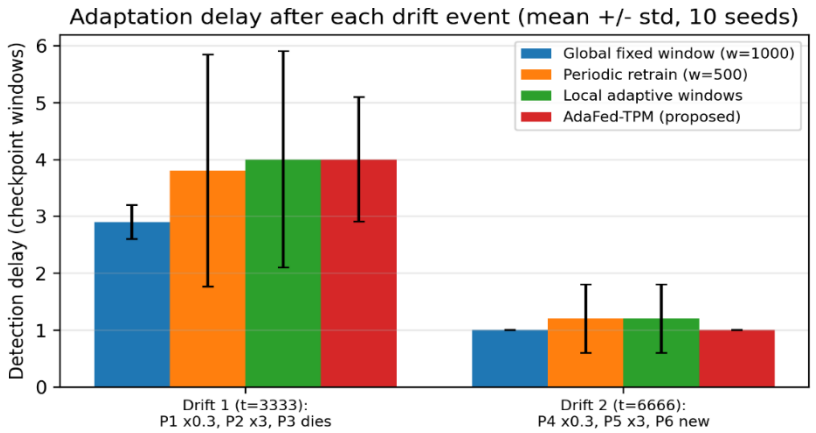

Fig. 2. Adaptation delay in checkpoint windows after each drift event (mean ± std over 10 seeds).

Delay counts checkpoints until every drift-affected pattern's reported status matches its post-drift status; drift 1 therefore penalises the retraction of the dead pattern P3, while drift 2 requires only the pickup of live patterns.

The reported dead-pattern re-discovery rate is 0.115 for AdaFed-TPM versus 0.033 for retraining, about 3.5 times as large; the paired source test favours retraining (W=0,p=0.0020). A carried-forward support can retain a dead pattern until refreshed. Fixed-window has the shortest first-drift delay, but its mean false re-discovery

of 0.078 exceeds retraining's 0.033; these outcomes describe distinct aspects of forgetting rather than a single universally fastest method.

5.3 | Ablation: Quality Weighting and Drift-Triggered Sync

Table 4 reports nearly identical checkpoint metrics for the full method and both ablations: nine of ten seeds are identical, and one favours an ablation by 0.010 recall. The reported p=0.317 is not an exact two-sided signed-rank value for one nonzero pair, whose exact p is 1; zero handling and approximation require raw-data reconciliation. Event-sync payload adds 204614 bytes relative to the no-sync variant, about 23.7% of 862360 bytes. The campaign provides no reliable measured improvement from either component at this scale. Distance from the pruning threshold is a possible explanation, not an isolated causal result or a theorem excluding weighting benefits.

Table 4. Reported ablation means and standard deviations over ten seeds.

Variant	Tracking Recall	False Re-Discovery	Uplink Bytes	Wilcoxon p vs Full (Recall)
AdaFed-TPM (full)	0.988 ± 0.013	0.115 ± 0.048	1,066,974 ± 74,985	-
Uniform weights (no quality weighting)	0.989 ± 0.014	0.115 ± 0.048	862,360 ± 49,225	0.317
No drift-triggered sync	0.989 ± 0.014	0.115 ± 0.048	862,360 ± 49,225	0.317

Note: Nine pairs are tied; the source p=0.317 requires reconciliation with the stated signed-rank procedure before inferential interpretation.

5.4 | Robustness and Stress Behaviour

Table 5 summarizes the same ten-seed campaign. AdaFed-TPM recall ranges from 0.963 to 1.000, and its lowest mean checkpoint recall is 0.950, versus 0.720 for retraining and 0.775 for local adaptation. Reported alarms decrease from 6.20 for the cleanest to 0.50 for the noisiest client, with one intermediate inversion. By the setup, the noisiest client has the highest arrival weight and therefore more, not fewer, transactions. Local batches, baselines, corruption and arrival volume remain confounded; raw alarm traces and a volume-by-corruption factorial design are required to explain the trend. Mean event synchronizations are reported as 23.40 per seed, outside scheduled checkpoint refreshes.

Table 5. Robustness summary per method (from the 10-seed campaign): Per-seed recall range, worst mean checkpoint, and mean recall at the first checkpoint after each drift event.

Method	Per-Seed Recall Range	Worst Checkpoint	Recall At Ck 7 (T = 3,500)	Recall At Ck 14 (T = 7,000)
Global fixed window (w = 1,000)	0.940 – 1.000	0.900	1.000	0.980
Federated periodic retrain (w = 500)	0.845 – 0.893	0.720	0.975	0.860
Local adaptive windows (CUSUM)	0.898 – 0.978	0.775	0.975	0.940
AdaFed-TPM (proposed)	0.963 – 1.000	0.950	1.000	1.000

5.5 | Statistical Analysis

The source reports paired Wilcoxon tests over ten seeds [30], Friedman chi-square(3)=28.6,p=2.7e-06 across four methods [31], and Holm correction across three recall comparisons [32]. The fixed-window p=0.112 supplies no evidence of a difference but does not establish equivalence. The ten random seeds are repeats under one Adult-derived generator and shared planted-change schedule; checkpoints and overlapping windows within a run are temporally dependent, not additional independent samples. The ablation has only one non-tied pair, for which an exact two-sided signed-rank p is 1. The reported p=0.317 remains unresolved until the raw paired values and software settings are supplied.

Table 6. Inferential summary (paired Wilcoxon signed-rank on tracking recall unless noted; Holm correction over the three-baseline family).

Comparison	Metric	W	p	p (Holm)	Significant at 0.05
AdaFed-TPM vs periodic retrain (primary)	Tracking recall	0.00	0.0020	0.0059	Yes
AdaFed-TPM vs fixed window	Tracking recall	3.00	0.1118	0.112	No
AdaFed-TPM vs local adaptive	Tracking recall	0.00	0.0020	0.0059	Yes
Four methods (Friedman)	Tracking recall	chi2(3) = 28.6	2.7e-06	-	Yes
AdaFed-TPM vs periodic retrain	False re-discovery	0.00	0.0020	-	Yes (baseline better)
AdaFed-TPM vs uniform-weight ablation	Tracking recall	0.00	0.317	-	No
AdaFed-TPM vs no-sync ablation	Tracking recall	0.00	0.317	-	No

5.6 | Discussion: Findings, Losses, and Scope

The reported realization reduces uplink payload relative to fixed-window while having similar observed mean recall, with no significant difference established. It does not show equally accurate performance in a formal equivalence sense. Retraction after the first drift is slower, and second-drift delay ties fixed-window. Ablations establish no measured component improvement at this scale; they do not causally prove that windowing alone generates every observed benefit. H1 has support for payload reduction at one design point, while a general delay or communication guarantee remains open.

The proposed method has the highest dead-pattern re-discovery mean and jointly slowest first-drift mean delay. H2 receives no reliable ablation support, and H3 is untested because the campaign substitutes fixed two-timescale windows for geometric decay. Jaccard stability was not logged. The scope is one Adult-derived generator, a 23-item alphabet, unordered itemsets of size at most three, 10000 transactions, six clients, common planted global drift times, one volume-corruption gradient, ten seeds, no centralized drift-aware oracle, and no privacy layer.

Independent regeneration of the reported campaign requires the original generator and analysis code, exact seeds and partitions, raw checkpoint outputs, message traces, and figure workflow. The manuscript does not identify an accessible versioned release.

6 | Discussion: Limitations and Counterarguments

6.1 | Limitations

Mean-based detection can miss changes in variance, event order or rare-tail combinations. Consensus weighting may suppress legitimate minority populations, stale supports can persist without expiry, and the proposed stability rule does not by itself enforce its floor. Uplink payload accounting omits downlinks and production overhead. The measured trade-off includes higher dead-pattern re-discovery, while privacy, async local drift, geometric decay and ordered episode mining remain untested.

6.2 | Alternative Viewpoints and Responses

Scheduled refresh provides a simple auditable baseline; event-responsive operation is useful when its savings justify alarm handling and stale-value risk. Regular heartbeats and event responses remain part of the cost. Reliability weighting offers an alternative to exclusion, but agreement-based weights need validation against independent quality evidence and legitimate population differences. *Eq. (2)* states a design objective rather than a proof that the implemented heuristic prevents trivial stability or satisfies its constraints.

Offline pooled analysis can be a useful alternative when timeliness and data access permit it. Here the simulation evaluates timely distributed refresh at one scale: payload savings are observed, while dead-pattern retraction remains a weakness.

7 | Conclusion and Future Directions

AdaFed-TPM evaluates adaptive windowed itemset discovery with digest suppression and change responses in a distributed simulation. Mean recall is 0.988 versus 0.983 for fixed-window and 0.869 for periodic retraining, at 30.0% fewer uplink bytes than fixed-window; the non-significant fixed-window contrast does not prove equivalence. First-drift delay and dead-pattern re-discovery are worse than relevant scheduled baselines. Weighting and event synchronization have no reliably established benefit in the reported ablations. The experiment uses unordered co-occurrences and common global drift times, leaving geometric decay, asynchronous site drift, stability constraints and privacy for further evaluation.

Further work should supply and reconcile raw runs, message traces and signed-rank settings, then add a validated stale-support expiry or retraction mechanism. It should independently vary corruption and arrival volume, assess minority-population bias, place pattern supports near decision boundaries, and test geometric decay and a complete stability constraint. Natural streaming benchmarks, true asynchronous client changes, ordered temporal semantics, larger alphabets, and a privacy layer require distinct experiments and analysis.

Author Contributions

P. M.: conceptualization, methodology, software, and writing—original draft. R. B. M.: data curation, formal analysis, and visualization. A. A. Sh.: validation, supervision, and writing—review and editing. All authors read and approved the final manuscript.

Funding

This research received no specific grant from any funding agency in the public, commercial, or not-for-profit sectors.

Data Availability

UCI Adult is public [25]. The stream resampling, planted itemsets, drift schedule, partitions and corruption profiles require the seeded generation procedure and raw output exp_24_results.json.

Code Availability

Reproduction requires exp_24.py, exp_24_results.json, the seed list, checkpoint and message traces, and the original figure-generation workflow. A stable repository or attached supplement is not identified in the manuscript.

Conflicts of Interest

The authors declare no conflicts of interest.

Generative AI Disclosure

AI-assisted drafting tools were used in preparing the manuscript text under the direction and scientific responsibility of the authors; all methodological choices, claims, experimental results, and final content were determined and verified by the authors.

References

[1] Mannila, H., Toivonen, H., & Inkeri Verkamo, A. (1997). Discovery of frequent episodes in event sequences. *Data mining and knowledge discovery*, *1*(3), 259–289. https://doi.org/10.1023/A:1009748302351

[2] Gama, J., Žliobaitė, I., Bifet, A., Pechenizkiy, M., & Bouchachia, A. (2014). A survey on concept drift adaptation. *ACM computing surveys (CSUR)*, *46*(4), 1-37. https://doi.org/10.1145/2523813

[3] Gama, J., & Zhang, G. (2019). Learning under concept drift: A review. *IEEE transactions on knowledge and data engineering*, *31*(12). https://doi.org/10.1109/TKDE.2018.2876857

[4] McMahan, H. B., Moore, E., Ramage, D., Hampson, S., & Arcas, B. A. y. (2017). Communication-efficient learning of deep networks from decentralized data. *The 20th international conference on artificial intelligence and statistics* (pp. 1273–1282). PMLR. https://proceedings.mlr.press/v54/mcmahan17a.html

[5] Konečný, J., McMahan, H. B., Yu, F. X., Richtárik, P., Suresh, A. T., & Bacon, D. (2016). *Federated learning: Strategies for improving communication efficiency*. https://doi.org/10.48550/arXiv.1610.05492

[6] Zhao, Y., Li, M., Lai, L., Suda, N., Civin, D., & Chandra, V. (2018). *Federated learning with non-iid data*. https://doi.org/10.48550/arXiv.1806.00582

[7] Kairouz, P., & McMahan, H. B. (2021). Advances and open problems in federated learning. *Foundations and trends® in machine learning* (pp. 1–210). Now Publishers Inc. https://doi.org/10.1561/2200000083

[8] Agrawal, R., & Srikant, R. (1995). Mining sequential patterns. *Proceedings of the eleventh international conference on data engineering* (pp. 3–14). IEEE. https://doi.org/10.1109/ICDE.1995.380415

[9] Han, J., Pei, J., Mortazavi-Asl, B., Pinto, H., Chen, Q., & Dayal, U. (2001). Prefixspan: mining sequential patterns efficiently by prefix-projected pattern growth. *Proceedings of the 17th international conference on data engineering* (pp. 215–224). IEEE, Piscataway, NJ, USA. https://doi.org/10.1109/ICDE.2001.914830

[10] Zaki, M. J. (2001). SPADE: An efficient algorithm for mining frequent sequences. *Machine learning*, *42*(1), 31–60. https://doi.org/10.1023/A:1007652502315

[11] Agarwal, R., & Srikant, R. (1994). Fast algorithms for mining association rules. *Proc. of the 20th vldb conference* (Vol. 487, p. 499). Morgan Kaufmann Publishers Inc. https://www.researchgate.net/publication/2460430_Fast_Algorithms_for_Mining_Association_Rules

[12] Han, J., Pei, J., & Yin, Y. (2000). Mining frequent patterns without candidate generation. *Proceedings of the 2000 acm sigmod international conference on management of data* (pp. 1–12). Association for Computing Machinery. https://doi.org/10.1145/342009.335372

[13] Hulten, G., Spencer, L., & Domingos, P. (2001). Mining time-changing data streams. *Proceedings of the seventh ACM sigkdd international conference on knowledge discovery and data mining* (pp. 97–106). Association for Computing Machinery. https://doi.org/10.1145/502512.502529

[14] Cormode, G., & Muthukrishnan, S. (2005). An improved data stream summary: The count-min sketch and its applications. *Journal of algorithms*, *55*(1), 58–75. https://doi.org/10.1016/j.jalgor.2003.12.001

[15] Widmer, G., & Kubat, M. (1996). Learning in the presence of concept drift and hidden contexts. *Machine learning*, *23*(1), 69–101. https://doi.org/10.1007/BF00116900

[16] Gama, J., Medas, P., Castillo, G., & Rodrigues, P. (2004). Learning with drift detection. *Brazilian symposium on artificial intelligence* (pp. 286-295). Berlin, Heidelberg: Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-540-28645-5_29

[17] Bifet, A., & Gavalda, R. (2007). Learning from time-changing data with adaptive windowing. *Proceedings of the 2007 SIAM international conference on data mining* (pp. 443-448). Society for Industrial and Applied Mathematics. https://doi.org/10.1137/1.9781611972771.42

[18] Kifer, D., Ben-David, S., & Gehrke, J. (2004). Detecting change in data streams. *VLDB* (Vol. 4, pp. 180–191). University of Waterloo. https://cs.uwaterloo.ca/~shai/vldb04.pdf

[19] Page, E. S. (1954). Continuous inspection schemes. *Biometrika*, *41*(1/2), 100–115. https://doi.org/10.1093/biomet/41.1-2.100

[20] Li, T., Sahu, A. K., Talwalkar, A., & Smith, V. (2020). Federated learning: Challenges, methods, and future directions. *IEEE signal processing magazine*, *37*, 50–60. https://doi.org/10.1109/MSP.2020.2975749

[21] Li, T., Sahu, A. K., Zaheer, M., Sanjabi, M., Talwalkar, A., & Smith, V. (2020). Federated optimization in heterogeneous networks. *Proceedings of machine learning and systems*, *2*, 429–450. https://proceedings.mlsys.org/paper/2020/hash/1f5fe83998a09396ebe6477d9475ba0c-Abstract.html

[22] Reddi, S., Charles, Z., Zaheer, M., Garrett, Z., Rush, K., Konečný, J., ... & McMahan, H. B. (2020). *Adaptive federated optimization*. https://doi.org/10.48550/arXiv.2003.00295

[23] Nishio, T., & Yonetani, R. (2019). Client selection for federated learning with heterogeneous resources in mobile edge. *ICC 2019-2019 IEEE international conference on communications (ICC)* (pp. 1–7). IEEE. https://doi.org/10.1109/ICC.2019.8761315

[24] Frénay, B., & Verleysen, M. (2013). Classification in the presence of label noise: A survey. *IEEE transactions on neural networks and learning systems*, *25*(5), 845–869. https://doi.org/10.1109/TNNLS.2013.2292894

[25] Kohavi, R., & Becker, B. (1996). Adult data set. *UCI machine learning repository*, *5*(2093), 25. https://doi.org/10.24432/C5XW20

[26] Harris, C. R., Millman, K. J., Van Der Walt, S. J., Gommers, R., Virtanen, P., & Cournapeau, D. (2020). Array programming with NumPy. *Nature*, *585*(7825), 357–362. https://doi.org/10.1038/s41586-020-2649-2

[27] Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., & Cournapeau, D. (2020). SciPy 1.0: fundamental algorithms for scientific computing in Python. *Nature methods*, *17*(3), 261–272. https://doi.org/10.1038/s41592-019-0686-2

[28] Seabold, S., & Perktold, J. (2010). Statsmodels: econometric and statistical modeling with python. *Scipy*, *7*(1), 92–96. https://doi.org/10.25080/Majora-92bf1922-011

[29] Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. *Computing in science & engineering*, *9*(3), 90–95. https://doi.org/10.1109/MCSE.2007.55

[30] Wilcoxon, F. (1945). Individual comparisons by ranking methods. *Biometrics bulletin*, *1*, 80. https://doi.org/10.2307/3001968

[31] Friedman, M. (1937). The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *Journal of the american statistical association*, *32*, 675–701. https://doi.org/10.1080/01621459.1937.10503522

[32] Holm, S. (1979). A simple sequentially rejective multiple test procedure. *Scandinavian journal of statistics*, *6*(2), 65–70. https://www.jstor.org/stable/4615733