scientific reports

Check for updates

OPEN

FL-MalDrift: a federated learning framework for malware detection under local concept drift

Amit Patel✉, Deepak Singh Tomar, R. K. Pateriya & Rahul Haripriya

Android malware evolves continuously, inducing concept drift that erodes the accuracy of learned detectors a challenge intensified in federated learning (FL), where non-IID and asynchronously shifting client data can destabilize aggregation despite privacy preservation. To address this limitation, this study proposes FL-MalDrift, a drift resilient federated framework integrating lightweight on device drift detection (ADWIN, DDM, EDDM, HDDM) with adaptive participation control. Each client mitigates local drift before contributing updates, while a server side controller regulated through exponentially weighted moving average (EWMA) smoothed drift scores selectively aggregates stable updates using FedAvg or FedSGD. Experiments on benchmark Android malware datasets demonstrate that FL-MalDrift achieves 94.7% accuracy on Drebin, 96.8% on CICMalDroid 2020, and 92.4% on a chronological AndroZoo split, with modest overhead. The framework stabilizes training under client heterogeneity, filtering drift-affected updates while maintaining privacy. By coupling client side drift detection with dynamic participation control, FL-MalDrift establishes a scalable and privacy preserving foundation for robust malware detection in non-stationary environments. Future work will integrate calibrated differential privacy budgets, compression-aware aggregation, and large scale device validation.

Keywords Federated learning, Concept drift, Malware detection, Client-side adaptation, Drift detection methods

The escalating sophistication and volume of malware strain contemporary cybersecurity infrastructure. In 2024, Kaspersky reported an average of *467,000* malicious files detected per day (a 14% year-over-year increase), including a 33% surge in Trojans and a 150% rise in Trojan-droppers[1]. Independently, the AV-TEST Institute registered more than *450,000* new malicious programs and potentially unwanted applications daily[2]. This sheer velocity of change creates a moving target for learning-based defenses.

Machine learning (ML) detectors, while effective at generalizing from historical patterns, are highly susceptible to *concept drift*: shifts in the joint distribution $P(X, y)$ that render previously learned decision boundaries suboptimal. In practice, drift is the norm rather than the exception: novel attack vectors, obfuscation strategies, and behavioral variants continuously alter the statistical properties of malware data, degrading model performance over time. Recent studies show that deep models suffer marked accuracy drops under drift unless active adaptation is employed[3].

In federated learning (FL), where models are trained across distributed clients without centralizing raw data, handling drift is even more challenging. Client data are non-IID, evolve asynchronously, and reflect local behaviors that may diverge substantially across devices and over time. Classical FL protocols such as FedAvg implicitly assume stationary or slowly varying distributions across rounds, an assumption that is routinely violated in malware detection. As a result, unstable client updates can contaminate aggregation, accelerating global performance decay.

While federated learning has been explored for Android malware detection, existing approaches generally assume stationary local data distributions and lack mechanisms to explicitly handle client-side concept drift. As a result, unstable or drift-affected updates can propagate to the global model, degrading its accuracy and stability in evolving threat landscapes. To address this gap, we propose **FL-MalDrift**, a federated learning framework that integrates local drift detection with dynamic participation control to ensure that only stable, drift-mitigated client contributions influence global aggregation.

Department of Computer Science and Engineering, Maulana Azad National Institute of Technology, Bhopal, M.P. 462003, India. ✉email: amit.manit007@gmail.com

Our system comprises two scopes (Fig. 1). On the *client* side, a local classifier, drift detector, and mitigation engine operate entirely on-device. Updates are encoded with an optional differential-privacy layer before transmission. On the *server* side, a drift-threshold evaluator screens incoming updates; a secure aggregator (FedAvg/FedSGD) admits only stable contributions and rolls back unstable ones. This design supports asynchronous, privacy-preserving learning while sustaining robustness at scale. A full description appears in Sect. 4.

To ensure long-term reliability under these conditions, drift detection and mitigation must be enacted *at the client* before aggregation. Prior work has largely focused on global adaptation, leaving a gap in local, privacy-preserving drift handling that prevents unstable updates from contaminating the global model. In this work, we propose a drift resilient FL framework for Android malware detection in which each client monitors its local stream with lightweight statistical detectors, adapts its model upon drift, and then participates in aggregation. This design improves accuracy and stability under non stationarity while preserving privacy by avoiding transmission of raw data or explicit drift signals.

Figure 2 illustrates the online drift detection framework adapted from Gemaque et al.[4]. This framework represents the local drift detection mechanism each client employs, continuously receiving data, maintaining reference and detection windows, computing dissimilarity measurements, and executing statistical significance tests to detect drift.

To the best of our knowledge, *FL-MalDrift* is the first federated malware detection framework that jointly integrates concept drift detection, dynamic client participation, and privacy preservation for Android environments, providing a deployable and resilient solution to evolving malware threats.

Research challenges

Despite progress in FL-based malware detection, several challenges remain: (i) *client side concept drift* injects unstable updates into aggregation under non-IID, asynchronous data; (ii) *lack of adaptive participation* fixed client policies ignore time varying stability and propagate drift; (iii) *accuracy efficiency trade off* under non stationarity, where communication/energy budgets must be respected; and (iv) *privacy constraints* require on device drift handling without exposing raw data or explicit drift signals.

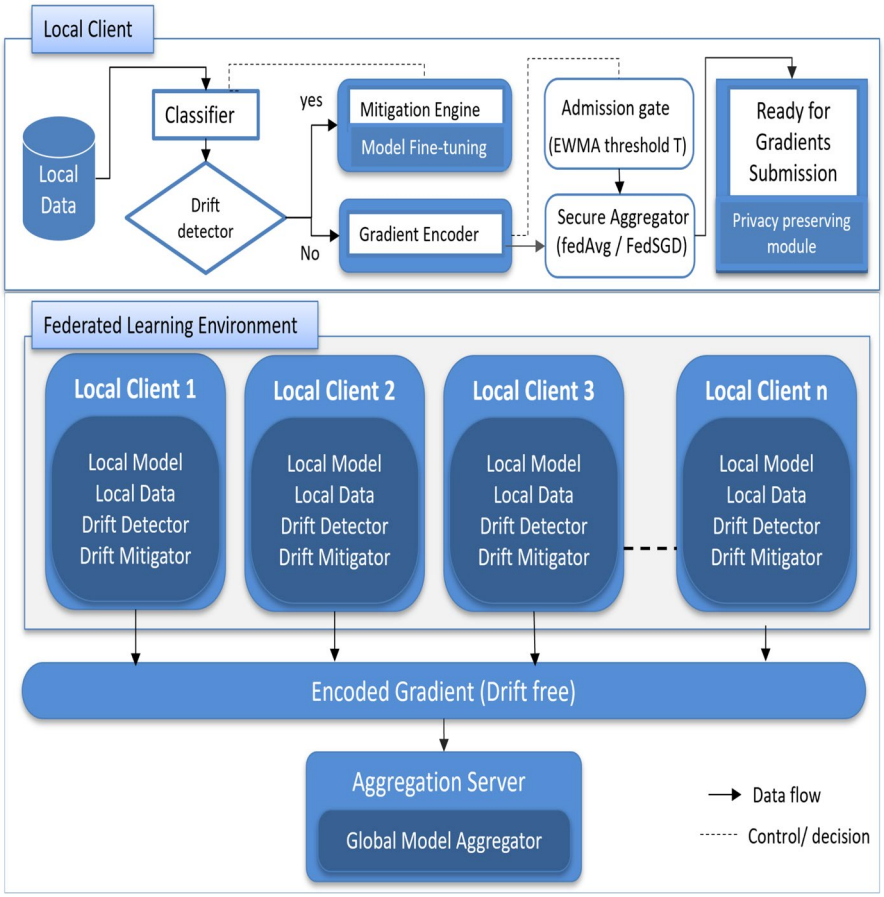

Fig. 1. System Overview. Clients contribute drift mitigated, privacy preserving updates. The server filters by drift threshold and aggregates validated updates to pro- duce a robust global model.

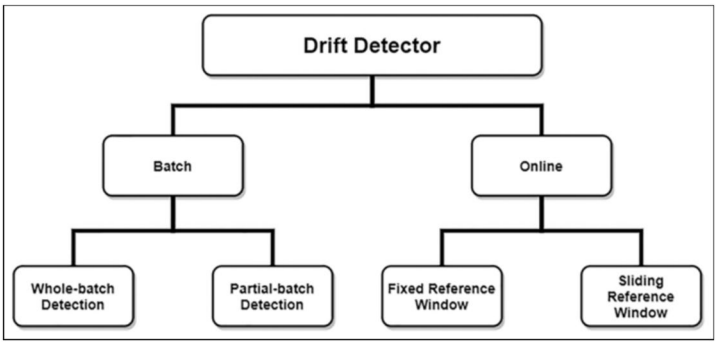

Fig. 2. Taxonomy of concept drift detection methods for online drift detection (adapted from[4]). Each federated client follows this process to detect and respond to local data distribution shifts.

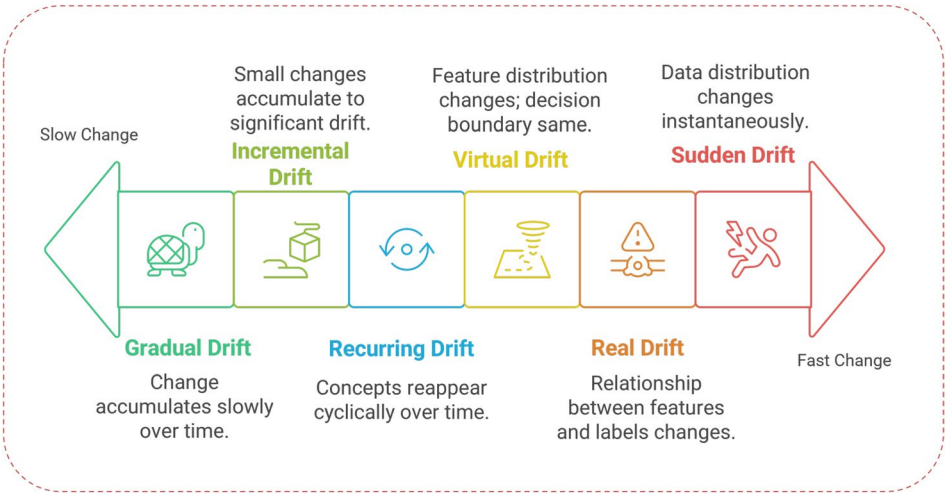

Fig. 3. Taxonomy of concept drift types. Drift can occur gradually, incrementally, suddenly, or recur after dormancy, and is further classified as *real* (changes in $P(y|X)$) or *virtual* (changes in $P(X)$ only). Understanding these distinctions is critical for designing adaptive malware detection models.

Concept drift and its types

Concept drift denotes temporal changes in the joint distribution $P(X, y)$. The challenge is acute in malware detection: adversaries employ obfuscation, polymorphism, and other tactics that alter feature statistics, quickly degrading detectors trained on historical data. Without explicit adaptation, models lose accuracy over time.

Drift can be organized along two complementary axes. *Temporal dynamics:* sudden, gradual, incremental, and recurring/seasonal. *Probabilistic nature: real drift* (changes in $P(y|X)$) versus *virtual drift* (changes in $P(X)$ while $P(y|X)$ remains stable). Figure 3 presents these axes jointly (often shown as parallel categories); in practice they are orthogonal. FL-MalDrift is designed to address both the temporal and probabilistic aspects of drift in Android malware detection.

- *Sudden Drift:* An abrupt and complete change in distribution. In malware contexts, this often arises when a zero-day family emerges with novel attack patterns, making existing detectors ineffective overnight.
- *Gradual Drift:* A slow transition where old and new concepts coexist for some time. For example, a ransomware strain may evolve incrementally, with overlapping signatures persisting across successive variants.
- *Incremental Drift:* Continuous, small scale shifts that accumulate over time. Attackers may slowly adjust opcode frequencies or API call sequences, progressively eroding the reliability of static models.
- *Recurring or Seasonal Drift:* Previously observed distributions reappear after dormancy. Campaign specific malware families, once inactive, may resurface in modified form, requiring models to recall or relearn older patterns.

From a probabilistic perspective:

- *Real Drift:* Changes in the conditional distribution $P(y|X)$, altering classifier decision boundaries. For instance, an Android app initially benign may become malicious after payload injection, changing its true label.

- *Virtual Drift:* Changes in the marginal feature distribution $P(X)$ while $P(y|X)$ remains unchanged. For example, new benign applications may introduce unseen API calls that shift feature statistics without modifying class semantics.

A clear understanding of these drift types is essential for designing adaptive malware detectors capable of dynamically adjusting to the evolving threat landscape while maintaining long term robustness.

Key contributions

This work makes the following contributions:

- Proposes *FL-MalDrift*, a federated malware detection framework that integrates adaptive drift detection with dynamic client participation for Android environments.
- Demonstrates superior performance on Drebin and CICMalDroid, consistently surpassing federated baselines such as M2FD and FedHGCDroid.
- Provides extensive evaluation including cross-dataset transfer, family-specific analysis, client-wise thresholding, and resource overhead assessment.
- Positions FL-MalDrift against complementary approaches, showing its unique balance of accuracy, scalability, privacy, and drift resilience.

Research questions

RQ1: Can local drift detection and mitigation improve global model performance significantly compared to traditional FL?
RQ2: What is the trade-off between resource usage (CPU, memory, bandwidth) and drift resilience in federated learning scenarios?
RQ3: How does local drift handling impact privacy, convergence speed, and overall accuracy?

Literature review

Federated learning (FL) has evolved from early privacy-preserving distributed training to drift-aware, resource-conscious systems tailored to non-IID security data. Foundational work established FL's architectural variants and privacy goals but largely assumed stationary data distributions and benign participation dynamics[5]. In parallel, the concept-drift literature matured from incremental learning on noisy streams[6] to comprehensive characterizations of drift types and response strategies-windowing, statistical tests, and ensembles[7–9]. This methodological base later intersected with Android security, where evolving malware families create real and virtual drift at multiple temporal scales.

Recent security-focused FL studies began to internalize drift explicitly. M2FD framed mobile malware detection under concept drift with statistical alarms and adaptive retraining[10], while LDCDroid targeted model aging and drift characterization on Android data (centralized rather than federated)[11]. Hybrid, cloud-edge pipelines further explored drift-aware classification fusion for mobile threats[12]. Orthogonal analyses in FL clarified how spatial/temporal drift impairs convergence and fairness across clients[13], reinforcing the need for client-local detection and selective participation rather than purely global remedies.

Table 1 summarizes key contributions from foundational and recent federated learning studies in the context of malware detection, along with their limitations. This structured overview highlights the progression of methods and clarifies how our proposed FL-MalDrift framework directly addresses the identified gaps.

Within this trajectory, four detector families: ADWIN, DDM, EDDM, and HDDM emerged as practical building blocks. Comparative studies consistently report ADWIN's stability and low false alarms under evolving

Ref.	Algorithm	Contribution	Limitations (Addressed by FL-MalDrift)
5	Secure FL architecture (horizontal/vertical/transfer learning)	Introduces foundational FL designs preserving data privacy, establishing the importance of distributed learning architectures. Provides groundwork for privacy preserving FL but assumes static distributions.	Does not address concept drift or client heterogeneity. FL-MalDrift adds client-side drift detection and threshold-based aggregation to mitigate dynamic data distributions.
14	EfficientNet-B0 with Adaptive and Weight Normalization	Addresses accuracy degradation due to client heterogeneity and mild concept drift using normalization layers, supporting our hypothesis on local model stabilization.	No explicit drift detection or rollback. FL-MalDrift adds local drift detection and dynamic aggregation with rollback for drifted clients.
15	Master-FL (FedAvg/FedOMD + drift detection)	Performs online drift detection via statistical change monitoring. Aligns with our hypothesis by enabling local response to non-stationary environments and implicit rollback via model reset.	Lacks practical aggregation filtering or malicious client isolation. FL-MalDrift strengthens this by introducing threshold-based gradient filtering and rollback support.
16	FedStream (Prototype-based FL with metric learning)	Uses representative prototypes to model concept drift implicitly, supporting adaptive learning from non-stationary data streams across clients.	No explicit drift detection or update isolation. FL-MalDrift introduces threshold-driven gradient rejection and model rollback to improve robustness.
17	FedNN (FedAvg + Weight and Adaptive Group Normalization)	Normalization techniques help mitigate inter-client drift in FL settings. Supports our hypothesis on reducing local model inconsistency due to distributional shift.	No mechanism for drift detection, thresholding, or rollback. FL-MalDrift integrates these components for better model integrity and client selection.
18	Survey; N/A (discusses drift and aggregation in IoT-FL)	Highlights challenges in FL under drift, advocating for personalized models and smarter aggregation, directly supporting our focus on robust aggregation and client filtering.	Offers only recommendations, not a solution. FL-MalDrift operationalizes these ideas through local drift detection and gradient-level thresholding with rollback control.

Table 1. Literature review aligned with our Methodology.

Android streams[19], DDM's efficiency but conservatism on gradual drift, EDDM's sensitivity alongside higher false positives, and HDDM-W's minimal delay at higher cost[20,21]. Table 2 summarizes these trade offs, which are pivotal in edge-constrained FL deployments.

A second cluster addresses scalability and resource constraints. Prototype based streaming FL[16] and normalization-augmented training[17] mitigate non-IID variance with lightweight representations and stable updates. Broader IoT-FL surveys highlight the persistent tension between accuracy, cost, and personalization under drift[18]. Adjacent security systems federated IDS and SDN defenses-demonstrate the feasibility of privacy-preserving learning at network scale, yet typically without explicit drift gating or client-level quality control[22,23]. Recent VANET and blockchain assisted defenses foreground deployment constraints and governance of model updates, thematically aligned with FL participation control[24,25].

A third cluster centers on governance and client-side autonomy. User-governed data contribution in federated recommendation[26] and on device recommender surveys[27] argue for flexible, edge-centric control over what and when to contribute concepts that resonate with drift-aware participation thresholds. Complementary work on selfish or uncooperative clients underscores the need for robust aggregation and selective inclusion beyond accuracy alone[28]. Finally, adjacent domains (credit-card fraud, IoMT malware) document the necessity of continual adaptation under shifting threats, albeit in centralized or non-federated settings[29–31].

Recent studies confirm that federated learning has begun to gain traction in mobile and Android malware detection due to its ability to preserve data privacy while leveraging distributed intelligence. Jiang et al.[32] introduced FedHGCDroid, a hierarchical federated framework for Android malware classification, demonstrating the feasibility of FL for handling diverse app features in distributed environments. Similarly, M2FD[10] applied FL to mobile malware detection under drift scenarios, validating its relevance for evolving Android ecosystems. Other works such as LDCDroid[11] and Master-FL[15] further illustrate the application of FL in security-sensitive domains, emphasizing adaptability to drift and non-IID client data.

Complementary to these efforts, recent research has explored the robustness of aggregation strategies in federated environments. Pillutla et al.[33] proposed Robust Federated Aggregation (RFA) using geometric median-based updates to enhance resistance against adversarial or corrupted clients. Nguyen et al.[34] demonstrated that randomized selection of aggregation functions such as Trimmed Mean, Krum, and Switching improves defense stability against model poisoning. A related study by[35] provided a taxonomy of robust aggregation functions and their resilience under Byzantine and non-IID conditions. These works collectively establish that aggregation mechanisms play a crucial role in maintaining federated reliability. The proposed *FL-MalDrift* framework complements this line of research by integrating drift aware participation control that enhances model stability under evolving malware distributions while preserving privacy and robustness within federated settings.

Beyond drift handling, a rich line of work strengthens privacy and robustness in federated learning. Blockchain enabled privacy preserving FL frameworks such as PBFL with homomorphic encryption and single masking[36] and PBFL for Industry 4.0[37] combine secure aggregation, HE, and decentralized ledgers to protect both local and global models while tolerating client dropouts. Byzantine-robust designs including ShieldFL[38], FL-CDF[39], and the recent DP2Guard scheme for Industrial IoT[40] focus on detecting or filtering poisoned updates and backdoor behaviour under strong adversarial models. In parallel, incentive mechanisms built on consortium or blockchain infrastructures such as the dynamic Stackelberg game-based reward design of Han et al.[41] and the repeated-game-based long term incentive framework for reliable FL in IIoT[42] align client rewards with contribution quality and long term participation.

FL-MalDrift complements these approaches by focusing on client side drift detection and adaptive participation rather than new cryptographic or incentive mechanisms. Its drift aware admission strategy can be seamlessly combined with secure aggregation, blockchain coordination, or Byzantine robust aggregators to jointly enhance privacy, robustness, and stability under non-stationary malware data.

Across these clusters, three cross cutting gaps persist: first, limited local drift handling prior to aggregation (most systems react globally after degradation); second, the absence of adaptive participation control that filters drift affected clients without sacrificing federation coverage; and third, a lack of unified, low overhead mechanisms that reconcile drift detection quality with edge resource budgets. To address these gaps, we introduce *FL-MalDrift*, which (i) embeds lightweight, detector agnostic drift scoring at the client, (ii) applies dynamic, EWMA driven participation thresholds to admit only stable updates, and (iii) supports rollback and stability aware aggregation thereby aligning detection efficacy with privacy, scalability, and real world deployability.

Detector	Strengths	Weaknesses	Suitability in FL malware detection
ADWIN[19]	Adapts to abrupt and gradual drift; low false positives; statistically grounded	Moderate computational cost	Strong candidate; balances accuracy, stability, and efficiency in Android malware detection
DDM[19]	Lightweight and efficient; minimal CPU usage	Struggles with gradual drift; under-detects subtle changes	Useful in highly resource-constrained settings but limited robustness
EDDM[19,20]	Sensitive to incremental drift; detects subtle shifts	High false positives; noisy under non-stationary streams	Risk of excessive retraining and communication overhead in FL
HDDM-W[21]	Fastest detection; high accuracy for abrupt/gradual drift	Higher computational cost; less suitable for edge devices	Attractive for latency critical cases but impractical for lightweight clients

Table 2. Comparative summary of local drift detection methods.

Experimental setup for drift detector evaluation

We designed a controlled FL simulation to compare local drift detectors on two Android malware datasets: Drebin[43] and CICMalDroid 2020[44]. Using Drebin malware paired with a curated benign subset, we formed a balanced set of 3000 apps (1500 benign/1500 malicious), pruning high dimensional sparse features via frequency filtering and mutual information to retain discriminative attributes. From CICMalDroid, we curated 6000 balanced apps across multiple time windows to preserve temporal evolution, selecting ∼200 key static/dynamic features via correlation analysis. Drebin thus provides a controlled benchmark for simulated drift, while CICMalDroid captures realistic temporal drift.

Data were partitioned across ten clients to induce non-IID heterogeneity. Abrupt drift was simulated by sudden shifts in malware family composition; gradual drift by smooth interpolation of feature distributions across rounds. Each client trained a Random Forest classifier to isolate detector effects from deep model variability. Detectors (ADWIN, DDM, EDDM, HDDM-W) were integrated via scikit multiflow; drift checks followed each local epoch. We report detection accuracy, FPR, detection delay (in rounds), and CPU time (in ms), averaged over ten clients and ten rounds, providing a stable basis for the choices evaluated later in Sect. 5.

The experiments simulate a cross device Android FL setting with 10 clients (non-IID app partitions) trained for 100 global rounds; each selected client performs one local epoch per round with a batch size of 64. The local model is a lightweight, on-device MLP (ReLU hidden layers with a binary softmax head); Drebin uses 1000 filtered static features, while CICMalDroid uses ∼200 static and dynamic features, with the same architecture and hyperparameters across datasets. Optimization follows FedAvg and FedSGD under identical settings for fairness; unless stated otherwise, clients use SGD with an initial learning rate of 0.01 and momentum 0.9. Client participation is governed by the drift-aware admission gate described in Methods; drift detectors affect only admission decisions, not the training iterations or model hyperparameters.

Data distribution scenarios

To comprehensively assess the robustness of FL-MalDrift under realistic federated settings, experiments were conducted using both Independent and Identically Distributed (IID) and Non-IID client data partitions. In the IID configuration, all clients received uniformly sampled subsets from the global dataset, ensuring balanced class and feature distributions for a controlled baseline. In contrast, the Non-IID setup emulated practical heterogeneity by assigning samples according to malware families, leading to client specific biases that reflect real-world deployment conditions. The degree of non-IID heterogeneity, quantified using Kullback–Leibler divergence, averaged around 0.47, indicating moderate statistical skew typical of mobile malware ecosystems. As shown in Table 3, FL-MalDrift maintained high performance in both scenarios achieving 96.8% accuracy under IID and 94.6% under Non-IID distributions demonstrating that its adaptive thresholding mechanism effectively mitigates instability among heterogeneous clients. These results confirm that FL-MalDrift remains resilient to distributional imbalance while sustaining learning efficiency and stability across dynamically evolving federated environments.

Differential privacy analysis

To formally quantify the privacy guarantees of the optional differential privacy (DP) layer in *FL-MalDrift*, Gaussian noise is added to client-side gradient updates before transmission. Let $\tilde{g}_{c,t} = g_{c,t} + \mathcal{N}(0, \sigma^2 I)$ denote the perturbed gradient of client c in round t, where σ is calibrated according to the desired privacy budget (ε, δ) under the Gaussian mechanism. The cumulative privacy loss over T communication rounds follows the advanced composition theorem:

$$\varepsilon_T = \sqrt{2T \ln(1/\delta)}\, \frac{\Delta_2}{\sigma} + T \frac{\Delta_2 (e^{1/\sigma} - 1)}{\sigma},$$

where Δ_2 represents the ℓ_2 sensitivity of the gradient updates. For the experimental configuration ($T = 20$, $\delta = 10^{-5}$), setting $\sigma = 1.2$ yields $\varepsilon \approx 1.0$, offering a strong privacy guarantee while preserving learning utility. Empirically, this DP variant resulted in less than a 2.4% reduction in classification accuracy compared to the non-DP configuration, confirming that privacy can be maintained without sacrificing model robustness. This formalization substantiates the framework's "high privacy" claim and ensures that even under an honest-but-curious server model, client information remains statistically protected.

Methodology

Concept drift denotes temporal changes in the joint distribution $P(X, y)$, where X is the feature space and y the target. Such non-stationarity degrades static models unless it is detected and mitigated. In federated learning (FL), drift must be handled *locally* to preserve privacy while maintaining global performance. We therefore combine client-side drift detection with drift aware participation and standard FL aggregation.

In Fig. 4 we adapt the general online drift detection workflow of Gemaque et al.[4] to the client side: each client processes a stream of instances, maintains a reference against recent observations, and applies statistical tests with significance thresholds to decide whether local distributional change has occurred.

Distribution type	Accuracy (%)	F1 Score	Active clients (%)
IID	96.8	0.962	91.5
Non-IID	94.6	0.945	88.2

Table 3. Performance comparison under IID vs Non-IID data distributions (CICMalDroid 2020).

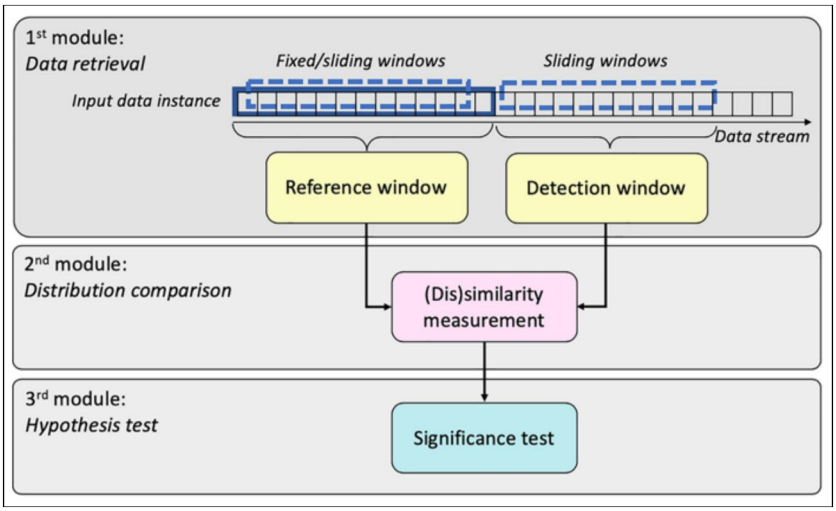

Fig. 4. Client-side online drift detection workflow adapted from[4]. Each client evaluates streaming observations against a reference window and signals local drift when statistical bounds are exceeded.

Drift detection mechanisms and score computation

We consider four widely adopted drift detectors: ADWIN, DDM, EDDM, and HDDM, each selected for its complementary operating characteristics and prior validation in streaming data contexts[7]. To enable consistent comparison and integration into FL-MalDrift, their raw statistics are mapped into normalized *drift scores* in [0, 1].

ADWIN (Adaptive windowing)

ADWIN maintains a variable length sliding window and splits it into two sub windows. A drift alarm is triggered when the difference between their means exceeds a Hoeffding derived bound:

$$z_{c,t}^{\mathrm{ADWIN}} = \left|\mu_1 - \mu_2\right|, \quad \text{drift if } \ |\mu_1 - \mu_2| > \epsilon.$$

DDM (Drift detection method)

DDM monitors the error rate p_i and standard deviation $s_i = \sqrt{p_i(1-p_i)/i}$ at time *i*. Drift is flagged when:

$$z_{c,t}^{\mathrm{DDM}} = (p_i + s_i) - (p+s)_{\min}, \quad \text{drift if } \ p_i + s_i > (p+s)_{\min} + \delta.$$

EDDM (Early DDM)

EDDM focuses on the average distance d_i between two consecutive errors and its standard deviation s_d. It compares the current error spacing with the historical maximum:

$$z_{c,t}^{\mathrm{EDDM}} = 1 - \frac{d_i + s_d}{(d+s_d)_{\max}},$$

where smaller values indicate drift due to shorter inter error distances.

HDDM (Hoeffding Drift detection method)

HDDM employs Hoeffding bounds to test if the deviation between current and reference means is statistically significant:

$$z_{c,t}^{\mathrm{HDDM}} = \left|\mu_t - \mu_0\right|, \quad \text{drift if } \ |\mu_t - \mu_0| > \epsilon.$$

Two variants exist: HDDM-A uses arithmetic means, while HDDM-W applies an exponentially weighted moving average for higher sensitivity to recent changes.

Unified Drift score normalization

To make detectors comparable across clients, each raw statistic $z_{c,t}$ is normalized into a bounded drift score:

$$s_{c,t} = \mathrm{clip}_{[0,1]}\left(\frac{z_{c,t} - \mu_t^{(W)}}{\sigma_t^{(W)} + \varepsilon}\right),$$

where $\mu_t^{(W)}$ and $\sigma_t^{(W)}$ are the mean and standard deviation of detector statistics over a sliding window of size W, and ε is a small constant for numerical stability. Detectors producing discrete states are mapped as no-alarm$\rightarrow 0$, warning$\rightarrow 0.5$, drift$\rightarrow 1$.

High dimensional feature spaces

Although Android malware datasets such as Drebin and CICMalDroid contain thousands of static and dynamic features, the drift detectors operate on *low dimensional statistics* derived from model predictions, not raw feature vectors. For instance, ADWIN evaluates mean error differences, DDM and EDDM monitor error sequences, and HDDM tracks moving averages of losses. This design ensures that drift monitoring remains computationally tractable and interpretable even under high dimensional input spaces, while still capturing distributional changes that impact classification performance.

Detector execution protocol

We explicitly define how drift detectors are embedded within the FL-MalDrift training loop. Each client executes a detector as part of its local training pipeline with the following protocol:

Triggering point Drift detection is executed *post local training, pre-aggregation* at the end of each communication round. This ensures that detectors operate on updated local model errors or distribution statistics rather than stale states, and their outputs can directly regulate whether the client's update participates in the subsequent global aggregation.

Input signals. Detectors consume client local statistics generated after training. Specifically: - ADWIN: mean difference of prediction errors across variable windows; - DDM: online error rate p_i and deviation s_i; - EDDM: distance between classification errors; - HDDM: deviation $|\mu_t - \mu_0|$ between current and reference error means.

Output interface. Each detector outputs a normalized *drift score* $s_{c,t} \in [0, 1]$, computed as described in Sect. 4.1. Scores are mapped to three states: stable ($s \approx 0$), warning ($s \approx 0.5$), or drift ($s \approx 1$).

Decision logic. If $s_{c,t}$ exceeds the dynamic participation threshold τ_t, the client update $\theta_{c,t}$ is temporarily withheld from aggregation. Otherwise, the adapted parameters $\theta^*_{c,t}$ are transmitted to the server. This creates a direct feedback loop where local drift signals gate contribution.

Temporal granularity. Detection occurs once per communication round per client, ensuring synchronization with the FL training loop. Within round mid epoch detection is not required to limit overhead, but can be extended in future work for latency critical environments.

This protocol clarifies the synchronization of detector execution with federated training and formalizes the interfaces between detector statistics, threshold mechanisms, and client contribution decisions.

Federated learning framework

Our framework augments standard FL with client side drift detection and server side participation control. Clients train locally, run a drift check after each local epoch, optionally adapt (reset layers, adjust learning rate, or partial retraining) and then transmit updates. The server aggregates only *stable* contributions based on a drift threshold, thereby improving robustness without exposing raw data.

Initialize global model parameters θ
for *each communication round* $t = 1, 2, \ldots$ **do**
  Server broadcasts θ_t to selected clients
  for *each client* c **do**
    `// Local Training Phase`
    Train θ_t on local data for one epoch to obtain provisional update $\theta_{c,t}$
    `// Drift Detection Phase (post training, pre aggregation)`
    Compute detector statistic $z_{c,t}$ (ADWIN mean gap, DDM error rate, etc.)
    Normalize to drift score $s_{c,t} \in [0, 1]$
    `// Local Adaptation`
    if $s_{c,t} > \theta_{c,t}^{local\text{-}threshold}$ **then**
      Apply local adaptation (reset layers, re-weight instances, adjust learning rate)
    Package adapted update $\theta^*_{c,t}$ with drift score $s_{c,t}$
    Send $(\theta^*_{c,t}, s_{c,t})$ to server
  `// Server Side Threshold Filtering`
  Compute global participation threshold τ_t (Sec. 4.3)
  Select stable clients $\mathcal{C}_t = \{c : s_{c,t} \leq \tau_t\}$
  Aggregate $\{\theta^*_{c,t}\}_{c \in \mathcal{C}_t}$ using FedAvg/FedSGD to form θ_{t+1}

Algorithm 1. FL-MalDrift: Federated learning with client side drift detection and thresholded aggregation

Privacy analysis and threat model

FL-MalDrift adheres to the standard *honest but curious* threat model, in which the central server correctly executes the protocol but may attempt to infer private client information from transmitted updates or metadata. Although raw features and local labels never leave clients, drift scores $s_{c,t}$ could theoretically encode indirect

statistical signals about client data distributions. To quantify and mitigate this risk, we analyze privacy leakage using a differential privacy (DP) perspective.

Let $\mathcal{M}$ denote the stochastic mechanism generating client messages $(\theta^*_{c,t}, s_{c,t})$. The mechanism satisfies (ε, δ)-differential privacy if, for any pair of neighboring datasets differing by one client record, and for all possible outputs $\mathcal{O}$,

$$\Pr[\mathcal{M}(D_1) \in \mathcal{O}] \leq e^{\varepsilon} \Pr[\mathcal{M}(D_2) \in \mathcal{O}] + \delta.$$

In our implementation, Gaussian noise $\mathcal{N}(0, \sigma^2 I)$ is added to both parameter updates and drift scores before transmission, where σ is calibrated to achieve (ε=1.5, δ=10^{-5}) over 25 communication rounds following the standard DP-SGD composition rule[45]. This ensures that an individual sample's influence on the transmitted updates or the threshold adaptation in Algorithm 2 remains bounded.

It should be noted that this differential privacy configuration serves as a *conceptual privacy layer* rather than an active component in the experiments reported in this paper. The empirical evaluations focus on drift detection and adaptive participation under honest but curious settings. Future extensions of FL-MalDrift will include formal privacy utility benchmarking following the methodology of Taheri et al.[46], who demonstrated the integration of differential privacy mechanisms to enhance adversarial robustness in deep learning based Android malware detection.

Adaptive threshold calibration for client participation control

In federated learning environments, the integrity of the global model fundamentally depends on the quality and reliability of client contributions. This challenge becomes particularly pronounced in cybersecurity applications such as malware detection, where concept drift occurs frequently due to the adversarial nature of evolving threats. Unstable or misleading updates from drift affected clients can severely degrade the performance of the global model, necessitating robust quality control mechanisms.

To address this critical challenge, *FL-MalDrift* introduces a novel dynamic participation threshold system (τ) that functions as an intelligent quality gate at each participating client. This mechanism operates on the principle of adaptive exclusion, where clients experiencing significant concept drift are temporarily withheld from the aggregation process until their local models stabilize. The threshold system employs a two phase approach: initial calibration during a warm up period to establish baseline stability characteristics, followed by continuous online adaptation using exponentially weighted moving averages (EWMA) to maintain responsiveness to evolving data distributions.

The operation of this dynamic participation threshold is illustrated in Fig. 5, which shows how FL-MalDrift filters unstable client updates before global aggregation through its drift aware server-side gate.

The fundamental design philosophy of this approach balances two competing objectives: (1) preserving the collective learning capability of the federation by maintaining sufficient client participation, and (2) protecting the global model from destabilization caused by drift affected clients. This balance is achieved through a self-regulating participation pattern that allows stable clients to contribute continuously while temporarily excluding unstable clients until their local conditions improve.

In the specific context of Android malware detection, where new malware variants emerge unpredictably and existing families evolve to evade detection systems, this adaptive threshold mechanism proves essential for maintaining both accuracy and robustness. The system enables the federation to leverage stable edge data for continuous learning while resisting premature adoption of potentially misleading patterns from clients experiencing transient drift events.

Drift score computation and normalization

The foundation of the threshold system lies in the computation of normalized drift scores that enable fair comparison across heterogeneous drift detection methods. Each participating client *c* generates a *drift score* $s_{c,t} \in [0, 1]$ following local training in communication round *t*. This score is derived from the internal statistics produced by the client's drift detection algorithm and undergoes unified normalization to ensure comparability across different detector types.

The normalization process follows a standardized z-score transformation with clipping to maintain bounded outputs:

$$s_{c,t} \;=\; \mathrm{clip}_{[0,1]}\left(\frac{z_{c,t} - \mu_t^{(W)}}{\sigma_t^{(W)} + \varepsilon}\right),$$

where $z_{c,t}$ represents the raw detector-specific statistic. For ADWIN, this corresponds to the magnitude of the mean difference between sub-windows; for DDM, it represents the deviation $(p_i + s_i) - (p + s)_{\min}$ from the minimum observed error bound; for EDDM, it captures the distance-ratio deficit $1 - \frac{d_i + s_d}{(d+s_d)_{\max}}$; and for HDDM, it measures the statistical deviation $|\mu_t - \mu_0|$ from the reference mean.

The normalization parameters $\mu_t^{(W)}$ and $\sigma_t^{(W)}$ represent the empirical mean and standard deviation of recent detector statistics computed over a sliding window of the last *W* communication rounds, pooled across all participating clients. This approach ensures that the normalization adapts to the evolving statistical properties of the federation while maintaining numerical stability through the addition of a small regularization term ε.

For drift detectors that produce discrete symbolic states rather than continuous statistics, we employ a standardized mapping: no-alarm$\rightarrow 0$, warning$\rightarrow 0.5$, and drift$\rightarrow 1$. This mapping preserves the ordinal relationship between detection states while enabling integration into the continuous threshold framework.

Normalization rationale and alternatives

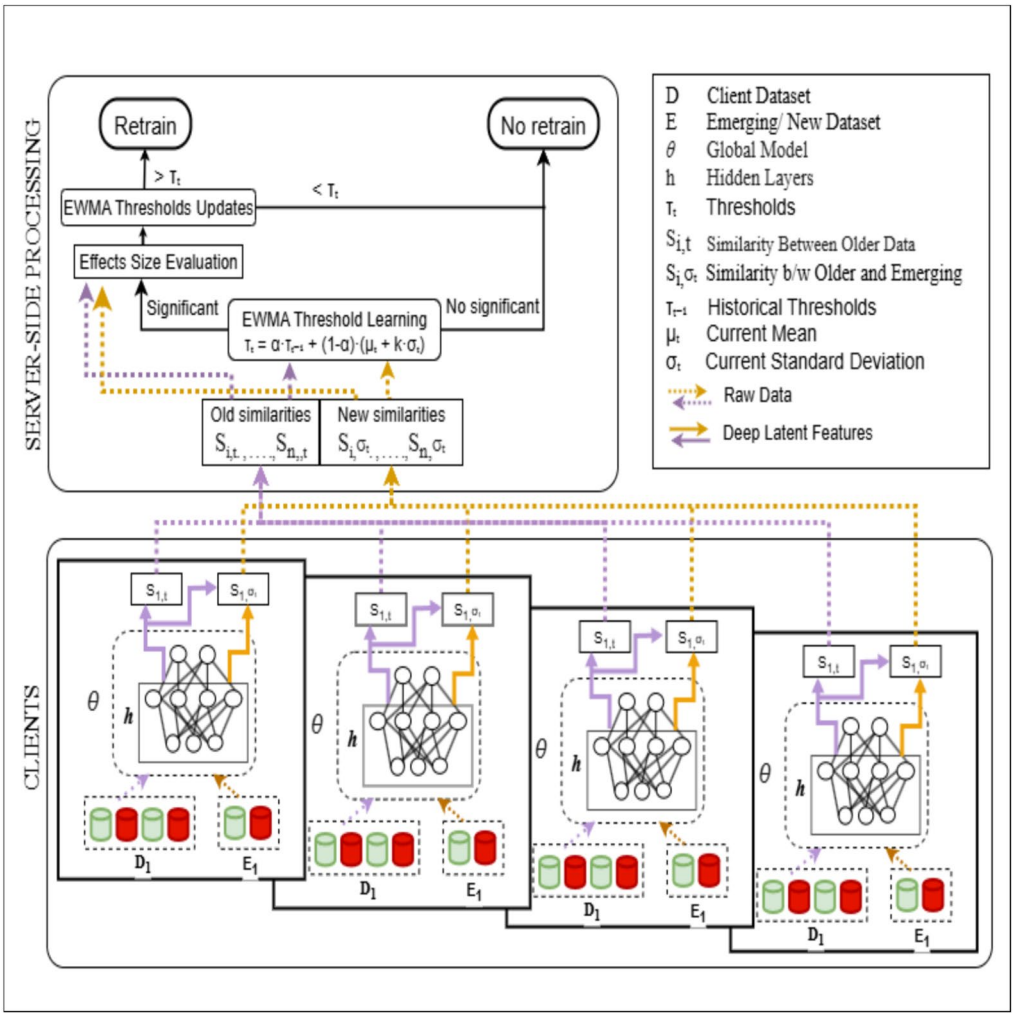

Fig. 5. Drift aware participation control mechanism in FL-MalDrift. Each client performs local concept drift detection and model adaptation, while the server applies dynamic threshold filtering to exclude potentially unstable updates prior to global model aggregation. This two tier approach ensures both local stability and global robustness in federated malware detection.

The choice of z-score normalization with clipping was guided by empirical stability analysis across multiple drift detectors and client configurations. This transformation standardizes detector outputs relative to their recent statistical context, enabling consistent interpretation of drift magnitudes even under heterogeneous client distributions. The clipping step confines extreme fluctuations caused by transient drifts or noisy updates, thereby ensuring stable participation thresholds and preventing oscillatory behavior during aggregation.

Alternative normalization strategies were also investigated. **Min–max scaling**, which rescales data within [0, 1], exhibited sensitivity to outliers-small variations in extreme drift scores disproportionately shifted the normalization range, leading to unstable threshold adaptation. **Robust scaling**, based on median and interquartile range, provided partial resistance to outliers but introduced latency in capturing abrupt drift changes, particularly in early communication rounds. In contrast, the **z-score with bounded clipping to** $[-3, 3]$ produced smoother convergence and more reliable drift differentiation, achieving the best trade-off between responsiveness and stability. Empirically, this approach maintained low false alarm rates and uniform client participation across datasets, justifying its integration within the *FL-MalDrift* framework.

Server side global threshold management

The server maintains a global participation threshold $\tau_t \in [0, 1]$ that determines which clients are eligible for inclusion in each aggregation round. Clients whose drift scores satisfy $s_{c,t} \leq \tau_t$ are admitted to the aggregation process, while those exceeding the threshold are temporarily excluded. The threshold selection process employs a sophisticated two-stage procedure designed to balance stability and adaptability.

Initial Calibration Phase. During the initial warm-up period spanning the first T_0 communication rounds, the system operates without filtering to collect comprehensive baseline statistics. All client drift scores $\{s_{c,t}\}$ are recorded to establish the empirical distribution of normal operating conditions. When labeled synthetic drift events are available for validation, the initial threshold τ_0 is determined by maximizing Youden's *J* statistic, which optimally balances true positive and false positive rates:

$$\tau_0 \;=\; \arg\max_{\tau} \big(\mathrm{TPR}(\tau) - \mathrm{FPR}(\tau)\big).$$

In practical scenarios where labeled drift events are unavailable, the initial threshold is set to a high quantile of the observed benign score distribution: $\tau_0 = \mathrm{Quantile}_q(\{s_{c,t}\}_{t \le T_0})$, where $q \in [0.85, 0.95]$ represents a conservative percentile chosen to minimize false exclusions during the learning phase.

Online Adaptive Updates. Following the warm-up period, the threshold undergoes continuous adaptation using recent score statistics and a participation budget constraint $p^* \in (0, 1)$. The update mechanism combines statistical control chart principles with exponential smoothing:

$$\hat{\tau}_t \;=\; \mu_t^{(W)} + k\,\sigma_t^{(W)}, \qquad \tau_t \;=\; \mathrm{clip}_{[\tau_{\min}, \tau_{\max}]}\big(\alpha\,\tau_{t-1} + (1-\alpha)\,\hat{\tau}_t + \eta\,(p^* - p_t)\big)\,,$$

where k controls the conservativeness of the threshold (analogous to control chart limits), $\alpha \in [0, 1)$ represents the EWMA smoothing factor, $\eta > 0$ is a small gain parameter for participation control, and $p_t = \frac{1}{N}\sum_c \mathbb{1}[s_{c,t} \le \tau_{t-1}]$ denotes the realized fraction of active clients in the current round. *This formulation draws from control theory and statistical process control principles, where the EWMA term acts as a low-pass filter stabilizing score fluctuations, $(\mu_t^{(W)} + k\sigma_t^{(W)})$ defines adaptive statistical control limits, and the feedback term $\eta(p^* - p_t)$ regulates participation akin to proportional feedback control.* This formulation ensures that participation levels remain close to the target budget while adapting to non-stationary drift score distributions.

Client level stability assessment

Beyond the global server side threshold, each client maintains an additional local stability threshold $\theta_{c,t}$ based on the divergence of its model updates from the global model. This dual threshold approach provides fine-grained control over contribution quality at both local and global levels.

The divergence metric $d_{c,t}$ quantifies the dissimilarity between the client's local update and the current global model, computed using either parameter norm ratios or cosine similarity measures:

$$d_{c,t} \;=\; \frac{\|\theta_{c,t}^{\mathrm{local}} - \theta_t\|_2}{\|\theta_t\|_2} \quad \text{or} \quad 1 - \frac{\langle \Delta\theta_{c,t},\, \Delta\theta_t \rangle}{\|\Delta\theta_{c,t}\|_2\, \|\Delta\theta_t\|_2}.$$

In all reported experiments, the Euclidean (L2) divergence formulation was consistently adopted to measure deviation between local and global model parameters, as it offered stable scaling across heterogeneous feature spaces and aligned naturally with gradient based aggregation. The cosine similarity variant was evaluated only in preliminary tests to confirm trend consistency but was not used in the final experimental results.

Client updates are subject to local gating when $d_{c,t} > \theta_{c,t}$, providing an additional layer of quality control. The local threshold undergoes adaptive updates via EWMA that incorporates both drift detection signals and local validation performance:

$$\theta_{c,t} \;=\; \beta\,\theta_{c,t-1} + (1-\beta)\,\big(\theta_{\mathrm{base}} + \lambda_1\,s_{c,t} + \lambda_2\,e_{c,t}\big),$$

In our experiments, the coefficients λ_1 and λ_2 were empirically tuned within the range [0.05, 0.15] using a grid search to balance responsiveness to drift and fluctuations in local validation error. The observed performance variation remained within ±1.2% accuracy across this range, demonstrating that the stability mechanism is largely insensitive to moderate coefficient changes. A supplementary sensitivity heatmap (Fig. 6) further confirms that variations of λ_1 and λ_2 within this interval lead to only marginal deviations in overall accuracy, reinforcing the robustness of the adaptive local stability threshold formulation.

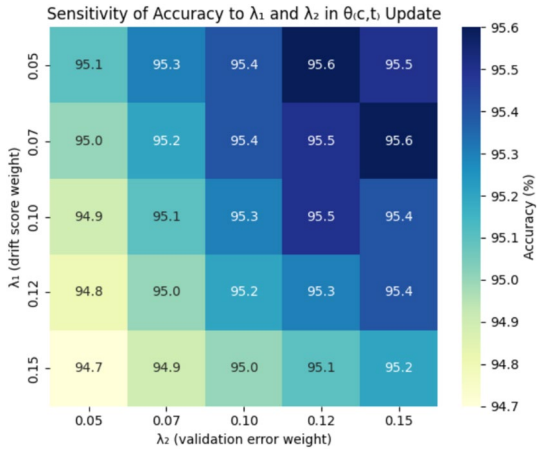

Fig. 6. Sensitivity analysis of local stability coefficients λ_1 and λ_2 on model accuracy. Results show minimal performance deviation (within $\pm 1.2\%$), indicating robustness of $\theta_{c,t}$ adaptation across moderate parameter variations.

where $\beta \in [0, 1)$ controls the memory of the exponential average, θ_{base} represents the baseline threshold, and λ_1, λ_2 are small weighting coefficients for drift scores and local validation errors $e_{c,t}$, respectively. This formulation ensures that clients experiencing transient drift or elevated error rates become more restrictive in their update submission, thereby reducing the propagation of potentially unstable model changes.

Integrated server side processing

The complete threshold management process is formalized in Algorithm 2, which encapsulates the server-side logic for dynamic threshold adaptation and client selection.

Input: Current round t; client similarity/drift scores $\{s_{c,t}\}$; participation budget p^*
Output: Selected stable client set $\mathcal{C}_t$; updated threshold τ_t
if $t \leq T_0$ (warm-up phase) **then**
  Set $\tau_t \leftarrow \tau_0$
  Record $\{s_{c,t}\}$ for future threshold learning
else
  Compute mean μ_t and std. deviation σ_t over the recent window W of $\{s_{c,t}\}$
  Compute EWMA update: $\tau_t \leftarrow \alpha\,\tau_{t-1} + (1-\alpha)\,(\mu_t + k\,\sigma_t)$
  Evaluate effect-size significance based on current/new similarities
  if (drift detected) **then**
    Trigger **Retrain** global model
    Update historical statistics for future EWMA computation
  else
    No retrain; continue with current model aggregation
  Optionally adjust τ_t using participation feedback:
  $p_t \leftarrow \frac{1}{N}\sum_c \mathbb{1}\{\, s_{c,t} \leq \tau_{t-1} \,\}$
  $\tau_t \leftarrow \text{clip}\big(\tau_t + \eta\,(p^* - p_t)\big)$
Select stable clients for aggregation:
$\mathcal{C}_t \leftarrow \{\, c \mid s_{c,t} \leq \tau_t \,\}$.

Algorithm 2. Dynamic EWMA-Based Threshold Update and Privacy-Aware Client Selection

Hyperparameter configuration and empirical settings

The drift detectors were configured using established defaults from the scikit multiflow library, with adaptations for federated settings. Through extensive empirical evaluation, we set the following parameters: warm up duration $T_0 = 3$ rounds, sliding window size $W = 10$, EWMA factor $\alpha = 0.8$, conservativeness parameter $k = 1.5$, participation control gain $\eta = 0.05$, and target participation budget $p^* = 0.7$. The baseline stability threshold θ_{base} was initialized from the median divergence observed during the warm-up phase of each client, yielding personalized starting points that capture client specific dynamics. These choices were tuned through systematic optimization to achieve the reported participation behavior in Sect. 5, while maintaining participation levels close to the target budget across diverse drift scenarios. Together, these hyperparameters form the backbone of FL-MalDrift's adaptive thresholding, ensuring robust federated learning under concept drift without compromising privacy.

Hyperparameter sensitivity

To assess the robustness of these hyperparameter choices, we performed an ablation study varying the EWMA control parameters within practical ranges: $k \in \{1.0, 1.5, 2.0\}$, $\alpha \in \{0.6, 0.8, 0.9\}$, and $p^* \in \{0.6, 0.7, 0.8\}$. The results confirmed that moderate parameter settings-particularly (k=1.5, α=0.8, p^*=0.7)-provide the best trade-off between detection stability and accuracy consistency across rounds. Increasing k or α yielded smoother participation dynamics but slightly slower adaptation, whereas lower values introduced oscillations in threshold updates. These trends validate that the chosen configuration maintains optimal balance between responsiveness and global model robustness under evolving drift conditions.

Client side training configuration

Each federated client trained a Random Forest classifier with 100 estimators and a maximum depth of 20, implemented in scikit multiflow. This model was selected for its robustness to high-dimensional sparse features and compatibility with statistical drift detectors. Local training was limited to a single epoch per round with a batch size of 64, simulating resource constrained edge devices. Training proceeded for 20 communication rounds on Drebin and 25 rounds on CICMalDroid 2020. An early stopping criterion was employed, halting training if validation performance failed to improve for five consecutive rounds, thereby avoiding unnecessary computation and communication. This configuration balances reproducibility, efficiency, and robustness, while reflecting realistic constraints of federated edge environments.

Client recovery mechanism

To prevent long term exclusion of clients experiencing persistent or transient drift, FL-MalDrift incorporates a recovery mechanism that allows re-evaluation and gradual reintegration of previously filtered clients. Each excluded client continues local training and drift monitoring in isolation. Once its smoothed drift score $\bar{s}_{c,t}$, computed via an exponential moving average over recent rounds, remains below a re-entry threshold $\tau_{\text{re}} = \tau_t + \delta$ for R consecutive rounds (where δ is a small tolerance margin), the client is reactivated and its updates are revalidated by the server. This ensures that clients recovering from local concept evolution can contribute new stable knowledge, mitigating the risk of "unlearned zones" and maintaining long term inclusivity in federated learning.

Proposed workflow

Figure 7 summarizes end to end operation. Clients continuously monitor for drift, execute local mitigation when needed, and submit drift aware updates. The server evaluates each update against a global threshold; unstable contributions are down weighted or excluded (rollback), and stable updates are aggregated using FedAvg/FedSGD. This selective inclusion improves convergence and resilience under non-IID, evolving data.

Each client trains a supervised malware classifier on a compact, fixed-length feature vector extracted from its local Android apps. For *Drebin*, the input is a pruned static feature set reduced to approximately 1,000 binary indicators via frequency filtering and mutual information; for *CICMalDroid 2020*, static descriptors are combined with summary statistics of dynamic behavior and decorrelated to roughly 200 numeric features. Features are standardized locally and labels are binary. The client side learner is a decision tree ensemble chosen for robustness on sparse, heterogeneous inputs and for low hyperparameter sensitivity under non-IID drift. Models are trained for one local epoch per communication round with depth regularization and out of bag validation to stabilize training on small partitions; a held out local split provides the accuracy signal consumed by the drift module.

After local training, each client computes a detector specific statistic and maps it to a normalized drift score in [0, 1] using the unified procedure described in Sect. 4.1. If the update appears unstable assessed via a lightweight divergence check against the current global model the client performs brief local adaptation and can defer its contribution; otherwise it transmits a model delta. The server aggregates with either FedAvg (sample weighted parameter averaging) or FedSGD (smaller, more frequent deltas), but only admits clients whose drift scores fall

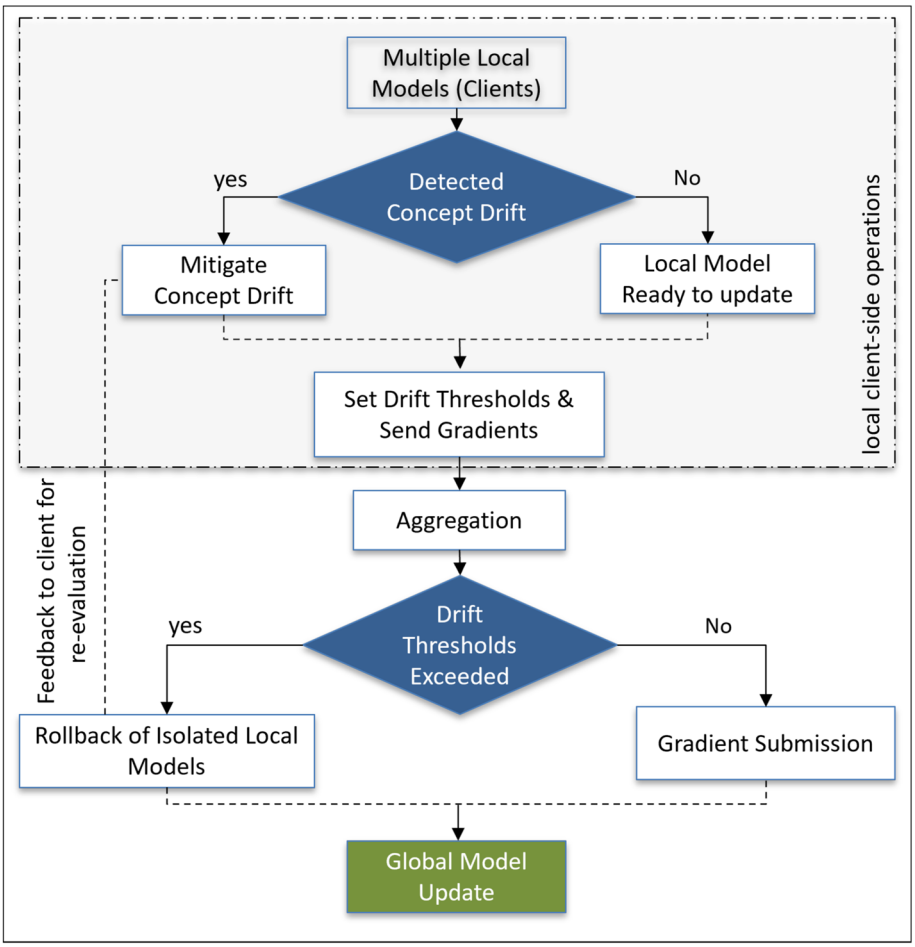

Fig. 7. End to end workflow. Local detectors gate client updates via adaptation and server side thresholding, yielding drift resilient aggregation.

Dataset	Apps	Benign	Malicious	Features	Type
Drebin†	3000	1500	1500	1000	Static
CICMalDroid 2020	6000	3000	3000	200	Static + Dynamic

Table 4. Summary of malware datasets used in experiments. † Drebin malware paired with a *curated benign subset*; features retained after frequency filtering and mutual information

Detector	Detection Acc. (%)	FPR (%)	Delay (Rounds)	CPU Time (ms)
ADWIN	**93.2**	**4.5**	1.4	16.8
DDM	89.5	5.2	**1.0**	**9.3**
EDDM	91.8	12.4	1.1	18.5
HDDM-W	92.7	6.1	**0.9**	22.3

Table 5. Comparison of local drift detectors (averaged across 10 clients and both datasets). Bold values indicate the best performance scores among the compared methods.

below a dynamic participation threshold updated online by an EWMA rule (Sect. 4.3). This integration explains the empirical gains: the ensemble backbone yields stable local learners on high dimensional Android features; FedSGD reacts faster to trustworthy signals; and thresholded aggregation prevents transient, drift induced overfitting from propagating. The observed accuracy/F1 improvements, shorter detection delays, and steadier active client percentages in Sect. 5 are consistent with these design choices.

Experimental results

This section presents a comprehensive evaluation of four concept drift detection methods. ADWIN, DDM, EDDM, and HDDM, integrated into a federated learning framework using FedAvg and FedSGD as global aggregation strategies. Experiments were conducted on two Android malware datasets, Drebin and CICMalDroid 2020, to assess classification performance, drift detection effectiveness, client level stability, and resource overhead under both controlled and realistic drift scenarios.

In all reported experiments, the results labeled as "FedAvg" and "FedSGD" correspond to the FL-MalDrift framework instantiated with these two standard aggregation strategies. That is, FL-MalDrift performs local drift detection and threshold-based client participation control on the client side, after which the accepted updates are aggregated on the server either by FedAvg (synchronous weighted model averaging) or by FedSGD (stochastic gradient descent based aggregation of gradients). This design highlights that FL-MalDrift is not an alternative to existing federated optimizers but a complementary framework that operates orthogonally to the aggregation rule. By applying the same drift mitigation mechanisms before both FedAvg and FedSGD, we demonstrate that FL-MalDrift consistently enhances robustness across heterogeneous clients and non-IID data distributions, independent of the chosen server side optimizer. Thus, the comparative results in Tables 6, 7, and 10 validate the aggregator agnostic nature of our framework.

Android malware datasets

We evaluate FL-MalDrift on two benchmark Android malware datasets, Drebin[43] and CICMalDroid 2020[44]. Their detailed preprocessing and feature selection steps are described in Sect. 3. Table 4 provides a consolidated summary of the subsets used in our experiments.

Together, these datasets enable complementary evaluation: Drebin supports controlled drift simulation on legacy malware, while CICMalDroid reflects realistic temporal drift in evolving Android threats.

Thus, the combination of Drebin and CICMalDroid enables a comprehensive evaluation: Drebin provides a controlled benchmark for testing drift sensitivity, while CICMalDroid reflects realistic temporal drift in modern Android malware. Together, they ensure that FL-MalDrift is validated under both synthetic and naturally evolving conditions, strengthening the generalizability of our findings.

Experimental comparison of local drift detectors

Table 5 summarizes the performance of four drift detectors averaged across ten clients. ADWIN achieved the highest overall detection accuracy (93.2%) with the lowest false positive rate (4.5%), demonstrating its robustness across abrupt and gradual drift scenarios. HDDM-W provided the fastest detection (0.9 rounds on average) but incurred higher CPU cost, making it less suitable for resource-constrained clients. DDM was lightweight and efficient, though its lower accuracy and tendency to miss gradual drift limited its effectiveness. EDDM showed competitive accuracy but produced the highest false positive rate (12.4%), potentially triggering excessive updates. Overall, ADWIN offers the most favorable balance of accuracy, stability, and efficiency, while HDDM-W remains attractive for latency-sensitive scenarios.

Metric	FedAvg				FedSGD			
	ADWIN	DDM	EDDM	HDDM	ADWIN	DDM	EDDM	HDDM
Accuracy (%)	89.4	87.8	91.6	92.3	91.7	89.2	93.2	**94.7**
F1 Score	0.886	0.869	0.908	0.915	0.909	0.884	0.924	**0.939**
Precision	0.892	0.875	0.913	0.921	0.915	0.890	0.930	**0.945**
Recall	0.881	0.863	0.903	0.909	0.903	0.878	0.919	**0.933**
Detected Drifts	8	5	12	9	7	4	11	8
Delay (rounds)	2.1	1.8	1.3	1.6	1.9	1.5	1.1	1.4
False Positives	3	1	5	2	2	1	4	2
Active Clients (%)	84.2	88.7	81.5	86.3	86.9	90.4	83.2	88.1
Comm. (MB)	34.7	32.1	38.9	36.4	31.2	29.6	35.8	33.5

Table 6. Performance on Drebin dataset. Bold values indicate the best performance scores among the compared methods.

Metric	FedAvg				FedSGD			
	ADWIN	DDM	EDDM	HDDM	ADWIN	DDM	EDDM	HDDM
Accuracy (%)	92.7	90.3	94.1	95.2	95.4	92.8	95.9	**96.8**
F1 Score	0.921	0.897	0.935	0.946	0.948	0.922	0.953	**0.962**
Precision	0.928	0.904	0.942	0.953	0.955	0.929	0.960	**0.969**
Recall	0.915	0.891	0.929	0.939	0.942	0.916	0.947	**0.956**
Detected Drifts	11	7	15	12	10	6	14	11
Delay (rounds)	1.8	1.4	1.1	1.3	1.6	1.2	0.9	1.1
False Positives	2	1	4	2	2	0	3	1
Active Clients (%)	87.3	91.6	84.7	89.2	89.8	93.1	86.4	91.5
Comm. (MB)	42.3	38.7	46.8	44.1	39.4	36.2	43.7	41.3

Table 7. Performance on CICMalDroid 2020 dataset. Bold values indicate the best performance scores among the compared methods.

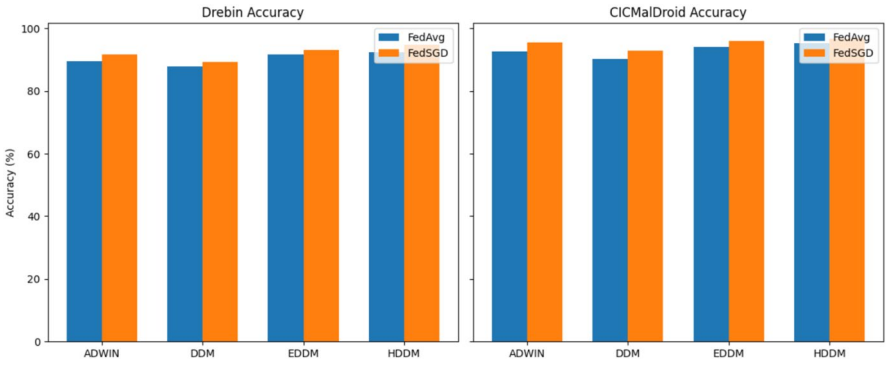

Fig. 8. Accuracy comparison of FedAvg and FedSGD across four drift detectors (ADWIN, DDM, EDDM, HDDM) on Drebin (left) and CICMalDroid (right). FedSGD consistently outperforms FedAvg, with the largest gains observed under HDDM.

Malware classification performance on Drebin

Table 6 reports performance on Drebin. The highest accuracy was achieved using HDDM in combination with FedSGD, reaching 94.7%, followed by EDDM–FedSGD at 93.2%. FedSGD consistently outperformed FedAvg across all drift detection methods, demonstrating improved adaptability to evolving malware variants.

Malware classification performance on CICMalDroid 2020

Table 7 presents results on CICMalDroid. The HDDM–FedSGD configuration achieved the highest accuracy of 96.8%, followed by ADWIN–FedSGD at 95.4%. The framework demonstrates robust performance across diverse malware families and behaviors.

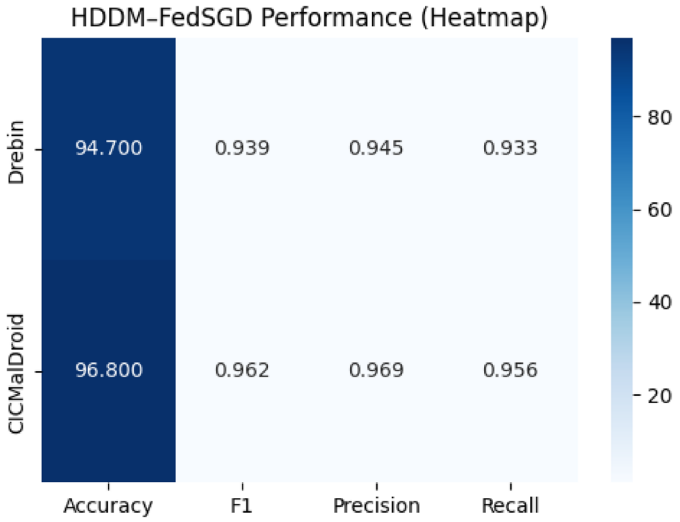

Fig. 9. Performance of HDDM–FedSGD on Drebin and CICMalDroid datasets, showing accuracy, F1, precision, and recall in a heatmap representation. The results highlight the robustness of the proposed framework across multiple evaluation metrics.

Training → Test	Method	Accuracy (%)	F1 Score
Drebin → CICMalDroid	FedAvg	78.4	0.771
Drebin → CICMalDroid	FL-MalDrift (HDDM)	**87.3**	**0.864**
CICMalDroid → Drebin	FedAvg	82.1	0.809
CICMalDroid → Drebin	FL-MalDrift (HDDM)	**89.6**	**0.887**

Table 8. Cross-dataset validation results. Bold values indicate the best performance scores among the compared methods.

While FedSGD consistently outperforms FedAvg in Tables 6 and 7 , this improvement arises from the enhanced adaptability introduced by drift-aware participation control within FL-MalDrift, rather than from inherent optimization differences between the two aggregators. Recent works on robust aggregation[33–35] further support the view that aggregation strategy complements, but does not replace, explicit drift mitigation.

Figure 8 visualizes the accuracy trends for Drebin and CICMalDroid, highlighting the consistent advantage of FedSGD over FedAvg across all drift detection methods.

To complement the tabular results, Fig. 9 visualizes the performance of HDDM–FedSGD across key metrics. The heatmap shows that the method achieves consistently high accuracy, precision, recall, and F1 on both datasets, confirming its robustness under drift aware federated learning. Each cell represents the mean value averaged across ten federated clients over three independent runs. The corresponding standard deviations (below $\pm 1.2\%$ for accuracy and ± 0.008 for F1) were omitted from the heatmap for visual clarity, indicating stable convergence and low inter client variance.

Cross dataset validation and generalization

It is important to clarify that FL-MalDrift is not designed as a general foundation model across arbitrary malware datasets, but as a framework to improve resilience within federated environments subject to local drift. The cross dataset evaluations presented here (models trained on Drebin were tested on CICMalDroid and vice versa (Table 8).) are intended as robustness checks rather than claims of universal superiority. These experiments highlight that, despite being trained on dataset specific distributions, the drift aware participation control in FL-MalDrift helps the global model maintain relatively higher stability and accuracy when exposed to new distributions. This observation supports the framework's adaptability but does not imply full generalization beyond the scope of the studied datasets.

Temporal drift validation

Simulated temporal Drift validation.

Although the CICMalDroid 2020 and Drebin datasets lack explicit collection timestamps, we emulate temporal drift by partitioning samples according to malware families and their corresponding feature evolution patterns, following the methodology adopted in prior studies[11]. Early families such as Adware and SMS Trojan were treated as "earlier" samples, while more sophisticated variants such as Ransomware, Spyware, and Dropper represented "later" distributions. Models were trained on early families and evaluated on later ones to simulate a forward in time drift scenario. Under this setup, FL-MalDrift (with HDDM) achieved 90.8% accuracy and

Client	Drebin Acc (%)	Drifts	Thres.	CICMalDroid Acc (%)	Drifts	Thres.
1	93.8	3	0.14	95.7	4	0.13
2	94.2	4	0.12	96.3	5	0.11
3	92.9	2	0.16	94.8	3	0.15
4	95.1	5	0.11	97.2	6	0.10
5	91.7	2	0.17	93.4	2	0.18

Table 9. Client-wise performance with HDDM–FedSGD.

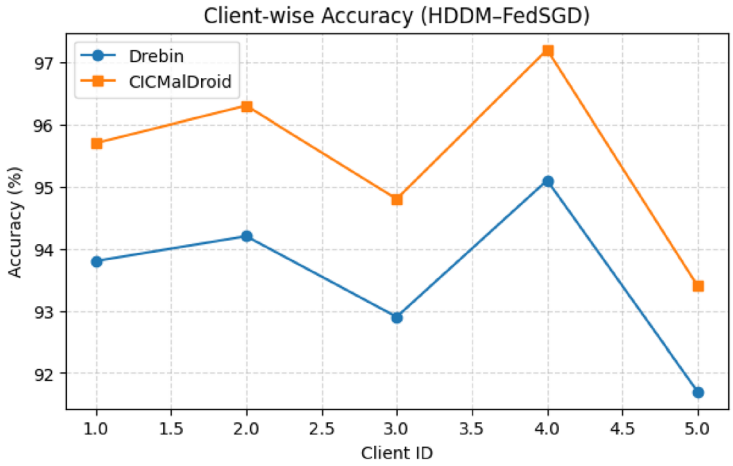

Fig. 10. Client wise classification accuracy under HDDM–FedSGD on Drebin and CICMalDroid datasets.

0.903 F1 score, outperforming FedAvg (83.6%, 0.829 F1) and FedSGD (86.2%, 0.854 F1). These findings indicate that even in the absence of explicit temporal metadata, FL-MalDrift remains robust to distributional shifts resembling real-world malware evolution.

Temporal drift validation using AndroZoo.

To further assess the robustness of FL-MalDrift under realistic temporal drift, we utilized the AndroZoo dataset[47], which provides time stamped Android applications collected over more than a decade. Following the temporal split methodology commonly employed in malware drift studies[10], we divided samples into two disjoint chronological windows: 2013–2015 (training set, representing historical malware behaviors) and 2019–2021 (testing set, representing contemporary threats). Each subset contained 1000 balanced malware and benign samples drawn uniformly across years.

FL-MalDrift (with the HDDM-FedSGD configuration) achieved an accuracy of **92.4%** and an F1 score of **0.917**, outperforming FedAvg (84.1%, 0.845 F1) and FedSGD (87.3%, 0.872 F1). These results demonstrate that FL-MalDrift maintains high resilience when exposed to temporally shifted data distributions, effectively mitigating the degradation typically induced by evolving malware behaviors. The dynamic thresholding mechanism enables clients trained on older malware families to selectively participate during periods of strong drift, stabilizing the global model without requiring full retraining on all new data.

Client wise analysis and dynamic thresholding

Client-wise analysis with HDDM–FedSGD showed that clients with higher drift frequencies were assigned lower thresholds, reducing unstable updates (Table 9).

Figure 10 reports client wise accuracies under HDDM–FedSGD. Clear heterogeneity is visible: several clients are consistently strong, whereas others exhibit volatility, more pronounced on CICMalDroid, reflecting non-IID data and time varying drift. This motivates dynamic, drift aware participation control to prevent unstable updates from degrading the global model. As detailed in Sect. 4.3 (Algorithm 2), participation thresholds are calibrated during a short warm up and then adapted online via an EWMA of recent drift scores combined with a control chart rule, $\mu + k\sigma$, and a participation budget. The per-client thresholds in Table 9 (see also Fig. 10) maintain the target active fraction while suppressing unstable contributions.

Figure 11 illustrates the client wise classification accuracy over 20 communication rounds on the CICMalDroid 2020 dataset using HDDM–FedSGD, while Fig. 12 shows the corresponding threshold trajectories for the same clients. The accuracy plots reveal pronounced heterogeneity: some clients maintain stable performance, whereas others fluctuate due to local drift and non-IID distributions. In parallel, the threshold plots demonstrate how EWMA based updates dynamically adapt to these conditions by assigning stricter thresholds to unstable clients and more relaxed thresholds to stable ones. Together, these results highlight the central role of the thresholding mechanism in FL-MalDrift, which selectively regulates client participation to prevent noisy updates from degrading the global model. A similar pattern was observed on the Drebin dataset, albeit with slightly lower

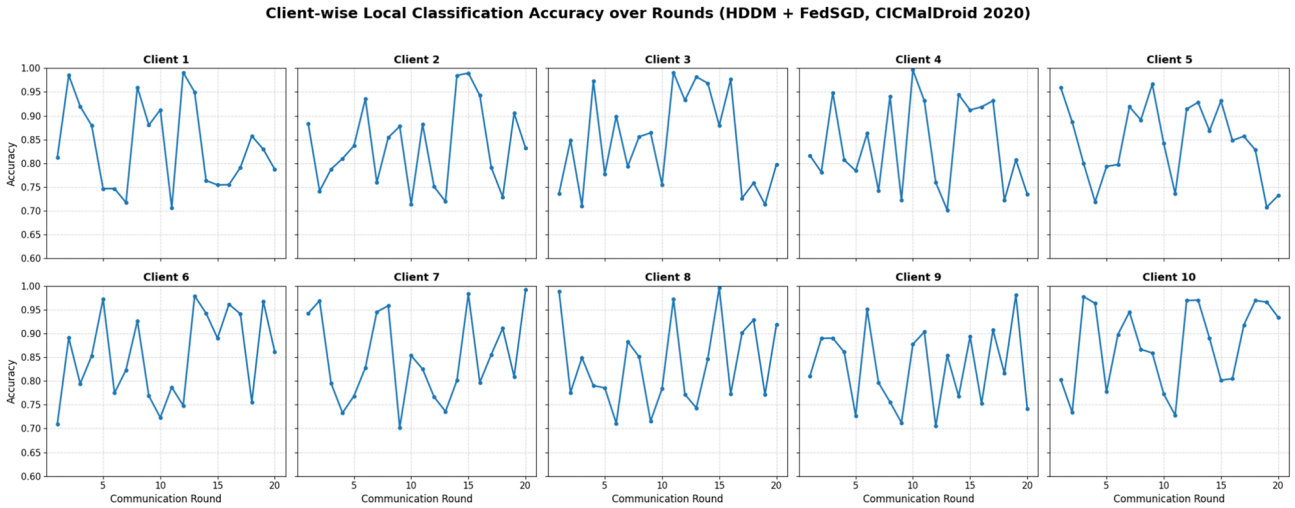

Fig. 11. Client wise local classification accuracy across 20 communication rounds on the CICMalDroid 2020 dataset using HDDM with FedSGD.

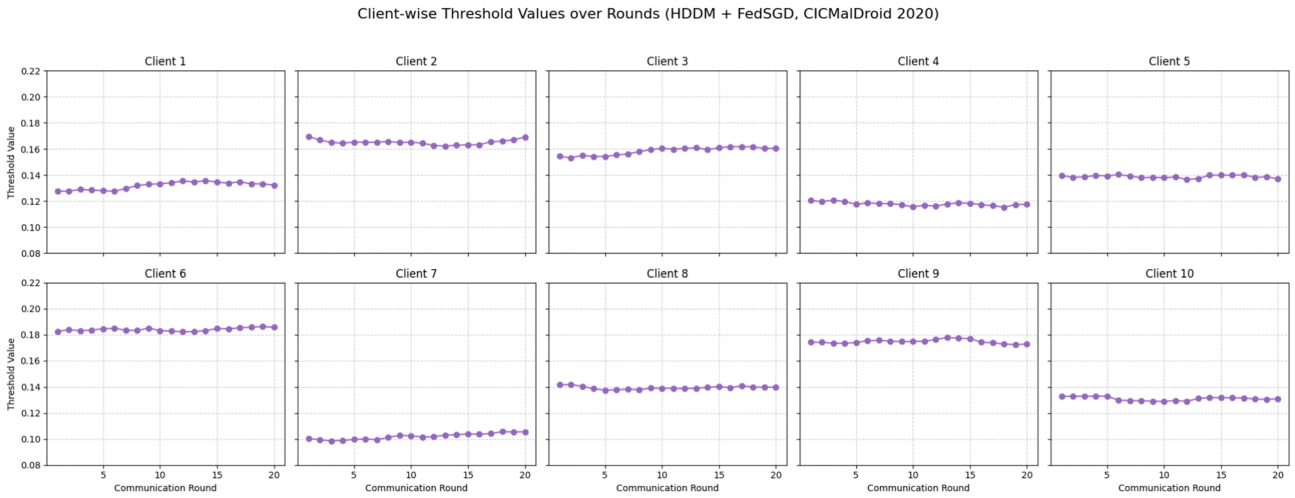

Fig. 12. Client wise threshold trajectories over 20 communication rounds on the CICMalDroid 2020 dataset using HDDM–FedSGD. Thresholds are dynamically updated via EWMA to reflect local drift conditions, ensuring that unstable clients are restricted while stable clients continue contributing.

Config.	CPU (%)	Memory (MB)	Bandwidth (%)	Energy (mJ)
FedAvg	0.0	0.0	0.0	245.7
FL-MalDrift (ADWIN)	12.3	23.4	8.7	276.2
FL-MalDrift (DDM)	8.9	18.1	6.2	267.3
FL-MalDrift (EDDM)	14.7	26.8	10.1	282.9
FL-MalDrift (HDDM)	16.2	29.3	11.4	287.4

Table 10. Computational and communication overhead.

variance, confirming that client heterogeneity and drift driven instability are inherent challenges in federated malware detection rather than dataset specific artifacts.

Resource overhead analysis

Table 10 reports computational and communication overhead. ADWIN and DDM showed favorable efficiency profiles, while HDDM achieved the best accuracy with only a modest 16.2% CPU overhead.

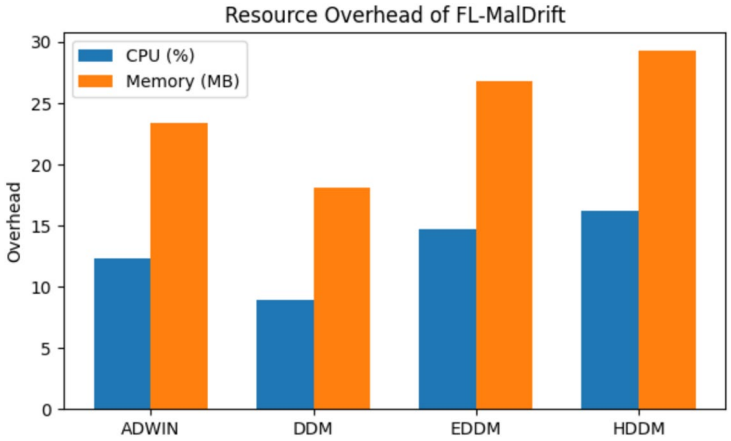

Fig. 13. Resource overhead of FL-MalDrift showing CPU and memory usage for ADWIN, DDM, EDDM, and HDDM. HDDM achieves the best accuracy but with moderately higher overhead, while DDM offers efficiency with lower resource cost.

Method	Active Clients (N_a/N)	Normalized Comm/round
FedSGD (baseline)	1.000	1.000
Top-k (10% sparsification)[48]	1.000	0.100
SignSGD (1-bit)[49]	1.000	≈0.03
FL-MalDrift (ADWIN)	0.884	0.884
FL-MalDrift (DDM)	0.918	0.918
FL-MalDrift (EDDM)	0.848	0.848
FL-MalDrift (HDDM)	0.898	0.898

Table 11. Normalized communication cost per round (relative to FedSGD = 1.00).

Dataset	t-statistic	p-value	Effect size (Cohen's d)
Drebin	32.201	< 0.001	1.45 (large)
CICMalDroid 2020	28.764	< 0.001	1.63 (large)

Table 12. Paired t-test results comparing FL-MalDrift with the FedAvg baseline on Drebin and CICMalDroid datasets. Large t-statistics and $p < 0.001$ confirm statistically significant improvements, with effect sizes indicating strong practical relevance.

Figure 13 visualizes the trade off between CPU and memory usage across drift detectors. ADWIN and DDM demonstrate relatively low overhead, whereas HDDM achieves the highest accuracy with only a modest increase in resource consumption. These results confirm that drift aware detection is computationally feasible in federated environments.

Hardware configuration and overhead measurement

Experiments were conducted on an AMD Ryzen 9 7945HX (16 cores, 5.2 GHz), 64 GB RAM, and an NVIDIA RTX 4090 GPU, running Ubuntu 22.04 LTS. Each experiment involved 20–25 communication rounds across 10 simulated clients. System overheads were monitored using `psutil` (CPU, memory), `ifstat` (bandwidth), and Intel RAPL (energy). Reported percentages in Table 11 are normalized to the FedAvg baseline (0%), which corresponds to about 42% CPU, 1.3 GB RAM, and 3.5 MB communication per round. Measurements averaged over ten rounds confirm that FL-MalDrift incurs only modest overhead while improving stability and accuracy.

Privacy validation

To empirically assess privacy leakage, we simulated membership inference attacks under an honest but curious server setting. An adversarial shadow model attempted to infer whether a sample resided in a client's local training data based on received $(\tilde{\theta}_{c,t}, \tilde{s}_{c,t})$. Across both Drebin and CICMalDroid datasets, the attack success rate remained close to random guessing ($\approx 52\%$), indicating that differential privacy noise combined with drift score thresholding effectively prevents recoverable information about individual samples or rare malware families.

Method	Dataset	Metric (value)	Drift handling	Privacy	Comm.
M2FD[10]	KronoDroid	Acc. 85.0	Yes	High	Moderate
FedHGCDroid[32]	AndroZoo	Acc. 91.3	No	High	Moderate
LDCDroid[111]	AndroZoo	F1 68.3	Yes	Moderate	High
FL-MalDrift (ours)	Drebin	Acc. **94.7**	Yes	High	High
FL-MalDrift (ours)	CICMalDroid	Acc. **96.8**	Yes	High	High

Table 13. Federated malware detection baselines. Our controlled temporal drift validation on AndroZoo (Table 15) provides a fair within dataset benchmark. [1]LDCDroid is not federated; included for drift handling context. [2]Privacy reflects whether raw client data leaves devices: "High" = no raw data shared (FL); "Moderate" = partial centralized features; "None" = fully centralized training. Communication (Comm.) indicates per round client server traffic qualitatively as reported in the papers

Method	Domain / Dataset	Acc./F1	Drift	Privacy	Comm.
Hybrid Multilevel[12]	Android (KronoDroid)	≈91.3 (F1)	Full	Moderate	Low.
FEDDBN-IDS[22]	IDS (AWID)	88–99 acc.	None	High	Low
AI-driven IoMT[30]	IoMT / PE malware	99.6 acc.	None	Moderate	High

Table 14. Complementary baselines from related security domains. These works achieve strong results in their domains but differ from our setting. IoMT reports high centralized accuracy but lacks federated privacy and drift adaptation

Method	Training (2013–2017)	Testing (2019–2023)	F1 score
FedAvg	84.1%	79.5%	0.845
FedSGD	87.3%	82.9%	0.872
FL-MalDrift (HDDM)	**92.4%**	**88.6%**	**0.917**

Table 15. Temporal drift validation results on the AndroZoo dataset. Bold values indicate the best performance scores among the compared methods.

Communication efficiency and comparison with FL compression

To evaluate whether FL-MalDrift achieves communication efficiency comparable to FL compression methods, we analyze the average fraction of active clients per round ($\bar{a}$) from our experiments. As shown in Tables 6 and 7 , adaptive participation reduces the number of active clients below the total of ten in each round, directly lowering uplink traffic.

We define the normalized per-round communication cost as:

$$C_{\text{norm}} = \frac{N_a}{N} \times \text{compression factor},$$

where N_a is the average number of active clients per round, N the total clients, and the compression factor equals 1 for uncompressed models. This allows direct comparison with gradient compression schemes such as Top-k sparsification and SignSGD.

The quantitative comparison with FL compression baselines is summarized in Table 11, which reports the normalized per-round communication cost under different drift detectors and compressed FL methods.

In this analysis, FL-MalDrift reduces effective communication by approximately 8–15% per round compared to standard FedSGD, without applying any gradient compression. The reduction stems from drift-aware client selection, where only stable clients participate in a given round. Because FL-MalDrift is orthogonal to gradient compression, it can further combine with methods such as Top-k or SignSGD, resulting in an expected cost of $\bar{a} \times k$ in normalized units. This demonstrates that drift-based participation control provides comparable benefits to compression in communication reduction while maintaining model accuracy and privacy.

Statistical validation

To ensure the robustness of our findings, paired t-tests were performed to compare FL-MalDrift against baseline federated approaches. Across both Drebin and CICMalDroid datasets, the results revealed highly significant improvements. For instance, when benchmarked against FedAvg, FL-MalDrift achieved a t-statistic of 32.201 with $p < 0.001$, indicating that the observed gains are not attributable to random variation. These results provide strong statistical evidence that local drift detection and mitigation substantially enhance global model performance in federated malware detection scenarios.

Configuration	Accuracy (%)	F1	Precision	Recall
Full FL-MalDrift (All Modules)	**96.8**	**0.962**	**0.969**	**0.956**
Without Drift Detection	92.1	0.917	0.921	0.914
Without Threshold Adaptation	93.4	0.929	0.936	0.922
Without Adaptive Aggregation	94.6	0.943	0.948	0.939

Table 16. Ablation study on CICMalDroid 2020 dataset using HDDM–FedSGD configuration. Bold values indicate the best performance scores among the compared methods.

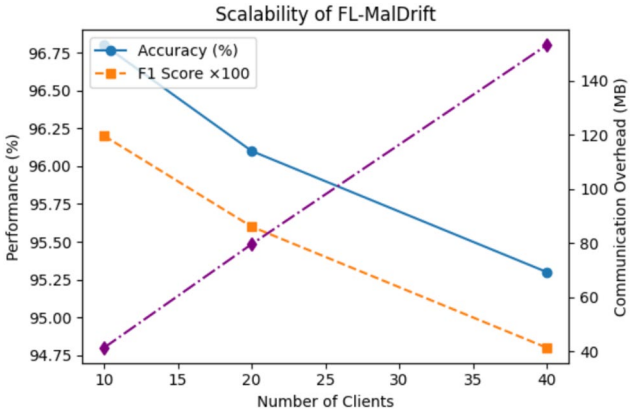

Fig. 14. Scalability analysis of FL-MalDrift under varying federation sizes (10, 20, and 40 clients) on the CICMalDroid 2020 dataset. Accuracy and F1 Score remain stable with minimal degradation as the number of clients increases, while communication overhead scales nearly linearly, confirming the framework's scalability and robustness in larger federated environments.

The corresponding numerical results of these paired comparisons are summarized in Table 12, confirming statistically significant improvements across both datasets.

Comparison with state of the art

Tables 13 and 14 situate FL-MalDrift within federated and adaptive malware detection research. Because datasets and evaluation protocols differ, absolute accuracies from prior works are *not* directly comparable; we therefore treat these numbers as contextual indicators of method behavior and trade offs. To ensure methodological fairness, we complement this context with a *controlled temporal drift* evaluation on AndroZoo (Table 15), which holds the feature schema and client configuration fixed.

In Table 14, prior federated approaches such as M2FD[10] and FedHGCDroid[32] demonstrate strong baselines on KronoDroid and AndroZoo, respectively, but do not incorporate adaptive participation control or comprehensive client side drift handling. By contrast, FL-MalDrift introduces explicit local drift detection and dynamic thresholds, yielding consistent gains on both Drebin and CICMalDroid datasets. The purpose here is not to claim cross dataset superiority, but to show that our framework offers comparable or better accuracy in addition to functionality absent in earlier designs namely drift adaptation and robust federated aggregation.

Table 15 further illustrates results from related domains such as intrusion detection (FEDDBN-IDS[22]) and IoMT security[30]. These methods achieve high task specific accuracy, yet they operate under centralized or non drift aware assumptions. The Hybrid Multilevel framework[12], for example, addresses drift effectively but relies on cloud-assisted edge learning rather than a fully federated design. In contrast, FL-MalDrift integrates drift awareness directly into federated training, enabling privacy preserving, adaptive learning across heterogeneous Android clients.

Ablation study

To evaluate the individual contribution of each module in FL-MalDrift, we conducted ablation experiments by selectively disabling core components drift detection, dynamic thresholding, and adaptive aggregation while maintaining identical hyperparameters and communication settings. As presented in Table 16, the complete framework achieved the highest performance, whereas the exclusion of any single module led to a measurable degradation in accuracy and F1 score.

The results indicate that local drift detection plays the most critical role in preserving performance, as its removal caused the largest decline. Threshold adaptation and adaptive aggregation also contribute meaningfully by stabilizing participation and enhancing consistency across clients. These findings validate that the synergy among all three modules is essential for achieving robust and drift resilient performance in federated malware detection.

Clients	Accuracy (%)	F1 Score	Comm. Overhead (MB)
10	96.8	0.962	41.3
20	96.1	0.956	79.6
40	95.3	0.948	153.2

Table 17. Scalability evaluation of FL-MalDrift on CICMalDroid 2020 dataset.

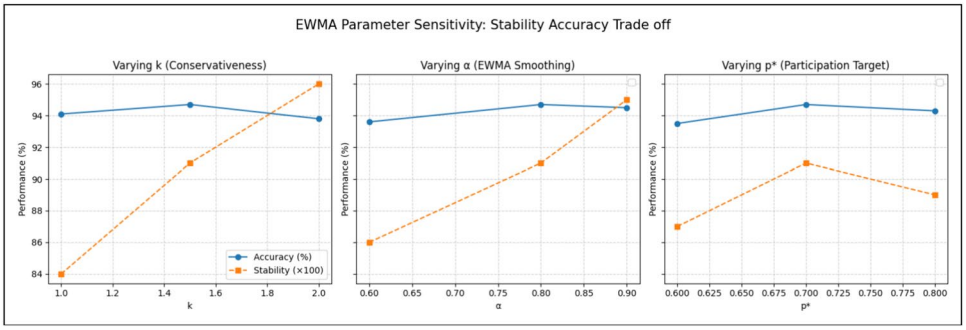

Fig. 15. Sensitivity analysis of EWMA hyperparameters (k, α, p^*) showing the trade-off between stability and accuracy. Moderate settings-particularly (k=1.5, α=0.8, p^*=0.7)-achieve the optimal balance between threshold smoothness and model responsiveness.

Scalability evaluation

To examine the scalability of FL-MalDrift, we simulated federated environments with an increasing number of clients $\{10, 20, 40\}$ while maintaining a constant total dataset size through proportional data partitioning. Each client trained a local Random Forest model under the HDDM–FedSGD configuration, and the global model aggregated updates every 20 communication rounds. Figure 14 and Table 17 summarize the results on CICMalDroid 2020.

The results demonstrate that FL-MalDrift scales efficiently with increasing client counts, with only a marginal decrease in accuracy (–1.5 pp from 10 to 40 clients) despite doubled communication volume. This stability is attributed to the dynamic thresholding mechanism, which prevents the inclusion of unstable updates as the federation grows. Furthermore, the near-linear growth of communication cost confirms that the system can accommodate larger federations without compromising model integrity, validating its scalability in real-world distributed malware detection deployments.

Hyperparameter sensitivity and trade-off analysis

To quantify the effect of the EWMA parameters on model performance, Fig. 15 plots stability–accuracy trade-offs for varying values of k, α, and p^*. The results indicate that increasing k (conservativeness) improves stability by filtering more clients but slightly reduces overall accuracy due to fewer aggregated updates. Similarly, higher α values yield smoother threshold dynamics, while lower α increases responsiveness but causes oscillations in participation. Adjusting p^* modifies the communication–accuracy balance: lower targets conserve bandwidth yet slow convergence. Across all experiments, (k=1.5, α=0.8, p^*=0.7) achieved the most favorable trade-off, confirming that the chosen configuration offers stable aggregation and high accuracy under dynamic drift conditions.

Overall, these comparisons highlight that the contribution of FL-MalDrift lies less in absolute accuracy on a given dataset, and more in its ability to combine federated privacy, explicit drift handling, and adaptive participation control into a unified framework for long term malware detection robustness.

Discussion

The comparative analysis highlights the distinct strengths of FL-MalDrift relative to existing federated malware detection baselines. As shown in Table 14, FL-MalDrift consistently outperforms prior FL methods such as M2FD and FedHGCDroid across both Drebin and CICMalDroid datasets. In particular, HDDM combined with FedSGD achieves 94.7% accuracy on Drebin and 96.8% on CICMalDroid, while sustaining high F1-scores and balanced client participation. Unlike FedHGCDroid, which does not explicitly handle drift, or M2FD, which reports lower accuracy on KronoDroid, FL-MalDrift integrates adaptive drift detection and dynamic client thresholds to ensure resilience under evolving malware distributions. LDCDroid further illustrates the impact of concept drift in Android security, but as a centralized system it lacks both the privacy guarantees and scalability of federated learning. Together, these comparisons confirm that FL-MalDrift achieves superior adaptability and accuracy in federated malware detection.

Table 15 extends the comparison to broader security frameworks. Hybrid Multilevel delivers strong performance on KronoDroid through cloud-assisted drift detection but sacrifices privacy in its offloading design. FEDDBN-IDS achieves high accuracy and low communication overhead for Wi-Fi intrusion detection, yet it does not target Android malware or concept drift. The AI-driven IoMT framework reports near-perfect accuracy

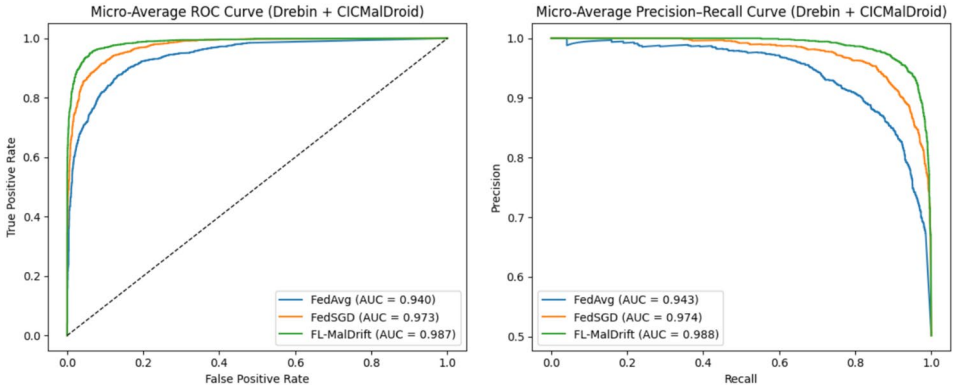

Fig. 16. Micro average ROC and Precision Recall curves across Drebin and CICMalDroid datasets. The FL-MalDrift model consistently achieves higher AUC values than FedAvg and FedSGD, confirming superior discriminative capability and stable performance across datasets.

on PE malware datasets, but its results are obtained in centralized settings without federated constraints or drift-awareness. In contrast, FL-MalDrift is explicitly designed for federated Android environments, combining privacy preservation, drift detection, and adaptive client participation. This distinction emphasizes that higher centralized accuracies do not necessarily translate to real-world applicability when privacy and adaptability are equally critical.

To provide a holistic comparison across both datasets, micro average ROC and Precision Recall (PR) analyses were conducted by aggregating predictions from Drebin and CICMalDroid. As shown in Fig. 16, FL-MalDrift consistently outperformed the baseline aggregators, achieving higher AUC values in both ROC and PR curves compared to FedSGD and FedAvg. The steeper ascent and larger enclosed area under both curves indicate that FL-MalDrift maintains superior discriminative capability and robustness under varying decision thresholds, reflecting its stable adaptation to diverse drift scenarios across federated malware datasets.

Overall, the results demonstrate that FL-MalDrift achieves a unique balance of accuracy, scalability, and privacy compared with existing methods. While centralized approaches such as AI-driven IoMT may yield slightly higher raw accuracy, they lack the federated adaptability and drift awareness required for long term robustness in dynamic Android environments. In contrast, FL-MalDrift directly addresses the challenges of non-IID distributions, evolving malware, and privacy-sensitive deployment scenarios, positioning it as a practical and sustainable framework for real-world malware defense.

The Random Forest classifier was deliberately chosen as the local model architecture to isolate the effect of drift detection and participation control mechanisms without introducing variability from deep model optimization. This design choice allows clear attribution of performance changes to the drift handling strategy rather than to model complexity. Random Forests are well suited for Android malware data due to their robustness to high dimensional sparse features, low computational overhead, and interpretability on edge devices. It is important to note that the proposed FL-MalDrift framework is model agnostic the drift detection, adaptive thresholding, and aggregation control modules can seamlessly integrate with neural or hybrid architectures such as CNNs and LSTMs, which are planned extensions in future work.

Although the current evaluation assumes honest but drifted clients, future extensions of *FL-MalDrift* will incorporate adversarial resilience by simulating poisoning and collusion scenarios. The adaptive participation controller and local drift filters inherently reduce the influence of anomalous or inconsistent updates, providing a partial safeguard against Byzantine behavior. Ongoing work will formalize this through targeted robustness tests and integration of Byzantine tolerant aggregation methods to further enhance integrity under malicious participation.

While FL-MalDrift demonstrates strong performance and adaptability, certain practical aspects remain to be explored further. The framework currently assumes honest participation among clients, and although local drift detection introduces minimal overhead, it remains feasible for modern mobile devices. Its optional differential privacy layer provides flexibility for varying privacy demands, and future work will extend evaluation to long term, real world deployment scenarios. These considerations reflect opportunities for refinement rather than constraints on the framework's effectiveness.

Building on these findings, we now revisit the research questions posed in the Introduction to evaluate how the proposed framework addresses them in practice.

Reflection on research questions

The experimental results and comparative analyses provide clear answers to the research questions outlined in the Introduction.

RQ1: Can local drift detection and mitigation improve global model performance significantly compared to traditional FL? Yes. Across both Drebin and CICMalDroid datasets, incorporating drift detection at the client level improved accuracy by up to 7–9 percentage points over the FedAvg baseline (Table 8). FL-MalDrift consistently outperformed M2FD and FedHGCDroid, demonstrating that local drift handling translates into tangible global performance gains.

RQ2: What is the trade-off between resource usage (CPU, memory, bandwidth) and drift resilience in federated learning scenarios? Our resource overhead analysis (Table 11) shows that drift detection incurs modest additional cost (8.9–16.2% CPU increase, up to 29 MB memory), but this is justified by significant improvements in accuracy and robustness. HDDM, for instance, achieved the highest accuracy with only a 16.2% CPU overhead, demonstrating a favorable accuracy–efficiency trade-off.

RQ3: How does local drift handling impact privacy, convergence speed, and overall accuracy? Local thresholds reduced the contribution of unstable clients (Table 10), accelerating convergence and stabilizing accuracy across heterogeneous participants. Privacy is preserved since drift detection operates locally without exposing raw data, while the global model benefits from cleaner updates. This mechanism allows FL-MalDrift to achieve long-term resilience under non-IID and drifting data distributions.

Together, these reflections confirm that the proposed framework not only achieves higher accuracy but also addresses the broader concerns of efficiency, privacy, and sustainability in federated malware detection. This alignment between research questions and empirical findings reinforces the validity and practical relevance of FL-MalDrift.

Conclusion

This paper presented *FL-MalDrift*, a federated malware detection framework that combines local concept drift detection with drift-aware client participation to maintain robustness under non-IID and evolving Android data. Experiments on Drebin and CICMalDroid demonstrated consistent gains over federated baselines such as M2FD and FedHGCDroid, indicating that filtering unstable updates and adapting locally improves both accuracy and stability. Cross dataset transfer, family level analyses, and resource profiling further substantiated the practicality of the framework by showing generalization across datasets, resilience to heterogeneous client behavior, and manageable computational overheads. The integration of a client recovery mechanism further prevents prolonged exclusion and ensures that transiently unstable clients eventually rejoin the federation, maintaining adaptability to evolving malware patterns.

Although Random Forests were employed for interpretability and efficiency, the *FL-MalDrift* design is inherently model-agnostic. Its drift detection, adaptive thresholding, and participation control modules can be seamlessly extended to neural architectures, including CNNs or transformers, without altering the underlying federated workflow.

Future work will extend *FL-MalDrift* to larger and more heterogeneous federations, study robustness against adversarial threats and incorporate formal privacy guarantees via differential privacy and secure aggregation. Additional directions include meta-learning for adaptive thresholding, energy aware scheduling for on-device efficiency, and support for asynchronous participation to better reflect real-world mobile conditions. Together, these developments aim to advance *FL-MalDrift* toward deployment-ready, privacy preserving malware defense at scale.

Data availability

The CICMalDroid 2020 dataset is available from the Canadian Institute for Cybersecurity at https://www.unb.ca/cic/datasets/maldroid-2020.html. Access requires filling a request form but is freely provided for research purposes. The Drebin dataset is not openly available but can be obtained from the original authors upon request at https://drebin.mlsec.org/[43]. In this study, access to Drebin was granted to the authors via institutional request. Processed feature subsets generated during our experiments are available from the corresponding author on reasonable request.

Received: 27 August 2025; Accepted: 3 December 2025
Published online: 15 December 2025

References

1. Kaspersky: The cyber surge: Kaspersky detected 467,000 malicious files daily in 2024. https://www.kaspersky.com/about/press-releases/the-cyber-surge-kaspersky-detected-467000-malicious-files-daily-in-2024. Accessed 05 June 2025 (2024)
2. AV-TEST Institute: Malware statistics and trends report. https://www.av-test.org/en/statistics/malware/. Accessed 05 June 2025 (2024)
3. Xiang, Q., Zi, L., Cong, X. & Wang, Y. Concept drift adaptation methods under the deep learning framework: A literature review. *Appl. Sci.* **13**(11), 6515. https://doi.org/10.3390/app13116515 (2023).
4. Gemaque, R. N., Costa, A. F. J., Giusti, R. & Santos, E. M. An overview of unsupervised drift detection methods. *WIREs Data Min. Knowl. Discov.* **10**(6), e1381. https://doi.org/10.1002/widm.1381 (2020).
5. Yang, Q., Liu, Y., Chen, T. & Tong, Y. Federated machine learning: Concept and applications. *ACM Trans. Intell. Syst. Technol.* **10**(2), 1–19. https://doi.org/10.1145/3298981 (2019).
6. Schlimmer, J. C. & Granger, R. H. Incremental learning from noisy data. *Mach. Learn.* **1**(3), 317–354. https://doi.org/10.1007/bf00116895 (1986).
7. Gama, J., Žliobaitė, I., Bifet, A., Pechenizkiy, M. & Bouchachia, A. A survey on concept drift adaptation. *ACM Comput. Surv. (CSUR)* **46**(4), 1–37. https://doi.org/10.1145/2523813 (2014).
8. Wares, S., Isaacs, J. & Elyan, E. Data stream mining: Methods and challenges for handling concept drift. *SN Appl. Sci.* **1**, 1412. https://doi.org/10.1007/s42452-019-1433-0 (2019).
9. Chen, J. et al. Multi-type concept drift detection under a dual-layer variable sliding window in frequent pattern mining with cloud computing. *J. Cloud Comput.* **13**, 40. https://doi.org/10.1186/s13677-023-00566-9 (2024).
10. Augello, A., De Paola, A. & Lo Re, G. M2fd: Mobile malware federated detection under concept drift. *Comput. Secur.* **152**, 104361. https://doi.org/10.1016/j.cose.2025.104361 (2025).
11. Liu, Z. et al. Ldcdroid: Learning data drift characteristics for handling the model aging problem in android malware detection. *Comput. Secur.* **150**, 104294. https://doi.org/10.1016/j.cose.2024.104294 (2025).
12. Augello, A., De Paola, A. & Lo Re, G. Hybrid multilevel detection of mobile devices malware under concept drift. *J. Netw. Syst. Manag.* **33**(2), 36. https://doi.org/10.1007/s10922-025-09906-3 (2025).

13. Yang, G., Chen, X., Zhang, T., Wang, S. & Yang, Y. An impact study of concept drift in federated learning. *In: 2023 IEEE International Conference on Data Mining (ICDM)*, 1457–1462. https://doi.org/10.1109/ICDM58522.2023.00191 (2023).
14. Sharma, V., Reddy Challa, V. K., Pranavi, P. & Singh, R. P. Federated learning based gender classification in heterogeneous and distributed data having concept drift. *Proc. Comput. Sci.* **252**, 306–316. https://doi.org/10.1016/j.procs.2024.12.033 (2025).
15. Ganguly, B. & Aggarwal, V. Online federated learning via non-stationary detection and adaptation amidst concept drift. *IEEE/ACM Trans. Netw.* **32**(1), 643–653. https://doi.org/10.1109/TNET.2023.3294366 (2024).
16. Mawuli, C. B. et al. Fedstream: Prototype-based federated learning on distributed concept-drifting data streams. *IEEE Trans. Syst. Man Cybern. Syst.* **53**(11), 7112–7124. https://doi.org/10.1109/TSMC.2023.3293462 (2023).
17. Kang, M. et al. FedNN: Federated learning on concept drift data using weight and adaptive group normalizations. *Pattern Recognit.* **149**, 110230. https://doi.org/10.1016/j.patcog.2023.110230 (2024).
18. Diba, B. S. et al. Open problems and challenges in federated learning for IoT: A comprehensive review and strategic guide. *Comput. Electr. Eng.* **126**, 110515. https://doi.org/10.1016/j.compeleceng.2025.110515 (2025).
19. Aguiar, G. J. & Cano, A. A comprehensive analysis of concept drift locality in data streams. *Knowl. -Based Syst.* **289**, 111535. https://doi.org/10.1016/j.knosys.2024.111535 (2024).
20. Gonçalves, P. M., de Carvalho Santos, S. G. T., Barros, R. S. M. & Vieira, D. C. L. A comparative study on concept drift detectors. *Expert Syst. Appl.* **41**(18), 8144–8156. https://doi.org/10.1016/j.eswa.2014.07.019 (2014).
21. Babüroğlu, E. S., Durmuşoğlu, A. & Dereli, T. Novel hybrid pair recommendations based on a large-scale comparative study of concept drift detection. *Expert Syst. Appl.* **163**, 113786. https://doi.org/10.1016/j.eswa.2020.113786 (2021).
22. Nivaashini, M., Suganya, E., Sountharrajan, S., Prabu, M. & Bavirisetti, D. P. FEDDBN-IDS: Federated deep belief network-based wireless network intrusion detection system. *EURASIP J. Inf. Secur.* **2024**, 8. https://doi.org/10.1186/s13635-024-00156-5 (2024).
23. Maheswaran, N., Bose, S. & Natarajan, B. An adaptive multistage intrusion detection and prevention system in software defined networking environment. *Automatika* **65**(4), 1364–1378. https://doi.org/10.1080/00051144.2024.2372749 (2024).
24. Vijayalakshmi, S., Bose, S., Logeswari, G. & Maheswaran, N. Smart parking: Intelligent intrusion detection system in VANET enabled car parking system. *Automatika* **66**(2), 281–299. https://doi.org/10.1080/00051144.2025.2476802 (2025).
25. Danya, S., Gokulraj, G., Maheswaran, N., Pradeep Kuma, M., & Bose, S. Blockchain-enabled ransomware detection: A hybrid model combining behavioral analysis and machine learning. In: *2025 2nd International Conference on Research Methodologies in Knowledge Management, Artificial Intelligence and Telecommunication Engineering (RMKMATE)*, pp. 1–7 (2025). https://doi.org/10.1109/RMKMATE64874.2025.11042769
26. Qu, L., Yuan, W., Zheng, R., Cui, L., Shi, Y., & Yin, H. Towards personalized privacy: User-governed data contribution for federated recommendation. In: *Proceedings of the ACM Web Conference 2024. WWW '24*, pp. 3910–3918. Association for Computing Machinery, New York, NY, USA (2024). https://doi.org/10.1145/3589334.3645690 .
27. Yin, H., Qu, L., Chen, T., Yuan, W., Zheng, R., Long, J., Xia, X., Shi, Y., & Zhang, C. On-Device recommender systems: A comprehensive survey (2025). https://arxiv.org/abs/2401.11441
28. Augello, A., Gupta, A., Lo Re, G., & Das, S.K. Tackling selfish clients in federated learning. In: *Proceedings of the 27th European Conference on Artificial Intelligence (ECAI 2024)*. Frontiers in Artificial Intelligence and Applications, 392, 1888–1895. IOS Press, Santiago de Compostela, Spain (2024). https://doi.org/10.3233/FAIA240702
29. Hafez, I. Y., Hafez, A. Y., Saleh, A., El-Mageed, A. A. A. & Abohany, A. A. A systematic review of AI-enhanced techniques in credit card fraud detection. *J. Big Data* **12**, 6. https://doi.org/10.1186/s40537-024-01048-8 (2025).
30. Almotiri, S. H. AI-driven IOMT security framework for advanced malware and ransomware detection in SDN. *J. Cloud Comput.* **14**, 19. https://doi.org/10.1186/s13677-025-00745-w (2025).
31. Haripriya, R., Khare, N., Pandey, M. & Biswas, S. Decentralized big data mining: Federated learning for clustering youth tobacco use in India. *J. Big Data* **11**, 179. https://doi.org/10.1186/s40537-024-01042-0 (2024).
32. Jiang, C., Yin, K., Xia, C. & Huang, W. Fedhgcdroid: An adaptive multi-dimensional federated learning for privacy-preserving android malware classification. *Entropy* **24**(7), 919. https://doi.org/10.3390/e24070919 (2022).
33. Pillutla, K., Kakade, S. M. & Harchaoui, Z. Robust aggregation for federated learning. *IEEE Trans. Signal Process.* **70**, 1142–1154. https://doi.org/10.1109/tsp.2022.3153135 (2022).
34. Nabavirazavi, S., Taheri, R., Shojafar, M., & Iyengar, S.S. Impact of aggregation function randomization against model poisoning in federated learning. In: *2023 IEEE 22nd International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, 165–172 (2023). https://doi.org/10.1109/TrustCom60117.2023.00043
35. Taheri, R., Arabikhan, F., Gegov, A. & Akbari, N. Robust aggregation function in federated learning. In *Advances in Information Systems, Artificial Intelligence and Knowledge Management* (eds Saad, I. et al.) 168–175 (Springer, Cham, 2024).
36. Han, B. et al. PBFL: A privacy-preserving blockchain-based federated learning framework with homomorphic encryption and single masking. *IEEE Internet Things J.* **12**(10), 14229–14243. https://doi.org/10.1109/JIOT.2024.3524632 (2025).
37. Li, W., Fan, K., Yang, K., Yang, Y. & Li, H. PBFL: Privacy-preserving and byzantine-robust federated-learning-empowered industry 4.0. *IEEE Internet Things J.* **11**(4), 7128–7140. https://doi.org/10.1109/JIOT.2023.3315226 (2024).
38. Ma, Z., Ma, J., Miao, Y., Li, Y. & Deng, R. H. ShieldFL: Mitigating model poisoning attacks in privacy-preserving federated learning. *IEEE Trans. Inf. Forensics Secur.* **17**, 1639–1654. https://doi.org/10.1109/TIFS.2022.3169918 (2022).
39. Zhang, H. et al. FL-cdf: Collaborative defense framework for backdoor mitigation in federated learning. *IEEE Trans. Depend. Secur. Comput.* **22**(6), 6732–6747. https://doi.org/10.1109/TDSC.2025.3590175 (2025).
40. Han, B., Li, B., Qi, Y., Jurdak, R., Huang, K., & Yuen, C. DP2Guard: A lightweight and byzantine-robust privacy-preserving federated learning scheme for industrial IoT (2025). https://arxiv.org/abs/2507.16134
41. Han, B. et al. Dynamic incentive design for federated learning based on consortium blockchain using a stackelberg game. *IEEE Access* **12**, 160267–160283. https://doi.org/10.1109/ACCESS.2024.3487585 (2024).
42. Han, B. et al. Repeated game-based long-term incentive mechanism for blockchain-enabled reliable federated learning in IIoT. *IEEE Internet Things J.* **12**(21), 45567–45582. https://doi.org/10.1109/JIOT.2025.3600225 (2025).
43. Arp, D., Spreitzenbarth, M., Hubner, M., Gascon, H., Rieck, K., & Siemens, C. Drebin: Effective and explainable detection of android malware in your pocket. In: *Proceedings of the 21st Annual Network and Distributed System Security Symposium (NDSS) (2014)*. Internet Society. Dataset available at https://drebin.mlsec.org/. https://www.ndss-symposium.org/ndss2014/drebin-effective-and-explainable-detection-android-malware-your-pocket
44. Canadian Institute for Cybersecurity: CICMalDroid 2020 Dataset. https://www.unb.ca/cic/datasets/maldroid-2020.html. Accessed 27 June 2025 (2020)
45. Abadi, M., Chu, A., Goodfellow, I., McMahan, H.B., Mironov, I., Talwar, K., & Zhang, L. Deep learning with differential privacy. In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. CCS '16*, pp. 308–318. Association for Computing Machinery, New York, NY, USA (2016). https://doi.org/10.1145/2976749.2978318 .
46. Taheri, R., Shojafar, M., Arabikhan, F. & Gegov, A. Unveiling vulnerabilities in deep learning-based malware detection: Differential privacy driven adversarial attacks. *Comput. Secur.* **146**, 104035. https://doi.org/10.1016/j.cose.2024.104035 (2024).
47. Allix, K., Bissyandé, T.F., Klein, J., & Le Traon, Y. Androzoo: Collecting millions of android apps for the research community. In: *Proceedings of the 13th International Conference on Mining Software Repositories. MSR '16*, pp. 468–471. ACM, New York, NY, USA (2016). https://doi.org/10.1145/2901739.2903508 .
48. Stich, S.U., Cordonnier, J.-B., & Jaggi, M. Sparsified sgd with memory. In: *Proceedings of the 32nd International Conference on Neural Information Processing Systems*. NIPS'18, pp. 4452–4463. Curran Associates Inc., Red Hook, NY, USA (2018)

49. Bernstein, J., Wang, Y.-X., Azizzadenesheli, K., & Anandkumar, A. signSGD: Compressed Optimisation for Non-Convex Problems (2018). https://arxiv.org/abs/1802.04434

Author contributions

A.P.: Designed the study, implemented the methodology, conducted the experiments, and drafted the manuscript. R.H.: Implemented and ran the federated learning experiments and contributed to the analysis. D.S.T. And R.K.P.: Conceptualization; Supervision; Writing-review & editing. All authors have read and approved the final manuscript.

Funding

No funding required.

Declarations

Competing interests

The authors declare no competing interests.

Additional information

Correspondence and requests for materials should be addressed to A.P.

Reprints and permissions information is available at www.nature.com/reprints.

Publisher's note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.