# Detect, Adapt, Overcome: Mitigating Concept Drift in Federated Learning

Iftekhar Rahman Nisal Hemadasa Dominik Kaaser Pierre-Alexandre Murena Stefan Schulte

*Christian Doppler Laboratory for Blockchain Technologies for the Internet of Things*
TU Hamburg, Hamburg, Germany

***Abstract*—*Federated Learning (FL)* is a distributed machine learning paradigm where a central server coordinates the training of a global model across multiple decentralized clients. Most FL algorithms assume stationary data-generating processes and neglect concept drift, i.e., the change of data distributions over time. This results in a gradual decline of the model's performance and eventually renders the model obsolete.**

**We investigate the impact of concept drift in FL and propose an active approach to detect and adapt to drift. Our detection method is based on local model loss which allows us to categorize clients based on the type of drift they experience and apply appropriate adaptive measures. Through extensive experiments, we demonstrate the efficiency, accuracy, and precision of our approach: the proposed adaptation approach successfully accelerates convergence for drifted clients, outperforming the baseline algorithm. Even under extreme drift conditions, our global model remains stable. This demonstrates the superior robustness of our approach compared to conventional methods.**

***Index Terms*—Machine Learning, Federated Learning, Concept Drift, Model Adaptation, Drift Detection, Non-IID Data**

## I. Introduction

The amount of data in application domains such as healthcare, smart cities, and autonomous systems is growing rapidly. Processing data in the cloud, however, is often not feasible due to the communication overhead or data protection requirements. Edge computing addresses this trend by processing data locally on devices at the periphery of a network [20]. In this context, *Federated Learning (FL)* [16] has gained a lot of attention. FL allows multiple clients to collaboratively train a global model without sharing raw data, thus minimizing data transfer while ensuring data privacy.

Despite extensive research in FL [22], most approaches assume a stationary data-generating process, where data distributions remain consistent over time [12]. While this assumption simplifies algorithm design and implementation, it is not realistic in real-world applications [1]. Data from edge devices are frequently subject to change due to seasonal patterns, user behavior, hardware failures, or environmental shifts. This change in the data distribution is known as *concept drift*. If left unaddressed, concept drift can severely degrade model performance over time [13].

Various strategies in (centralized) machine learning address concept drift [7]. These methods typically rely on centralized control, access to raw data, or pre-trained models [14], making them effective in traditional machine learning pipelines. In contrast, in FL each client can be subject to different drift patterns, and global model updates obscure individual client's behavior. The decentralized nature of FL, the heterogeneity of client data, and the inability to access raw data require novel approaches to overcome concept drift in FL.

We propose a novel approach to mitigate concept drift in FL. Our method involves a detection step to identify clients affected by concept drift and an adaptation step to counter its impact. We focus specifically on *local* drift: our detection method is based on *client loss*, which reflects how a client's model degrades over time. This allows us to develop a robust and efficient approach to overcome concept drift in FL systems.

## II. Related Work

In their seminal paper, McMahan et al. [16] introduce FL, and their algorithm *FederatedAveraging (FedAvg)* remains one of the most widely used implementations to date. Canonaco et al. [1] build upon FedAvg. Their algorithm *Adaptive-FedAvg* uses exponential moving averages to calculate the current effective learning rate to continuously adapt the model to potential concept drift. This can be seen as a server-side federated variant of the AdaGrad [4] and Adam [10] optimizers. While this adaptation improves over FedAvg, it is primarily tailored to sudden changes but fails to capture slower, gradual, or incremental changes spanning multiple training epochs.

*Concept-Drift-Aware Federated Averaging (CDA-FedAvg)* [2] follows an active approach. It uses an algorithm based on the data distribution to detect drift, leveraging both long-term and short-term memory of local data on the clients to adjust the local model accordingly. Drift is detected if changes in statistical properties of the data distribution surpass a predefined threshold. CDA-FedAvg shows promising results in the narrow domain of human activity recognition on smartphones but its detection accuracy and precision have not yet been thoroughly analyzed in more general environments.

*FedNN* [9] combines weight normalization and adaptive group normalization. They simulate concept drift using images with different styles and backgrounds, but with the same labels, creating heterogeneous concept drift. While their approach appears to achieve better accuracy than FedAvg, its performance under sudden or incremental drift has not been thoroughly analyzed.

*Adaptive Monitoring and Elimination (FLAME)* [15] focuses on active adaptation. It monitors client training by comparing validation loss and training loss, incorporating a confidence

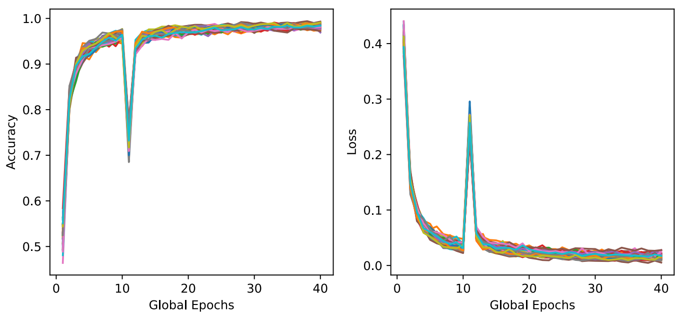

Fig. 1. Client accuracy and loss, global sudden drift. The plots show accuracy and loss for each individual client.

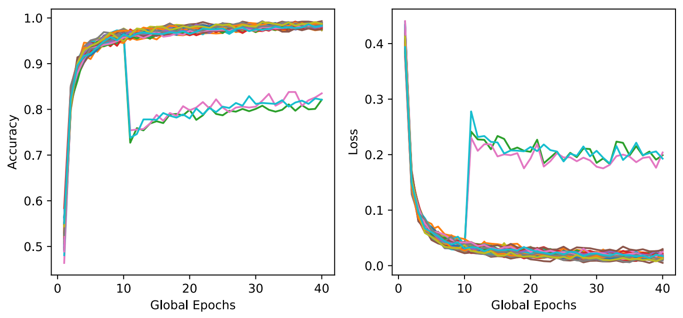

Fig. 2. Client accuracy and loss, local sudden drift at three clients. The plots show accuracy and loss for each individual client.

value for validation results using the Kolmogorov-Smirnov test to detect drift or instability. For adaptation, it applies a memory retention method where, instead of discarding all past samples, it selectively samples data from different concepts when multiple concepts are present. This method demonstrates good F1 scores under concept drift, but as with FedNN, there is no thorough performance analysis under sudden or incremental drift.

Saile et al. [17] build upon Adaptive-FedAvg [1] and focus on the client side. They calculate an adaptive learning rate using exponential moving averages for mean, variance, and variability of client loss, increasing the learning rate for clients with high loss to accelerate convergence. While this method demonstrates fast convergence under sudden drift, it faces challenges in adapting to incremental drift.

Thomas et al. [19] analyze the impact of client selection. They propose a probabilistic client selection algorithm that introduces a bias towards clients with higher local loss. Their approach recovers from sudden drift faster than random sampling but they do not analyze other drift types.

Kumar et al. [11] treat concept drift as a selective forgetting problem and use techniques from federated unlearning to remove obsolete knowledge while adapting to new data patterns.

In contrast to centralized learning, few FL algorithms are specifically designed to *detect* drift. Stallmann et al. [18] follow a fuzzy clustering approach to identify sudden drift, and Manias et al. [14] detect drift using $k$-Means clustering after dimensionality reduction via principal component analysis.

Algorithms like Adaptive-FedAvg, CDA-FedAvg, FedNN, and FLAME improve upon FedAvg when addressing a single type of drift. However, these methods rely on a single global model and are thus limited in dynamic environments where clients experience multiple types of concept drift. To address this, Jothimurugesan et al. [8] introduce FedDrift, which employs a multiple model strategy. In this approach, a drift detection mechanism monitors model loss. When the loss exceeds a threshold, the affected clients are flagged as drifting. These clients are then isolated and reassigned to a new model and cluster. FedDrift significantly outperforms traditional FL algorithms in managing concept drift, particularly in scenarios involving multiple drifting concepts.

## III. Detecting Drift

We analyze two types of concept drift, *sudden drift* and *incremental drift*. Following the related work, we use image classification for our evaluation. We implement sudden drift via class swaps [1], where we swap the labels for two pairs of classes in some round $r_0$. As suggested in [17], this introduces a drastic drift. In contrast, we introduce incremental drift by applying a gradual affine transformation over multiple epochs. In each epoch of a drifting period $[r_s, r_e]$, we apply a rotation of the image data until an overall rotation of $30°$ is reached. In addition to this increasing rotation, we also increase the number of affected images linearly in the drifting period $[r_s, r_e]$. The affected images are selected randomly.

Preliminary simulations (see Figure 1) show that our model is robust to *global drift* where all clients undergo the same shift in their data distribution at the same time. This is a consequence of the fact that we apply the Adam optimizer, which adjusts to a complex learning plane by considering past gradient descents during backpropagation in an ideal way. We therefore focus on *local drift*, which occurs only at a subset of the clients that experience a change in their data distributions. In this case, the model performance of affected clients degrades significantly and does not recover as in the case of global drift (see Figure 2).

### A. Sudden Drift

Sudden drift in local datasets often results in abrupt shifts in client loss or accuracy values. The most straightforward way to detect this pattern is to monitor clients for significant changes in training performance and, to tell actual drift from outliers apart, to monitor whether these shifts persist. To this end, we use the loss value of the clients. Compared to accuracy, loss values are more sensitive to subtle changes in model performance, since the loss function captures the magnitude of the errors, allowing it to reflect even minor shifts in the data distribution. This more granular view of model performance is crucial for detecting small or gradual changes that could indicate the onset of concept drift. Finally, cross-entropy loss can reveal drift that affects certain classes or subsets of data more than others, which commonly occurs in federated environments where data are not *independent and identically distributed (IID)*.

**Algorithm 1** Sudden Drift Detection

```
Input:    $t$                   current epoch
          $\langle \ell \rangle_0^t$   sequence of client losses for each epoch
          $\delta$              drastic change factor
          $\theta$              sustained high loss threshold
Output:   TRUE                  if client has sustained sudden drift
          FALSE                 otherwise

1: if $\ell_{t-1} > \delta \cdot \ell_{t-2}$ and $\ell_t \geq \theta$ then
2:     return TRUE                          ▷ report client as drifting
3: else
4:     return FALSE                         ▷ no sudden drift detected
```

Our algorithm for detecting sudden local concept drift is given in Algorithm 1. It is run on each client. If a client's loss in epoch $t-1$ exceeded a predefined factor of its previous loss in epoch $t-2$, it is flagged for a drastic change. If the loss remains high beyond a certain threshold in the subsequent epoch $t$, the client is reported to the server as experiencing sustained sudden drift. The server maintains a list of clients that are currently undergoing drift.

For the sake of presentation, Algorithm 1 only considers three consecutive rounds. However, it is straightforward to generalize this to settings where outliers frequently span multiple rounds. In this case, a sustained drift would only be reported if the loss value remains above the threshold $\theta$ for a sufficiently large number of rounds after the drastic change.

### B. *Incremental Drift*

Incremental drift gradually increases over time. To capture this pattern, we propose a separate algorithm that uses two loss windows, a short-term window (sensitive to sudden drift) and a long-term window (required to identify incremental drift). Our main approach is to compare the current loss value against the average loss in these windows in order to analyze whether the trend of increasing loss values persists.

Our algorithm to detect incremental drift is given in Algorithm 2. We monitor two moving averages calculated over different window lengths to detect potential drifts (Lines 1 and 2). Intuitively, if a client's loss is greater than both moving averages, drift is detected. More formally, a client is reported to the server as experiencing incremental drift when its loss exceeds both the short-term average loss $\nu$ and the long-term average loss $\mu$ (Line 3). The short-term average loss $\nu$ is the mean of the client's losses over the last $s$ epochs, while the long-term average loss $\mu$ is the mean of the client's losses over a significantly larger window spanning the last $d$ epochs. Once a drift is detected, we *freeze* the long-term average by storing it in the value $\mu^*$ (Line 5) and use only $\mu^*$ for the long-term average in Line 2. Only when the loss drops below the short-term average and the long-term average (Line 7), the long-term average $\mu$ is released again.

**Algorithm 2** Incremental Drift Detection

```
Input:    $t$                   current epoch
          $\langle \ell \rangle_0^t$   sequence of client losses for each epoch
          $d$                   long window length
          $s$                   short window length
Data:     $\mu^*$               frozen average, initially $\perp$
Output:   TRUE                  if client has sustained incremental drift
          FALSE                 otherwise

1: $\nu \leftarrow \text{avg}\{\ell_{t-s+1}, \ell_{t-s+2}, \ldots, \ell_t\}$
2: $\mu \leftarrow \begin{cases} \mu^* & \text{if } \mu^* \neq \perp \\ \text{avg}\{\ell_{t-d+1}, \ell_{t-d+2}, \ldots, \ell_t\} & \text{otherwise.} \end{cases}$
3: if $\ell_t > \nu$ and $\ell_t > \mu$ then
4:     if $\mu^* = \perp$ then
5:         $\mu^* \leftarrow \mu$                   ▷ freeze the long average
6:     return TRUE                          ▷ report client as drifting
7: if $\ell_t \leq \nu$ and $\ell_t \leq \mu$ then
8:     $\mu^* \leftarrow \perp$                     ▷ release the long average
9: return FALSE                             ▷ no incremental drift detected
```

**Algorithm 3** Adaptation and Global Model Computation

```
Input:    $C$                        set of clients
          $\alpha_s, \alpha_i, \alpha_n$   scaling factors
Data:     $G_{\text{opt}}$           global model
          $G_s, G_i, G_n$            drift models

Client-Side Algorithm:
1: for each epoch do
2:     train client model $M$ on local data
3:     use Algorithms 1 and 2 to detect client drift
4:     send client model $M$ and drift type $t$ to the server

Server-Side Algorithm:
5: for each epoch do
6:     for each client $i = 1$ to $n$ do
7:         receive model $M_i$ from client $i$
8:         receive drift type $t_i$ from client $i$
9:     aggregate sudden drift model $G_s$ from
           $\{M_i \mid t_i = \textit{sudden drift}\}$
10:    aggregate incremental drift model $G_i$ from
           $\{M_i \mid t_i = \textit{incremental drift}\}$
11:    aggregate non-drift model $G_n$ from
           $\{M_i \mid t_i = \textit{no drift}\}$
12:    based on $t_i$, send model $G_s$, $G_i$, or $G_n$ back to client $i$
13:    compute global model $G_{\text{opt}} \leftarrow \alpha_s G_s + \alpha_i G_i + \alpha_n G_n$
```

## IV. Adapting to Drift

We follow an active approach to adapt our model under concept drift: we first detect concept drift and then adapt accordingly. Our adaptation algorithm integrates both detection algorithms Algorithms 1 and 2 and thus applies in scenarios where both types of drift occur simultaneously. The method used for drift detection determines the adaptation strategy, with slight variations depending on the nature of the detection method used. We use a multiple model approach for concept drift adaptation, similarly to [8]. Our algorithm for adaptation to concept drift is given in Algorithm 3.

Our approach uses a loss-based grouping method to detect concept drift and generate multiple models, where each group corresponds to a particular type of drift. We create separate drifted models that are trained exclusively on drifted clients' datasets (Lines 1 to 4). Specifically, clients experiencing drift (either sudden or incremental) are handled differently from non-drifted clients, leading to separate aggregation paths for each type of drift and ultimately enabling finer adaptation to changes in the data distribution.

The loss-based drift detection method described in Section III plays a crucial role in this approach. By monitoring the evolution of the clients' loss values over time, we can effectively distinguish between sudden drift and incremental drift. This two-pronged detection strategy allows us to apply the appropriate adaptation technique, assigning clients to either a sudden drift model or an incremental drift model based on the nature of the drift they experience.

Once all epochs are completed (Lines 5 to 8), a global model $G_{\text{opt}}$ is computed as a weighted average of the model without drift $G_n$ trained on non-drifted clients (Line 11) and the drifted models $G_s$ for sudden drift (Line 9) and $G_i$ for incremental drift (Line 10) with corresponding weights $\alpha_n$, $\alpha_s$, and $\alpha_i$ (Line 13). This allows for a dynamic and flexible adaptation to concept drift by leveraging multiple specialized models and combining them in a way that reflects the contribution of each drift type to the overall system. The final global model $G_{\text{opt}}$ can thus be expressed as $G_{\text{opt}} = \alpha_s G_s + \alpha_i G_i + \alpha_n G_n$.

The global model $G_{\text{opt}}$ is kept on the server rather than being sent back to the clients. However, when new clients join the FL process during the training phase, they are initialized using the global model $G_{\text{opt}}$, as it best represents the various drift scenarios encountered across all clients.

To handle incremental drift more effectively, we also adapt the Adam optimizer [10] to prioritize more current gradient descent values, especially in the presence of slowly changing data. Adam is an advanced optimization algorithm that leverages two key parameters known as moment estimates, $m_t$ and $v_t$. Initially set to zero, these moment estimates are updated based on the computed gradients of the loss function. An adaptive learning rate based on the two-moment estimates allows the algorithm to dynamically adjust to the learning space. More formally,

$$m_t = \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$$

and

$$v_t = \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2,$$

where $\beta_1$ and $\beta_2$ are hyperparameters that control the scaling of the past gradient $m_{t-1}$ and the current gradient $g_t$ when computing the moment estimates. Typically, $\beta_1$ and $\beta_2$ are set to 0.9 and 0.999, respectively, as these values have demonstrated effective performance in practice [10]. Nevertheless, these hyperparameters may be adjusted depending on the specific learning problem and the characteristics of the dataset.

Incremental drift, by its nature, involves a gradual shift in the data distribution and thus requires more dynamic model updates during the learning process. To this end, we modify the hyperparameters $\beta_1$ and $\beta_2$ of the Adam optimizer such that the current gradient descent steps are given more weight relative to past updates, provided the model remains stable in the presence of noise. This ensures that the model remains sensitive to recent changes in the data, which is crucial when dealing with incremental drift: reducing $\beta_1$ and $\beta_2$ gives higher priority to the most recent gradients.

## V. Evaluation

We now present the evaluation of our approach. First we describe our architecture in Section V-A. Then we evaluate our detection algorithms in Section V-B. Finally, we evaluate the performance of our adaptation approach in Section V-C.

Our evaluation is based on the MNIST dataset [3], which is widely used in machine learning and image classification as a benchmark for various models and systems. It contains 70,000 grayscale images of size $28 \times 28$ pixels showing handwritten digits. The dataset has a total of 10 classes, one for each digit from 0 to 9, and a train-test split ratio of 6 : 1.

### *A. Architecture*

We use in our experiments 60 clients and 40 epochs. As a baseline we use the FedAvg algorithm [16]. For training, we partition the training dataset into equally sized subsets based on the number of clients. Each subset is processed in local batches of 32 data objects at a time. When we split the data randomly into the subsets, we create variations in class distributions across the subsets. This introduces non-IID data among the clients, simulating realistic conditions where data available to different clients vary significantly.

The *Convolutional Neural Network (CNN)* used for our experiments consists of two 2D convolution layers with ReLU activation function and maximum pooling. Two fully connected layers are designed to learn various combination of features and their interactions. The model's output probabilities are computed using the softmax function. We use the Adam optimizer for the CNN model with beta values of $\beta_1 = 0.9$ and $\beta_2 = 0.999$ [10]. The learning rate is set to 0.001, which is optimal for our setting [21]. We measure client performance using model loss and accuracy. Loss is computed using the cross-entropy loss function during every epoch. The accuracy for each client is defined as the number of correct predictions over the total number of predictions.

We use PyTorch 2.4.0 to implement CNNs and FedAvg. Our experiments run in Kaggle's cloud environment, using two NVIDIA TESLA P100 GPUs to accelerate the training of our models. Our implementation uses SciPy 1.11.4 and NumPy 1.26.4. For the reference implementation of DDM and HDDM, we use the Scikit-Multiflow framework. Our implementation is available in our public GitHub repository.[1]

### *B. Evaluation of Detection Approach*

For the evaluation of the detection methods, we define a set of drifting clients which remains consistent throughout the experiments. Note that in our evaluation, we do not immediately activate the detection method. Instead, we allow the model to stabilize over a few epochs. Once this period is complete, we activate the drift detection method. Preliminary experiments show that it takes four epochs for the model parameters to stabilize in our experimental setting, and we therefore start the drift detection in epoch five. The ground truth for detection is determined by the set of predefined drifting clients who start their drift at epoch 10.

[1]https://github.com/ifte110/Concept_drift_detection_adaptation

TABLE I
GENERAL SETTING AND PARAMETERS FOR SUDDEN DRIFT DETECTION

| **Parameters** | **Values** | **Parameters** | **Values** |
|---|---|---|---|
| epochs | 40 | number of clients | 60 |
| drift introduced at epoch | 10 | local batch size | 32 |
| detection starts at epoch | 5 | training set size | 60,000 |
| learning rate | 0.001 | client dataset size | 1,000 |
| drastic change factor $\delta$ | 3 | swapped class labels | $(3,8)$, $(5,6)$ |
| sustained high loss threshold $\theta$ | 4 | | |

TABLE II
PERFORMANCE OF DRIFT DETECTION FOR SUDDEN DRIFT

| | **Algorithm 1** | **DDM+HDDM** |
|---|---|---|
| true positives | 290 | 116 (11+113) |
| true negatives | 1800 | 1800 |
| false positives | 0 | 0 |
| false negatives | 10 | 184 (289+187) |
| accuracy | 0.9952 | 0.9123 |
| precision | 1.0 | 1.0 |
| recall | 0.9666 | 0.3866 |
| F1 score | 0.9830 | 0.5576 |

*1) Sudden Drift:* To evaluate our loss-based detection method under sudden drift, we simulate drift in a federated learning system and use Algorithm 1 to detect the drift. The corresponding parameters are shown in Table I. The performance of the drift detection methods is given in Table II. We observe that the loss-based detection method defined in Algorithm 1 performs exceptionally well, achieving 100% precision, with only a slight drop in accuracy and recall due to the presence of false negatives. The reason for the false negatives is illustrated in the loss monitoring plot in Figure 3: We observe that the incorrect classification appears only in epoch 10 when the drift is introduced. This is to be expected since Algorithm 1 only flags a client as drifting when two conditions, drastic change and sustained change, are met. As a result, the drifting clients in Figure 3 cannot be detected immediately in epoch 10, simply due to the fact that no sustained drift could exist at this epoch. While this results in a minor loss in accuracy, this mechanism is nonetheless crucial for filtering out outliers.

As a baseline, we compare our detection approach to the prominent error-based *Drift Detection Method (DDM)* [6] and *Hoeffding's-inequality-based DDM (HDDM)* [5]. While being precise and avoiding false positives, DDM+HDDM suffer from poor recall and fail to detect many drift events. As observed from the data, the 91% accuracy is predominantly driven by HDDM, which accounts for 113 true positives out of 300, whereas DDM only identified 10 true positives at the point of drift introduction (see numbers in parentheses in Table II). The performance of DDM+HDDM is shown in Figure 4. This highlights a key limitation of DDM: it is effective at detecting abrupt, singular changes but struggles to track sustained losses post-drift. HDDM successfully detects drift in approximately 50% of the drifted clients, with detection occurring throughout all epochs. Compared to the loss-based drift detection method, DDM+HDDM achieve lower accuracy and significantly worse F1 scores and recall. The failure to consistently identify all drifted clients indicates that, despite the ability to detect some instances of sudden drift, the overall consistency and reliability of DDM+HDDM are inadequate for effectively monitoring dynamic environments.

*2) Incremental Drift:* To evaluate our detection algorithm for incremental drift, we apply the loss-based detection method for incremental drift defined in Algorithm 2 and evaluate its performance in three different incremental drift scenarios: We set the numbers of clients experiencing drift to 6, 16, and 30, respectively. As before, the total number of clients is set to 60. The ground truth for drift detection contains the clients drifting during the drifting period 19 to 30.

We use a short window length of $s = 15$ and a long window length of $d = 20$. These values have been selected based on multiple experiments to provide a good trade-off between sensitivity and stability, minimizing false detection while effectively capturing drifts. This observation is also supported by the findings from CDA-FedAvg [2], where long sliding windows result in improved detection and adaptation to incremental drift.

The performance of our algorithm is shown in Table III. It performs well for six drifting clients. Its performance degrades only when 30 out of 60 clients are drifting, in which case we

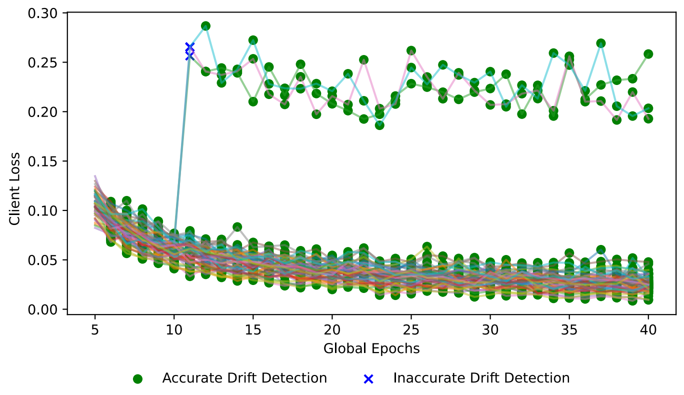

Fig. 3. Loss monitoring and detection accuracy using Algorithm 1

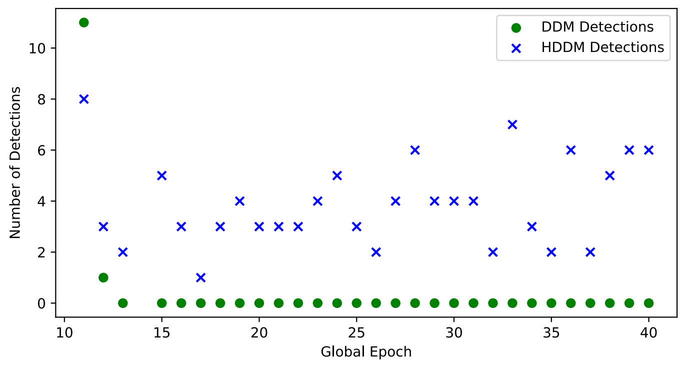

Fig. 4. Sudden drift detection using DDM and HDDM

TABLE III
PERFORMANCE OF DRIFT DETECTION FOR INCREMENTAL DRIFT

| | **Algorithm 2** | | | **DDM+HDDM** | |
|---|---|---|---|---|---|
| drifting clients | 6 | 16 | 30 | 16 | 30 |
| true positives | 59 | 145 | 270 | 73 | 62 |
| true negatives | 1378 | 1275 | 1097 | 1280 | 1140 |
| false positives | 2 | 5 | 43 | 0 | 0 |
| false negatives | 1 | 15 | 30 | 87 | 238 |
| accuracy | 0.9979 | 0.9861 | 0.9493 | 0.9395 | 0.8347 |
| precision | 0.9672 | 0.9666 | 0.8626 | 1.0 | 1.0 |
| recall | 0.9833 | 0.9063 | 0.9000 | 0.4563 | 0.2066 |
| F1 score | 0.9752 | 0.9355 | 0.8809 | 0.6266 | 0.3425 |

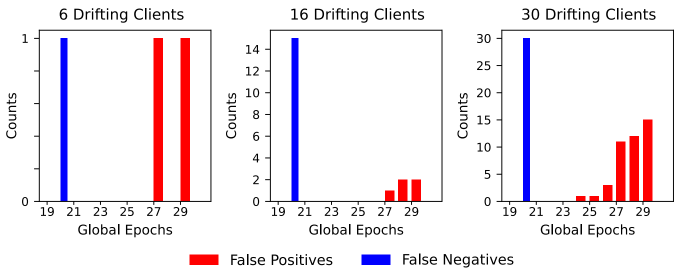

Fig. 7. Errors in Algorithm 2 over epochs for 6, 16, 30 drifting clients

see a rising number of false positives. To further investigate when these false positives occur, we need to closely examine the detection classification at each epoch. The bar plots in Figure 7 show the absolute numbers of false positives and false negatives over the epochs. Recall that drift is applied in the drifting period from epochs 19 to 30. We therefore observe that most errors occur at the time of the start of the drift and toward the end of the drift. The false negatives at the beginning of each epoch are due to the drifted clients' loss values not yet being severe enough for detection. The false positives towards the end of the drifting period are attributed to outliers: the non-IID nature of the client data and imbalanced class distributions cause performance fluctuations, leading to some clients being incorrectly detected as drifting. Additionally, the large number of false positives for 30 drifting clients stems from the fact that a large portion of the clients are drifting. This impacts the global model, as the non-drifted client parameters incorporate a significant portion of the drifted parameters. This leads to increased fluctuations in losses, which in turn result in a higher number of false positives.

The DDM+HDDM approach yields results comparable to those observed in sudden drift experiments. A significant concern is that both DDM and HDDM fail to detect all drifting clients until epoch 22: they only capture 45% and 21% of drifting clients for 16 and 30 selected clients, respectively. As a result of their inability to identify the majority of drifted clients, both methods exhibit poor recall and F1 scores.

*3) Mixed Drift Scenarios:* We introduce mixed drift in two ways. First, in the *separate setting*, we let two subsets of clients experience a different type of drift. Second, in the *simultaneous setting*, we let a subset of clients experience both sudden and incremental drift. We maintain the same experimental setup as before and focus on the detection phase of Algorithm 3 to assess the effectiveness of Algorithms 1 and 2 in accurately identifying different types of drift before the adaptation stage. In the separate setting, six clients experience incremental drift while five different clients experience sudden drift. In the simultaneous setting, three clients experience both drift types at the same time.

From our results in Figures 5 and 6, we see that our detection algorithms perform well in both settings of mixed drift. This is also reflected by our data in Table IV. For the separate setting, our algorithms miss the sudden drift only at epoch 10, and the second set of false detection results occurs for the clients under incremental drift at epoch 20. For the simultaneous setting, false positives occur at the start of the drift and around epoch 20. Additionally, one outlier is incorrectly reported as drift and one missed detection occurs. The observed behavior for both settings corresponds to the behavior of the individual detection algorithms as shown in the previous sections. Overall, the number of false detection results is low, demonstrating excellent performance across the experiments.

Despite the improved performance of HDDM under mixed drift in Table IV, our loss-based method continues to outperform both error-based detection methods DDM and HDDM. This is primarily due to the high number of false negatives associated with the latter methods, which ultimately leads to poor recall and F1 scores. Regarding the clients which experience

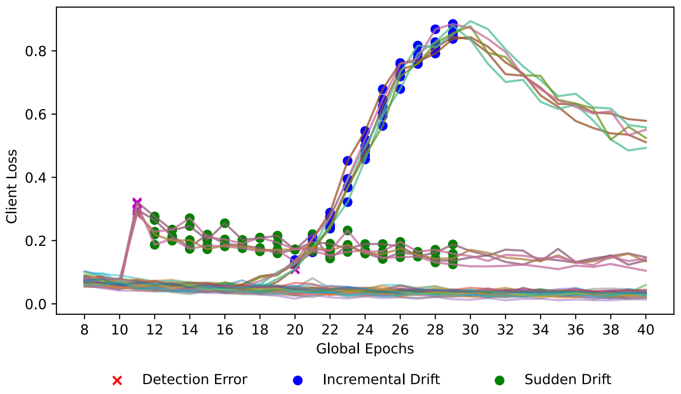

Fig. 5. Drift detection using Algorithms 1 and 2, separate setting

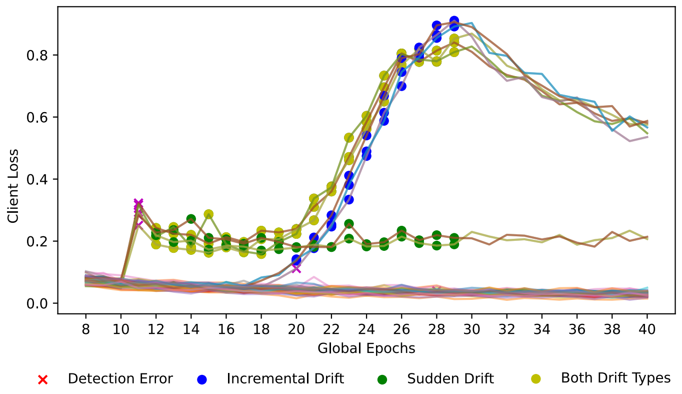

Fig. 6. Drift detection using Algorithms 1 and 2, simultaneous setting

TABLE IV
PERFORMANCE OF DRIFT DETECTION FOR MIXED DRIFT

| | Algorithms 1 and 2 | | DDM+HDDM | |
|---|---|---|---|---|
| setting | separate | simultaneous | separate | simultaneous |
| true positives | 147 | 119 | 55 (24+43) | 52 (17+47) |
| true negatives | 1285 | 1314 | 1285 | 1315 |
| false positives | 0 | 1 | 0 | 0 |
| false negatives | 8 | 6 | 100 (131+112) | 73 (108+78) |
| accuracy | 0.9944 | 0.9951 | 0.9305 | 0.9493 |
| precision | 1.0 | 0.9916 | 1.0 | 1.0 |
| recall | 0.9438 | 0.9520 | 0.3548 | 0.4160 |
| F1 score | 0.9735 | 0.9714 | 0.5238 | 0.5875 |

both types of drift, we make the following observation. Due to the delay in incremental drift impacting model performance, the drifting clients initially show high loss due to sudden drift. The effects of incremental drift only become apparent around epoch 19. Therefore, our algorithm first detects these clients as experiencing sudden drift, followed by incremental drift only later. Our detection method classifies these clients as having both types of drift and adjusts their models accordingly.

## C. Evaluation of Adaptation Approach

We now evaluate our adaptation approach. In our plots, we always compare the behavior of the system without adaptation in place to the behavior when using Algorithm 3.

First, we consider sudden drift. We let the system run for ten rounds before we switch two pairs of class labels for six out of 60 clients. From Figure 8 we observe that without our adaptation method, the model accuracy of drifted clients drops to around 70%. Although the model regains some accuracy between epochs 10 and 40, the drifted clients never converge with the non-drifted clients. When we apply the adaptation method from Algorithm 3, the model converges rapidly, as the drifted clients are assigned to a new drifted model.

Next, we evaluate the impact of incremental drift. As shown in Figure 9, incremental drift leads to a significant drop in performance. The accuracy of drifted clients falls below 50%, and although we observe some recovery after epoch 30, the accuracy remains below 70%. With our adaptation approach, we are able to mitigate this drop: Algorithm 3 keeps the minimal accuracy at 80%, and ultimately, the model converges.

Finally, our results for mixed drift are shown in Figures 10 and 11. Our adaptation algorithm not only retains the accuracy, but also allows drifted clients to quickly converge. For the mixed drift experiment, we assign beta values of 0.6 and 0.7 to incrementally drifting clients as discussed in Section IV.

When our algorithm computes the global model $G_{\text{opt}}$ (see Algorithm 3), the scaling factors must be chosen carefully to ensure that the global model remains stable on test datasets that exhibit similar drift patterns. Extensive simulations show optimal results when setting $(\alpha_s, \alpha_i, \alpha_n)$ to $(0.35, 0.00, 0.75)$ for sudden drift and to $(0.0, 0.35, 0.75)$ for incremental drift. Table V shows our results for incremental drift. We compare the performance of the FedAvg model against the global model. For a small fraction of drifting data items and small numbers of drifting clients, both models show a similar performance for both types of drift. As soon as drift is introduced to the entire test data, the global model shows a significant improvement over the FedAvg model. In the final set of experiments shown in Table V, the fraction of drifted test data is kept constant at 1.0, while the number of drifting clients increases up to 70%. This high proportion of drifted clients contributes to the significant performance degradation observed in the FedAvg model, while the optimized model maintains its robustness.

TABLE V
COMPARISON OF ACCURACIES UNDER INCREMENTAL DRIFT

| Drift | Drift Fraction | | Accuracy | |
|---|---|---|---|---|
| | Data | Clients | FedAvg | Algorithm 3 |
| Sudden | 0.2 | 3 | 0.9482 | 0.9420 |
| Sudden | 0.4 | 3 | 0.9120 | 0.9064 |
| Sudden | 0.6 | 3 | 0.8703 | 0.8667 |
| Sudden | 0.8 | 3 | 0.8289 | 0.9228 |
| Sudden | 1.0 | 3 | 0.7921 | 0.9785 |
| Incremental | 0.2 | 3 | 0.9594 | 0.9602 |
| Incremental | 0.4 | 3 | 0.9350 | 0.9387 |
| Incremental | 0.6 | 3 | 0.9170 | 0.9279 |
| Incremental | 0.8 | 3 | 0.8858 | 0.9163 |
| Incremental | 1.0 | 3 | 0.8544 | 0.9066 |
| Incremental | 1.0 | 6 | 0.8451 | 0.9037 |
| Incremental | 1.0 | 16 | 0.8510 | 0.9056 |
| Incremental | 1.0 | 30 | 0.7644 | 0.9109 |
| Incremental | 1.0 | 42 | 0.6279 | 0.9143 |

## VI. CONCLUSIONS

We investigate the impact of concept drift on model accuracy in FL, with a particular focus on local drift, where only a subset of clients experience changes in their data distribution. This scenario poses unique challenges, as unaffected clients continue to train on stationary data. We examine both sudden and incremental drift across multiple experimental setups.

Our main contribution is an active approach that detects drift based on client loss. The detection mechanism, based solely on training loss, identifies distinct drift patterns with low false positives and false negatives, outperforming DDM and HDDM. Adaptation is achieved through a multi-model strategy: drifting clients are assigned a separate model. In addition, we use optimizers that favor recent data of drifting clients, significantly improving model accuracy.

Our detection and adaptation strategies are computationally lightweight and impose virtually no overhead on client devices. We leverage loss values which are already computed during training, and avoid additional communication. As a result, our solution can be readily deployed in a resource-constrained environment. Our experiments show that our method consistently outperforms FedAvg across a wide range of drift scenarios.

## ACKNOWLEDGMENT

The financial support by the Austrian Federal Ministry of Economy, Energy and Tourism, the National Foundation for Research, Technology and Development and the Christian Doppler Research Association is gratefully acknowledged.

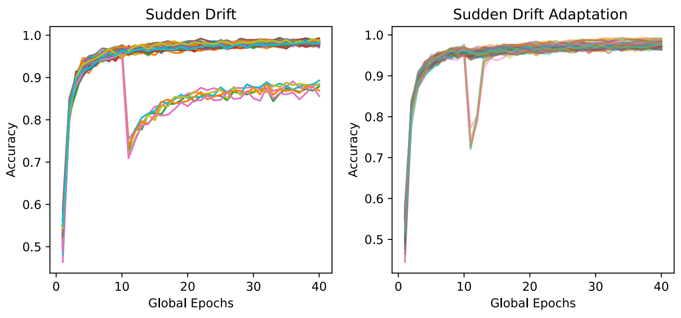

Fig. 8. Client accuracy, sudden drift

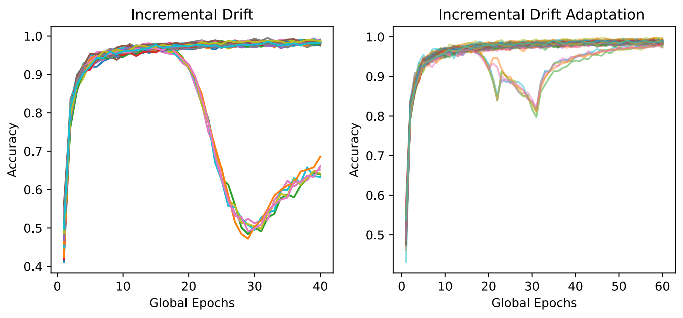

Fig. 9. Client accuracy, incremental drift

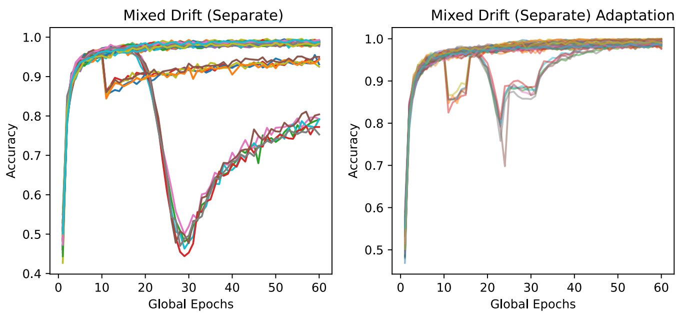

Fig. 10. Client accuracy, mixed drift,separate setting

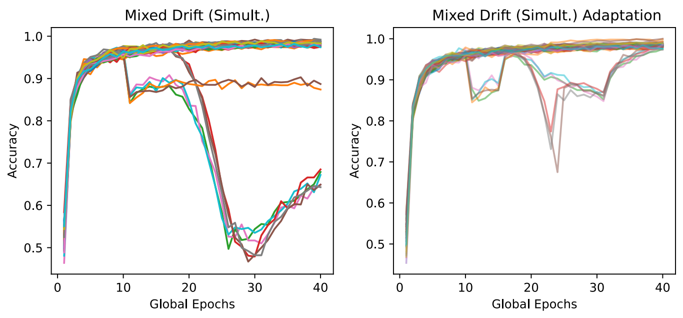

Fig. 11. Client accuracy, mixed drift,simultaneous setting

## References

[1] G. Canonaco, A. Bergamasco, A. Mongelluzzo, and M. Roveri, “Adaptive federated learning in presence of concept drift,” in *International Joint Conference on Neural Networks*, 2021, pp. 1–7.

[2] F. E. Casado, D. Lema, M. F. Criado, R. Iglesias, C. V. Regueiro, and S. Barro, “Concept drift detection and adaptation for federated and continual learning,” *Multim. Tools Appl.*, vol. 81, no. 3, pp. 3397–3419, 2022.

[3] L. Deng, “The MNIST database of handwritten digit images for machine learning research [best of the web],” *IEEE Signal Process. Mag.*, vol. 29, no. 6, pp. 141–142, 2012.

[4] J. C. Duchi, E. Hazan, and Y. Singer, “Adaptive subgradient methods for online learning and stochastic optimization,” *J. Mach. Learn. Res.*, vol. 12, pp. 2121–2159, 2011.

[5] I. Frías-Blanco, J. del Campo-Ávila, G. Ramos-Jiménez, R. Morales-Bueno, A. Ortiz-Díaz, and Y. Caballero-Mota, “Online and non-parametric drift detection methods based on hoeffding’s bounds,” *IEEE Trans. Knowl. Data Eng.*, vol. 27, no. 3, pp. 810–823, 2015.

[6] J. Gama, P. Medas, G. Castillo, and P. P. Rodrigues, “Learning with drift detection,” in *17th Brazilian Symposium on Artificial Intelligence*, 2004, pp. 286–295.

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

[8] E. Jothimurugesan, K. Hsieh, J. Wang, G. Joshi, and P. B. Gibbons, “Federated learning under distributed concept drift,” in *International Conference on Artificial Intelligence and Statistics*, 2023, pp. 5834–5853.

[9] M. Kang, S. Kim, K. H. Jin, E. Adeli, K. M. Pohl, and S. Park, “Fednn: Federated learning on concept drift data using weight and adaptive group normalizations,” *Pattern Recognit.*, vol. 149, p. 110 230, 2024.

[10] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” in *3rd International Conference on Learning Representations*, 2015.

[11] A. Kumar, N. Hemadasa, D. Kaaser, and S. Schulte, “Fluid: Federated learning with unlearning and instant drift-recovery,” in *3rd IEEE International Conference on Federated Learning Technologies and Applications*, 2025.

[12] J. Lu, A. Liu, F. Dong, F. Gu, J. Gama, and G. Zhang, “Learning under concept drift: A review,” *IEEE Trans. Knowl. Data Eng.*, vol. 31, no. 12, pp. 2346–2363, 2019.

[13] N. Lu, G. Zhang, and J. Lu, “Concept drift detection via competence models,” *Artif. Intell.*, vol. 209, pp. 11–28, 2014.

[14] D. M. Manias, I. Shaer, L. Yang, and A. Shami, “Concept drift detection in federated networked systems,” in *IEEE Global Communications Conference*, 2021, pp. 1–6.

[15] I. Mavromatis, S. D. Feo, and A. Khan. “FLAME: adaptive and reactive concept drift mitigation for federated learning deployments.” arXiv: 2410.01386. (2024).

[16] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, “Communication-efficient learning of deep networks from decentralized data,” in *20th International Conference on Artificial Intelligence and Statistics*, 2017, pp. 1273–1282.

[17] F. Saile, J. Thomas, D. Kaaser, and S. Schulte, “Client-side adaptation to concept drift in federated learning,” in *2nd International Conference on Federated Learning Technologies and Applications*, 2024, pp. 71–78.

[18] M. Stallmann, A. Wilbik, and G. Weiss, “Towards unsupervised sudden data drift detection in federated learning with fuzzy clustering,” in *IEEE International Conference on Fuzzy Systems*, 2024, pp. 1–8.

[19] J. Thomas, F. Saile, M. Fischer, D. Kaaser, and S. Schulte, “Adaption via selection: On client selection to counter concept drift in federated learning,” in *11th European Conference on Service-Oriented and Cloud Computing*, 2025, pp. 3–17.

[20] X. Wang, Y. Han, C. Wang, Q. Zhao, X. Chen, and M. Chen, “In-edge AI: intelligentizing mobile edge computing, caching and communication by federated learning,” *IEEE Netw.*, vol. 33, no. 5, pp. 156–165, 2019.

[21] Y. Wu, L. Liu, J. Bae, K. H. Chow, A. Iyengar, C. Pu, W. Wei, L. Yu, and Q. Zhang, “Demystifying learning rate policies for high accuracy training of deep neural networks,” in *IEEE International Conference on Big Data*, 2019, pp. 1971–1980.

[22] C. Zhang, Y. Xie, H. Bai, B. Yu, W. Li, and Y. Gao, “A survey on federated learning,” *Knowl. Based Syst.*, vol. 216, p. 106 775, 2021.