Federated Learning under Distributed Concept Drift

Ellango Jothimurugesan
Carnegie Mellon University
ejothimu@cs.cmu.edu

Kevin Hsieh
Microsoft Research
kevin.hsieh@microsoft.com

Jianyu Wang
Carnegie Mellon University
jianyuw1@andrew.cmu.edu

Gauri Joshi
Carnegie Mellon University
gaurij@andrew.cmu.edu

Phillip B. Gibbons
Carnegie Mellon University
gibbons@cs.cmu.edu

Abstract

Federated Learning (FL) under distributed concept drift is a largely unexplored area. Although concept drift is itself a well-studied phenomenon, it poses particular challenges for FL, because drifts arise staggered in time and space (across clients). Our work is the first to explicitly study data heterogeneity in both dimensions. We first demonstrate that prior solutions to drift adaptation, with their single global model, are ill-suited to staggered drifts, necessitating multi-model solutions. We identify the problem of drift adaptation as a time-varying clustering problem, and we propose two new clustering algorithms for reacting to drifts based on local drift detection and hierarchical clustering. Empirical evaluation shows that our solutions achieve significantly higher accuracy than existing baselines, and are comparable to an idealized algorithm with oracle knowledge of the ground-truth clustering of clients to concepts at each time step.

1 Introduction

Federated learning (FL) [22, 29] is a popular machine learning (ML) paradigm that enables collaborative training without sharing raw training data. FL is crucial in the era of pervasive computing, where massive IoT and mobile phones continuously generate relevant ML data that cannot be easily shared due to privacy and communication constraints. FL also enables different organizations such as hospitals [33] and retail stores [42] to jointly obtain valuable insights while preserving data privacy. FL has become an important technology in the real world with massive deployments (500+ million installations on Android devices) as well as a growing market with many solution providers [28].

Existing FL solutions generally assume the training data comes from a *stable* underlying distribution, and the training data in the past is sufficiently similar to the test data in the future. Unfortunately, this assumption is often violated in the real world, where the underlying data distribution is non-stationary and constantly evolves. For instance, user sentiment and preference change drastically due to external environments such as the pandemic and macroeconomics [13, 21]. Data collected by cameras are also subject to various data changes such as unexpected weather and novel objects, which can lead to significant ML model performance losses [2, 37].

This *concept drift* problem [41] has been studied extensively in a centralized learning environment [12, 38]. These centralized solutions, however, cannot address the fundamental challenges of concept drifts in FL where data is heterogeneous over *time* and across different *clients*. When different clients experience the data drift *at different times*, no single global model can perform well for all clients. Similarly, when *multiple concepts exist simultaneously*, no centralized training decision works well for all clients. Several recent works have recognized the problem of FL under concept drift and proposed solutions that adapt learning rates or add regularization terms [8, 9, 16, 26]. Although these solutions perform better than drift-oblivious algorithms such as FedAvg [29], the solutions still

Preprint. Under review.

use a single global model for all clients, and hence fail to address the aforementioned fundamental challenges of heterogeneity over time and across clients.

In this work, we present the first FL solution that employs multiple models to address FL under distributed concept drift. Our solution aims to create one model for each new concept so that all clients under the same concept can train that model collaboratively, similar to what is done for *personalized* or *clustered* FL [10, 14, 15, 27, 34]. We introduce two new algorithms for model creation and client clustering so that our solution addresses all the challenges of distributed concept drift. Our first algorithm, FedDrift-Eager, is a specialized algorithm that creates models based on drift detection. FedDrift-Eager is effective if new concepts are introduced one at a time. Our second algorithm, FedDrift, is a general algorithm that leverages hierarchical clustering to adaptively determine the appropriate number of models. FedDrift isolates drifted clients and conservatively merges clients via hierarchical clustering, so that FedDrift can effectively handle general cases where an unknown number of new concepts emerge simultaneously.

We empirically evaluate our solution using four popular concept drift datasets, and we compare our solution against state-of-the-art centralized concept drift solutions (AUE [6] and DriftSurf [38]) and a recent FL solution that adapts to concept drifts (Adaptive-FedAvg [7]). Our results show that (i) FedDrift-Eager and FedDrift consistently achieve much higher and more stable model accuracy than existing baselines (average accuracy 93% vs. 88% for the best baseline, across six dataset/drift combinations); (ii) FedDrift performs much better than FedDrift-Eager when multiple new concepts are introduced at the same time; and (iii) our solution achieves a similar model accuracy as Oracle (94% accuracy), an idealized algorithm that knows the timing and distribution of concept drifts.

2 Background and Motivation

2.1 Problem Setup

We consider a FL setting with P clients, assumed to be stateful and participating at each round, and a central server that coordinates training across the clients. Training data are decentralized and arriving over time. The data at each client $c = 1, 2, \dots, P$ and each time $t = 1, 2, \dots$ are sampled from a distribution (concept) $\mathcal{P}_c^{(t)}(x, y)$. We consider that data may be non-IID in two dimensions, varying across clients and across time. We say that there is a concept drift at time t and at client c if $\mathcal{P}_c^{(t)} \neq \mathcal{P}_c^{(t-1)}$ (the standard definition of drift with respect to a single node [12]).

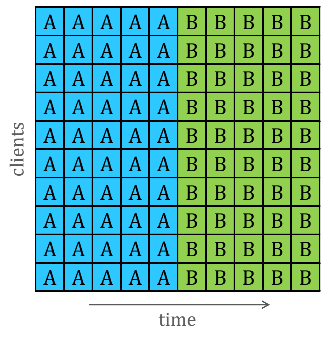
One option is to learn a single global model h (which is a function of time but is notationally suppressed) that is used for inference at all clients. In this case, the objective is to minimize over all time t, $\sum_{c=1}^{P} \mathbb{E}_{(x,y)\sim\mathcal{P}_c^{(t)}}[\ell(h(x), y)]$, where ℓ is the loss function. However, the optimal single model may not be well-suited in the presence of concept drifts. By decomposing the joint distribution $\mathcal{P}(x, y) = \mathcal{P}(x)\mathcal{P}(y|x)$, we distinguish between drifts where only $\mathcal{P}(x)$ changes versus drifts where the feature-to-label mapping $\mathcal{P}(y|x)$ changes. Under the former case (which goes by the names virtual drift [39], covariate drift [36], and feature-distribution skew [19]) the optimal single model can perform well (although achieving fast convergence still requires a specialized strategy; e.g., FedProx [24]). But under the latter case where the feature-to-label mapping changes (real drift or concept drift [39]), lower loss can often be obtained by using specialized models for different concepts.

The multi-model option is to learn a set of global models $\{h_m\}$, and a time-varying clustering of clients. For notation, we denote the cluster identities by one-hot vectors $w_c^{(t)}$, where $w_{c,m}^{(t)} = 1$ when the client c at time t uses model h_m for inference; we denote $h_{w_c^{(t)}}$ to represent the unique model h_m where $w_{c,m}^{(t)} = 1$. The objective is to minimize over all time t,

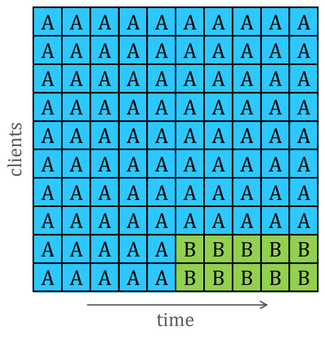
$$\sum_{c=1}^{P} \mathop{\mathbb{E}}_{(x,y)\sim\mathcal{P}_c^{(t)}} [\ell(h_{w_c^{(t)}}(x), y)]. \tag{1}$$

2.2 Motivation

The prior work on drift adaptation in FL has considered only restrictive settings such as (i) drifts occurring simultaneously in time (e.g., Figure 1a), where a centralized approach works well [7], or (ii) drifts with only minor deviations from a majority concept (e.g., Figure 1b), where updates from



(a) Simultaneous timing



(b) One majority concept

Figure 1: Simplistic drifts studied in prior work.

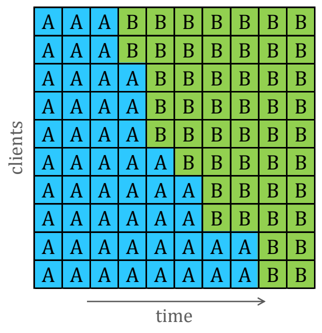


Figure 2: Distributed drift pattern (2 concepts).

drifting clients are suppressed and the minority concept goes unlearned [9, 26]. Our work is the first to acknowledge and explicitly study the more general settings arising in distributed drifts, with heterogeneous data across clients and over time.

Consider the distributed drift pattern depicted in Figure 2. This is representative of an emerging trend (e.g., a breaking news event) that effects different clients at different times (e.g., due to their lag in learning of the news). Even for this simple case of a single staggered transition between two concepts, prior work results in significant accuracy loss. In particular, their use of a single global model (and at best a single global drift detection test) results in poor accuracy during the transition period (time steps 4–8, see Figure 4(left)).

We also consider more challenging cases, as depicted in Figure 3, where multiple concepts emerge at the same time and concept drifts may be recurring (a.k.a. periodic).

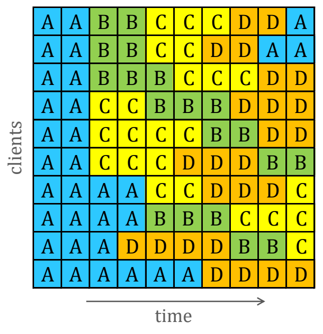


Figure 3: Distributed drift pattern (4-concepts).

2.3 Related Work

Concept drift has been studied extensively in the centralized setting for decades. We refer the reader to the surveys by Gama et al. [12] and Lu et al. [25]. As previously discussed, directly applying these centralized algorithms in FL is not well-suited for distributed concept drifts with heterogeneous data across time and clients. We demonstrate this in our experimental evaluation, where we compare against state-of-the-art algorithms such as DriftSurf [38] and AUE [6].

Drift in FL, on the other hand, has so far seen only preliminary study. One line of work considers the setting where there is one concept in the system to be learned (either like the example in Figure 1b when a minority of clients drift, or when clients observe the main concept under random noise), and seek to speed up the convergence of a model for that one concept by suppressing clients with heterogeneous data via regularization [9, 16] or drift detection [26]. When it comes to adapting to a new concept over time, we are only aware of two works, and both only consider drifts with uniform timing like the example in Figure 1a. First, Casado et al. [8] consider only the *virtual drift* setting (where the labeling $\mathcal{P}(y|x)$ is fixed and only $\mathcal{P}(x)$ changes) and uses drift detection to partition data from distinct concepts, in order to train a single model accurately in the course of revisiting each partition (i.e., rehearsal). Second, Canonaco et al. [7] propose Adaptive-FedAvg, in which the server tunes the learning rate used by all clients as a function of the variability across updates, with the goal of reacting fast when drift occurs while also achieving stable performance in the absence of drift. In our experimental evaluation, we compare against Adaptive-FedAvg.

Our solution to drift in FL relies on learning multiple models, which has been studied in prior work on *personalized FL* and *clustered FL*. Clients with similar data can be grouped into clusters, where each cluster of clients is associated with a global model that they collaboratively train [5, 10, 14, 15, 27, 34]. As we extend the problem of data heterogeneity in FL with an additional dimension of time, we train multiple models with the algorithm in §3, which is heavily inspired by the prior clustering algorithms IFCA [15] and HypCluster[27]. This serves as the starting point of our solution, where our main contribution is the creation of new clusters as new concepts arrive over time. Finally, our solution in §4 to handle an unknown number of concepts relies on hierarchical clustering, which has been studied in FL (in the static case) previously by Briggs et al. [5]. In the prior work, the clustering is based on clients' local updates, and it is unclear how to set the distance threshold at which to stop merging. In contrast, an advantage of our approach is that the stopping criterion is identical to the drift detection threshold, which has an intuitive interpretation of performance loss.

3 Multi-Model Training in FL

As discussed above, distributed concept drift often means that multiple concepts are present simultaneously, necessitating the need for multi-model training. In this section, we present an algorithm for multi-model training in FL over time. Then in §4, we will show how to apply the algorithm to react to drifts to new concepts.

Our approach (depicted in Algorithms 1 and 2) trains a set of global models, each trained by a cluster of clients. We define a time step as the granularity at which new data may arrive at a client. A time step may consist of multiple communication rounds. The set of data arriving at client c and time t is denoted by $S_c^{(t)}$. The global models being trained are denoted by h_m for $m \in [M]$, where M is the total number of models at a given time. Each model is trained by a cluster of clients, where the clustering may vary over time as concept drifts occur. The cluster identity of client c at time t is denoted by the one-hot vector $w_c^{(t)}$, where $w_{c,m}^{(t)} = 1$ when assigned to the cluster associated with model h_m and 0 otherwise. The cluster identities $w_{c,m}^{(t)}$ indicate whether the data $S_c^{(t)}$ that arrived at client c at time t are sampled when computing a local update to the global model h_m. Further, the cluster identity of a client at a given time indicates which model is used for inference.

Within each time, the training of the global models in Algorithm 1 is equivalent to Federated Averaging [29], since the aggregation weight of each client within each cluster is fixed at time τ. So the convergence of Algorithm 1 can be guaranteed by directly using previous analyses for Federated Averaging, such as [24, 40]. The difference here is that the objective function that clients are minimizing at time τ is replaced by the following:

$$\tilde{F}_m^{(\tau)}(h_m) = \sum_{c=1}^{P} \tilde{w}_{c,m}^{\tau} F_c^{(\tau)}(h_m) \tag{2}$$

where $F_c^{(\tau)}$ denotes the local objective function on client c, and the normalized weight is defined as $\tilde{w}_{c,m}^{\tau} = \sum_{t=1}^{\tau} w_{c,m}^{(t)} N_c^{(t)} / \sum_{c=1}^{P} \sum_{t=1}^{\tau} w_{c,m}^{(t)} N_c^{(t)}$.

In the ideal case where each cluster maps to one concept in the system, each h_m is specialized for each concept that is sampled from a unique data distribution ($\mathcal{P}(x, y)$), and these h_m form a strong solution to our overall objective in §2.1. This ideal solution is the Oracle algorithm in our evaluation in §5, and we empirically demonstrate that our proposed solutions achieve comparable accuracy.

Algorithm 1 Multi-model training at time τ

```
Input: Cluster identities $w_{c,m}^{(t)}$
  for each round $i = 1, 2, \ldots, R$ do
    for each client $c = 1, 2, \ldots, P$
    and each model $m = 1, 2, \ldots, M$ in parallel do
      $h_{c,m} \leftarrow \textsc{LocalUpdate}(c, h_m, \{w_{c,m}^{(t)}\}_{t=1}^{\tau})$
    for each model $m = 1, 2, \ldots, M$ do
      $h_m \leftarrow \frac{\sum_{c=1}^{P} h_{c,m} \sum_{t=1}^{\tau} w_{c,m}^{(t)} N_c^{(t)}}{\sum_{c=1}^{P} \sum_{t=1}^{\tau} w_{c,m}^{(t)} N_c^{(t)}}$

  $\textsc{LocalUpdate}(c, h_m, \{w_{c,m}^{(t)}\}_{t=1}^{\tau})$:
  for each local step $j = 1, 2, \ldots, K$ do
    $b \leftarrow$ random minibatch of size $B$ from
        $\bigcup_{t: w_{c,m}^{(t)}=1} S_c^{(t)}$
    $h_m \leftarrow h_m - \eta \nabla \ell(h_m; b)$
  return $h_m$
```

τ	current time (prior time indexed by t)
P	# clients (indexed by c)
M	# global models (indexed by m)
R	# communication rounds (indexed by i)
K	# local steps per model per round (by j)
$S_c^{(t)}$	new data arriving at client c at time t
$N_c^{(t)}$	$= \lvert S_c^{(t)} \rvert$
B	minibatch size
η	step size
h_m	global model m
$h_{c,m}$	local update of h_m by client c
$w_{c,m}^{(t)}$	is $S_c^{(t)}$ used to update h_m?

Algorithm 2 Clustering to the lowest loss

```
  $\ell_{c,m}^{(\tau)} \leftarrow$ loss of $h_m$ on client data $S_c^{(\tau)}$
  $w_{c,m}^{(\tau)} \leftarrow \mathbf{1}\{m = \arg\min_{m'} \ell_{c,m'}^{(\tau)}\}$
  Run Algorithm 1
```

Note that, as stated, each client c in Algorithm 1 retains its complete history of both the cluster indicators $w_{c,m}^{(t)}$ and the local data arrivals $S_c^{(t)}$. To reduce this overhead, each client could instead maintain just a sliding window of the most recent time steps, as long as the window suffices for the minibatch sampling in LocalUpdate.

Thus, we have separated the problem of concept drift in FL into two components: (i) determining the time-varying clustering of clients in response to concept drifts, which is then used as input for (ii) multi-model training in Algorithm 1. Suppose, hypothetically, that there is a global model already initialized for each concept up to some moderate accuracy. In this restrictive setting, Algorithm 2 can be used to determine the cluster identities for each new time step. Each client tests the global models from the previous time step over its newly arrived data and chooses to identify with the model with the best loss (breaking ties randomly).[1] The setting considered encompasses time steps involving drifts that occur between concepts known to the system; e.g., the later stages of a staggered drift from concept A to concept B after some clients have already observed concept B (Figure 2). However, Algorithm 2 does not have any mechanism to spawn new clusters or determine the number of clusters. In §4, we will show how to determine the input for Algorithm 1 with clustering algorithms that can spawn clusters over time to react to drifts to *new* concepts.

4 Clustering Algorithms

Under concept drift in FL, data is heterogeneous both over time and across clients. The concept at each time and client is the ground-truth clustering that we seek to learn. Ideally, the models trained by each cluster correspond 1-to-1 to the concepts present in the system. We want to avoid spawning multiple clusters that correspond to a single concept, in which case each model is trained over only a subset of the relevant data, not taking full advantage of collaborative training. We also want to avoid merging clients corresponding to multiple concepts into a single cluster (model poisoning).

We present two clustering algorithms for adapting to concept drift. First, in §4.1 we handle the case where only one new concept emerges at a time, which includes the example drift pattern in Figure 2, by incorporating a straightforward drift detection algorithm. Second, in §4.2 we give a general algorithm that handles the general case where multiple new concepts may emerge simultaneously, which includes the example drift pattern in Figure 3, by incorporating a bottom-up technique that *isolates clients* that detect drift and *iteratively merges* clusters corresponding to the same concept.

In the remainder of this section, we assume that the first time step starts with one concept and one model, and that our clustering is run for each time step $\tau > 1$ as new data arrives.

4.1 Special Case: One New Concept at a Time

When a new concept emerges, the clients that observe the drift should be split off to a new cluster to start training a new model. Drift detection has been well-studied in the centralized, non-FL, setting [1, 3, 11, 17, 30, 32, 38]. As we noted in §2.2, for staggered drifts in FL, trying to apply a drift detection test *globally* at the server over the aggregate error results in poor performance during the transition period. Instead, we propose applying drift detection *locally* at each client.

Drift detection tests commonly work by monitoring the prediction error of a model and signaling a drift whenever the error increases by a set significance threshold, which represents a trade-off between false positives and negatives in detection. The literature contains an abundance of tests refined over the years of increasing sophistication to attain better accuracy and lower detection delay. The particular test is not a focus of this paper, and for simplicity we consider a test of the following form. A drift is signaled at client c at time τ with respect to model h_m if the loss of the model over the newly arrived data, denoted as $\ell_{c,m}^{(\tau)}$, degrades by a threshold δ relative to the loss measured at the previous time:

$$\ell_{c,m}^{(\tau)} > \ell_{c,m}^{(\tau-1)} + \delta. \tag{3}$$

This test checks for any drift that incurs performance degradation with respect to a given model. However, the desired condition for creating a *new model* should check only for concept drifts that

[1] If there are no new data at a particular client, then we say its cluster identity is carried over from the previous time step so the model used for inference is well-defined.

correspond to a concept *previously unobserved* and *ill-suited* for all existing models. For other drifts, such as the later stage of the staggered drift from concept A to concept B in Figure 2 (after concept B has already been detected and an appropriate model created), a client should join an existing cluster (in this case, the cluster for B). Hence, we extend the test for model creation to compare against the *best performing* model:

$$\min_m \ell_{c,m}^{(\tau)} > \min_m \ell_{c,m}^{(\tau-1)} + \delta. \tag{4}$$

Algorithm 3 FedDrift-Eager at time τ

```
$\ell_{c,m}^{(\tau)} \leftarrow$ loss of model $h_m$ on client data $S_c^{(\tau)}$
$w_{c,m}^{(\tau)} \leftarrow \mathbf{1}\{m = \arg\min_{m'} \ell_{c,m'}^{(\tau)}\}$
if $\min_m \ell_{c,m}^{(\tau)} > \min_m \ell_{c,m}^{(\tau-1)} + \delta$ at any client $c$ then
    $M \leftarrow M + 1$
    Initialize a new global model $h_M$
    $w_{c,*}^{(\tau)} \leftarrow \mathbf{0}$; $w_{c,M}^{(\tau)} \leftarrow 1$
Run Algorithm 1
```

We note that detection tests that compare across multiple models have been previously studied in the centralized learning setting in the context of adapting to recurring drifts; a system that learns one model at a time may still maintain previously learned models for the case that a drift corresponds to a previously observed concept (e.g., the concept sequence A–B–A) [20]. The clustering in Algorithm 3 (FedDrift-Eager) applies this multi-model drift detection test at each client, and creates a new cluster for all the clients that detect a new concept; otherwise, each client identifies with the cluster with the best-performing model. This algorithm relies on the assumption that only one new concept occurs at a time by assigning the drifted clients to a single cluster. Despite this limitation, Algorithm 3 still merits interest as it experimentally performs well on the non-trivial case of the staggered drift in Figure 2 that has not been addressed by the prior work, as shown in §5. However, for the drift in Figure 3 in which concepts B and C emerge simultaneously at different clients, this algorithm creates only one cluster and sub-optimally tries to train a single model for both new concepts. Next, we extend this algorithm to address the general case where an unknown number of new concepts can occur at a time.

4.2 General Case

When drifts to new concepts are detected at multiple clients, in general we do not know whether the drifts all correspond to one concept or multiple concepts (or even zero concepts in the event of false positives in detection). We designed Algorithm 4 (FedDrift) for clustering in the face of this uncertainty. For each client that detects drift to a new concept, Algorithm 4 conservatively isolates the clients to individual clusters, and then merges clusters corresponding to the same concept slowly and safely over time by iteratively applying classical hierarchical agglomerative clustering [35].

The generic hierarchical clustering procedure is specified by a distance function over the set of elements to be clustered and a stopping criterion, and at each step until the stopping criterion is met, merges the two closest clusters, where the distance between clusters of multiple elements is commonly defined to be the maximum distance between their constituents (known as a max-linkage clustering). In Algorithm 4, the Merge subroutine combines two clusters i and j by averaging their models with weight proportional to the size of each model's training dataset (over all clients) and unifying the cluster identities.

To specify a distance function for hierarchical clustering, Algorithm 4 first aggregates at the server the loss estimates L_{ij} of the model h_i evaluated over a subsample of the data associated with the cluster for model h_j. [2] Then the distances between each cluster are initialized as $D(i,j) \leftarrow \max(L_{ij} - L_{ii}, L_{ji} - L_{jj}, 0)$.[3] The first term $L_{ij} - L_{ii}$ measures the loss degradation of model h_i when evaluated over the data associated with h_j, relative to the loss over its own data. We informally interpret this difference as the magnitude of drift between the concept associated with h_i to the concept associated with h_j, analogous to the drift detection condition in Eq (3) (although not identical due to the bias of L_{ii} measuring a model's accuracy over its own training data). The term $D(i,j)$ is

[2]More precisely, at client c, the data clustered to h_j are sampled proportionate to the size of the local dataset relative to the global dataset for h_j, $\sum_t w_{c,j}^{(t)} N_c^{(t)} / \sum_{c'} \sum_t w_{c',j}^{(t)} N_{c'}^{(t)}$.

[3]We note that $D(i,j)$ is not necessarily a true distance function as there is no guarantee that it satisfies the triangle inequality.

defined to be symmetric by also accounting for the magnitude of the drift $L_{ji} - L_{jj}$ in the reverse direction from concept j to concept i.

In addition to defining the cluster distances $D(i, j)$, employing hierarchical clustering also requires setting a stopping criterion. Typically, that corresponds to specifying either the desired number of clusters (which in our case is unknown), or an upper limit on the distance between clusters to stop merging. By our identification of the cluster distance as a magnitude of drift, we naturally re-use the drift detection threshold δ to also represent the tolerance level up to which clusters can be merged in Algorithm 4, which avoids introducing another hyperparameter.

One subtlety to Algorithm 4 is that the hierarchical clustering is iteratively run at every time step, because the cluster distances vary with time. A simpler alternative would be to only try merging newly created clusters of local models after one time step of training. However, at that one time step, even models corresponding to the same concept may fail to merge given the limited sample size and limited number of training iterations. In other words, while the models are still warming-up, they may still be separated by a distance exceeding δ. As the models converge over time, the distance may drop below δ, which Algorithm 4 accounts for by iteratively attempting to merge.

Algorithm 4 FedDrift at time τ

```
$\ell^{(\tau)}_{c,m} \leftarrow$ loss of model $h_m$ on client data $S^{(\tau)}_c$
**for** each client $c = 1, 2, \ldots, P$ in parallel **do**
    **if** $\min_m \ell^{(\tau)}_{c,m} > \min_m \ell^{(\tau-1)}_{c,m} + \delta$ **then**
        Initialize a local model at client $c$ to be added to the set of global models at $\tau + 1$, and assign client $c$ to its own cluster
    **else**
        $w^{(\tau)}_{c,m} \leftarrow \mathbf{1}\{m = \arg\min_{m'} \ell^{(\tau)}_{c,m'}\}$
**for** each $i, j$ from $1, 2, \ldots, M$ in parallel **do**
    $L_{ij} \leftarrow$ loss of model $h_i$ on subsample of $\cup_{c,t:w^{(t)}_{c,j}=1} S^{(t)}_c$
Cluster distances $D(i,j) \leftarrow \max(L_{ij} - L_{ii}, L_{ji} - L_{jj}, 0)$
**while** $\min_{i \neq j} D(i,j) < \delta$ **do**
    MERGE$(i, j, D)$
Run Algorithm 1

MERGE$(i, j, D)$:
Add a new model $h_k \leftarrow \frac{h_i \sum_{c,t} w^{(t)}_{c,i} N^{(t)}_c + h_j \sum_{c,t} w^{(t)}_{c,j} N^{(t)}_c}{\sum_{c,t} w^{(t)}_{c,i} N^{(t)}_c + \sum_{c,t} w^{(t)}_{c,j} N^{(t)}_c}$
$w^{(t)}_{c,k} \leftarrow w^{(t)}_{c,i} + w^{(t)}_{c,j}$ for all $c, t$
$D(k, l) = \max(D(i, l), D(j, l))$ for all $l$
Delete models $h_i, h_j$
```

The hierarchical clustering strategy of Algorithm 4 allows it to adaptively determine the appropriate number of clusters even when an unknown number of new concepts emerge at a time, but it also incurs additional computational resources relative to Algorithm 3. Algorithm 4 creates more global models to be broadcasted, adding to the communication cost. Additionally, the hierarchical clustering adds an $O(M^2 \log M)$ time complexity at the server at every time step (using a heap data structure for finding the minimum pairwise distance), where M is the number of global models.

Similar to Algorithm 1, each client c could maintain $w^{(t)}_{c,m}$ and $S^{(t)}_c$ for just a sliding window of the most recent time steps, as long as the window suffices for Algorithm 4's subsampling step.

5 Experimental Results

We empirically demonstrate that FedDrift-Eager and FedDrift are more effective than prior centralized drift adaptation and achieve high accuracy that is comparable to an oracle algorithm in the presence of distributed concept drifts. Prior work on FL under drifts is limited to simple cases such as in Figure 1, as noted in §2.2. Our evaluation is on the drifts in Figures 2 and 3, which represent more complex scenarios where drifts (i) occur across clients with staggered timing, (ii) correspond to different concept changes across different clients, and (iii) involve recurring concepts (e.g., the sequence A–B–C–D–A).

We investigate these drift patterns with respect to the following datasets studied in the literature on centralized drift adaptation [6, 38], FL for personalization [5], and FL under drifts [7, 26]. The datasets SINE and CIRCLE [31] each have two defined concepts, and we generate partitions of the data under the 2-concept staggered drift of Figure 2. In SINE, the first concept is a decision boundary of a sine curve and the second concept reverses the direction (swapping the labels). In CIRCLE, the two concepts are each decision boundaries of two different circles, representing a smaller concept

Table 1: Average accuracy (%) across all clients and time (over 5 trials)

	SINE-2	CIRCLE-2	SEA-2	MNIST-2	SEA-4	MNIST-4
Oblivious	50.44 ± 1.52	88.36 ± 0.27	86.37 ± 0.34	87.25 ± 0.14	85.38 ± 0.28	82.97 ± 0.04
DriftSurf	83.90 ± 1.01	92.54 ± 0.67	87.27 ± 0.34	91.71 ± 1.60	85.48 ± 0.28	82.99 ± 0.05
AUE	86.06 ± 0.60	92.74 ± 0.51	87.46 ± 0.12	92.19 ± 0.07	85.55 ± 0.08	81.29 ± 0.19
Window	86.42 ± 0.74	93.67 ± 0.15	88.08 ± 0.10	92.15 ± 0.34	85.76 ± 0.16	81.16 ± 0.46
Adaptive-FedAvg	78.02 ± 10.73	86.26 ± 0.00	86.69 ± 0.39	92.16 ± 0.04	85.32 ± 0.25	81.62 ± 0.07
FedDrift-Eager	98.46 ± 0.03	97.86 ± 0.20	88.35 ± 0.37	$\mathbf{95.99 \pm 0.06}$	88.08 ± 0.24	89.21 ± 2.02
FedDrift	$\mathbf{98.48 \pm 0.01}$	$\mathbf{97.88 \pm 0.17}$	$\mathbf{88.65 \pm 0.43}$	95.93 ± 0.01	$\mathbf{88.41 \pm 0.29}$	$\mathbf{94.09 \pm 0.08}$
Oracle	98.46 ± 0.01	97.57 ± 0.59	88.53 ± 0.23	96.00 ± 0.02	88.75 ± 0.20	94.60 ± 0.04

change than SINE. The datasets SEA [4] and MNIST [23] have more concepts defined, and we generate partitions of the data under both the 2-concept and 4-concept drift patterns of Figures 2 and 3. In SEA, each concept corresponds to a shifted hyperplane. In MNIST, concept A corresponds to the original labeling of the hand-drawn digits, and under each other concept, the labels of two of the digits are swapped (B swaps digits 1 and 2, C swaps digits 3 and 4, and D swaps digits 5 and 6).

We compare our algorithms FedDrift-Eager and FedDrift against the following baselines. First, the Oblivious algorithm learns a single model with FedAvg and has no mechanism for drift adaptation. Second, we consider traditional (non-FL) drift adaptation algorithms applied centrally at the server on top of FedAvg. Drift adaptation is typically classified into three categories, and we compare against algorithms representative of each: the drift detection method DriftSurf [38], the ensemble method AUE [6], and a Window method that forgets data older than one time step (more are reported in Appendix B). Third, Adaptive-FedAvg [7] is an FL algorithm that learns a single model and adapts to drifts by centrally tuning the learning rate used by all clients as a function of the variability across updates. Fourth, Oracle is an idealized algorithm that has oracle access to the concept ID at training time and runs the multi-model training of Algorithm 1 with the ground-truth clustering. Appendix B reports results with less competitive baselines, such as clustered FL [34] and personalized FL [15].

We run our experiments using the FedML framework [18]. Over the course of 10 time steps, each of 10 clients observes a new batch of training data. The models trained under each algorithm are fully connected neural networks with a single hidden layer of size $2d$ where d is the number of features. After training for each time step, we test each algorithm over the batch of data arriving at the following time step. Each experiment is run for 5 trials, and we report the mean and the standard deviation. Additional details on the experimental setup regarding datasets, algorithms, and hyperparameters are in Appendix A.

In Table 1, we report the accuracy averaged across all clients and all time steps except for the times of drifts. We omit the times of drift because there is no chance for a client to adapt to the drift yet, and all algorithms suffer from the inevitable performance loss. By omitting the time of drift, we eliminate the noise from beneficial clustering mistakes if by chance a client was clustered to the model appropriate for the test data after the drift. For completeness, the results averaging over all time steps including drifts are in Appendix B.

Across all the 2-concept datasets under the staggered drift, we observe that the multi-model algorithms FedDrift-Eager and FedDrift outperform the prior centralized solutions. In Figure 4(left), the accuracy is broken down per time step on CIRCLE-2, where we observe that centralized algorithms particularly suffer during the transition period. The fundamental issue is that when both concepts simultaneously exist, there is no single model that is an accurate fit at all clients. Even AUE, which uses a weighted combination over its ensemble of models, has poor performance because all of its models are updated by all clients, and during the transition period, none of its models are trained solely over data from the second concept. On the other hand, FedDrift-Eager and FedDrift, with multiple models trained by different clusters of clients, learn models specialized for the second concept immediately after it emerges, and learn to apply the appropriate model at each client during the transition, matching the performance of Oracle.

Another challenge that the 2-concept staggered drift poses for DriftSurf, AUE, and Adaptive-FedAvg is that their adaptation strategies are a function of estimators that, from the perspective at the central server, are aggregated over some clients that are drifting and others that are not. It is muddy whether drift is truly occurring, and even the unsophisticated window-based algorithm performs slightly better.

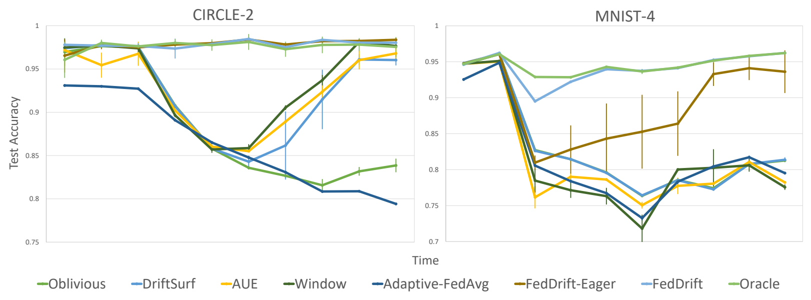
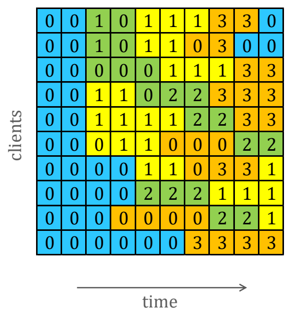


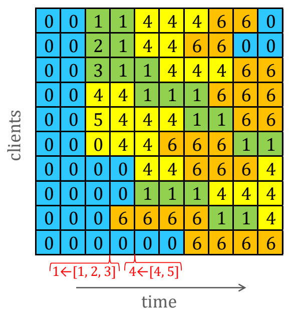
Figure 4: Accuracy at each time (averaged across clients) on CIRCLE-2 and MNIST-4.

Regarding the 4-concept drift, we visualize the accuracy per time step on MNIST-4 in Figure 4(right). The performance of the centralized baselines never recover as long as multiple concepts exist. To better understand the performance of FedDrift-Eager and FedDrift, we refer to the clustering learned by each in Figure 5. In the ideal case (Oracle), there is exactly one model for each concept.

For FedDrift, at time 3 one new model is created for 5 of the 6 clients that drifted, and one false negative where a drifted client stays on the original model. With hierarchical clustering applied at the beginning of time 4, the 3 clusters corresponding to the green concept are correctly merged, while all clients on the yellow concept cluster to model 4 which had the lowest test loss over the new data. Also at time 4, model 6 is created for the new orange concept. Then at time 5, hierarchical clustering merges models 4 and 5 (due its iterative application in FedDrift, as the distance between models 4 and 5 decreases after model 4 is further trained). After time 5, FedDrift has a distinct model for each concept, and no excess models.



(a) FedDrift-Eager



(b) FedDrift

Figure 5: The clustering learned on MNIST-4. Each cell indicates the model ID at each client and time step, and the background color indicates the ground-truth concept.

On the other hand, for FedDrift-Eager, when drifts occur at time 3, only model 1 is created for both the yellow and green concepts. At time 4, clustering to the lowest loss separates the yellow and green concepts, but then only a single model is applied for the blue and green concepts, as well as the orange concept due to a missed drift detection. Ultimately, FedDrift-Eager does eventually learn to use a distinct model for each concept over time in response to later drifts (when the green concept occurs again at time 5 and the orange concept at 7), and the accuracy recovers at the end in Figure 4. Meanwhile, the accuracy of FedDrift is close to Oracle throughout, with one gap at time 3 when each local model is created.

As noted in §4, one of the drawbacks of FedDrift is that it can create more models compared to FedDrift-Eager, adding to the communication cost. Appendix B shows that restricting FedDrift to just one new global model per time step (additional local models are still permitted) decreases its accuracy by only 0.61 percentage points on the MNIST-4 dataset, while saving communication.

6 Conclusion

Federated learning under distributed concept drift is a largely unexplored area, posing particular challenges because drifts can arise staggered in time and space (across clients). This paper presented FedDrift-Eager and FedDrift, the first algorithms explicitly designed to mitigate these challenges. Empirical evaluation on a variety of dataset/drift combinations showed that these algorithms achieve significantly higher accuracy than existing baselines, and are comparable to an idealized algorithm with oracle knowledge of the ground-truth clustering of clients to concepts at each time step. Future work includes exploring ways to address the privacy implications of clustering clients.

References

[1] Manuel Baena-García, José del Campo-Ávila, Raúl Fidalgo, Albert Bifet, R Gavaldà, and R Morales-Bueno. Early drift detection method. In *StreamKDD*, pages 77–86, 2006.

[2] Romil Bhardwaj, Zhengxu Xia, Ganesh Ananthanarayanan, Junchen Jiang, Nikolaos Karianakis, Yuanchao Shu, Kevin Hsieh, Victor Bahl, and Ion Stoica. Ekya: Continuous learning of video analytics models on edge compute servers. *CoRR*, abs/2012.10557, 2020.

[3] Albert Bifet and Ricard Gavaldà. Learning from time-changing data with adaptive windowing. In *ICDM*, pages 443–448, 2007.

[4] Albert Bifet, Geoff Holmes, Richard Kirkby, and Bernhard Pfahringer. MOA: Massive online analysis. *JMLR*, 11:1601–1604, 2010.

[5] Christopher Briggs, Zhong Fan, and Peter Andras. Federated learning with hierarchical clustering of local updates to improve training on non-iid data. In *International Joint Conference on Neural Networks (IJCNN)*, 2020.

[6] Dariusz Brzezinski and Jerzy Stefanowski. Reacting to different types of concept drift: The accuracy updated ensemble algorithm. *IEEE Trans. Neural Netw. Learn. Syst*, 25(1):81–94, 2013.

[7] Giuseppe Canonaco, Alex Bergamasco, Alessio Mongelluzzo, and Manuel Roveri. Adaptive federated learning in presence of concept drift. In *International Joint Conference on Neural Networks*, pages 1–7, 2021.

[8] Fernando E Casado, Dylan Lema, Marcos F Criado, Roberto Iglesias, Carlos V Regueiro, and Senén Barro. Concept drift detection and adaptation for federated and continual learning. *Multimedia Tools and Applications*, pages 1–23, 2021.

[9] Yujing Chen, Zheng Chai, Yue Cheng, and Huzefa Rangwala. Asynchronous federated learning for sensor data with concept drift. In *IEEE International Conference on Big Data*, pages 4822–4831, 2021.

[10] Moming Duan, Duo Liu, Xinyuan Ji, Yu Wu, Liang Liang, Xianzhang Chen, Yujuan Tan, and Ao Ren. Flexible clustered federated learning for client-level data distribution shift. *IEEE Transactions on Parallel and Distributed Systems*, 2021.

[11] Joao Gama, Pedro Medas, Gladys Castillo, and Pedro Rodrigues. Learning with drift detection. In *Advances in Artificial Intelligence-SBIA*, pages 286–295, 2004.

[12] João Gama, Indre Zliobaite, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia. A survey on concept drift adaptation. *ACM Comput. Surv.*, 46(4), 2014.

[13] Abhinav Garg, Naman Shukla, Lavanya Marla, and Sriram Somanchi. Distribution shift in airline customer behavior during COVID-19. *CoRR*, abs/2111.14938, 2021.

[14] Avishek Ghosh, Justin Hong, Dong Yin, and Kannan Ramchandran. Robust federated learning in a heterogeneous environment. *arXiv preprint arXiv:1906.06629*, 2019.

[15] Avishek Ghosh, Jichan Chung, Dong Yin, and Kannan Ramchandran. An efficient framework for clustered federated learning. In *NeurIPS*, pages 19586–19597, 2020.

[16] Yongxin Guo, Tao Lin, and Xiaoying Tang. Towards federated learning on time-evolving heterogeneous data. *arXiv preprint arXiv:2112.13246*, 2021.

[17] Maayan Harel, Koby Crammer, Ran El-Yaniv, and Shie Mannor. Concept drift detection through resampling. In *ICML*, pages 1009–1017, 2014.

[18] Chaoyang He, Songze Li, Jinhyun So, Mi Zhang, Hongyi Wang, Xiaoyang Wang, Praneeth Vepakomma, Abhishek Singh, Hang Qiu, Li Shen, Peilin Zhao, Yan Kang, Yang Liu, Ramesh Raskar, Qiang Yang, Murali Annavaram, and Salman Avestimehr. Fedml: A research library and benchmark for federated machine learning. *arXiv preprint arXiv:2007.13518*, 2020.

[19] Peter Kairouz, H Brendan McMahan, et al. Advances and open problems in federated learning. *Foundations and Trends® in Machine Learning*, 14(1–2):1–210, 2021.

[20] Ioannis Katakis, Grigorios Tsoumakas, and Ioannis Vlahavas. Tracking recurring contexts using ensemble classifiers: an application to email filtering. *Knowledge and Information Systems*, 22 (3):371–391, 2010.

[21] Pang Wei Koh, Shiori Sagawa, Henrik Marklund, Sang Michael Xie, Marvin Zhang, Akshay Balsubramani, Weihua Hu, Michihiro Yasunaga, Richard Lanas Phillips, Irena Gao, Tony Lee, Etienne David, Ian Stavness, Wei Guo, Berton Earnshaw, Imran Haque, Sara M. Beery, Jure Leskovec, Anshul Kundaje, Emma Pierson, Sergey Levine, Chelsea Finn, and Percy Liang. WILDS: A benchmark of in-the-wild distribution shifts. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2021.

[22] Jakub Konečný, H. Brendan McMahan, Felix X. Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. Federated learning: Strategies for improving communication efficiency. *CoRR*, abs/1610.05492, 2016.

[23] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[24] Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. Federated optimization in heterogeneous networks. *Proceedings of Machine Learning and Systems*, 2:429–450, 2020.

[25] Jie Lu, Anjin Liu, Fan Dong, Feng Gu, Joao Gama, and Guangquan Zhang. Learning under concept drift: A review. *IEEE Transactions on Knowledge and Data Engineering*, 31(12): 2346–2363, 2018.

[26] Dimitrios Michael Manias, Ibrahim Shaer, Li Yang, and Abdallah Shami. Concept drift detection in federated networked systems. *arXiv preprint arXiv:2109.06088*, 2021.

[27] Yishay Mansour, Mehryar Mohri, Jae Ro, and Ananda Theertha Suresh. Three approaches for personalization with applications to federated learning. *arXiv preprint arXiv:2002.10619*, 2020.

[28] MarketsAndMarkets. Federated learning solutions market by application (drug discovery, industrial iot), vertical (healthcare and life sciences, bfsi, manufacturing, retail and ecommerce, energy and utilities), and region - global forecast to 2028. `https://www.marketsandmarkets.com/Market-Reports/federated-learning-solutions-market-151896843.html`, 2021.

[29] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2017.

[30] Ali Pesaranghader and Herna L Viktor. Fast hoeffding drift detection method for evolving data streams. In *ECML PKDD*, pages 96–111, 2016.

[31] Ali Pesaranghader, Herna L Viktor, and Eric Paquet. A framework for classification in data streams using multi-strategy learning. In *ICDS*, pages 341–355, 2016.

[32] Ali Pesaranghader, Herna L Viktor, and Eric Paquet. McDiarmid drift detection methods for evolving data streams. In *International Joint Conference on Neural Networks (IJCNN)*, 2018.

[33] Nicola Rieke, Jonny Hancox, Wenqi Li, Fausto Milletari, Holger R Roth, Shadi Albarqouni, Spyridon Bakas, Mathieu N Galtier, Bennett A Landman, Klaus Maier-Hein, et al. The future of digital health with federated learning. *NPJ digital medicine*, 3(1), 2020.

[34] Felix Sattler, Klaus-Robert Müller, and Wojciech Samek. Clustered federated learning: Model-agnostic distributed multitask optimization under privacy constraints. *IEEE transactions on neural networks and learning systems*, 32(8):3710–3722, 2020.

[35] Shai Shalev-Shwartz and Shai Ben-David. *Understanding machine learning: From theory to algorithms*. Cambridge university press, 2014.

[36] Hidetoshi Shimodaira. Improving predictive inference under covariate shift by weighting the log-likelihood function. *Journal of statistical planning and inference*, 90(2), 2000.

[37] Abhijit Suprem, Joy Arulraj, Calton Pu, and Joao Ferreira. ODIN: Automated drift detection and recovery in video analytics. *Proceedings of the VLDB Endowment*, 13(11), 2020.

[38] Ashraf Tahmasbi, Ellango Jothimurugesan, Srikanta Tirthapura, and Phillip B. Gibbons. Drift-Surf: Stable-state / reactive-state learning under concept drift. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2021.

[39] Alexey Tsymbal. The problem of concept drift: definitions and related work. *Computer Science Department, Trinity College Dublin*, 106(2), 2004.

[40] Jianyu Wang and Gauri Joshi. Cooperative sgd: A unified framework for the design and analysis of local-update sgd algorithms. *Journal of Machine Learning Research*, 22(213):1–50, 2021.

[41] Gerhard Widmer and Miroslav Kubat. Learning in the presence of concept drift and hidden contexts. *Machine learning*, 23(1), 1996.

[42] Qiang Yang, Yang Liu, Tianjian Chen, and Yongxin Tong. Federated machine learning: Concept and applications. *ACM Trans. Intell. Syst. Technol.*, 10(2), 2019.

A Experimental Parameters

For each dataset in our experiments, the training data are distributed across 10 clients and arrive over 10 time steps. The partition of the data at each client and time is a constant 500 number of samples from the concept corresponding to the concept drift patterns in Figures 2 and 3 in §2.2. In our experimental results, after training at each time τ we report the test accuracy over the data at $\tau + 1$. For clarification, in reporting the accuracy at the last time step 10, we test over an 11th sample of data at each client that is from the same concept observed during training at time 10.

Across all algorithms we evaluate, the algorithms that learn a single model use FedAvg for training, and the clustering algorithms that learn multiple models use Algorithm 1 in §3 for training (which reduces to FedAvg when there is one cluster). The training parameters used in our experiments are shown in Table 2.

Table 2: Training parameters

Parameter	Description	Experimental setting
P	# clients	10
R	# communication rounds	100
K	# local steps per model per round	50
$N_c^{(t)}$	number of new data arrivals at client c at time t	500
B	minibatch size	50
η	step size	varies

Regarding the learning rate selection, first we discuss all algorithms excluding Adaptive-FedAvg. We searched for learning rates of the form 10^{-a} for $a = 1, 2, 3, 4$, for each dataset, and found that $\eta = 10^{-2}$ was the best for SINE-2, CIRCLE-2, SEA-2, and SEA-4, and that $\eta = 10^{-3}$ was best for MNIST-2 and MNIST-4. (This held for both of the two extremes among our baselines, Oblivious and Oracle, and we apply the same learning rate across all the algorithms.) Also note that for computing the Local Update at each client, we use the implementation of Adam in PyTorch with the options weight decay = 10^{-3} and amsgrad = True. We treat Adaptive-FedAvg separately, because it uses SGD with its own internal learning rate scheduler as its mechanism to react to drifts. We found that the initial learning rate of 10^{-2} was the best for each dataset with the exception of SINE-2, instead using 10^{-1}. (This higher learning rate explains the high standard deviation in the reported accuracy of Adaptive-FedAvg on SINE-2.)

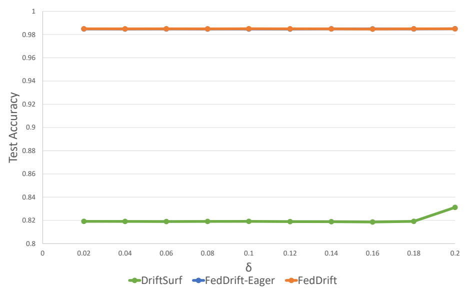
Next, we report the selection of the drift detection threshold δ in the algorithms DriftSurf, FedDrift-Eager, and FedDrift. While the optimal δ is expected to vary across datasets, even for a fixed dataset, different algorithms can peak in performance at varying δ. The performance of each of these three algorithms for each dataset across δ in the range $0.02, 0.04, \ldots, 0.20$ is shown in Figure 6. To not bias towards any one algorithm, the experimental results are reported for each algorithm and dataset using its best δ. However, using a fixed $\delta = 0.04$ for FedDrift-Eager and FedDrift makes at most a 1 pp difference in the results reported in Table 1 (on one trial).

For all other hyperparameters of the algorithms we evaluate, we follow the parameter choices stated in the original papers, with the following exceptions: for DriftSurf, we use $r = 3$ (which performed better than their suggested $r = 4$) and for AUE, we use $K = 5$ as the total ensemble size (compared to the $K = 10$ in their paper they consider over a significantly longer time horizon.)

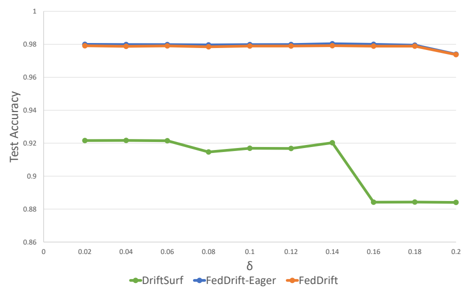
Finally, regarding the model training in Algorithm 1 at time τ, we apply one optimization for efficiency to only train models that are currently clustered to. (Although note that any such models are still retained by FedDrift-Eager and FedDrift in order to react to recurring drifts even if they are not actively being trained.)

B Additional Experimental Results

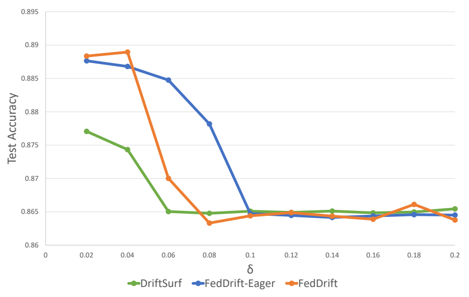
We present additional experimental results on more baseline algorithms and on variants of our algorithms restricted to limited memory or communication.



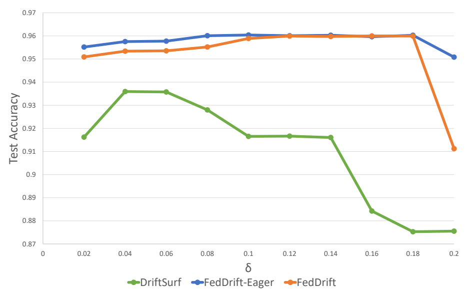
(a) SINE-2



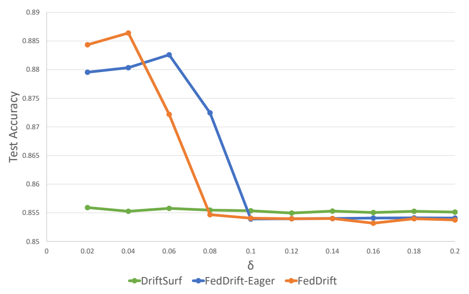
(b) CIRCLE-2



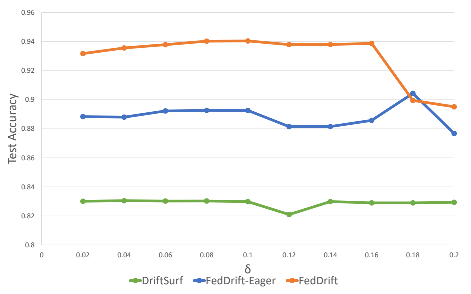
(c) SEA-2



(d) MNIST-2



(e) SEA-4



(f) MNIST-4

Figure 6: Average accuracy of each drift detection-based algorithm under varying thresholds δ.

Additional Baseline Algorithms. The additional algorithms presented in this appendix are:

- **Four traditional drift adaptation algorithms.** AUE-PC is a variation of the ensemble method AUE with the ensemble weights set *per-client*. Window-2 is a window method like Window, except that it forgets data older than two time steps instead of one. Weighted-Linear and Weighted-Exp also forget older data like window methods, but do so more gradually by down-weighting older data with either linear or exponential decay.
- **The FL clustering algorithm CFL [34].** In extending the original static algorithm to our time-varying setting, we also consider a variant CFL-W, in which during training, each client samples only from the window of the newest data arriving at each time.
- **Three variations of the IFCA clustering algorithm [15]** that we considered for extending the original algorithm to the time-varying setting. First, IFCA(T) is exactly Algorithm 2 in §3, which defines cluster identities for each client and each time, in order to associate the data within a client that are heterogeneous over time across multiple clusters. IFCA(T) chooses the cluster identity once per *time step* (where time steps consist of multiple communication rounds)—this differs from the original algorithm described by Ghosh et al. [15], which

recomputes the cluster identity once per *round*. Second, IFCA does the per-round clustering; more precisely, for each time step τ, the cluster identity $w_{c,m}^{(\tau)}$ is recomputed at every round under the same equation used at the beginning of the time step in Algorithm 2. Third, IFCA-W is a variant of IFCA that trains only over the most recent data arrivals at each time, and the cluster identities of data from previous time steps are forgotten. In general, the IFCA-based algorithms require the number of clusters as input, which we provide as oracle knowledge—either 2 or 4 depending on the total number of concepts over time in each dataset. This gives IFCA-based algorithms an advantage over all other algorithms we evaluate, which do not know the number of clusters a priori. For the initialization of all three variations, at time 1 and round 1, all clients are assigned to a single cluster, matching the assumption we made for FedDrift and FedDrift-Eager in §4.

- **A more communication-efficient variant of FedDrift.** FedDrift-C is the algorithm referred to in the last paragraph of §5 that is restricted to introducing one new global model per time step. More details on this algorithm are described later in this section.
- **Sliding window variants of FedDrift-Eager and FedDrift.** FedDrift-Eager-W and FedDrift-W are restricted to using only the most recent time step of data $S_c^{(t)}$ and cluster identities $w_{c,m}^{(t)}$.
- **A baseline sliding window variant** Oracle-W, which has oracle access to the ground-truth clustering but only uses the most recent time step of data in training.

In general, we use the -W suffix in the name of an algorithm to indicate a limited memory of a window of one time step. This memory restriction reduces the number of samples used for training at a time and might reduce the accuracy achievable under ground-truth clustering (Oracle-W vs. Oracle). Yet, the window is not strictly a drawback: (i) forgetting the older data builds in a passive adaptation to drift and (ii) in our setting it also guarantees that each client's training data at a step are all drawn from the same distribution—this is why we also investigate -W variants when extending the prior static clustering algorithms CFL and IFCA to our setting when data arrive over time.

Test Accuracy Results. Table 3 (extending Table 1 in §5) shows the test accuracy of all algorithms, averaged across all clients and time steps, but omitting the times of drifts. As noted in §5, we omit the times of drift when all algorithms suffer from the performance loss. For completeness, the test accuracy averaged over all time steps including drifts is shown in Table 4. In this latter table, note that Oracle and Oracle-W suffer a performance loss too at the time of drift. Under the test-then-train evaluation, Oracle has access to the concept ID of the data at training time but not at test time, where at each client, the model used for inference corresponds to the observed concept in the most recently arrived training data.

Based on these tables, we make the following observations on the additional algorithms. The AUE-PC variant of AUE extends the model weights in the ensemble method to be individualized per-client, based on the performance of each model over each client's local data (as opposed to weights chosen based on the aggregate performance at the server). This additional flexibility leads to only a marginal accuracy improvement over AUE across all datasets. While it is generally valuable for clients at different stages of a staggered drift to use different models for inference, the more fundamental obstacle is that each global model trained by AUE-PC is updated by all clients. In the course of the 2-concept staggered drift, all of the models in the ensemble are trained either over a mixture of data from both concepts or solely from the first concept, and there is no accurate model available that is a good fit for the second concept.

The Window-2 algorithm and the weighted sampling algorithms Weighted-Linear and Weighted-Exp are techniques for forgetting older data, but less abruptly compared to Window-1, and in general they all perform similarly. On the sharp drift of SINE-2, the fastest forgetting algorithm Window performs the best of these. On the other hand, on the 4-concept drift of MNIST-4 in which the time axis does not well separate different concepts, the slowest forgetting algorithm Weighted-Linear performs best. Meanwhile, the performance of all four algorithms are close on the SEA datasets, which have greater overlap between the concepts.

The clustering algorithms CFL and CFL-W start with each client in one cluster, and recursively split clusters over rounds and over time based on the intra-cluster similarity of their local updates. We observe that the CFL-W variant is the better-performing of the two on each dataset except MNIST-4 (which is also the only dataset where Oblivious outperforms Window), and is a consequence of the

Table 3: Average test accuracy (%) across clients and time, omitting drifts (5 trials)

	SINE-2	CIRCLE-2	SEA-2	MNIST-2	SEA-4	MNIST-4
Oblivious	50.44 ± 1.52	88.36 ± 0.27	86.37 ± 0.34	87.25 ± 0.14	85.38 ± 0.28	82.97 ± 0.04
DriftSurf	83.90 ± 1.01	92.54 ± 0.67	87.27 ± 0.34	91.71 ± 1.60	85.48 ± 0.28	82.99 ± 0.05
AUE	86.06 ± 0.60	92.74 ± 0.51	87.46 ± 0.12	92.19 ± 0.07	85.55 ± 0.08	81.29 ± 0.19
AUE-PC	87.67 ± 1.70	93.05 ± 0.19	87.61 ± 0.08	92.22 ± 0.09	85.60 ± 0.05	81.43 ± 0.22
Window	86.42 ± 0.74	93.67 ± 0.15	88.08 ± 0.10	92.15 ± 0.34	85.76 ± 0.16	81.16 ± 0.46
Window-2	85.21 ± 1.67	93.03 ± 0.46	87.71 ± 0.33	92.54 ± 0.37	85.67 ± 0.16	82.16 ± 0.32
Weighted-Linear	72.78 ± 1.23	89.91 ± 0.65	87.00 ± 0.01	89.70 ± 0.12	85.49 ± 0.17	82.79 ± 0.05
Weighted-Exp	82.77 ± 0.64	92.69 ± 0.25	87.59 ± 0.15	92.19 ± 0.17	85.59 ± 0.09	82.55 ± 0.06
CFL	60.27 ± 4.82	88.39 ± 0.40	86.36 ± 0.28	86.97 ± 0.40	85.33 ± 0.26	81.95 ± 0.55
CFL-W	95.15 ± 0.32	95.62 ± 1.14	87.66 ± 0.36	90.53 ± 0.81	85.67 ± 0.21	79.99 ± 0.58
IFCA(T)	98.45 ± 0.03	91.72 ± 5.19	86.46 ± 0.23	87.33 ± 0.15	85.44 ± 0.14	82.90 ± 0.05
IFCA	98.46 ± 0.02	92.20 ± 5.32	86.45 ± 0.25	87.55 ± 0.25	85.35 ± 0.09	82.89 ± 0.04
IFCA-W	98.49 ± 0.13	94.31 ± 1.62	88.04 ± 0.17	91.76 ± 0.50	86.17 ± 1.00	81.27 ± 0.43
Adaptive-FedAvg	78.02 ± 10.73	86.26 ± 0.00	86.69 ± 0.39	92.16 ± 0.04	85.32 ± 0.25	81.62 ± 0.07
FedDrift-Eager	98.46 ± 0.03	97.86 ± 0.20	88.35 ± 0.37	**95.99 ± 0.06**	88.08 ± 0.24	89.21 ± 2.02
FedDrift	98.48 ± 0.01	**97.88 ± 0.17**	**88.65 ± 0.43**	95.93 ± 0.01	**88.41 ± 0.29**	**94.09 ± 0.08**
FedDrift-C	98.51 ± 0.11	97.42 ± 0.57	88.30 ± 0.53	95.85 ± 0.05	87.46 ± 0.42	93.22 ± 0.44
FedDrift-Eager-W	98.51 ± 0.12	97.34 ± 0.76	88.43 ± 0.23	94.05 ± 0.02	87.90 ± 0.25	89.31 ± 0.38
FedDrift-W	**98.58 ± 0.17**	97.68 ± 0.09	88.43 ± 0.22	93.95 ± 0.02	88.17 ± 0.39	91.47 ± 0.07
Oracle	98.46 ± 0.01	97.57 ± 0.59	88.53 ± 0.23	96.00 ± 0.02	88.75 ± 0.20	94.60 ± 0.04
Oracle-W	98.47 ± 0.03	97.84 ± 0.11	88.70 ± 0.17	94.04 ± 0.02	88.74 ± 0.13	91.89 ± 0.05

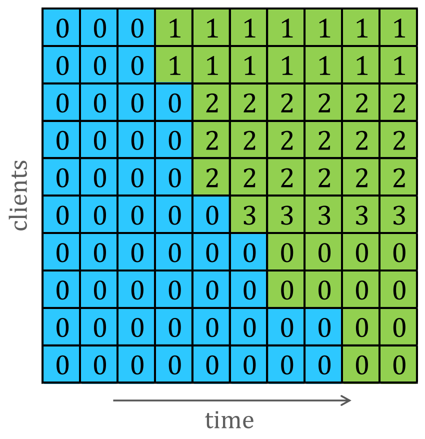


Figure 7: The clustering learned by CFL-W on SINE-2. The number at each client and time indicates the model ID and the background color indicates the ground-truth concept.

passive drift adaptation of its sliding window which forgets older data. The performance of CFL-W is relatively high on SINE-2 and CIRCLE-2. As an example, the clustering learned on SINE-2 is shown in Figure 7. We observe that, for the first 6 time steps, it correctly distinguishes the two concepts by using distinct models. The disadvantage of the clustering of CFL-W is that it creates excess models for the same concept and does not take full advantage of collaborative training. At time 5, it is limited to splitting its cluster for model 0 when the green concept occurs, but cannot merge the drifted clients to the existing cluster created for the green concept at the previous time step.

For IFCA, IFCA-W, and IFCA(T), the clustering is pre-initialized with a random model for each concept that can occur over time for each dataset. In general, we observe that this is not a reliable method for reacting to drift. All the IFCA variants perform well under the sharp label-swap drift of SINE-2. When the new concept occurs, the drifted clients cluster to the second model, and the learned clustering matches the ground-truth. On CIRCLE-2, we found that IFCA and IFCA(T) learned the correct clustering in 2 out of 5 trials, and otherwise used only a single model in the other 3 trials. IFCA-W learned the correct clustering in 1 out of 5 trials. (Note the high standard deviation in Table

Table 4: Average test accuracy (%) across clients and time, including drifts (5 trials)

	SINE-2	CIRCLE-2	SEA-2	MNIST-2	SEA-4	MNIST-4
Oblivious	45.77 ± 1.52	87.12 ± 0.26	86.12 ± 0.35	86.28 ± 0.12	85.11 ± 0.24	81.60 ± 0.03
DriftSurf	79.19 ± 0.88	91.16 ± 0.68	87.00 ± 0.35	90.55 ± 1.68	85.13 ± 0.19	81.62 ± 0.04
AUE	81.28 ± 0.81	91.50 ± 0.46	87.21 ± 0.11	91.07 ± 0.07	85.15 ± 0.07	79.65 ± 0.25
AUE-PC	82.18 ± 2.01	91.75 ± 0.17	87.34 ± 0.08	91.07 ± 0.09	85.16 ± 0.04	79.70 ± 0.24
Window	81.92 ± 0.88	92.40 ± 0.11	87.86 ± 0.08	91.35 ± 0.43	85.33 ± 0.10	78.88 ± 0.62
Window-2	80.35 ± 2.02	91.73 ± 0.49	87.45 ± 0.34	91.47 ± 0.47	85.24 ± 0.15	80.06 ± 0.61
Weighted-Linear	67.20 ± 1.43	88.67 ± 0.64	86.77 ± 0.02	88.56 ± 0.12	85.16 ± 0.11	81.38 ± 0.04
Weighted-Exp	76.80 ± 0.88	91.30 ± 0.26	87.34 ± 0.16	91.05 ± 0.18	85.19 ± 0.06	80.96 ± 0.07
CFL	54.41 ± 4.33	87.08 ± 0.31	86.10 ± 0.30	86.00 ± 0.38	85.00 ± 0.25	80.45 ± 0.64
CFL-W	86.83 ± 0.55	93.72 ± 0.93	87.36 ± 0.42	89.47 ± 0.74	85.25 ± 0.17	77.35 ± 0.81
IFCA(T)	88.77 ± 0.02	90.06 ± 4.62	86.22 ± 0.22	86.36 ± 0.14	85.12 ± 0.09	81.53 ± 0.05
IFCA	88.78 ± 0.02	90.49 ± 4.73	86.21 ± 0.28	86.56 ± 0.21	85.06 ± 0.04	81.51 ± 0.03
IFCA-W	88.80 ± 0.12	92.84 ± 1.19	87.84 ± 0.14	90.81 ± 0.67	85.52 ± 0.50	79.17 ± 0.39
Adaptive-FedAvg	73.82 ± 10.75	85.60 ± 0.00	86.55 ± 0.35	91.31 ± 0.05	85.01 ± 0.21	79.45 ± 0.06
FedDrift-Eager	88.76 ± 0.01	95.51 ± 0.18	87.86 ± 0.33	**94.09 ± 0.05**	86.64 ± 0.18	83.58 ± 0.79
FedDrift	88.77 ± 0.02	**95.54 ± 0.15**	**88.13 ± 0.39**	94.03 ± 0.02	**86.68 ± 0.20**	**85.72 ± 0.07**
FedDrift-C	88.82 ± 0.09	95.12 ± 0.50	87.78 ± 0.44	93.97 ± 0.05	86.21 ± 0.40	85.62 ± 0.47
FedDrift-Eager-W	88.82 ± 0.12	95.05 ± 0.67	87.87 ± 0.23	92.04 ± 0.03	86.44 ± 0.20	82.15 ± 0.32
FedDrift-W	**88.88 ± 0.15**	95.35 ± 0.08	87.95 ± 0.15	91.93 ± 0.03	86.46 ± 0.31	83.29 ± 0.06
Oracle	88.77 ± 0.01	95.25 ± 0.52	87.99 ± 0.20	94.11 ± 0.02	86.89 ± 0.17	86.10 ± 0.03
Oracle-W	88.77 ± 0.03	95.51 ± 0.10	88.15 ± 0.14	92.03 ± 0.01	86.83 ± 0.06	83.58 ± 0.03

3.) On the remaining datasets, none of the three algorithms ever used more than a single model (with one exception—on SEA-4, in 1 out of 5 trials, IFCA-W used a distinct model for the yellow concept). For these remaining datasets, we observe that IFCA and IFCA(T) degrade to the Oblivious algorithm, and that IFCA-W degrades to the Window algorithm. The authors of the original paper on IFCA note that the accuracy of the clustering is sensitive to the initialization of the models, and propose random restarts to address this issue, but restarts do not translate well to the time-varying setting we study. In our work, FedDrift-Eager and FedDrift address the initialization problem by using drift detection to deal with new concepts as they occur and to cultivate new clusters.

We consider a more communication-efficient variant of FedDrift that we call FedDrift-C. As noted in §4, one of the drawbacks of FedDrift is that it can create more models compared to FedDrift-Eager, adding to the communication cost. The goal is to only use a number of global models close or equal to the number of distinct concepts, and while FedDrift can hierarchically merge created models of the same concept, FedDrift can observe temporary spikes in the number of global models. To mitigate this cost, we consider the variant FedDrift-C, which differs from FedDrift in that, at each time after drift occurs, only one random client that drifted contributes its local model as a global model. In the case that multiple new concepts occur at a time, only one of the new concepts will be learned immediately, but clients that are still at an unlearned concept can detect drift again at the following time step, and get another chance to contribute its local model. Meanwhile, while a concept goes unlearned globally, the drifted clients do not contribute to any of the global models. We observe that the accuracy of the FedDrift-C variant is only marginally sub-optimal relative to FedDrift on MNIST-4, and is superior to FedDrift-Eager.

For FedDrift-Eager-W and FedDrift-W, restricting to a window has minimal impact on the accuracy for the SEA dataset. There is a significant loss of accuracy for the MNIST dataset relative to the non-windowed versions, but note that the same significant loss occurs when going from Oracle to Oracle-W, so this loss is a result of windowing, not specific to our algorithm. Indeed, the accuracy of FedDrift-W is quite close to Oracle-W.

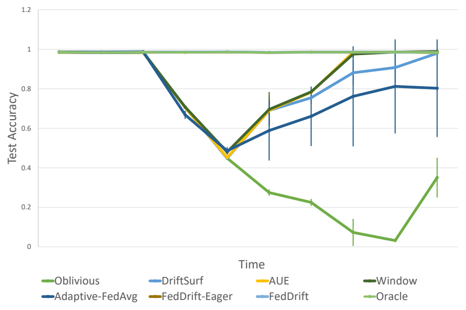
Random Drift Patterns. Throughout this paper, we have considered the 4-concept drift pattern in Figure 3 in §2.2 as a specific concrete example in order to depict the challenges in distributed concept drift, motivate the design of FedDrift, and discuss the experimental performance by comparing the learned clustering matrix to the ground-truth. To demonstrate the performance of our algorithms

Table 5: Average accuracy (%), omitting drifts

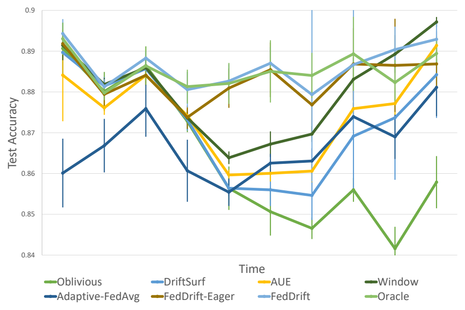
	MNIST-R
Oblivious	85.12 ± 1.37
FedDrift-Eager	89.85 ± 1.49
FedDrift	94.06 ± 0.38
Oracle	95.03 ± 0.15

Table 6: Average accuracy (%), including drifts

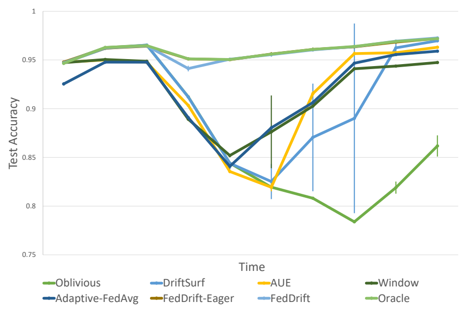
	MNIST-R
Oblivious	83.92 ± 1.23
FedDrift-Eager	85.26 ± 0.81
FedDrift	86.77 ± 0.76
Oracle	87.32 ± 0.86



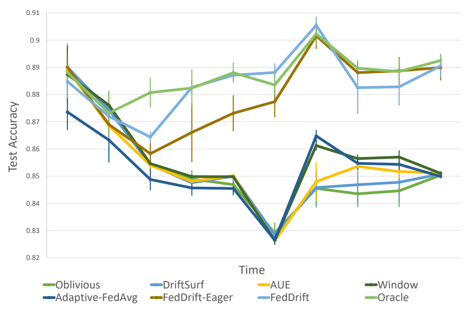
(a) SINE-2



(b) SEA-2



(c) MNIST-2



(d) SEA-4

Figure 8: Test accuracy at each time on SINE-2, SEA-2, MNIST-2, and SEA-4. Vertical lines represent standard deviations.

more generally, we consider a family of datasets MNIST-R with random concept changes. Using the same four concepts as in MNIST-4, MNIST-R is generated with all clients at the first concept to start, and then each client independently randomly observes one of the four concepts every two time steps (as opposed to every time step which is not possible to adapt to). We evaluate both FedDrift-Eager and FedDrift, compared to the single-model baseline Oblivious and the ideal multi-model baseline Oracle. Across 5 random seeds, the average accuracy is shown in Table 5 (and in Table 6 for all time including drifts). We observe that the performance of FedDrift is close to that of Oracle. Meanwhile, the performance of FedDrift-Eager is behind, given that it is likely to have multiple new concepts occurring simultaneously in MNIST-R.

Test Accuracy Over Time. Finally, in Figure 8, we include the plots of the accuracy over time that were omitted from the body of the paper for space, supplementing Figure 4 in §5. (Note the varying scales of the y-axes.) Similarly to Figure 4, here we observe the same general trends: (i) all of the centralized drift adaptation algorithms suffer in performance, particularly during the transition period when no one model works well across all clients; (ii) for the 4-concept drift on SEA-4, centralized baselines never recover in performance when multiple concepts are present; and (iii) FedDrift-Eager lags behind FedDrift on SEA-4 because it creates only one model after the drifts at time 3 corresponding to two concepts, but eventually recovers.