CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Detection and Adaptation of Staggered Concept Drift in Federated Online Learning

A Dynamic Cluster-Based Framework for Machine Learning with Distributed Heterogenous Data Streams

Master's thesis in Computer science and engineering

Fabian Forsman
Josef Jakobson

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2026

Master's thesis 2026

Detection and Adaptation of Staggered Concept Drift in Federated Online Learning

A Dynamic Cluster-Based Framework for Machine Learning with Distributed Heterogenous Data Streams

FABIAN FORSMAN & JOSEF JAKOBSON

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2026

Detection and Adaptation of Staggered Concept Drift in Federated Online Learning
A Dynamic Cluster-Based Framework for Machine Learning with Distributed Heterogenous Data Streams
FABIAN FORSMAN & JOSEF JAKOBSON

Supervisors: Martin Hilgendorf & Vinh Ngo, Department of Computer Science and Engineering
Advisor: Carl-Magnus Wall, Volvo Group
Examiner: Marina Papatriantafilou, Department of Computer Science and Engineering

Master's Thesis 2026
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Gothenburg, Sweden 2026

Detection and Adaptation of Staggered Concept Drift in Federated Online Learning
A Dynamic Cluster-Based Framework for Machine Learning with Distributed Heterogenous Data Streams
FABIAN FORSMAN & JOSEF JAKOBSON
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

Federated Learning (FL) enables collaborative model training across distributed edge devices without requiring the exchange of raw data. However, most FL approaches assume stationary data distributions, whereas real-world environments often exhibit concept drift, where the relationship between input data and target variables changes over time. In federated settings, such drifts may occur independently across clients, creating staggered concept drift that can lead to model poisoning, degraded predictive performance, and ineffective aggregation. This thesis investigates how staggered concept drift can be automatically detected and adapted to in a federated online learning setting under the computational constraints of edge devices. A drift-aware FL framework is proposed in which clients continuously train local online models, monitor prediction errors using lightweight drift detectors, and perform local adaptation when drift is detected. To prevent aggregation of models representing different concepts, a server-side clustering mechanism dynamically groups clients according to their current concepts and maintains separate cluster models for aggregation. Additionally, the framework is evaluated on the computational cost of the edge-level algorithms, the accuracy of the detectors and performance of the global models.

Keywords: Concept drift, federated learning (FL), online learning (OL), machine learning (ML), computer science

Acknowledgements

We would like to thank our supervisors Martin Hilgendorf and Quang Vinh Ngo, as well as our examiner Marina Papatriantafilou for their incredible and continued support throughout the entire process. The discussions during our meetings were crucial to making this thesis a possibility. We also thank our advisor Carl-Magnus Wall from Volvo Group for his invaluable guidance and encouragement. We thank him and Volvo for giving us the opportunity to write the thesis in the first place and for involving us in meetings regarding real-world applications of the thesis domain. Lastly, we want to thank our peers in the thesis group who provided useful feedback during the reviews.

Fabian Forsman & Josef Jakobson, Gothenburg, 2026-06-30

Contents

List of Figures

List of Tables

1

Introduction

The increasing deployment of sensor-equipped systems has enabled the continuous collection of large-scale operational data from complex environments. Such data have become important for statistical and machine learning methods that support predictive analytics and automated decision-making across a broad range of domains. In transportation, modern heavy-duty vehicles continuously generate telemetry streams of vehicle operation, environmental conditions, and energy consumption. Leveraging these data streams for predictive modelling offers substantial opportunities for improving fuel efficiency, reducing operational costs, and enabling more sustainable transportation systems. Many machine learning systems assume a centralized training setting in which data is collected and processed at a single location. In large-scale vehicle fleets, however, transmitting raw sensor measurements from individual vehicles to centralized server introduces substantial communication overhead and storage costs, while also raising concerns related to privacy and data ownership. These limitations motivate distributed learning approaches capable of exploiting decentralized data without requiring direct access to all of it in one place.

Federated Learning (FL) is as a promising framework for distributed machine learning in such settings. In FL, multiple edge devices collaboratively train a shared model while retaining training data locally [1]. Instead of transferring raw data, participating clients periodically communicate model updates to a server, which aggregates the local models into a global model. By decoupling model training from centralized data collection, federated learning enables large-scale collaborative learning while reducing communication costs and mitigating privacy risks. In transportation systems, sensor data are generated continuously rather than collected as static datasets. This setting motivates the integration of federated learning with online learning, where models are updated incrementally as new data become available [2] and is particularly suitable for streaming environments.

Challenges. A key challenge in online learning is that data distributions are rarely stationary over time. In long-haul transportation, changes in weather conditions, road topology, traffic patterns, vehicle load, or driving behaviour may alter both the statistical properties of sensor measurements and their relationship to predictive targets. Changes in this relationship is referred to as *concept drift* [3]. If left unaddressed, concept drift can significantly degrade predictive performance by rendering previously learned patterns obsolete.

Handling concept drift becomes considerably more challenging in federated online settings. Data evolution may occur asynchronously across distributed clients, with drifts emerging at different times and under different local operating conditions. Consequently, drift may remain localized to individual vehicles rather than appearing globally across the fleet. Existing federated learning frameworks are not inherently designed to account for such heterogeneous and asynchronous distribution shifts, creating the need for a framework that performs drift-aware adaptation across both local and global concept drift.

Thesis Objective. This thesis addresses these challenges by developing a framework for online federated learning under concept drift, with particular emphasis on asynchronous drift across clients and time. The proposed framework combines techniques from federated learning, online learning, and concept drift detection to enable automatic detection and adaptation of staggered concept drift in distributed streaming environments.

Thesis Contributions. The primary contribution of this thesis is the design and implementation of a drift-aware federated online learning framework that integrates lightweight client-side concept drift detection with a server-side clustering mechanism to separate clients according to their current concepts and thereby reduce model poisoning caused by staggered drift. This framework combines existing concept drift detectors and dynamic cluster management into a unified architecture suitable for computationally constrained edge devices. Furthermore, this thesis introduces an evaluation methodology for analysing such systems by jointly considering detector performance, adaptation quality, computational overhead, cluster behaviour, and predictive performance. The proposed framework is evaluated through a series of controlled experiments spanning multiple datasets, drift types, and detector configurations, providing an analysis of the interactions between drift detection, adaptation, clustering, and federated aggregation under asynchronous concept drift.

2

Background

This chapter introduces the theoretical foundations for the framework developed in this thesis. It begins with the basics of machine learning and model training, followed by online learning and the challenges it introduces. The concept of concept drift is then formalized and its detection and adaptation strategies are surveyed. Finally, federated learning is introduced along with the specific complications that arise when concept drift occurs in a federated online setting.

2.1 Machine Learning Models

Machine Learning (ML) is a technique where a statistical model is trained to predict outcomes based on a set of inputs [4]. More formally, it is defined as follows. Given a pair of input features $\mathbf{x} \in \mathbb{R}^d$, try to predict the target value $y \in \mathcal{Y}$. For instance, $\mathbf{x}$ could be the set of sensor data from a truck in Sweden on the 14th of February at a specific timestamp and y is the true energy consumption at said time.

There are a multitude of different ML structures, however this thesis will focus on the *multi-layer perceptron* (MLP) which is an architecture defined as a fully connected sequence of *layers* containing *perceptrons* (commonly referred to as *neurons*) [4]. Each neuron has a set of weights $\mathbf{w}$ which define the estimated function. More explicitly, an MLP works as follows:

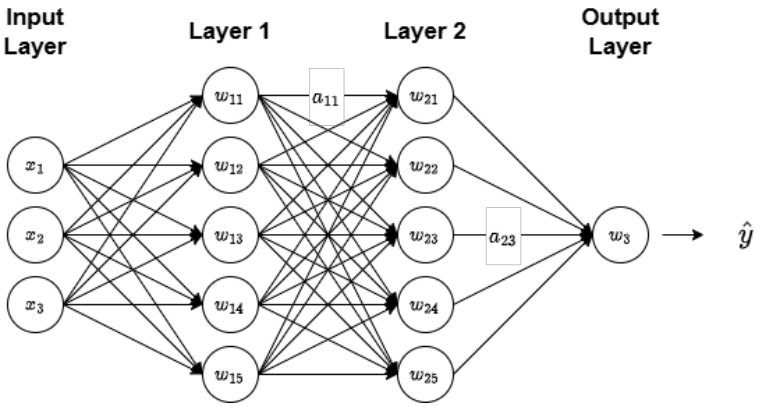


Figure 2.1: A 3-layer MLP.

A neuron i in layer l performs the dot product $\mathbf{w}_{li}^\top \cdot \mathbf{a}_{l-1} = a_{li}$, where $\mathbf{a}_{l-1} \in \mathbb{R}^{l-1}$ is a vector of all outputs from the previous layer and $\mathbf{w}_{li} \in \mathbb{R}^{l-1}$ is a vector of the

weights related to that neuron. For example, Figure 2.1 displays a 3-layer MLP which takes an input of three features and outputs a single value, where:

$$a_{11} = \mathbf{w}_{11}^{\top} \cdot [x_1, x_2, x_3]$$

$$a_{23} = \mathbf{w}_{23}^{\top} \cdot [a_{11}, a_{12}, a_{13}, a_{14}, a_{15}]$$

$$\hat{y} = \mathbf{w}_{3}^{\top} \cdot [a_{21}, a_{22}, a_{23}, a_{24}, a_{25}]$$

To simplify the definition, an MLP model can be defined as a function

$$F(\theta; \mathbf{x}) = \hat{y}$$

where θ is the set of all weights of all neurons. The goal is to iteratively update these weights to increase the accuracy of this estimation, meaning the model output $\hat{y}$ should be as close to y as possible across all possible combinations for $\mathbf{x}$. ML is a broad field, but to keep to this section focused and relevant to the thesis it will only be discussed in terms of MLP *classification* models. Classification refers to the problem of identifying the *class* or *label* of the input sample. In the training data, each sample has a discrete label as the target value y out of a set possible labels, and the model output $\hat{y}$ is a set of probabilities that the sample belongs to each label.

2.2 Training a Model

In traditional ML (henceforth referred to as *Offline* ML), a model is typically trained by giving it a single complete dataset, which is assumed to be independent and identically distributed (i.i.d)., containing data pairs $(\mathbf{x}, y)$ which it repeatedly uses until the model converges to an acceptable level of accuracy [4]. The training mechanism relies on the model making predictions $\hat{y}$ using the input features $\mathbf{x}$ to compute loss $\mathcal{L}(y, \hat{y})$. This loss is then used to update the model weights by computing the gradient $\nabla_\theta \mathcal{L}$ for every weight $\mathbf{w} \in \theta$ in the entire network and performing an optimization step:.

$$\theta \leftarrow \theta - \eta \nabla_\theta \mathcal{L}(y, \hat{y})$$

for some *learning rate* η. This optimization process is called gradient descent (GD) and is the gold standard for training ML models[4]. The direction of the gradient is always towards some minimum, but due to the potentially non-convex nature of the loss function w.r.t. the MLP, it is not necessarily the global minimum. This means that there may exist several basins with local minima, and which one is converged to depends on the initial weights [5].

This is typically done in *batches*, meaning that the dataset is partitioned into collections of samples, using one batch at a time to compute the average loss for an optimization step. This is then repeated a certain number of times, meaning that the one dataset is reused multiple times to further optimize the model as it gets better and better [4]. Another metric relevant to classification models is the *prediction error*, which is simply 1 if $\hat{y} \neq y$ and 0 otherwise. While not part of the

optimization process, it is a metric to determine the performance of a model and is more interpretable than raw loss values.

The loss function varies depending on the task for which the model is designed, but in general it is some sort of dissimilarity measure between what was predicted and the true value. For binary classification models (binary meaning that there are only two classes to choose from) the dissimilarity between $\hat{y}$ and y is computed using Binary Cross Entropy (BCE):

$$\mathcal{L}(y, \hat{y}) = -\frac{1}{N} \sum_{i=1}^{N} [y_i \log(\hat{y}) + (1 - y_i) \log(1 - \hat{y})] \tag{2.1}$$

where N is the number of samples per batch.

2.3 Online Learning

Offline ML requires that the full dataset is accessible at time of training, which may not always be feasible in practice. Limitations in storage, or the fact that the data is recorded in real time, means that the model cannot be trained on all of the data at once. Additionally, offline methods introduce scalability issues since updating such a model with new data means full or partial retraining, which is expensive. To compensate for these restrictions, new training methods were designed which centred around having a constant stream of individual data samples being fed to the model one by one. These methods are encompassed under the ML subfield of *online learning* (OL) [6], [7], [8].

When the data arrives to the model as a continuous stream and without the option to wait and save all of the data on device, the aforementioned batch training is clearly not possible. Instead, OL training methods aim to rely only on the current state of the model weights and the next incoming data point (although in certain cases a small batch of the most recent data points are kept) to perform optimization [7], [8]. The training mechanisms vary between implementations, however gradient descent is still common and alternate methods such as parameter-based or higher order algorithms [6], [8] will be omitted in this thesis. Online Gradient Descent (OGD) works similarly to its offline counterpart, with the limitation that the loss is computed over one or very few samples instead of larger batches.

2.4 Concept Drift

Concept drift refers to changes in the data-generating process over time, violating the assumption that data is i.i.d. [9], and is an ongoing issue with online learning [6]. These issues are a natural consequence of distributions and relationships in data in dynamic systems changing over time due to factors not considered by the ML model [10]. This *heterogeneity* in data results in the models *drifting* and needing to learn new patterns. In turn, previously learned patterns may lose their relevance, leading to poor model performance [10].

2.4.1 Setting and Definition

A *classification* is described by the prior probabilities of classes $p(y)$ and the *class conditional probability density functions* $p(\mathbf{x}|y),\ y = 1, \ldots, c$, where c is the number of classes. For class y, the *classification decision* $p(y|\mathbf{x})$ is made according to the posterior probabilities of the classes, following Bayes' rule in Bayesian Decision Theory [3]:

$$p(y|\mathbf{x}) = \frac{p(y)p(\mathbf{x}|y)}{p(\mathbf{x})} \tag{2.2}$$

Formally, a concept is defined as:

$$p(\mathbf{x}, y) = p(\mathbf{x}) \times p(y|\mathbf{x}) = p(y) \times p(\mathbf{x}|y) \tag{2.3}$$

where p denotes the joint distribution between the input features $\mathbf{x}$ and the target variables y [3] i.e., the distribution of $\mathbf{x}$ and y pairs. Concept drift is defined by the change in the joint distribution between time points t and $t + 1$:

$$\exists \mathbf{x} : p_t(\mathbf{x}, y) \neq p_{t+1}(\mathbf{x}, y) \tag{2.4}$$

Concept drift can happen in three ways:

(i) $p_t(\mathbf{x}) \neq p_{t+1}(\mathbf{x})$ while $p_t(y|\mathbf{x}) = p_{t+1}(y|\mathbf{x})$ [11]. This is *virtual drift*, also known as *covariate shift*, which describes situations in which the input distribution $p(\mathbf{x})$ changes while the conditional distribution $p(y|\mathbf{x})$ remains unchanged. Although virtual drift does not necessarily invalidate the optimal classifier, it may still influence data coverage, confidence estimates, and the stability of learning algorithms [3].

(ii) $p_t(y|\mathbf{x}) \neq p_{t+1}(y|\mathbf{x})$ while $p_t(\mathbf{x}) = p_{t+1}(\mathbf{x})$ [11]. This is *real concept drift*, which refers to changes in the conditional distribution $p(y|\mathbf{x})$, implying that the relationship between inputs and targets has altered and that the previously learned decision boundary may no longer be valid. This type of drift may occur with or without changes in the input distribution $p(\mathbf{x})$ [3].

(iii) Combining the two main categories of concept drift, i.e., virtual and real, results in: $p_t(\mathbf{x}) \neq p_{t+1}(\mathbf{x})$ and $p_t(y|\mathbf{x}) \neq p_{t+1}(y|\mathbf{x})$ [11].

The drifts are symmetrical in $p(y)$ and $p(\mathbf{x}|y)$. When data changes, the probabilistic relationships may change: class priors $p(y)$, class-conditional densities $p(\mathbf{x}|y)$, and posterior probabilities $p(y|\mathbf{x})$ *may* all be affected [3]. A classifier, for example an ML model, is trying to learn the relationship between input features $\mathbf{x}$ and target variables y. It is learning the conditional distribution $p(y|\mathbf{x})$.

For example, consider a fleet of trucks where sensor measurements such as speed, engine load, road gradient, and temperature are used to predict energy consumption. Virtual drift, i.e., change in $p(\mathbf{x})$, occurs when the distribution of sensor measurements changes while the relationship between sensors and energy consumption remains unchanged, for example when the fleet is assigned to different routes

with steeper terrain. A change in the input distribution $p(x)$ or the prior probability of classes $p(y)$ occurs when certain operating conditions become more frequent, such as an increased proportion of heavily loaded transports, leading to higher energy consumption without altering the underlying vehicle behaviour. Real concept drift arises when the relationship between sensor readings and energy consumption changes, i.e., the posterior probabilities $p(y|\mathbf{x})$, such as after engine software updates, hardware degradation or driver patterns change. In this case, the model which trained on a previous posterior may no longer be valid.

2.4.2 Understanding concept drift

A useful and common way to reason about concept drift is to decompose it into three fundamental questions: *when* the drift occurs, *how severe* the drift is, and *where* in the input distributions $p(x)$ or $p(y)$ the drift occurs. In other words, these factors describe the temporal behaviour of drift, the magnitude of change between concepts, and the regions of the feature space that are affected [11].

When. When a concept changes is the most basic metric. Formally, concept drift occurs when the joint distribution of data and labels changes between consecutive time steps, i.e., recall Equation 2.4: $p_t(\mathbf{x}, y) \neq p_{t+1}(\mathbf{x}, y)$. Accurate timing is important, since delayed detection can cause a prolonged degradation in predictive performance. Drift is rarely instantaneous in real systems [11]. Instead, it may have a start time, a transition period during which the distribution gradually changes, and an end time when the new concept stabilizes.

Severity. The severity of the drift describes how different the new concept is from the previous one. It can be formalized as a discrepancy measure

$$\Delta = \delta(p_t(\mathbf{x}, y), p_{t+1}(\mathbf{x}, y)), \tag{2.5}$$

where $\delta(\cdot)$ quantifies the cumulative *distance* between the joint distributions [11]. These distances include the relative entropy of the distribution [12] and confidence levels of sampling from normal distributions [13]. Higher values of Δ indicate stronger drift.

Where. This correspond to subsets of the feature space in which the new and old concepts differ significantly [11]. Intuitively, these are the areas where the decision boundary, class distribution, or data density have changed. The possibility of localization of drift depends strongly on the underlying data representation and detector [11]. Knowledge of drift regions supports targeted adaptation strategies, such as preserving models in stable regions, prioritizing retraining in unstable regions, and distinguishing true novel behaviour from noise or obsolete data [11].

2.4.3 Types of drift

Independent of the where in the data the drifts happen, i.e., virtual or real drift, there are several ways of *how* concept drift occurs, as can be seen in Figure 2.2.

Sudden/abrupt drifts are quick changes, such as a sensor failing. A challenge for concept drift detection is to differentiate between true drifts and outliers or noise [3]. *Gradual* and *incremental* drifts include long-term changes [3]. Lastly, *reoccurring* is a result of concepts re-appearing in a similar fashion. Seasonal or cyclical changes, such as price patterns in electricity markets, are examples of this. In practice, all these types of drift may occur simultaneously.

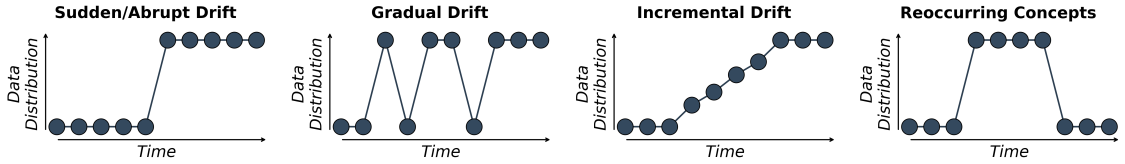


Figure 2.2: Patterns of data distribution changes over time, resulting in different types of concept drift.

2.4.4 Concept Drift Detection

Concept drift detection algorithms are a core component of learning under non-stationary conditions and are typically structured around a general four-stage framework [11], as can be seen in Figure 2.3. The first stage, *data retrieval*, partitions the incoming stream into batches or *windows* of batches. A window $W_{i,n}$ denotes a sequence of a stream of objects $x_1, x_2, \ldots$ ending at x_i of size n: $W_{i,n} = (x_{i-n+1}, \ldots, x_i)$. Since individual samples do not contain sufficient information to characterize changes in distribution, organizing data into windows enables a more reliable estimation of distributional properties [11]. In the second stage, *data modeling*, these saved batches in the window can be summarized or sketched to emphasize features of the data that are most likely to reflect meaningful changes. This step is optional and often used when dimensionality reduction or feature extraction is required to meet real-time processing constraints [11]. It is also used to sketch distributions rather than saving historical data.

Once the data modeling is prepared, the third stage, *test statistics calculation*, quantifies the dissimilarity between the reference distribution (e.g., an earlier window) and the current distribution. This can be done using statistical distances, such as measures of divergence as described in Equation 2.5. Choosing an accurate and robust measure of dissimilarity is challenging and essential to effective drift detection [11]. In the final stage, *hypothesis testing*, the calculated dissimilarity measure is evaluated by determining whether the observed change is significant or could be explained by random variance. Without this assessment, raw differences in distributions cannot be reliably interpreted as drift rather than noise [11].

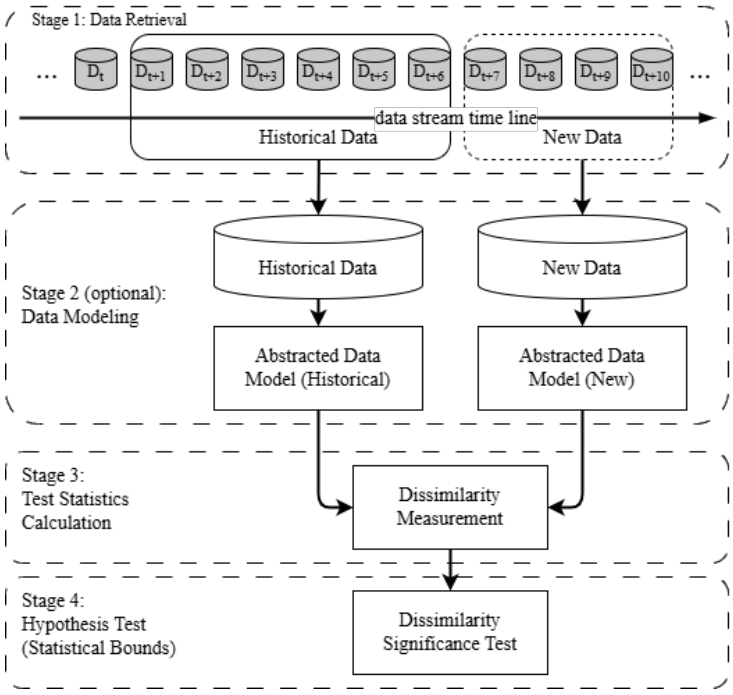


Figure 2.3: An overall framework for concept drift detection in streamed data on the edge [11].

Concept drift detection algorithms aim to identify statistically significant changes in a data stream that may invalidate an already deployed predictive model. Existing methods are commonly classified according to the type of test statistics they employ, namely *data distribution-based detection* for virtual drift, *error rate-based detection* for real drift, and *multiple hypothesis test-based detection* for both [11].

2.4.5 Detecting Real Concept Drift $p(y|\mathbf{x})$

Error rate-based drift detection constitutes the largest class of drift detectors [11]. These algorithms monitor the online prediction error and raise a warning or drift alarm when the change in error becomes statistically significant enough to indicate changes in $p(y|\mathbf{x})$ [11]. Since real drift corresponds to changes in the relationship between inputs and targets, detectors typically rely on signals derived from model outputs, such as prediction errors or loss values [11].

In online supervised settings, the most straight forward approach to detect concept drift is to monitor the prediction loss or error rate over time. A sliding window $W_{i,n}$ can then be defined over recent loss values, analogously to the windowing in subsection 2.4.2. A sustained increase in loss or the variance of loss provides evidence that $p_t(y|\mathbf{x}) \neq p_{t+1}(y|\mathbf{x})$, indicating that the learned decision function no longer reflects the true underlying mapping.

2.4.6 Concept Drift Adaptation

Adaptation algorithms for learning in non-stationary environments are commonly categorized as *active* or *passive* approaches [14]. Both aim to maintain a predictive model under concept drift, but they differ fundamentally in how and when adaptation is performed.

Active approaches follow a detect-and-react structure, meaning any adaptation processes that uses a detector. They are particularly effective in settings with abrupt drift and in online learning scenarios, where timely detection enables fast model adjustment [14]. In contrast, passive approaches do not attempt to explicitly detect drift. Instead, they assume that the data distribution may change at any time and continuously update the model, regardless of whether drift is present. Passive adaptation is therefore inherent in the learning process and is not triggered by a separate detector [14]. Additionally, the literature commonly groups drift adaptation strategies into three main categories: *retraining* new models and *ensemble-based* methods, which are both active approaches, and the passive approach *incremental* model adjustments [11].

The simplest adaptation is to discard the existing model and retrain it on recent data, assuming previous learned model is no longer valid. The selection of recent data is determined by the detector. Sliding-window detectors directly provide a window of samples for retraining, while warning-based detectors buffer incoming samples after a warning signal and use the accumulated data once drift is confirmed.

Ensemble-based adaptation maintains a collection of models rather than a single learner, and each model typically captures one concept or time period. This adaptation is especially effective for recurring drift, where previously observed concepts reappear over time. Instead of retraining from scratch, older models can be reactivated, reducing training cost, and improving response time [11]. The main limitations of ensemble methods are increased computational and memory requirements compared to data distribution-based and error rate-based methods, which only use one model.

Instead of full retraining, passive approaches integrate adaptation directly into the learning algorithm by expanding the capacity of the model or introducing forgetting mechanisms, allowing the model to gradually accommodate new concepts [11]. Passive methods also include single-classifier models that incrementally update their weights, i.e., regular online learning, and commonly increase or decrease the learning rate whenever drift occurs [14]. They are used because although active retraining-based strategies are simple and broadly applicable, they can be computationally expensive because of having to retrain multiple new models and discard potentially useful historical knowledge, particularly in environments where concepts recur.

However, the full-retraining approach on only the most recent data is not always suitable for purely virtual drift, where the underlying decision boundaries remain unchanged. In such cases, previously learned knowledge is still valid, and discarding it by training a new model from scratch leads to unnecessary loss of useful information and slower adaptation. Instead, reusing and updating the existing model can be

more efficient, as it leverages prior knowledge to adapt more quickly to changes in the data distribution [15]. Casado et al. [10] propose a solution which involves storing collections of samples of each concept that it has observed historically. Whenever drifts are detected, the first couple of samples are saved and added to a long-term memory, after which the model trains on the full memory. This solution ensures that the model learns from every possible distribution in the input features, improving generalization.

2.5 Federated Learning

Section 2.3 highlighted some shortcomings of the traditional offline approach of machine learning, such as the problem of dataset accessibility. However there may be additional issues which impose restrictions on the design of an ML system. When training an ML model, it is not always the case that the data is accessible on a single central device. Instead, the data may be spread across many different devices, which leaves two options; either all of the data is sent to where the model is to be trained, or a completely different approach must be adopted which lets the data stay at their respective devices. The former may not be ideal due to communication costs, security and privacy concerns or storage limitations. To solve this issue, a new subfield of ML called Federated Learning (FL) was introduced [16].

FL is an ML system where structurally identical models are trained separately across several devices (often referred to as edge devices, nodes or clients) using only the data accessible at their respective device. After some training, the model weights are sent to a central server, which aggregates the weights into a single model [16], [17]. The idea is that this aggregated model, known as the central or global model, has now indirectly been trained on all of the data by the aggregation process. There are multiple different aggregation algorithms, the most common being `FedAvg` [16] which takes all of the edge models' weights $\theta_1, \theta_2, ..., \theta_n$ and averages them weighted by their respective fraction of data on which they each trained.

$$\theta^* = \sum_{i=1}^{n} \frac{|D_i|}{|D|} \theta_i \tag{2.6}$$

Because of how gradient descent optimizes model weights, the edge models must have the same initial weights; otherwise, the clients weights may not correspond to optimization towards a common minimum, which would cause the average of these weights ending up outside of said basin and the model would never converge [16].

Figure 2.4 shows a sketch of how FL works, with the central model $\mathbf{M}^*$ and the edge models $M_1, M_2, \ldots, M_n$ for n different devices. Each model M_i trains only on its respective dataset D_i and contains its own set of model weights. After some time of training, the weights are sent to the central server and $\mathbf{M}^*$ is the aggregation of the models.

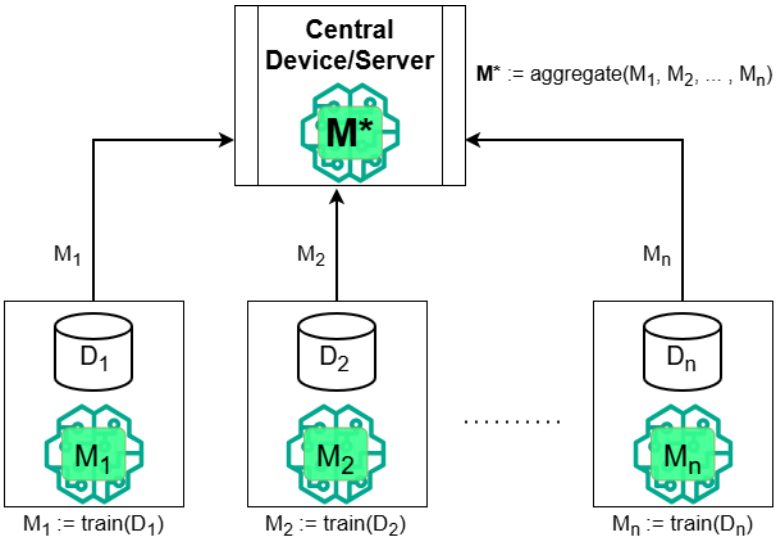


Figure 2.4: Diagram showing general idea of the Federated Learning structure.

In the context of online learning, data arrives as a continuous stream. Therefore, this process is repeated continuously with $\mathbf{M}^*$ being sent to the edge devices to continue training and then back to the server to be aggregated as a loop [10]. The model itself is smaller than the size of the data used for training on the device, thus communication costs are clearly reduced compared to sending data from all the clients to the central server.

The framework which controls the FL system must consider training, communication and aggregation of these edge models in tandem. Each aspect raises its own challenges [17]:

- Training – each edge model must learn effectively from its local data despite limited computational resources and potentially non-IID data distributions. Poorly trained local models can contribute low-quality updates that degrade the performance of the aggregated model.
- Communication – the communication pipelines between the edge devices and the central server must be efficient and not introduce excessive overhead, otherwise the benefits of distributed learning may be outweighed by communication costs compared to centralized training.
- Aggregation – the frequency and timing of aggregation has a significant impact on the quality and stability of the resulting global model.

In addition to these requirements which are prevalent in most FL applications [17], the online setting, the aforementioned concept drift and hardware limitations on the edge devices introduce another layer of complexity [10], [17], [18].

2.5.1 Concept Drift in Federated Learning

Concept drift is a difficult problem in FL because different edge devices may experience concept drift independently, and the drift itself could be either *local* or

global [10]. Local drift refers to concept drift occurring in a subset of edge models, while global drift refers to drift that will occur in all models eventually. This asynchronous occurrence of drift across clients and over time is referred to as *staggered drift* [18]. It is important to note that any global drift will likely manifest as local drift at first, as there is no guarantee that all edge devices will experience drift at the same time.

This raises the following questions: should local drift be accepted in the global model in anticipation of global drift across all clients, or does the new concept actually differ from other edge models and ought to be separated from the current FL network? Jothimurugesan et al. [18] highlight the danger of the first option, since if only local drift has occurred and different clients hold different concepts, accepting this drift into the global model leads to *model poisoning*. This means that the global model has now been trained on two different concepts and performance deteriorates. Conversely, it may be the case that the global concept is indeed drifting but has only been experienced in a few edge models, thus accepting this drift into the global model is the correct decision because it pre-emptively teaches the non-drifted clients the new concept [10], [18].

The issue of asynchronous drift in federated online learning when it is purely *virtual* has been solved by Casado et al. [10], who proposed the adaptation algorithm described in subsection 2.4.6 for purely virtual drift. Their algorithm was specifically tailored for a FL framework with online learning models at the clients. Therefore, the focus of this thesis will be on *real* concept drift, as the problem of model poisoning is much more relevant when it is the decision boundary that is changing across clients and over time.

2.5.2 Clustered FL

Jothimurugesan et al. [18] proposed an ensemble-based approach, called **FedDrift**, of handling concept drift in a federated setting, where different concepts are identified and separate FL networks are created dynamically for each one. The idea is to have several *clusters* of clients with a per-cluster global model (henceforth referred to as a *cluster model*) each of which map onto one of the concepts experienced by clients thus far. When clients drift, they are automatically reassigned to a cluster whose model corresponds to the new concept. If no such cluster model exists, a new cluster is created containing only the drifted client with the client's model becoming the initial cluster model.

Algorithm 1 describes a single timestep of the algorithm, which is run repeatedly until all the data has been used or until convergence. The algorithm is initialized with a single cluster model with all clients assigned to it, with new cluster models being created and the assignments being updated as the algorithm runs. It is important to note that the algorithm does not distinguish between cluster and edge models, rather each client is considered a source of data to which their respective cluster model is sent to train on said data.

At each time step, every client receives a batch of data, and the server sends **all**

Algorithm 1: FedDrift at time τ

Input: # of clients P, # of models M, One-hot encoded client-cluster assignments $w_{c,m}$, Batch of data at clients $\{S_c\}_{c=1}^{P}$

1 $\ell_{c,m}^{(\tau)} \leftarrow$ loss of model h_m on client data $S_c^{(\tau)}$;
2 **for** $c = 1, 2, \ldots, P$ **do in parallel**
3 **if** $\min_m \ell_{c,m}^{(\tau)} > \min_m \ell_{c,m}^{(\tau-1)} + \delta$ **then**
4 Initialize a local model at client c;
5 Add it to the set of global models at $\tau + 1$;
6 Assign client c to its own cluster;
7 **else**
8 $w_{c,m}^{(\tau)} \leftarrow \mathbf{1}\left\{m = \arg\min_{m'} \ell_{c,m'}^{(\tau)}\right\}$;
9 **end**
10 **end**
11 **for** $i, j = 1, 2, \ldots, M$ **do in parallel**
12 $L_{ij} \leftarrow$ loss of model h_i on sample of $\bigcup_{c,t:\, w_{c,j}^{(t)}=1} S_c^{(t)}$;
13 $D(i,j) \leftarrow \max(L_{ij} - L_{ii},\ L_{ji} - L_{jj},\ 0)$;
14 **end**
15 **while** $\min_{i \neq j} D(i,j) < \delta$ **do**
16 MERGE(i, j, D);
17 **end**
18 Train models at respective clients;

models to **all** clients, where the models are evaluated on each batch resulting in model-client loss value pairs (line 1). For each client, the lowest-loss model is selected as a candidate cluster model, and the loss is compared to the lowest loss of the previous timestep (line 2). If this new smallest loss is larger (by some threshold δ) than the smallest loss in the previous time step, a new model is created and the client is assigned to it (lines 4–6). This is the case where a client has drifted to a previously unknown concept. Otherwise, assign the client to the lowest-loss model (line 8).

This entire process runs at all clients in parallel, meaning that if multiple clients drift to the same new concept then multiple models have been created for the same concept. To ensure that there is only one model per concept, a merging algorithm is performed (lines 11–17), the implementation details of which are skipped). The clients then supply their assigned model with all historical data that matches the current concept, and send this updated model to the server where it is aggregated with all other updates from the other clients within the cluster.

2.6 Drift Detection and Adaptation Algorithms

The detection algorithms used in this thesis were selected based on the state-of-the-art in error-rate based drift detection. They are all based on historical model error rate and use statistical estimates and bounds to determine changes in the data stream. The algorithms, their designs and motivations can be found in their respective papers. Adaptation is an *active retraining* algorithm as described in sub-

section 2.4.6 which uses data provided by the detectors to train a fresh model.

Drift Detection Method (DDM) [13] DDM (the pseudocode is provided in algorithm 2) detects concept drift by monitoring the online prediction error rate of the stream of data. For each incoming prediction error of the latest sample or batch of samples, the method estimates the average running probability of error p_i and its standard deviation $s_i = \sqrt{p_i(1-p_i)/i}$ (lines 8–11), where i denotes the number of processed instances. The algorithm maintains the minimum observed value of $p_i + s_i$, corresponding to the historically best performance of the learner (lines 12–15). A warning state is triggered when the current error estimate exceeds this minimum by two standard deviations (lines 20–23), while drift is declared when the deviation exceeds three standard deviations (lines 16–19).

Whenever DDM determines that the error rate is above the warning threshold, the latest sample is added to a *memory buffer* (line 17). This means that while DDM is trying to determine if the concept has truly drifted, it collects samples from what it suspects to be the new concept. When it finally decides that drift has occurred, `Adapt()` is called (lines 30–34), which uses the buffer to retrain the new model.

Early Drift Detection Method (EDDM) [19] EDDM extends DDM by focusing on the distance between classification errors rather than the error rate itself. Instead of monitoring the frequency of errors, EDDM tracks the average distance between consecutive misclassifications and its associated standard deviation. The underlying rationale is that gradual drift often manifests as increasingly frequent errors before the overall error rate rises substantially. The algorithm compares the current estimate of the mean error distance to its historically maximal value, normalized by the corresponding standard deviation. Warning and drift is signalled when this normalized ratio decreases below predefined thresholds. Adaptation is performed identically as with DDM, i.e. samples are collected during the warning period which are then used to retrain a new model after drift is detected.

McDiarmid Drift Detection Method (MDDM) [20] MDDM replaces the normal approximation employed in DDM with McDiarmids inequality [21], thereby providing a concentration bound that does not rely on distributional assumptions. MDDM operates on a sliding window of recent prediction errors rather than running means. A weighted mean of the window contents is computed, with more recent observations assigned larger weights in order to emphasize current performance. Drift is declared when the difference between the maximum observed weighted mean and the current weighted mean exceeds a threshold derived from McDiarmids bound. The variants MDDM-A, MDDM-E, and MDDM-G differ solely in their weighting schemes. MDDM-A employs linearly increasing arithmetic weights, MDDM-E applies exponentially increasing weights to strongly prioritize recent observations, and MDDM-G uses geometrically increasing weights that provide an intermediate emphasis on recency. Adaptation in this case requires that a sliding window of the batches is collected in addition to their respective prediction errors, which is then used for retraining.

Hoeffdings Drift Detection Method (HDDM) [22] HDDM similarly relies on concentration inequalities, but employs Hoeffding bounds to compare error rate statistics over time. HDDM-A uses Hoeffdings inequality to monitor the average error rate directly by partitioning observations into historical and recent segments and testing whether the difference between their empirical means exceeds the theoretical confidence bound. Similarly to DDM and EDDM, HDDM has both a warning and drift signal. HDDM-W extends this approach through exponentially weighted moving averages, assigning progressively greater importance to recent observations while preserving statistical guarantees. By integrating temporal weighting into the estimation process, HDDM-W improves responsiveness to gradual and recurring drift patterns while maintaining robustness to noise. Adaptation is identical to DDM and EDDM.

Fast Hoeffding Drift Detection Method (FHDDM) [23] FHDDM adopts a sliding-window strategy in conjunction with Hoeffdings inequality. The method continuously computes the proportion of correct predictions within a fixed-size window and records the maximum accuracy observed so far. Drift is detected when the decrease between the current accuracy and the historical maximum exceeds the Hoeffding confidence bound associated with the window size. Adaptation is identical to the sliding window-based method of MDDM.

Fast Hoeffding Drift Detection Method Stack (FHHDMS) [24] FHDDMS extends FHDDM's principle by employing two overlapping windows of different sizes simultaneously. Typically, a short window is used to capture abrupt drift quickly, while a longer window provides robustness against gradual changes and transient fluctuations. By combining statistical evidence across temporal scales, FHDDMS theoretically improves adaptability across heterogeneous drift scenarios without substantially increasing computational complexity. The adaptation is similar to the other sliding window-based methods, however the size of the adaptor memory window corresponds to the size of the short detector window.

Algorithm 2: Drift Detection Method (DDM)

Input: Input feature X, True Label y, Prediction error $e_t \in [0, 1]$, Adaptor Memory $\mathcal{M}$
Output: New model θ_{new}

$t \leftarrow 0$
$p_t \leftarrow 0$
$s_t \leftarrow 0$
$p_{\min} \leftarrow \infty$
$s_{\min} \leftarrow \infty$
state $\leftarrow$ normal
Function `Detect()`**:**
  $t \leftarrow t + 1$
  Update running mean error rate: $p_t = p_{t-1} + \frac{1}{t}(e_t - p_{t-1})$
  Update running standard deviation: $s_t = \sqrt{\frac{p_t(1-p_t)}{t}}$
  if $p_t + s_t \leq p_{\min} + s_{\min}$ **then**
    $p_{\min} \leftarrow p_t$
    $s_{\min} \leftarrow s_t$
  end
  if $p_t + s_t \geq p_{\min} + 3s_{\min}$ **then**
    state $\leftarrow$ drift
    $\mathcal{M} \leftarrow \mathcal{M} \cup \{X, y\}$
    return `Adapt()`$(\mathcal{M})$
  end
  else if $p_t + s_t \geq p_{\min} + 2s_{\min}$ **then**
    state $\leftarrow$ warning
    $\mathcal{M} \leftarrow \mathcal{M} \cup \{X, y\}$
    return
  end
  else
    state $\leftarrow$ normal
    $\mathcal{M} \leftarrow \emptyset$
    return
  end
Function `Adapt()`$(\mathcal{M})$**:**
  Create fresh model θ_{new}
  $\theta_{new} \leftarrow \text{Train}(\theta_{new}, \mathcal{M})$
  $\mathcal{M} \leftarrow \emptyset$
  return θ_{new}

3

Problem

This chapter describes the problem and operational requirements that the thesis aims to solve. Additionally, it presents and motivates the metrics of interest that will be used to evaluate the proposed solution.

3.1 Problem Description and Requirements

The goal of this thesis is to design and implement a federated online learning framework with the capability of automatically detecting and adapting to staggered concept drift at both the client and server level. The use case and context of the framework concern the use of onboard sensor readings of heavy duty vehicles for machine learning. The research question is: *How can staggered concept drift be automatically detected and adapted to, in a federated online setting with hardware constrained edge devices?*

A federated online learning system in a dynamic, real-world environment must satisfy several *operational requirements*:

1. It must detect shifts in the underlying data distribution quickly, since a delayed response to the shift can lead to sustained prediction errors on the client and impact other models in the framework. In this setting, timely detection means that drift indicators must be computed online and with low latency during normal vehicle operations.

2. All detection and adaptation algorithms must operate within the limited computational budget available on edge hardware. Heavy-duty vehicles typically host low-power CPUs with restricted memory and storage, which constrains the feasible class of drift detectors to lightweight, streaming-based methods.

3. Data is non-i.i.d. across clients and over time, with potentially significant concept drifts occurring unpredictably and in different directions. As a result, the framework must account for both local drift that affects only specific clients and global drift that emerges collectively across all of them. This has to be done such that poisoning is minimized, i.e. no two clients with different concepts are aggregated with one another.

3.2 Evaluation Metrics

To evaluate the efficacy of the framework, the relevant aspects are how well the detection algorithms react to drifting data, as well as the performance of the global models. Additionally, the computational cost of the edge-level algorithms in terms of how long they take to compute and how much data it needs to save on the device will be evaluated. The evaluation of each aspect will be performed in the context of the full framework. This is so that any potential interplay between the performance of the individual components and the framework as a whole can be analysed. It should be noted that any detection and adaptation algorithms that may be used in this thesis have been studied previously in isolation in some of the following metrics on their own [24].

3.2.1 Drift Detection Metrics

Detection performance evaluates how accurately and quickly a detector identifies concept drift. This includes the ability to detect true drift events with minimal delay and the ability to avoid false alarms caused by natural variability in the data. Detection accuracy is measured in terms of the frequency of true positives, false positives and false negatives, and detection timeliness is measured in terms of *delay*.

A true positive (TP) is recorded when the detector fires for the first time after a drift has occurred. A false positive (FP) is recorded when the detector fires either before any drift has occurred, or after a drift has already been detected, i.e. as a spurious re-trigger within the same drift. A false negative (FN) is recorded a detection is never detected after a drift. A reliable detector should ensure that the adaptation is triggered promptly when necessary while avoiding unnecessary processes due to unpredictable fluctuations in the data stream. The detector accuracy performance is further evaluated using *precision, recall and F1-score*, which are defined based on the number of TP's, FP's and FN's.

Precision Precision measures the proportion of predicted positive samples that are actually positive:

$$\text{Precision} = \frac{TP}{TP + FP} \tag{3.1}$$

Recall Recall measures the proportion of actual positive samples that are correctly identified:

$$\text{Recall} = \frac{TP}{TP + FN} \tag{3.2}$$

F1-score The F1-score is the harmonic mean of precision and recall:

$$\text{F1-score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{3.3}$$

Equivalently, the F1-score can be written directly in terms of TP, FP, and FN as:

$$\text{F1-score} = \frac{2TP}{2TP + FP + FN} \tag{3.4}$$

The detection delay is the number of batches it takes for drift to be detected from the point it started. Let t_d denote the index of the sample where true drift occurred for a client and let $\hat{t}_d$ denote the sample index at which the detector first raises an alarm after the drift has occurred. The detection delay for a client is then defined as

$$\Delta = \hat{t}_d - t_d, \qquad \text{with } \hat{t}_d \geq t_d.$$

3.2.2 Computational Cost Metrics

One of the operational requirements of the thesis is that detectors and adaptation must adhere to the limited computational and memory budget. Computational cost of each algorithm is therefore measured so that they can be compared to one another. The Time-to-Compute (TTC) is the average detection time over all processed batches B, computed as

$$\texttt{TTC}_{\texttt{Detector}} = \frac{1}{|B|} \sum_{b \in B} t_b(\texttt{Detect()})$$

where t_b denotes the execution time for batch b to be processed by any detection algorithm `Detect()`. For adaptation, the TTC computation follows the same formula.

Memory usage is the number of batches used by the adaptation to retrain the new model. For detectors using sliding windows of constant size, the number of historical batches saved is the same number. For the other detectors which use *warning* thresholds to start collecting batches of data, the average number of batches used every adaptation is recorded instead.

3.2.3 Cluster Model Performance Metrics

Classification models can be evaluated using a number of different metrics such as loss, accuracy, F1-score, etc. [25]. The simplest and most interpretable metric is accuracy, which is the measure of how often the model correctly predicts the class of the input sample across the dataset:

$$\text{acc} = \frac{1}{N} \sum_{i=1}^{N} \mathbf{1}(\hat{y}_i = y_i)$$
$$\text{where } \hat{y}_i = F(\theta; \mathbf{x}_i)$$

With FL, the global models are by definition aggregates of their clients. In the online setting with staggered drift, the performance of the entire framework and system is fully represented in the predictive correctness of the cluster models. This

is due to the fact that every component on the client level, such as training, detection, adaptation etc., has direct implications on them through the FL aggregation. For example, poor neural network design, bad detection and/or bad adaptation will lead to poor or conflicting models getting aggregated which will then lead to poisoned cluster models. If no cluster model is able to make accurate predictions on any of the concepts that have existed within the system, then the framework has failed. However, if the cluster models consistently perform similarly regardless of the framework configuration, including those lacking any drift awareness at all, this would mean that the framework is redundant.

To this end, the cluster models will be evaluated using validation data. Validation data is data that the model has not seen during training, and as such is a measure of the actual "learning" capabilities of the model rather than just pattern-matching specific input feature combinations to class labels within the data that it has seen [25].

4

Challenges

This chapter formalizes the challenges that arise when combining federated learning, online learning, and concept drift detection in a resource-constrained edge setting. Although each of these components have been studied individually in the literature, their joint operation introduces compound complexity that motivates the design decisions in chapter 5. The challenges are organized into three areas: the distributed and asynchronous nature of drift in a federated setting, the correctness and efficiency of detection and adaptation on individual clients, and the communication and coordination overhead introduced by the federated architecture. The chapter concludes with an explicit statement of the scope that bound the thesis.

4.1 Drift-Aware Federated Learning

Concept drift in a federated setting introduces challenges that are not present in centralized or single-stream learning. Although each individual component of drift detection, adaptation, and model aggregation can be studied in isolation, their combined interaction under decentralized and asynchronous conditions leads to *compounded complexity*. In particular, the presence of heterogeneous data distributions across clients and the absence of a global notion of discrete time steps result in interdependent challenges, where resolving one aspect often exposes limitations in another. This means that assumptions at one level propagate constraints to subsequent levels of the system design.

4.1.1 Staggered Drift

As described in subsection 2.5.2, there exist algorithms designed to deal with staggered drift. In particular, **FedDrift** (algorithm 1) is a significant source of inspiration. However, there are two major problems with FedDrift when considering their algorithm in the setting of this thesis.

Firstly, in **FedDrift**, the full clustering procedure is performed at every timestep, however this is a burden on a client since it requires evaluating several models locally. If there has not been a drift, there is no point in constantly checking if the clusters are correctly organized. This means that computational resources are potentially wasted on performing the clustering. A solution to this problem is that instead of performing the clustering using a predefined schedule, it could instead only be

run every time a drift is detected. This means that the procedure is only executed when it is necessary, decreasing the amount of computations wasted on pointless processes.

The second major problem stems from the fact that FedDrift relies on identifying concepts within large segments of incoming data in advance, performing the clustering procedure of clients before the data is used for training. In the setting of the thesis, this would require waiting for the clients to save up large collections of data on disk as well as batch training, both of which are contrary to the operational requirements. Instead, the framework will have to rely on the state of the client models and some minimal amount of historical data of the current concept to evaluate the cluster models during the clustering procedure.

4.1.2 Correctness of Detection and Adaptation

When using clustering approaches to manage staggered drift, correctness becomes a challenge. Correctness in this context refers to the ability to accurately detect true drifts and apply appropriate adaptation that improve model performance under the new concept, such that the clustering algorithm has something meaningful to evaluate.

In practice, correctness is difficult to guarantee due to the reliance on limited local data. Detection algorithms must balance sensitivity and robustness, while adaptation algorithms must balance how much historical data to retrain on versus initial model performance after drift has been adapted to. Errors in detection propagate directly to adaptation, resulting in either delayed responses to drift or unnecessary retraining. This has direct implications on the clustering aspect as well. Failed detection means that clients are not assigned correctly before an aggregation. This means that the old cluster model is continuously being poisoned because of the client that has not been re-assigned to its new, correct cluster.

The instantiation of the adaptor directly follows the detector, making adaptation fully dependent on detection. However, even if the detection performs correctly, adaptation may fail. Failing means that the new model does not fully represent the new concept, for example, by training on too few batches. When the concept is not actually represented by the client model by having too low accuracy after adaptation, the clustering does not have anything meaningful to evaluate. This results in clusters being created for incomplete concepts during cluster reassignment.

Furthermore, the choice of hyperparameters for detection algorithms affects the sensitivity of the detector, which is dependent on both the type and severity of the drift. Hyperparameters that are suitable for abrupt and severe drifts may fail to detect gradual drifts in a timely manner, while overly sensitive detectors can generate false positives and trigger unnecessary adaptation. It is unnecessary because the cluster is simply re-evaluated and the client should return to the same one, making no difference in the assignments although a full framework pipeline is instantiated. Since adaptation is only initiated after a detected drift, these configuration choices indirectly influence the quality of the resulting client models and, consequently, the

clustering process. As the framework assumes that client models accurately reflect their underlying concepts, any degradation in detection or adaptation correctness propagates to cluster assignment and aggregation. Therefore, evaluating the clustering mechanism in isolation is difficult, as its performance is inherently coupled to the correctness of the preceding detection and adaptation.

4.1.3 Resource-Constrained Edge Devices

Assuming the correctness of detection and adaptation is infeasible, since algorithms often rely on maintaining sufficient historical data and repeated retraining. This directly conflicts with the constraints imposed by edge devices, where computational power and memory are limited. It is infeasible to store large amounts of data, perform computationally intensive detection algorithms, or frequently retrain models from scratch.

This creates a fundamental trade-off between correctness and efficiency. Methods that aim for high detection accuracy and robust adaptation typically require more computational resources, while lightweight algorithms may sacrifice accuracy performance. For instance, larger sliding windows improve statistical reliability in characterizing the concept of the data, but increase memory usage. More sophisticated detection algorithms improve accuracy at the cost of computation time. In addition, adaptation strategies introduce further resource demands by requiring temporary data storage for retraining. If drift is incorrectly detected or detection warnings trigger unnecessary storing of retraining data, these costs are incurred without the corresponding benefits. As a result, federated OL on edge devices must carefully balance the need for accurate detection and effective adaptation against the practical limitations of the environment.

4.2 Asynchronous Communication

In a federated setting with independent clients communicating with a centralized server, asynchronous communication is expected. Unlike synchronous FL, where updates are aggregated in well-defined rounds, asynchronous systems must continuously handle updates and signals from the clients as they arrive. This removes the need for explicit, top-down coordination across clients, but introduces challenges on the server.

A fundamental challenge is the absence of a shared notion of time across clients. Since each client processes its own local data stream independently, model updates are generated at different points in time and under potentially different underlying data distributions. These model updates are performed *round-based*, meaning that aggregation only occurs after a client has processed a predefined number of local samples or training iterations. Consequently, the notion of an aggregation round is local to each client rather than globally synchronized across the system. Whenever a client triggers aggregation, the server has to force all other clients to stop training and partake in the aggregation, resulting in system-wide communication.

4.3 Information Loss vs. Communication overhead

In online FL, how often aggregations are performed versus how much time should be spent on communication is an important trade-off. Cluster models have to be aggregated relatively regularly, and each aggregation requires some baseline amount of communication between the server and the clients regardless of implementation. This raises the question of how to regulate the time between aggregations such that the global model remains sufficiently up-to-date without incurring excessive overhead. Frequent updates in round-based aggregation may lead to diminishing returns, where small local changes are repeatedly transmitted with little impact on the global model. In contrast, a lack of consistent aggregations may result in poor or even outdated global models, which is a problem if the global models are being used for inference at the server while clients are still training. In the context of this thesis, the issue is more problematic when using an active approach in adaptation. If a client detects a drift long after its previous aggregation, a new model is instantiated and the old model is discarded before it is aggregated again, meaning all the data the model had trained on is lost.

The challenge persists in clustering-based approaches with staggered drift. Since cluster assignments are updated only when drifts are detected, and it will take some amount of batches between a drift occurring and the detector noticing, there exists an inherent delay between a client experiencing drift and the server becoming aware of it. During this interval, the client may continue contributing updates to a cluster model via aggregation when there is a mismatch of concepts, i.e., model poisoning. This means that cluster models are potentially poisoned into being unable to represent a single concept if enough of its clients drift in separate directions without any reassignments occurring. In addition to being a useless cluster; if, at some point, a client drifts into the poisoned clusters' old concept and the clustering procedure is triggered, the poisoned model will never be a candidate for reassignment. This will lead to a new cluster being created for a concept that should already have been represented, thus leading to more cluster models than necessary.

4.4 Scope

The following scope allows the thesis to concentrate on a clearly defined problem: evaluating the feasibility and behaviour of concept-drift-aware federated learning in an edge-based vehicle setting. The thesis investigates how drift detection, adaptation and clustering interact under constrained computational and communication resources.

The thesis therefore implements a prototype framework for evaluating different approaches to handling concept drift in federated online learning. The implementation is designed to capture the core learning behaviour of the system while remaining lightweight enough to run in an experimental edge-computing environment. By concentrating on the learning pipeline itself, the evaluation can more clearly assess the

feasibility and effectiveness of the proposed approach under realistic edge-device constraints.

4.4.1 Model Training

There are numerous methods that exist for training an online model, each with their advantages and disadvantages, and deciding on which one to implement is significant in OL. Known algorithms which specifically account for concept drift are well documented [7], [8], [26], [27]. However, the restrictions of fitting these on lightweight hardware as well as the fact that they must be compatible with a federated setting requires adjustments. Therefore, the framework uses Online Gradient Descent (OGD) as the learning algorithm. OGD is lightweight and widely used in online learning on resource-constrained edge devices. Its simplicity also allows the evaluation to focus on the interaction between concept drift detection, adaptation and clustering, rather than on the behaviour of a particular online optimization strategy. Using a well-understood baseline learner makes identifying differences in performance more directly applicable to the drift-aware federated framework itself.

4.4.2 Framework Design

The framework evaluates a set of drift detection algorithms selected from the literature. These methods show different trade-offs in computational cost, memory consumption, and detection sensitivity. Therefore, the evaluation captures how such design choices influence clustering and federated learning performance. The goal is not to identify a universally optimal detector, but rather to assess how different categories of approaches behave within the proposed framework and what the consequences are.

The framework is implemented as a research prototype intended to evaluate concept-drift in federated learning under realistic edge-device constraints. The implementation therefore focuses on the learning pipeline itself, including local training, drift detection, adaptation, clustering, and model aggregation. This allows the evaluation to isolate the behaviour of the proposed approach while maintaining a manageable system complexity suitable for experimental analysis.

To enable a controlled evaluation of the learning framework, the communication assumes reliable connectivity between clients and the server. Communication is therefore lossless and cooperative, allowing the experiments to focus on the effects of concept drift, adaptation, and clustering rather than on network-level failures, issues and communication delays, etc. This assumption provides a stable setting in which the behaviour of the learning framework can be analyzed without introducing additional variability from the communication.

4.4.3 Focus on real drift

The issue of staggered virtual drift with on-drift detection is already solved [10], as described in subsection 2.4.6 and subsection 2.5.1, and the algorithm does not require re-clustering for adaptation on virtual staggered drift data.

Therefore, the proposed framework will focus on real concept drift, which implies that adaptation strategies are designed under the assumption that changes in the data stream correspond to shifts in the underlying decision boundaries, rather than solely in the input distribution. Alternative approaches that explicitly distinguish between real and virtual drift, or that aim to reuse models across both types of drift, are not explored in this study. Although such methods may improve performance in cases where both virtual and real concept drift occurs, they introduce additional complexity in both detection and adaptation. This choice ensures a more controlled evaluation of the proposed framework, while acknowledging that the findings may not directly generalize to scenarios where virtual drift is prominent.

4.4.4 Meta-Learning

Meta-learning, understood here as the use of external or higher-level contextual information to anticipate, rather than merely react to, concept drift is a powerful tool in OL [28]. Meta-learning is particularly relevant for recurrent or seasonal drift patterns, where changes are not purely stochastic but follow predictable cycles. In the context of heavy-duty vehicles, seasonal effects on energy consumption constitute a typical example. Energy usage during winter months is systematically higher due to engine warm-up, auxiliary heating systems, and increased rolling resistance. A model trained primarily on summer data may therefore exhibit predictable degradation during colder periods.

Incorporating such contextual knowledge would allow the system to anticipate performance degradation and adapt models proactively, for instance by conditioning training or aggregation on temporal context or by maintaining season-specific representations. However, doing so requires explicit modelling of external variables, domain knowledge between contextual signals and data streams, and additional decision logic governing when and how meta-information should influence learning. In a federated setting, this complexity is further amplified by heterogeneity between clients; for example, seasonal effects may differ by geography, usage patterns, or operational schedules. In this thesis, drift adaptation is restricted to reactive methods driven by observed data and model behaviour rather than proactive contextual cues.

5

Approach

This chapter details the main conceptual design of the framework, detailing how different known algorithms for detection, adaptation and federated learning will be used or adjusted to fit the specific setting and operational requirements of the thesis detailed in chapter 3. The framework will consist of several components: model training, drift detection and adaptation on the client, federated learning on the server as well as server-level drift adaptation in the form of clustering, as seen in Figure 5.1

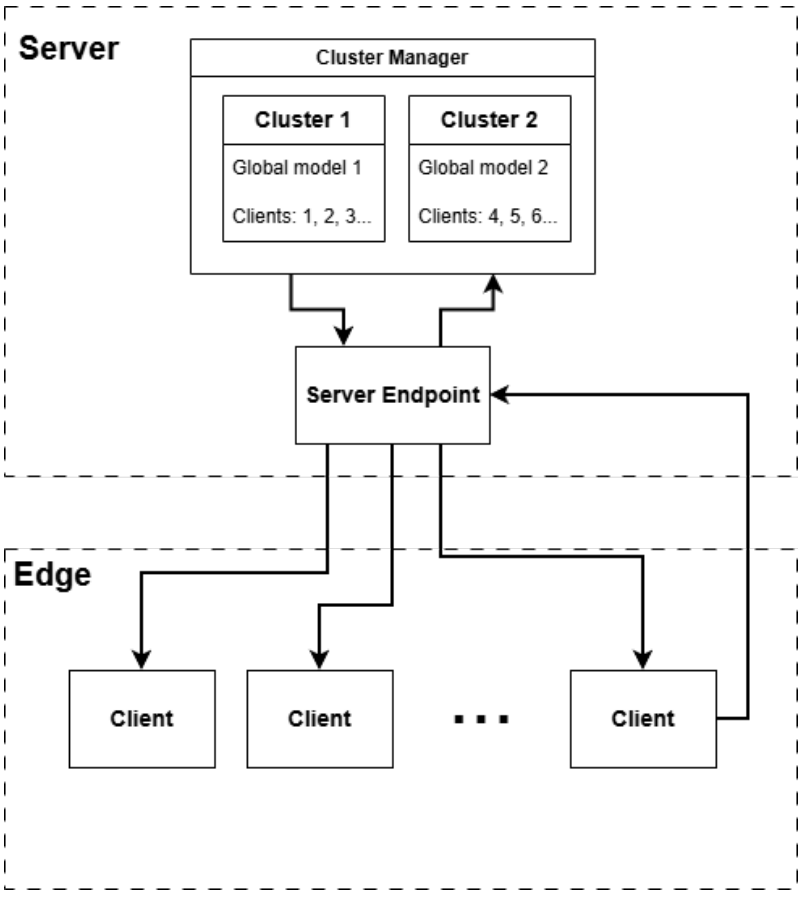


Figure 5.1: Framework Overview.

5.1 Setting

The system is implemented as a prototype using 3 physical devices (Raspberry Pi 5), with multiple *client* instances executed sequentially on each device. This allows for a larger number of edge clients for experimentation than the number of available devices. The implementation is built using the programming language Python. Although lower-level languages such as C++ would provide improved computational efficiency, Python is chosen for its ease of use and the available machine learning and communication modules. As a result, the study does not investigate absolute performance optimizations at the level of programming language or low-level system design, but rather the *relative* time-to-compute of the detection and adaptation algorithms on the same hardware. The network is built using the Python Flask and Requests libraries. Endpoints are set up at a laptop acting as the server and at each Raspberry PI as Flask Apps. The communication itself is performed using HTTP requests to send and receive data between the endpoints.

5.2 System Parameters

The framework is governed by a set of global parameters controlling aggregation, memory, stability, and cluster selection behaviour. These parameters are summarised in Table 5.1.

Table 5.1: System parameters used throughout the framework.

Symbol	Name	Description
T	Aggregation interval	Number of batches processed by a client before requesting model aggregation within its cluster. Controls aggregation frequency in FL. Also referred to as round-based aggregation.
M	Retraining window	The buffer of recent batches for retraining after drift detection. Defines how much historical data is retained for adaptation by the detector.
N	Stability window size	Minimum number of running error observations required before stability evaluation can begin.
W	Evaluation window	Evaluation buffer used for comparing cluster models after drift adaptation with error rate statistics. Populated only after stability is reached.
τ	Stability ratio threshold	Fraction of satisfied stability conditions required over the stability window to declare stable error rate. Higher values enforce stricter stability requirements and thus delay reclustering.
$\mathcal{D}_\phi$	Detector configuration	Set of hyperparameters defining the behaviour of the drift detector, including internal window's size and parameters for detection sensitivity.

In addition to the parameters listed in Table 5.1, each detector configuration $\mathcal{D}_\phi$ may include internal hyperparameters such as sliding window lengths, warning thresholds, and sensitivity controls depending on the specific drift detection algorithm. These are not treated as global system parameters since they are detector-specific and are kept fixed at their default values per experiment configuration, unless stated otherwise.

5.3 Client

The client is responsible for receiving data samples $(\mathbf{x}_i, y_i)$, adaptation, training the model and notifying the server when any server-level actions are required.

The general flow for the client is the following:

Algorithm 3: Client Loop

Input: Local model θ
Drift detector $\mathcal{D}$
Aggregation interval T
Retraining window M
Evaluation window W
Output: Continuously updated local model θ

```
1  repeat
2      Receive data batch (x, y)
3      Compute prediction error e ← 1[ŷ ≠ y] using θ
4      Insert e into detector D's memory
5      if warning signal active from D or D requires it then
6          Save (x, y) to retraining window M
7      end
8      if drift detected by D then
9          θ ← new model trained on M. Clear M
10         repeat
11             Receive new batch (x', y')
12             Save (x', y') to evaluation window W
13             Compute error e' ← 1[ŷ' ≠ y'] using θ
14         until W is full and IsStable(e)*                    // *algorithm 4
15
16         θ ← ClusterSelection(θ, W)*. Clear W                // *algorithm 5
17     else
18         Train θ on sample (x, y)
19         if trained on T samples since last aggregation then
20             Request aggregation within cluster
21             Await aggregated weights from server
22             Replace θ with received weights
23         end
24     end
25 until experiment terminates
```

Algorithm 3 shows the following repeated steps:

1. A batch of data is received by the client (line 2).
2. Use the current model to compute error on the sample (1 if predicted class is correct, 0 otherwise) (line 3).
3. Insert the error rate value into memory (line 4). If detector has previously triggered a warning signal, or always requires it, save the full sample as well (lines 5-6).

4. Perform detection algorithm (line 8). $\mathcal{D}$ outputs a ternary signal: {normal, warning, drift}, where warning enables memory buffering and drift triggers retraining in some detectors. If not detected, skip to step 7. Otherwise perform adaptation until stabilization (lines 10-14).
5. Notify server that drift has been adapted to. Await cluster models for cluster reassignment. Evaluate all cluster models. If matching model exists, send its cluster ID to the server. If no cluster is good enough, notify server to create a new cluster and send current model weights (line 16).
6. Train on the sample (line 18).
7. If the client has trained for a predefined number of samples, call for aggregation within its cluster. Wait until server responds with aggregated model weights and replace current model (lines 19-23).

Detection and Adaptation The detection component is based on the algorithms as they are described in section 2.6, using code from their public GitHub repositories[1] or recreated according to the pseudo-code provided in their respective papers. Adaptation will always entail full retraining of the model, however the data used depends on the detector. After adaptation is triggered, it blocks any aggregation requests from the server and trains until stable.

Memory The memory configuration depends on the detector used in the framework. For example, a detector such as DDM or HDDM mentioned in section 2.6 will instantiate a fresh memory of arbitrary length and fill it with samples after the model passes the warning threshold until it reaches drift. On the other hand, MDDM [20] and others will always have a sliding window it uses for detection. After adaptation is triggered, a sliding window will be created saving samples until stabilization. This is because, for a framework using clustering, some number of recent data must always exist to calculate the error of the cluster models during reassignment. The memory components are implemented to be functionally similar to a python Deque. Inserting a sample while it is full will automatically remove the oldest entry.

Stability For the cluster reassignment or new cluster creation to work as intended, the drifted client's model must truly represent the new concept as discussed in subsection 4.1.1. The ideal case would be to wait until the model performs "well", meaning it has converged in its error rate. However, deciding on a fixed error rate threshold to define convergence, or setting a static number of samples to train on, is non trivial and dataset-dependent. While possible to tune for one concept, this threshold approach becomes problematic when considering the possibility that two concepts may differ in complexity, meaning that a threshold defined for one concept may not work for another. Instead, the readiness of the client in terms of reclustering after drift is determined by its *stability*. An unstable error rate would explicitly mean that the model is unable to represent the concept. Importantly, this does not

[1] GitHub repo for MDDM, HDDM, FHDDMS

necessarily guarantee a low error rate, meaning that the model may still not perform well on the new concept, rather it is some minimum requirement on the model that it should not be in the early stages of learning where it definitely will not perform well. The algorithm can be seen in 4.

Algorithm 4: Stability Checker

Input: Error e
Stability window F of size N
Required ratio τ
Output: Boolean stability decision

1 $p_0 \leftarrow 0$, $s_0 \leftarrow 0$, $n \leftarrow 0$, $p_{min} \leftarrow \infty$, $s_{min} \leftarrow \infty$
2 **Function** `IsStable(`e`):`
3 Update samples of errors count: $n \leftarrow n + 1$
4 Update running mean error: $p_i \leftarrow p_i + \frac{e - p_i}{n}$
5 Update running standard deviation: $s_i \leftarrow \sqrt{\frac{p_i(1-p_i)}{n}}$
6 **if** $p_i + s_i < p_{\min} + s_{\min}$ **then**
7 $p_{\min} \leftarrow p_i, \quad s_{\min} \leftarrow s_i$
8 **end**
9 $f_1 \leftarrow p_i + s_i \leq p_{\min} + s_{\min}$
10 $f_2 \leftarrow p_i + s_i \geq p_{\min} - s_{\min}$
11 Append $f_1 \wedge f_2$ to stability window F
12 **if** $|F| < N$ **then**
13 **return** *False*
14 **end**
15 **return** $\frac{\sum F}{|F|} \geq \tau$

Lines 3–5 maintain running estimates of mean error and variance. Lines 6–8 updates the running minimum p_i and s_i. Lines 9–11 construct a binary stability indicator over a sliding window. Lines 12–14 ensure that enough batches of data have been evaluated to produce N errors. Line 15 computes final binary stability decision based on the ratio threshold τ.

This process is somewhat based on drift detection in DDM [13], where running average error rates and standard deviations are recorded and checked to be within a confidence interval. The stability checker in algorithm 4 introduces two parameters: the window size N, and required ratio τ. Larger values of N will increase the probability that the model has actually *converged* in its error rate and reduce the probability of premature stabilization, but increase memory usage. The parameter τ specifies the fraction of samples in the stability window that must satisfy the convergence criterion before stabilization is declared (line 18). The strictness of the framework directly follows τ, i.e. increasing τ increases strictness as well as the reverse. Empirically, $\tau = 0.8$ worked well because it gave a good balance between client model accuracy within a reasonable number of batches.

In general, the trade-off between strictness and earlier reassignment is about how long the clients are not contributing to the overall framework against how well the clients represent the new concepts. If a client is considered stable too early, it will not represent the new concept properly and increase the risk of creating a new unnecessary cluster during re-clustering. If the client stays too long in this state, it is

essentially the same argument as for how often a client should be aggregated; any detected drifts reset the model, meaning that all the training between two drifts is lost if it was never able to aggregate due to being considered unstable.

Cluster Selection When the drift has been adapted to and the model is stable, the client will receive the weights of all cluster models. Using the samples stored in the stability memory window, the mean p and standard deviation s of the error rate of each cluster model is computed, and the model with the lowest $p + s$ is identified. This error is then compared to the p_{min}, s_{min} computed during the stability process:

$$p + s < p_{min} + 2s_{min} \tag{5.1}$$

If this holds, the candidate cluster model is considered a match, and its ID is sent to the server for reassignment. If not, send the client model weights to the server and instruct it to create a new cluster and reassign the drifted client to it.

Algorithm 5: Cluster Selection and Reassignment

Input: Local model θ;
Evaluation window W;
Set of cluster models $\mathcal{C}$;
Stability statistics $(p_{\min}, s_{\min})$ of local model θ on W
Output: Updated local model and cluster assignment

1 **Function** `ClusterSelection`(θ, W)**:**
2 Receive weights of all cluster models $\mathcal{C}$ from server
3 **foreach** *cluster model* $c \in \mathcal{C}$ **do**
4 Compute mean error $p_c = \frac{1}{|W|} \sum_{(x,y)\in W} \mathbf{1}[c(x) \neq y]$
5 Compute standard deviation $s_c \leftarrow \sqrt{\frac{p_c(1-p_c)}{|W|}}$
6 Compute upper bound error $e_c \leftarrow p_c + s_c$
7 **end**
8 Get cluster model with lowest error $c^* \leftarrow \arg\min_{c\in\mathcal{C}} e_c$
9 **if** $e_{c^*} \leq p_{\min} + 2s_{\min}$ **then**
10 Replace θ with weights of c^*
11 Send cluster ID c^* to server
12 **return** θ
13 **else**
14 Request creation of a new cluster
15 Send local model weights θ to server
16 **return** θ
17 **end**

The reason for using this selection method in algorithm 5 is based on the work done behind DDM [13]. According to the creators of DDM, the upper confidence bound shown in Equation 5.1 represents the warning threshold, i.e. the moment where drift is considered probable (lines 6 and 9). The rationale is that if the cluster model is not considered close to drifting on the most recent data at the client, then it *likely* represents the same concept and the client should be reassigned to its cluster. The stability window size N is shared between the stability checker (algorithm 4) and the evaluation phase after drift in algorithm 5. This ensures that the same last data

from the stability checking is used for cluster evaluation, which should accurately represent the new concept.

It is required that the framework should work on non-convex problems as this is the case with neural networks. As described in section 2.2, two independently trained models can reach sufficient performance on the same data despite having different parameters due to landing in separate basins during the gradient descent optimization process. This has direct implications for aggregation in the case of retraining a model after drift. When a fresh client model has converged to a concept that is already represented by a cluster model, aggregating it may degrade the cluster model due to incompatible weights, despite both models performing well on the same concept. Instead, the new retrained model on the client is replaced by the cluster model. This will lead to some loss of training, but is necessary to avoid breaking FedAvg.

The Model And Training Once the sample has been processed by the steps above, it is used to train the edge model by gradient descent. The model is an MLP neural network built using the PyTorch module[2] and uses the BCE loss function and the Adam optimizer[3] for training which is the standard loss function and optimizer for classification models [4]. The training process and inference is implemented using the module's native functionality.

Aggregation As the client trains, every new sample increments a counter. After the counter exceed a predefined threshold, it sends an aggregation request to the server. Any other clients within the same cluster will be called to send their weights and pause their training until the server sends the aggregate model back. Any clients currently in the process of stabilization after a drift are ignored. After a client is involved in aggregation, their counter is reset, even if it was not the one which called for the aggregation. This makes sure that the same clients are not aggregated several times with minimal updates in the case that there are differences between their individual counters due to latency in the system.

5.4 Server

The server runs an ongoing process that holds all cluster models, keeps track of the client memberships of each, reassigns clients as they drift and runs aggregation to update said cluster models.

Aggregation The framework uses standard Federated Averaging [16] as its aggregation algorithm, which it uses on the client models it receives to compute the cluster model. It then sends the new model back to the clients.

Cluster Reassignment When a client drifts, adapts and has stabilized, the server will receive a call to match the drifted client to a new cluster. The server will send

[2]PyTorch documentation
[3]Adam optimizer

the models of all clusters and receive back either a cluster ID or a set of weights. If the result is an ID, the model changes the mapping of that client to that cluster. If the result is a set of weights, it means that no existing cluster represents the new concept according to the stability and model evaluation algorithms and a new cluster is created, containing only the drifted client and its model weights as the cluster model. The reason for only performing reassignment when a drift is detected stems from the arguments in subsection 4.1.1, where this method saves computational resources by not performing reassignment unless necessary.

6

Evaluation

This chapter describes how the framework is evaluated. It begins by re-introducing the relevant metrics discussed in section 3.2 which will be tracked throughout the evaluation. It then introduces the datasets that will be used to feed the clients of the framework, followed by the descriptions of the different experiments and hyperparameter configurations. The chapter concludes with a guide to reading and interpreting the visualisations used throughout the results chapter.

6.1 Detector and Adaptor Evaluation

Every experiment described in the following sections are evaluated on the same detectors. The choice of detectors was based on the state-of-the-art in concept drift detection literature. Their descriptions are found in section 2.6. They are:

- DDM
- EDDM
- MDDM (A, E, G)
- HDDM (A, W)
- FHDDM
- FHDDMS

The performance of the detector is measured in how accurate its detections are based on the true positives (TP), false positives (FP) and false negatives (FN) together with their aggregate representations precision, recall and F1. Additionally, the delay between a drift occurring in the data stream to the detector noticing is measured by taking the number of batches between these two points. Furthermore, the wall-clock running time it takes for the detector to process a single batch (TTC) is recorded in milliseconds. The precision, recall and F1 are averaged across every client, the delay is averaged across every client and for every *true drift* occurrence. True drift is the ground truth of when the drift occurs in the data stream of a client. The TTC is averaged across every client and for every single execution of a detection (line 8 in algorithm 3). The adaptor is evaluated by the wall-clock running time of creating a new model and training it (line 9 in algorithm 3) as well as how many

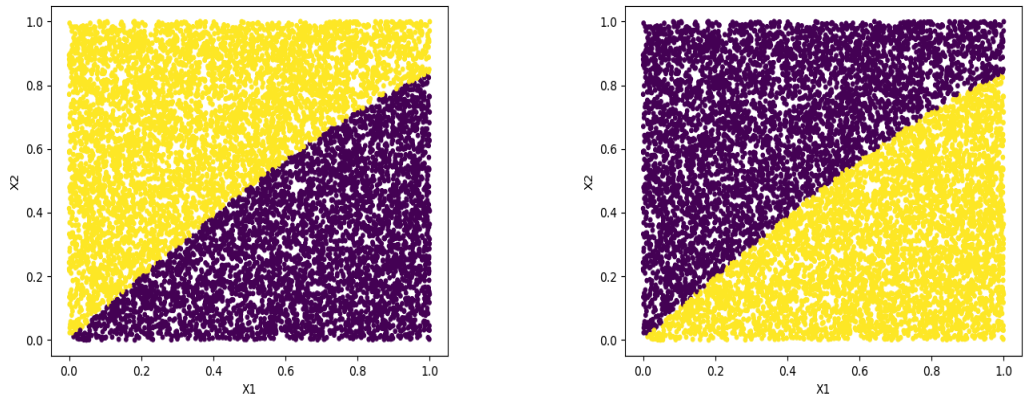
samples were provided to the new model. These are both averaged for every time it was triggered and across every client.

As discussed in subsection 4.1.2, there is a strong relationship between the detector, the adaptation and the clustering which have implications on the framework as a whole. The metrics discussed in this section will be used together with the plots and tables described in section 6.4 to construct an analysis of the results as described in section 6.5. For example, EDDM is not only not in the state-of-the-art, but actually quite poor [20]. It was specifically included to highlight these implications, rather than being a candidate as the ideal detector.

6.2 Datasets

To test the framework, a few different datasets were chosen. They are very common in other studies in the field of concept drift (see [13], [18], [20] among others). Additionally they vary in drift severity, class balance and general complexity, which means the framework can be tested under different conditions to highlight possible strengths and weaknesses.

SINE1 SINE1 (Figure 6.1) consists of two numerical input features $x_1, x_2 \in [0, 1]$ and label $y \in \{0, 1\}$, as well as two concept. In the first concept, if $x_2 > \sin(x_1)$ then $y = 1$ and 0 otherwise. In the other concept, the decision boundary is flipped, i.e. $y = 1$ when $x_2 < \sin(x_1)$ and vice versa. The drift here is as severe as it can get; with the decision boundary reversed, a perfectly trained model will perform with maximum error after the drift, meaning that if the drift detectors are unable to detect the change then the framework will never work for more subtle drifts. This is the simplest dataset that is used as an effective baseline for detectors.

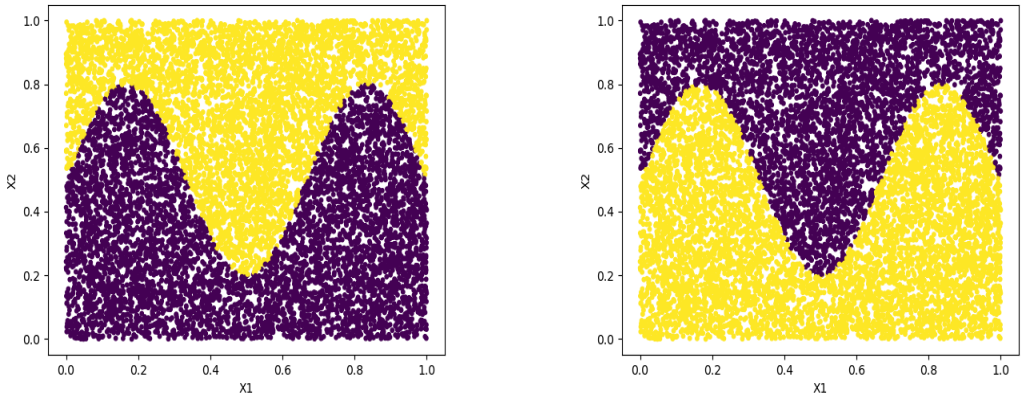


(a) SINE1 - Concept 1

(b) SINE1 - Concept 2

Figure 6.1: The decision boundaries for both SINE1 concepts. Yellow dots are where $y = 1$ and purple for $y = 0$.

SINE2 SINE2 (Figure 6.2) is similar to SINE1 but adjusts the decision boundary to be more complicated. In the first concept, if $x_2 > 0.5 + 0.3\sin(3\pi x_1)$ then $y = 1$ and 0 otherwise. In the other concept, the decision boundary is flipped. The drift situation is identical to SINE1. However, the added complexity makes it more difficult to adjust, which means that the adaptation is also thoroughly tested. Therefore, the drifts in the SINE2 dataset are equally easy to detect as SINE1, but require longer adaptation.



(a) SINE2 - Concept 1

(b) SINE2 - Concept 2

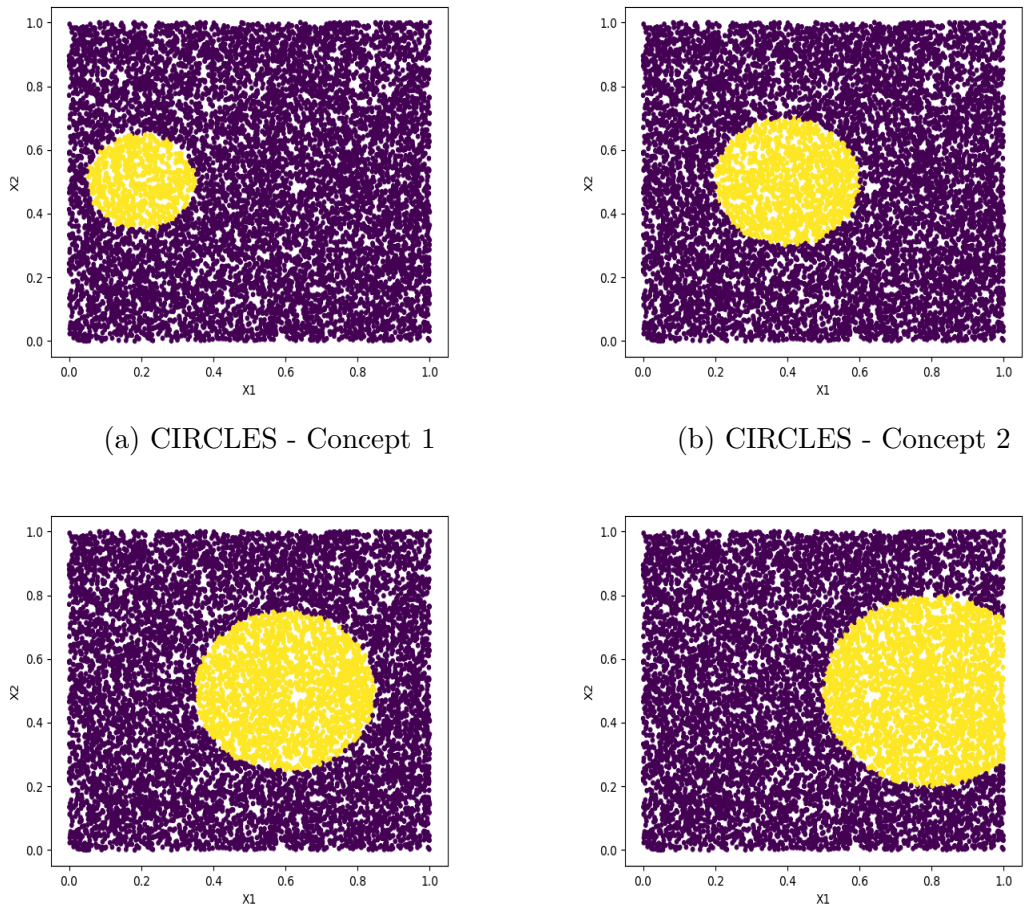
Figure 6.2: The decision boundaries for both SINE2 concepts. Yellow dots are where $y = 1$ and purple for $y = 0$.

CIRCLES CIRCLES (Figure 6.3) is a dataset with 4 concepts which are ordered as seen in the figure. It has the same two input features and label structure, and each concept is defined by a centre coordinate (a, b) and a radius, creating a decision boundary in the form of a circle. The label is 1 if the coordinate defined by the input features is within the circle and 0 otherwise. The center coordinates and radius of each concept is the following:

- Concept 1: Center = (0.2, 0.5), radius = 0.15
- Concept 1: Center = (0.4, 0.5), radius = 0.2
- Concept 1: Center = (0.6, 0.5), radius = 0.25
- Concept 1: Center = (0.8, 0.5), radius = 0.3

This dataset is significantly different from the two previous datasets, namely in the difference between concepts. The decision boundary travels and grows in size across several smaller steps, rather than reversing completely. This makes detection significantly more difficult, since a model that performs well in one concept may still perform decently in the next, meaning that the change in error rate may not be fast or severe enough for detection.

The datasets were implemented in the evaluation by creating specific generators for each, with input features being randomly drawn from a constant normal distribu-



(a) CIRCLES - Concept 1

(b) CIRCLES - Concept 2

(c) CIRCLES - Concept 3

(d) CIRCLES - Concept 4

Figure 6.3: The decision boundaries for all 4 CIRCLES concepts. Yellow dots are where $y = 1$ and purple for $y = 0$. The concepts appear in the order 1, 2, 3, 4.

tion and applying the labelling function to produce the $(\mathbf{x}, y)$ sample pairs. The implementation allows for setting the drift to be abrupt or gradual for some number of samples. During gradual drift, samples are drawn at random from the preceding concept to the impending one, with the probability balance shifting from the former to the latter during the transition period. In addition to creating generators for each dataset, there is the functionality to create a custom dataset, containing individual concepts from each to create tailored sequences of concepts of different lengths for different clients. All of this is to emulate the staggered drift presented throughout the thesis such that every aspect of the experiments can be controlled and true drifts are known during the evaluation. True drifts are therefore defined at generation time by the dataset generators. This ensures that drift detection metrics such as true positives, false positives, false negatives, and detection delay can be computed against a known ground truth. Without this information, it would not be possible to directly evaluate whether a detected drift corresponds to an actual change in the underlying data distribution.

6.3 Experiment Setup

The framework is tested by creating different unique configurations with respect to the datasets/concepts and types of drift. Each configuration is characterized as an *experiment*, with each experiment being run on a number of clients across the three available Raspberry PI's.

Table 6.1: Experiment configurations. **Clients** denotes the number of clients used across all Raspberry PI's. **Concepts** denotes the datasets and concepts used throughout the experiment. **Agg** denotes the maximum number of samples a client trains on before triggering an aggregation within its cluster.

Ex.	Clients	Concepts	Agg.	Data	Drift type(s)
1	15	$A : \texttt{SINE1}_1 \rightarrow B : \texttt{SINE1}_2$	300	Figure 6.4	Abrupt
2	15	$A : \texttt{SINE2}_1 \rightarrow B : \texttt{SINE2}_2 \rightarrow A : \texttt{SINE2}_1 \rightarrow B : \texttt{SINE2}_2$	300	Figure 6.5	Abrupt & Recur.
3	15	$A : \texttt{SINE2}_1 \rightarrow B : \texttt{SINE2}_2 \rightarrow A : \texttt{SINE2}_1 \rightarrow B : \texttt{SINE2}_2$	300	Figure 6.6	Gradual & Recur.
4	15	$A : \texttt{CIRCLES}_1 \rightarrow B : \texttt{CIRCLES}_3 \rightarrow C : \texttt{CIRCLES}_2 \rightarrow D : \texttt{CIRCLES}_4$	400	Figure 6.7	Abrupt
5	9	$A : \texttt{SINE1}_1, B : \texttt{SINE1}_2, C : \texttt{SINE2}_1$ & $D : \texttt{SINE2}_2$. Mixed order.	600	Figure 6.8	Abrupt & Recur.

The number of clients changes from 15 in experiments 1–4 to 9 experiment 5, simply for visualization purposes of the results. Having as many clients in Experiment 5 as the other experiments would reduce the readability of the cluster evaluation plots described in section 6.4.

6.3.1 Experiment 1 – SINE1 Abrupt

Experiment 1 is the baseline evaluation of the framework. Using the SINE1 dataset, which contains only two highly distinct concepts separated by global abrupt drifts, the experiment assesses whether the framework can correctly detect a very severe type of concept drift and reassign clients accordingly. Since the decision boundary is completely reversed after each drift, a previously well-performing model immediately becomes inaccurate, making drift detection relatively straightforward. Therefore, the experiment validates the fundamental functionality of detection, adaptation, and clustering under ideal conditions before continuing with more challenging scenarios.

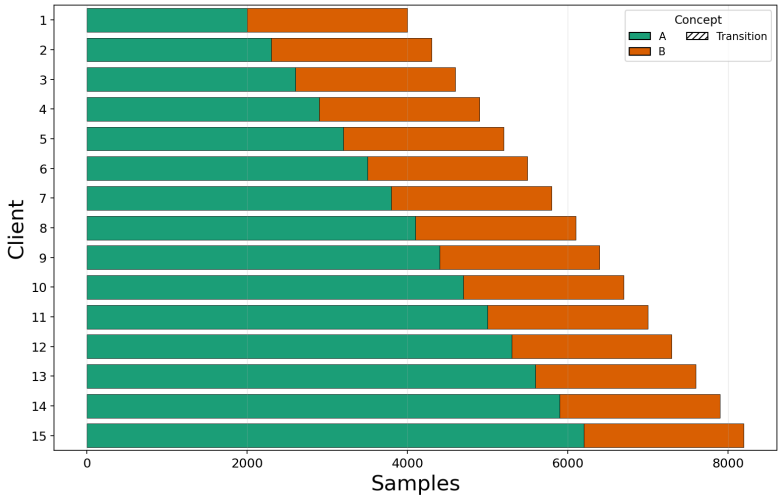


Figure 6.4: Input data in concepts for each client in experiment 1.

6.3.2 Experiment 2 – SINE2 Abrupt & Recurring

Experiment 2 evaluates the framework under recurring abrupt concept drift using the more complex SINE2 dataset. Although the drifts remain abrupt and severe, the underlying concepts contain a more complicated decision boundary, which requires longer adaptation before models regain high accuracy. Furthermore, the experiment investigates whether the framework can correctly identify recurring concepts. This is to ensure that the framework does not create more clusters than necessary when recurring concepts occur.

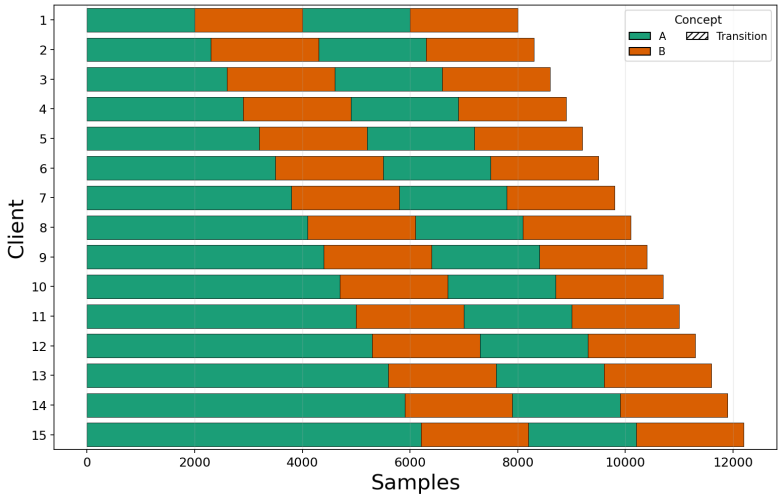


Figure 6.5: Input data in concepts for each client in experiment 2.

6.3.3 Experiment 3 – SINE2 Gradual & Recurring

Experiment 3 extends Experiment 2 by replacing abrupt drifts with gradual drifts. During each transition period, samples are drawn from both the previous and new

concepts. This setting is more challenging for drift detectors because the error rate increases gradually rather than suddenly. The experiment evaluates the sensitivity of the detectors and how the potentially delayed detections affect the clusters.

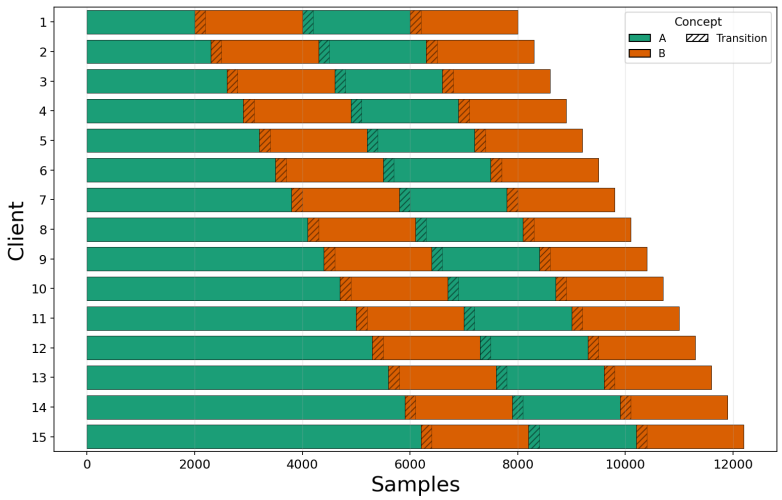


Figure 6.6: Input data in concepts for each client in experiment 3. The transition-time between concepts is 300 samples, i.e. samples are drawn from both concepts randomly during this period.

6.3.4 Experiment 4 – CIRCLES Abrupt

Experiment 4 evaluates the framework using the CIRCLES dataset with a sequence of non-recurring abrupt drifts. Unlike the SINE datasets, concepts remain partially similar because the decision boundary moves and changes size, rather than reversing completely. Consequently, models trained on one concept may still achieve reasonable performance on the next, making drift detection significantly more difficult. The purpose of this experiment is therefore to assess the framework's ability to identify subtle drifts and show the effects of not detecting them.

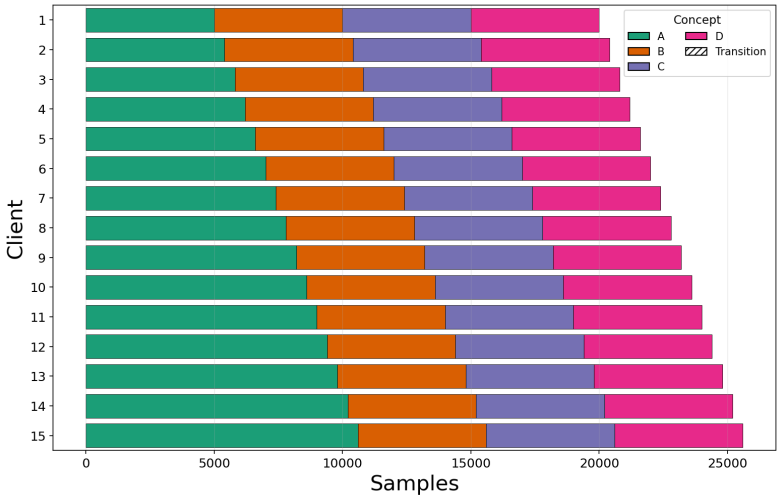


Figure 6.7: Input data in concepts for each client in experiment 4.

6.3.5 Experiment 5 – Mixed

Experiment 5 is the most important experiment and represents the most realistic and challenging evaluation scenario. Multiple concepts from both the SINE1 and SINE2 datasets are distributed across clients in different orders and lengths, creating a heterogeneous federated environment where several concepts coexist simultaneously and clients experience abrupt and recurring drifts independently. The experiment primarily evaluates the clustering component of the framework by testing whether clients sharing the same concept can be clustered despite experiencing different previous concepts. It also evaluates the error propagation of FP, FN and delayed detections and how it affects the clustering.

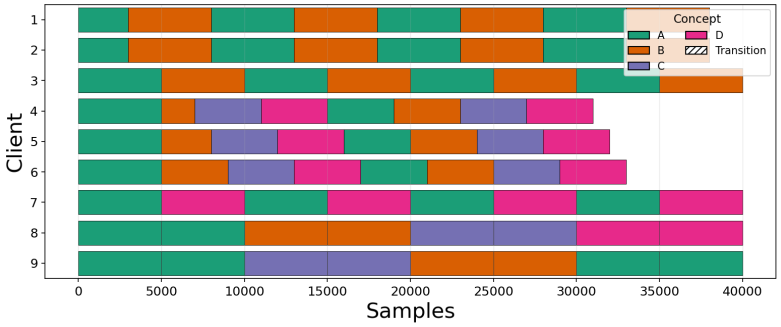


Figure 6.8: Input data in concepts for each client in experiment 5.

6.3.6 Hyperparameters for ML Models

Table 6.2: Hyperparameters for ML models and detectors used in experiments.

Parameter	Experiment 1	Experiment 2 & 3	Experiment 4 & 5
Input size	2	2	2
Hidden size	8	16	16
Hidden layers	1	1	2
Learning rate	0.01	0.01	0.01
Batch size	1	1	1

These model parameters were chosen to balance performance on each dataset and low computational cost. Tests were performed on each dataset to determine the most practical model parameters for use in the real experiments. However, in principle the number of neurons could be scaled up to perform similarly or better depending on how much time the models are given to train. A learning rate of 0.01 has been used previously for these datasets in FedDrift [18]. Since all experiments start with the clients on the same concept, the initial cluster model is trained on the server to save time while performing the experiments.

6.3.7 Hyperparameters for Detectors

For Experiments 4 and 5, detectors MDDM (A, E, G) and FHDDM(S) increased detector configuration $\mathcal{D}_\phi$ parameter δ from 1×10^{-6} to 1×10^{-3} for more sensitive detection [20]. All other detector hyperparameters are default from the original papers. This was done because drifts within CIRCLES and between SINE1 and SINE2 datasets require more aggressive detection, since for example SINE1 concept 1 and SINE2 concept 1, shown in section 6.2, have moderately similar decision boundaries.

Table 6.3: Detector memory configurations. The memory size denotes the number of recent error-rate observations retained by the detector during detection. Detectors without explicit windows only maintain running statistics.

Detector	Saved error rates
DDM	0
EDDM	0
MDDM-A	25
MDDM-E	25
MDDM-G	25
HDDM-A	0
HDDM-W	0
FHDDM	25
FHDDMS	100

Table 6.3 summarizes the detector memory configurations used throughout the evaluation. Detectors such as DDM, EDDM and HDDM maintain running statistics and therefore do not store historical error rates explicitly. MDDM and FHDDM-based methods instead rely on sliding windows of recent prediction errors, requiring memory proportional to the window size. The values shown correspond to the detector configuration parameter $\mathcal{D}_\phi$ introduced in Table 5.1.

6.4 Clustering Evaluation

To present the results of the clustering evaluation, a combination of plots and tables are used. The clustering aspect is evaluated through a set of visualizations that compare the results of the cluster reassignment algorithm described in section 5.3 with the true concepts experienced by the clients. The plots and tables allow for analysis of how the stability and cluster selection algorithms (algorithms 4 and 5 respectively) perform in correctly executing the clustering procedure in terms of if drifted clients are assigned to the correct clusters or how many spurious clusters are created.

6.4.1 Cluster Creation and Concept Visualizations

The clustering evaluation uses two complementary grid-style plots to make the relationship between clients' true concepts and their cluster assignments explicit. Furthermore, the accompanying cluster accuracy plots show how the predictive performance of each cluster model changes throughout the experiment.

Concept Drift Progression (on round aggregation). In the following example, three clients are used and there is a staggered drift for two recurring concepts, as shown in Figure 6.9.

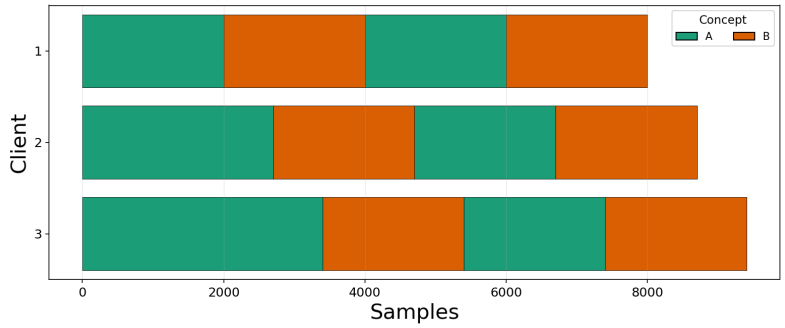


Figure 6.9: Example of staggered drift with three clients and two recurring concepts.

For each experiment, the results provide a time-series sequence of records produced by round-based aggregations. Each record contains a cluster identifier and the set of clients currently associated with that cluster, together with each client's true concept. Furthermore, a model snapshot of the aggregated cluster model is added to the record. A fully correct clustering of clients using the framework is seen in Figure 6.10.

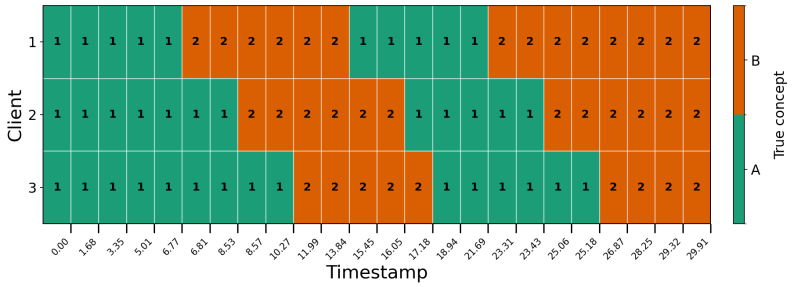


Figure 6.10: Example of expected behaviour of successful framework. The two concepts have their own separate clusters, the number of clusters is the same as the number of concepts and each client is always correctly assigned.

The x-axis shows the time at which the record was created, measured from the start of the experiment. The colour of the cells shows the true concept, while the number inside each cell indicates the cluster to which the client is assigned at that time. To avoid redundant columns where no cluster reassignments or drifts occur, timestamps are removed where the columns are identical to the subsequent column, which can be seen in Figure 6.11. The resulting visualization is referred to as the *concept drift progression grid* and is shown in Figure 6.11.

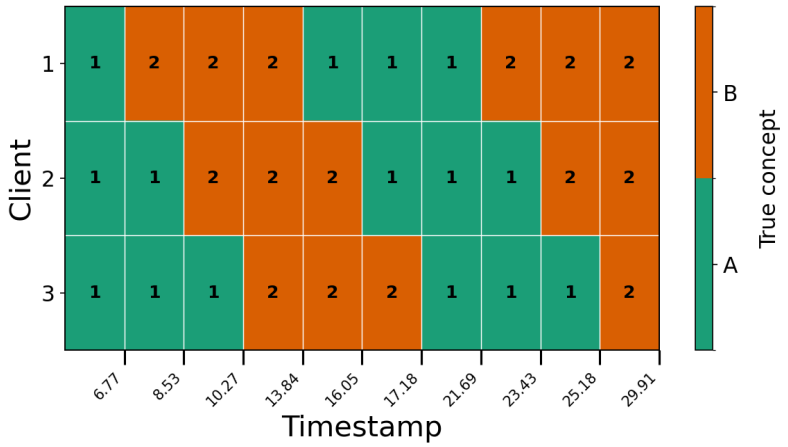


Figure 6.11: Example of concept drift progression grid over time for three clients.

The visualization contains the same information as Figure 6.10, but omits timestamps where neither drifts nor cluster reassignments occur.

Summarized Concept Cluster Grid. The summarized grid in Figure 6.12 is derived from the full concept drift progression history by grouping consecutive timestamps into intervals such that no client drifts more than once within an interval. The columns are processed in order, and an interval ends when adding the next column would introduce a second drift for any client. At each interval boundary, the last column that still satisfies the one-drift constraint is retained, and a new interval begins.

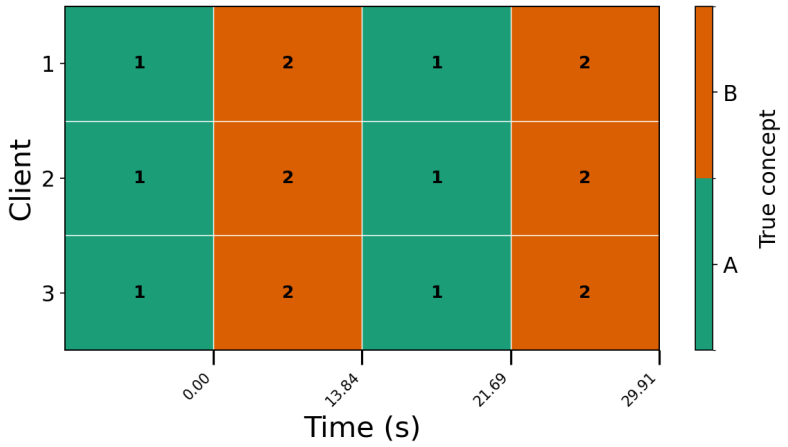


Figure 6.12: Example of *concept drift cluster grid* over time for three clients.

Consequently, the x-axis values in Figure 6.12 form a subset of the timestamps shown in Figure 6.11. As in the full grid, cell colour encodes the true concept and the cell annotation indicates the assigned cluster. This acts as the qualitative visualization for comparing cluster assignments with the true concepts experienced by the clients. The colour of each cell encodes the client's true concept and the annotation shows the assigned cluster. In a correct clustering, clients assigned to the same cluster should share the same concept and therefore have the same cell colour. Mixed colours within a cluster indicate that clients with different concepts are being aggregated together, which may be caused by a missed detection, a delayed reassignment, or an incorrect cluster assignment.

The summarized concept cluster grid is inspired by the visualization methodology proposed in FedDrift [18]. Similar to their work, the grid provides a compact representation of the relationship between the true concepts experienced by clients and the clusters produced by the framework. While the primary conclusions of this thesis are drawn from the quantitative metrics presented throughout the evaluation, the summarized grid remains valuable because it provides a direct qualitative comparison with FedDrift under equivalent scenarios. In particular, it visualizes the correspondence between concepts and clusters over time, including whether clients sharing the same concept are assigned to the same cluster, whether recurring concepts result in cluster reuse, and whether additional clusters are created unnecessarily. Consequently, the visualization serves both as a validation of the clustering behaviour and as a means of comparing the proposed framework against the cluster formation patterns reported in the FedDrift paper.

6.4.2 Cluster Model Performance

To evaluate the framework from a machine learning performance perspective, the clusters are evaluated as presented in subsection 3.2.3. This performance is validated with accuracy, using snapshots of the cluster models' weights, saved with a timestamp, the cluster ID and which clients were connected to it. After an experiment is completed, the evaluation process iterates across this history and records

the accuracy of the models at every time step. Due to the complications from having different concepts at each time step, each cluster model is evaluated on unseen data against all concepts of the experiment. This means that for each time step, the highest performing accuracy of any of these concepts then determines the concept of the snapshot model. To further validate the performance of the framework, the true concept of the clients assigned to each cluster is also recorded in a *cluster membership* table.

The following example presents a well-performing framework with four concepts: A: SINE-1 concept 1, B: SINE-1 concept 2, C: SINE-2 concept 1, and D: SINE-2 concept 2. It shows the result of staggered abrupt and recurring drifts for some data. The specific configuration is not important. The graph should be interpreted as a toy-example to understand how to read the results of the evaluation.

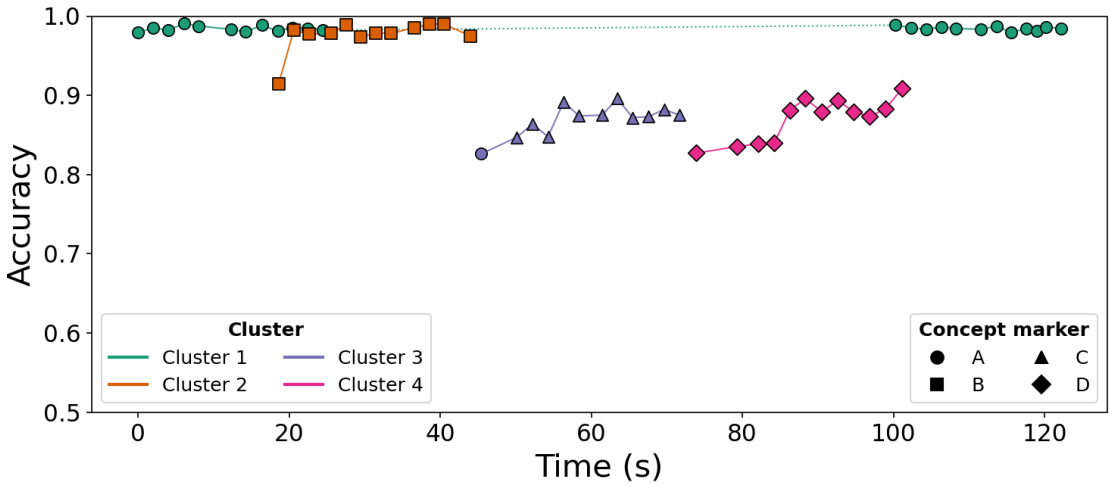


Figure 6.13: An example of a cluster accuracy plot for well-performing framework with 4 concepts and 9 clients. The colour of the series identifies the cluster and the symbol markers define the best concept in a time step. Each marker is an aggregation within the specified cluster. A dotted line means that there are no clients in the cluster between a pair of aggregation time steps.

To complement the accuracy plot and explicitly show the number of clients and concepts associated with each cluster over time, the corresponding cluster membership table is presented in Table 6.4.

Table 6.4: Cluster membership table for Figure 6.13. **Best** is the highest-performing concept of the cluster and **True** shows the number of clients assigned to it with their respective concepts.

Cluster 1

Time	Best	True	Acc
0.00	A	A (9)	0.9792
2.03	A	A (9)	0.9854
4.04	A	A (9)	0.9823
6.07	A	A (9)	0.9906
8.04	A	A (9)	0.9875
12.29	A	A (9)	0.9833
14.28	A	A (9)	0.9802
16.40	A	A (9)	0.9885
18.49	A	A (8)	0.9812
20.48	A	A (7)	0.9854
22.48	A	A (5)	0.9844
24.48	A	A (3)	0.9823
100.26	A	A (2)	0.9885
102.31	A	A (3)	0.9854
104.35	A	A (9)	0.9833
106.34	A	A (9)	0.9865
108.32	A	A (9)	0.9844
111.62	A	A (9)	0.9833
113.65	A	A (9)	0.9875
115.59	A	A (9)	0.9792
117.59	A	A (9)	0.9844
118.97	A	A (9)	0.9812
120.17	A	A (9)	0.9865
122.16	A	A (9)	0.9844

Cluster 2

Time	Best	True	Acc
18.63	B	B (1)	0.9146
20.62	B	B (2)	0.9823
22.63	B	B (3)	0.9771
25.58	B	B (6)	0.9781
27.55	B	B (9)	0.9885
29.52	B	B (9)	0.9740
31.50	B	B (9)	0.9781
33.51	B	B (9)	0.9781
36.54	B	B (9)	0.9854
38.55	B	B (7)	0.9896
40.55	B	B (7)	0.9896
44.04	B	B (2), C (1)	0.9750

Cluster 3

Time	Best	True	Acc
45.45	A	C (1)	0.8260
50.11	C	C (2)	0.8469
52.21	C	C (3)	0.8635
54.28	C	C (2)	0.8479
56.35	C	C (3)	0.8917
58.38	C	C (5)	0.8740
61.43	C	C (7)	0.8750
63.45	C	C (7)	0.8958
65.49	C	C (7)	0.8719
67.57	C	C (6)	0.8729
69.63	C	C (6)	0.8823
71.69	C	C (5)	0.8750

Cluster 4

Time	Best	True	Acc
73.92	D	D (1)	0.8271
79.34	D	D (1)	0.8354
82.08	D	D (2)	0.8385
84.22	D	D (3)	0.8396
86.28	D	D (4)	0.8813
88.36	D	D (4)	0.8958
90.52	D	D (4)	0.8792
92.64	D	D (5)	0.8938
94.73	D	D (4)	0.8792
96.88	D	D (3)	0.8729
98.99	D	D (3)	0.8833

Similarly to how the concept grids are summarized in subsection 6.4.1, the cluster membership tables are summarized as well. The first and last rows of each cluster are kept to determine how long the cluster exists. Then, for each row, only the first occurrence of subsequent equal rows in columns **Best** and **True** are kept. This removes all rows where the cluster does not change the best performing concept (**Best**) nor the number of client-concept mappings (**True**) between aggregations. The result is shown in Table 6.5.

Table 6.5: Summarized cluster membership table for Figure 6.13. Each subsequent recurring row in columns Best and True are removed and only the first instance is kept.

Cluster 1

Time	Best	True	Acc
0.00	A	A (9)	0.9792
18.49	A	A (8)	0.9812
20.48	A	A (7)	0.9854
22.48	A	A (5)	0.9844
24.48	A	A (3)	0.9823
100.26	A	A (2)	0.9885
102.31	A	A (3)	0.9854
104.35	A	A (9)	0.9833
122.16	A	A (9)	0.9844

Cluster 2

Time	Best	True	Acc
18.63	B	B (1)	0.9146
20.62	B	B (2)	0.9823
22.63	B	B (3)	0.9771
25.58	B	B (6)	0.9781
27.55	B	B (9)	0.9885
38.55	B	B (7)	0.9896
44.04	B	B (2), C (1)	0.9750

Cluster 3

Time	Best	True	Acc
45.45	A	C (1)	0.8260
50.11	C	C (2)	0.8469
52.21	C	C (3)	0.8635
54.28	C	C (2)	0.8479
56.35	C	C (3)	0.8917
58.38	C	C (5)	0.8740
61.43	C	C (7)	0.8750
67.57	C	C (6)	0.8729
71.69	C	C (5)	0.8750

Cluster 4

Time	Best	True	Acc
73.92	D	D (1)	0.8271
82.08	D	D (2)	0.8385
84.22	D	D (3)	0.8396
86.28	D	D (4)	0.8813
92.64	D	D (5)	0.8938
94.73	D	D (4)	0.8792
96.88	D	D (3)	0.8729
101.12	D	D (2)	0.9083

A consequence of this summarization is that exact timing between aggregations is no longer directly visible. For example, in **Cluster 1**, the time difference between the second-to-last and last recorded rows is approximately 18 seconds. This interval may

be misinterpreted as the cluster being inactive; however, as shown in Figure 6.13, the cluster remains continuously active during this period.

6.5 How to Draw Conclusions from the Results

This section provides guidance on how to interpret the accuracy plots, cluster membership tables, and concept cluster grids presented in the following sections. The three visualizations are complementary and should be analyzed together to fully assess framework behaviour.

6.5.1 Accuracy Plots and Cluster Membership Tables

The cluster accuracy plot shows the validation accuracy of each cluster model over time. A drop in accuracy within a cluster, or an inconsistent best concept over time, is a signal that the cluster has been contaminated by clients carrying a different concept, i.e., model poisoning due to missed/delayed detections, mistakes in the stability checker or cluster selection algorithms. The cluster membership table provides the same information in tabular form, with the added detail of which concepts, and how many clients of each, were present in the cluster at the time of aggregation. This allows a contamination to be identified precisely. If the **True** column contains a mix of concepts at a given row, poisoning is occurring at that aggregation round.

For example, as can be seen in Figure 6.13, there was almost no poisoning and the framework worked as intended. Even when the results seem to point to poor performance of the framework, it does not always have to be the case. For example, the first marker of cluster 3 does not match the rest. This means that either the cluster has changed concept over time due to poisoning, or the cluster performed better on one concept while actually representing another. This can be further investigated by looking at the cluster membership table (Table 6.5). In **Cluster 3**, the first **Best** value is A, yet its only client is of concept C. This being the first and only client means that no poisoning has occurred, but the client would still technically perform better with concept A, indicating that it has not yet trained enough to perform well with concept C. This is likely the result of the stability checker (algorithm 4) determining stability too early meaning that it has a low accuracy on that concept, but the cluster selection (algorithm 5) was able to flag the client as not part of **cluster 1** despite it performing better at concept **A**. As time goes on, more and more clients of concept C are assigned to it and the cluster model's best concept is now aligned with its clients. All this points to the cluster being created with a slightly suboptimal client and thus a suboptimal initial cluster model rather than model poisoning or poor reassignment.

6.5.2 Concept Cluster Grids

The summarized concept cluster grid, described in section 6.4, compresses the full time series into representative snapshots of concept drifts. To use it, two consecutive time steps are observed. Deviations in the correct cluster assignments are found in

two cases. If the colour of the cell and the cluster ID number changes such that the new ID is different from previous instances of the new colour, then an incorrect reassignment has been made. In other words, the framework did not identify the correct cluster nor did it create a new one for the newly drifted client. In the second case, a client changes colour but does not change cluster ID. This means that a drift has occurred but was either missed by the detector, or a reassignment was performed but the client ended up in the same cluster. The concept cluster grid therefore acts as a qualitative summary of clustering correctness, complementing the quantitative cluster membership tables.

6.5.3 Relating False Negatives to Poisoning

A false negative, meaning a drift event that the detector fails to raise an alarm for has direct consequences at the cluster level. When a client drifts without the detector noticing, it continues to be aggregated in its current cluster while training on the new concept. This introduces conflicting updates into the cluster model, degrading its accuracy on the original concept. When the aggregated cluster model is sent back to the client, it will produce a lower accuracy than before the aggregation due to the model poisoning.

Furthermore, if the local model recovers sufficient accuracy between aggregations, the client may never be reassigned correctly, as the clustering mechanism is never instantiated since the drift is never detected, as explained in subsection 4.1.2. This creates a scenario in which the client remains in an incorrect cluster potentially indefinitely, continuously introducing conflicting updates into the cluster model while evading reassignment due to the detector never detecting. Each aggregation essentially acts as an abrupt drift for the local model which may or may not be detected. This effect is visible as a sudden accuracy drop in the cluster accuracy plot and a mixed-concept entry in the **True** column of the cluster membership table. The concept cluster grid presented in section 6.4 will show the affected client remaining in the wrong cluster (same cluster ID, different colour) during the interval spanning the missed drift.

6.5.4 Delayed Detections

A delayed detection (high detection delay rather than a full false negative) can produce a similar poisoning effect to a false negative, but limited to the interval between the true drift point and the alarm. If aggregation occurs during this window, the cluster model is updated with a partially poisoning set of weights. Whether this causes a visible accuracy drop depends on how many clients in the cluster were simultaneously on the wrong concept and for how long. The delayed detections is not present in the concept grid, but can be found in the cluster membership tables by looking at the **True** column. For example, assume a cluster initially contains three clients, all on concept A. At time t, one client drifts to concept B, but the client was able to be aggregated twice before the detector raised the alarm. During those two rounds, the cluster still contains two clients on concept A and one client on concept B. In the cluster membership table, the corresponding rows would show

entries such as **True** = A (2), B (1). The cluster model may still report **Best** = A, but with a lower accuracy than before due to the conflicting update from the client on concept B. Once the drift is detected and the client is reassigned, the **True** column returns to A (2), and the cluster accuracy recovers.

7

Results

This chapter presents the results for some of the detectors for all five experiments described in chapter 6. The detectors included in this chapter were chosen specifically to highlight the strengths and weaknesses of the framework, while the results of the rest of them can be found in Appendix A. Each experiment is reported with its detection and adaptation performance tables, cluster accuracy plots for highlighted detectors, and some include concept cluster progression grids. In addition, the results are followed by explanations and observations which are then tied into an overall discussion in the next chapter.

7.1 Experiment 1 – SINE1 Abrupt

Table 7.1: Concept drift detector performance on Experiment 1 using the SINE1 dataset. TTC and delay are reported across clients; TP/FP/FN are totals across clients; precision, recall, and F1 are mean ± std of per-client rates.

Detector	TTC [ms]	Delay [Batches]	TP	FP	FN	Prec	Rec	F1
None	–	–	0	0	15	–	0.000 ± 0.000	–
DDM	0.0887 ± 0.0012	21.7 ± 7.6	15	9	0	0.700 ± 0.254	1.000 ± 0.000	0.800 ± 0.169
EDDM	0.0738 ± 0.0017	23.1 ± 11.2	15	10	0	0.667 ± 0.244	1.000 ± 0.000	0.778 ± 0.163
MDDM A	0.0890 ± 0.0008	12.2 ± 0.7	15	0	0	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000
MDDM E	0.0861 ± 0.0027	13.1 ± 1.1	15	0	0	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000
MDDM G	0.0844 ± 0.0010	11.7 ± 0.7	15	0	0	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000
HDDM A	0.0795 ± 0.0015	8.3 ± 8.7	15	0	0	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000
HDDM W	0.0826 ± 0.0016	10.2 ± 1.0	15	0	0	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000
FHDDM	0.0820 ± 0.0033	14.1 ± 1.6	15	0	0	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000
FHDDMS	0.0908 ± 0.0028	13.5 ± 0.8	15	0	0	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000

Table 7.2: Adaptation performance per detector on Experiment 1 (mean ± std across all clients).

Detector	Adapt TTC [ms]	Adaptor Mem [Batches]
None	–	–
DDM	95.9 ± 327.3	127.7 ± 439.4
EDDM	10.2 ± 2.4	12.7 ± 3.2
MDDM A	18.2 ± 0.2	25.0
MDDM E	24.8 ± 25.8	25.0
MDDM G	18.8 ± 0.2	25.0
HDDM A	2.1 ± 1.1	2.1 ± 1.5
HDDM W	2.8 ± 0.7	2.9 ± 0.5
FHDDM	18.6 ± 0.4	25.0
FHDDMS	24.9 ± 25.8	25.0

For Experiment 1, consisting of the simplest dataset in terms of data complexity and drift severity, SINE1, most detectors performed perfectly. Only DDM and EDDM had issues with several false positives, while the others only detected true drifts with minimal delay and with very similar TTC. Regarding adaptation, the detectors that do not use predefined window sizes for retraining used significantly fewer samples and thus took much less time to perform. DDM had a noticeably higher TTC and adaptor memory, which was most likely the result of an outlier detection. The detector likely signalled a warning for an extended amount of time such that the retraining buffer was filled with many more samples than for the other detectors.

7.1.1 None detector

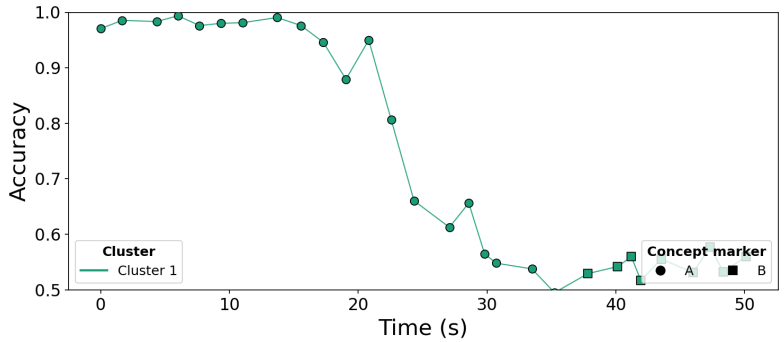


Figure 7.1: Cluster accuracy plot for Experiment 1 with no detector.

Having no detector, and thus no re-clustering mechanism, significantly impacted the performance of the global model as clients started to drift. The low accuracy is the result of model poisoning from the clients with different concepts being aggregated.

7.1.2 HDDM A detector

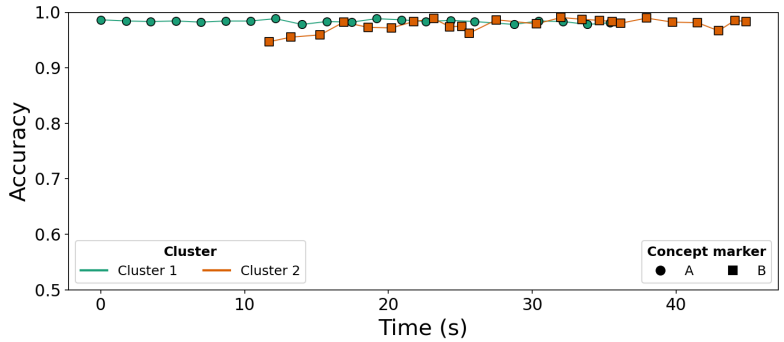


Figure 7.2: Cluster accuracy plot for Experiment 1 with HDDM A detector.

With re-clustering, the framework worked as expected. The plot for HDDM A is the ideal situation, and the plots for the other detectors were similar.

7.2 Experiment 2 – SINE2 Abrupt & Recurring

Table 7.3: Concept drift detector performance on Experiment 2. TTC and delay are reported as mean ± std across clients; TP/FP/FN are totals across clients; precision, recall, and F1 are mean ± std of per-client rates.

Detector	TTC [ms]	Delay [Batches]	TP	FP	FN	Prec	Rec	F1
None	–	–	0	0	45	–	0.000 ± 0.000	–
DDM	0.0865 ± 0.0033	89.6 ± 73.1	43	0	2	1.000 ± 0.000	0.956 ± 0.172	0.967 ± 0.129
EDDM	0.0742 ± 0.0015	168.5 ± 65.7	42	22	3	0.689 ± 0.178	0.933 ± 0.138	0.780 ± 0.135
MDDM A	0.0861 ± 0.0029	11.8 ± 1.4	45	0	0	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000
MDDM E	0.0875 ± 0.0024	13.9 ± 1.4	45	0	0	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000
MDDM G	0.0814 ± 0.0014	11.5 ± 1.5	45	0	0	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000
HDDM A	0.0807 ± 0.0015	21.0 ± 11.5	44	2	1	0.967 ± 0.088	0.978 ± 0.086	0.968 ± 0.068
HDDM W	0.0808 ± 0.0032	11.4 ± 3.6	45	3	0	0.950 ± 0.104	1.000 ± 0.000	0.971 ± 0.059
FHDDM	0.0816 ± 0.0023	13.5 ± 1.1	45	0	0	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000
FHDDMS	0.0941 ± 0.0039	13.6 ± 1.7	45	0	0	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000

Table 7.4: Adaptation performance per detector on Experiment 2 (mean ± std across all clients).

Detector	Adapt TTC [ms]	Adaptor Mem [Batches]
None	–	–
DDM	44.5 ± 35.1	59.0 ± 47.4
EDDM	16.0 ± 2.0	21.2 ± 2.6
MDDM A	18.4 ± 0.3	25.0
MDDM E	18.9 ± 0.5	25.0
MDDM G	18.7 ± 0.4	25.0
HDDM A	4.7 ± 2.5	5.5 ± 3.4
HDDM W	4.1 ± 1.2	4.6 ± 1.6
FHDDM	19.0 ± 0.9	25.0
FHDDMS	18.7 ± 0.6	25.0

Although the drifts in SINE2 are as severe as those in SINE1 and would therefore be expected to produce similar detection performance, the results suggest otherwise. The lower F1 scores for HDDM A and HDDM W with three times the number of drifts indicate that they are not perfectly accurate and that there is some probability that a drift is missed or a false drift is signalled as the number of drifts increases. The lack of a significant increase in detection delay means that the issue was unlikely to be caused by the differences in the dataset between SINE1 and SINE2. When it comes to adaptation, the results are similar regarding TTC, with a slightly higher number of samples saved for retraining for the non-windowing detectors, meaning that there was as longer period of time between a warning and a drift signal allowing for more samples to be saved.

7.2.1 None detector

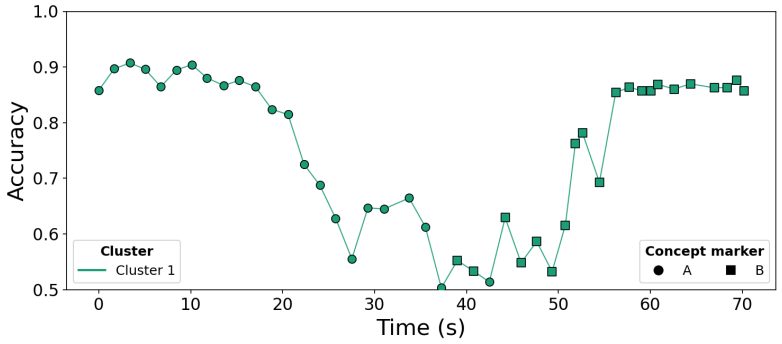


Figure 7.3: Cluster accuracy plot for Experiment 2 with no detector.

Table 7.5: Cluster membership table for Figure 7.3.

Cluster 1

Time	Best	True	Acc
0.00	A	A (15)	0.8583
11.77	A	A (14), B (1)	0.8802
13.60	A	A (13), B (2)	0.8667
15.25	A	A (12), B (3)	0.8760
17.03	A	A (11), B (4)	0.8646
18.82	A	A (9), B (6)	0.8240
22.33	A	A (8), B (7)	0.7250
27.53	A	A (7), B (8)	0.5552
29.28	A	A (8), B (7)	0.6469
31.04	A	A (9), B (6)	0.6448
35.53	A	A (8), B (7)	0.6125
37.26	A	A (7), B (8)	0.5031
39.01	B	A (7), B (8)	0.5521
40.76	B	A (8), B (7)	0.5333
42.48	A	A (7), B (8)	0.5135
44.21	B	A (6), B (9)	0.6292
47.63	B	A (7), B (8)	0.5865
50.76	B	A (6), B (9)	0.6156
51.85	B	A (5), B (10)	0.7625
54.46	B	A (4), B (11)	0.6927
56.24	B	A (3), B (12)	0.8542
57.72	B	A (2), B (13)	0.8635
59.06	B	A (1), B (14)	0.8573
60.79	B	B (15)	0.8687
70.17	B	B (15)	0.8573

As was the case for Experiment 1, the framework struggled as clients started to drift and performed very poorly during the middle period, and then improving again. This is because between 20s–60s the fraction of clients of each concept is relatively even, meaning that the two completely conflicting concepts are being aggregated together making the global model useless. As the balance of concepts starts to strongly favour one over the other, the performance improves as would be expected.

7.2.2 MDDM E detector

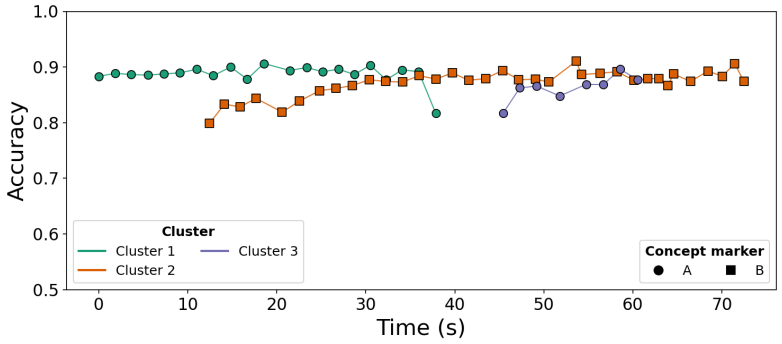


Figure 7.4: Cluster accuracy plot for Experiment 2 with MDDM E detector.

Table 7.6: Cluster membership table for Figure 7.4.

Cluster 1

Time	Best	True	Acc
0.00	A	A (15)	0.8833
11.01	A	A (14)	0.8958
14.80	A	A (13)	0.9000
16.66	A	A (12)	0.8781
18.53	A	A (11)	0.9062
21.46	A	A (10)	0.8938
23.35	A	A (9)	0.8990
25.13	A	A (8)	0.8917
26.96	A	A (7)	0.8958
28.71	A	A (6)	0.8865
30.49	A	A (4)	0.9031
32.30	A	A (3)	0.8771
34.09	A	A (2), B (1)	0.8948
35.93	A	A (1), B (1)	0.8917
37.86	A	B (1)	0.8167

Cluster 2

Time	Best	True	Acc
12.45	B	B (1)	0.7990
28.48	B	A (1), B (1)	0.8667
30.39	B	B (2)	0.8771
32.23	B	B (3)	0.8740
34.12	B	B (2)	0.8729
37.90	B	B (4)	0.8781
43.46	B	B (2)	0.8792
45.35	B	B (3)	0.8938
47.19	B	B (2)	0.8771
50.55	B	B (3)	0.8729
53.63	B	B (5)	0.9104
56.29	B	B (6)	0.8885
60.14	B	B (9)	0.8760
63.93	B	B (10)	0.8667
68.43	B	B (11)	0.8927
72.47	B	B (11)	0.8740

Cluster 3

Time	Best	True	Acc
45.41	A	A (1)	0.8167
51.78	A	A (2)	0.8479
58.59	A	A (1)	0.8958
60.53	A	A (1)	0.8771

While the overall performance of the framework was very high for most detectors as shown in section A.2, the results for MDDM E highlight a weakness of the framework. The plot and table above shows what happens when the detection delay is slightly too long and aggregation occurs in the period between a drift and the detector noticing. The experiment only contains two concepts, yet three clusters were created. It can also be seen that the final point in **Cluster 1** is at a noticeably lower accuracy than previous points, indicating that some poisoning has occurred, even if the best concept is still concept A. This is confirmed by Table 7.6, where the true concept of the last client in **Cluster 1** is actually concept B. This means that the last client was able to drift and then aggregate before the detector realised. The error of the client had not yet reached the point where a drift was detected, but enough to cause some small poisoning in the cluster model. When a client now drifts back into concept A, **Cluster 1** performs poorly, as determined by the cluster selection, and a new cluster is created.

7.3 Experiment 3 – SINE2 Gradual & Recurring

Table 7.7: Concept drift detector performance on Experiment 3. TTC and delay are reported as mean ± std across clients; TP/FP/FN are totals across clients; precision, recall, and F1 are mean ± std of per-client rates.

Detector	TTC [ms]	Delay [Batches]	TP	FP	FN	Prec	Rec	F1
None	–	–	0	0	45	–	0.000 ± 0.000	–
DDM	0.0865 ± 0.0022	160.3 ± 28.2	45	1	0	0.983 ± 0.065	1.000 ± 0.000	0.990 ± 0.037
EDDM	0.0738 ± 0.0018	274.6 ± 134.3	29	16	16	0.533 ± 0.371	0.644 ± 0.479	0.835 ± 0.109
MDDM A	0.0875 ± 0.0011	91.5 ± 17.2	45	0	0	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000
MDDM E	0.0863 ± 0.0032	102.6 ± 12.7	45	0	0	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000
MDDM G	0.0820 ± 0.0042	100.4 ± 15.1	45	0	0	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000
HDDM A	0.0793 ± 0.0028	102.0 ± 17.4	45	1	0	0.983 ± 0.065	1.000 ± 0.000	0.990 ± 0.037
HDDM W	0.0797 ± 0.0036	88.4 ± 12.0	45	1	0	0.983 ± 0.065	1.000 ± 0.000	0.990 ± 0.037
FHDDM	0.0815 ± 0.0010	103.0 ± 9.4	45	0	0	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000
FHDDMS	0.0928 ± 0.0028	96.7 ± 8.1	45	0	0	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000

Table 7.8: Adaptation performance per detector on Experiment 3 (mean ± std across all clients)

Detector	Adapt TTC [ms]	Adaptor Mem [Batches]
None	–	–
DDM	91.6 ± 134.1	123.3 ± 178.7
EDDM	16.0 ± 4.2	20.7 ± 5.4
MDDM A	18.8 ± 0.7	25.0
MDDM E	19.0 ± 0.8	25.0
MDDM G	21.1 ± 8.9	25.0
HDDM A	7.5 ± 2.2	9.2 ± 2.9
HDDM W	5.8 ± 1.7	7.1 ± 2.3
FHDDM	18.9 ± 0.8	25.0
FHDDMS	22.0 ± 8.9	25.0

Experiment 3 adds an additional element to the drift, namely a gradual transition between concepts instead of an abrupt switch from one to the other. This, as shown in the results, increases the delay of detection substantially since the incorrect predictions on the impending concept are obscured by the existence of samples from the old concept, which the model still is able to predict correctly. Overall, the detectors still performed well albeit with significant delays, which are counted from the start of the transition.

EDDM, however, degrades substantially, detecting only 29 out of 45 drift events (recall 0.644 ± 0.479) with 16 false positives. The large standard deviation in recall suggests that EDDM succeeded on some clients but failed systematically on others. The gradual transition period of 200 samples introduces a mixed-concept signal during the transition, which causes EDDM's error distance statistic to behave erratically [19]. The other detectors remain largely unaffected by the gradual transition, which suggests that their sliding-window or concentration-bound designs provide sufficient robustness to the diluted error signal during transition.

7.3.1 None detector

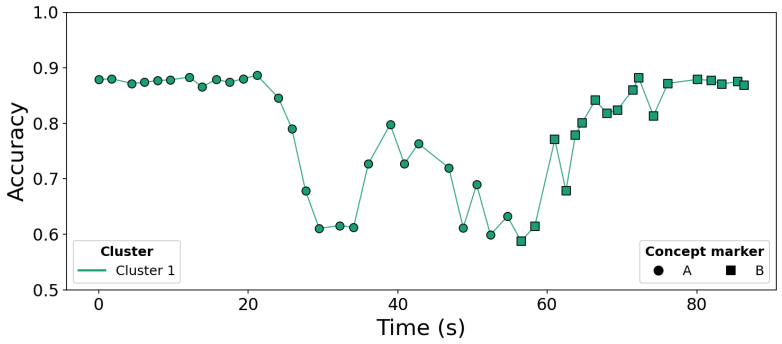


Figure 7.5: Cluster accuracy plot for Experiment 3 with no detector.

The results without a detector with gradual drift shows very similar results to Experiment 2 with the same dataset described in subsection 7.2.1.

7.3.2 FHDDM detector

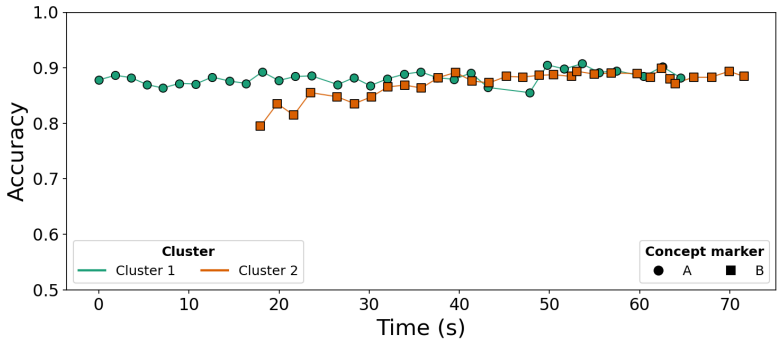


Figure 7.6: Cluster accuracy plot for Experiment 3 with FHDDM detector.

Table 7.9: Cluster membership table for Figure 7.6.

Cluster 1

Time	Best	True	Acc
0.00	A	A (15)	0.8781
10.76	A	A (14), B (1)	0.8708
12.57	A	A (14)	0.8833
14.51	A	A (13)	0.8760
16.32	A	A (12)	0.8719
18.15	A	A (11)	0.8927
19.95	A	A (9), B (1)	0.8771
21.80	A	A (9)	0.8844
23.61	A	A (8)	0.8854
26.51	A	A (7)	0.8698
28.32	A	A (6)	0.8823
30.10	A	A (5)	0.8677
32.02	A	A (6)	0.8802
37.53	A	A (4)	0.8823
39.42	A	A (2)	0.8792
41.29	A	A (1), B (1)	0.8906
43.17	A	B (1)	0.8646
47.79	A	A (1)	0.8552
53.60	A	A (2)	0.9073
55.48	A	A (1)	0.8917
62.52	A	A (2)	0.9021
64.53	A	A (1)	0.8823

Cluster 2

Time	Best	True	Acc
17.94	B	B (1)	0.7948
21.63	B	B (2)	0.8146
26.42	B	B (3)	0.8479
30.27	B	A (1), B (5)	0.8479
32.08	B	B (5)	0.8656
33.97	B	B (3)	0.8687
35.79	B	A (1), B (2)	0.8635
37.67	B	A (1), B (4)	0.8823
39.58	B	A (1), B (3)	0.8917
43.31	B	A (1), B (4)	0.8729
45.21	B	A (1), B (3)	0.8844
47.06	B	B (3)	0.8833
48.89	B	B (4)	0.8865
50.47	B	B (5)	0.8875
52.45	B	B (6)	0.8844
56.84	B	B (7)	0.8906
61.19	B	B (8)	0.8833
63.38	B	B (9)	0.8802
66.04	B	B (10)	0.8833
71.57	B	B (10)	0.8844

The results with the FHDDM detector in Table 7.9 shows good results with only some model poisoning. Both **Cluster 1** and **Cluster 2** are correctly defined by their concepts A and B throughout the run, with at most one incorrectly assigned client at any time step, likely due to the increased delay. The poisoning of the few number of incorrectly assigned clients essentially does not reduce the accuracy, as seen in **Cluster 2** in time steps 35.79–47.06. It should be noted that the **True** concepts shown in the tables will show the incoming concept if the client is currently transitioning.

7.3.3 MDDM G detector

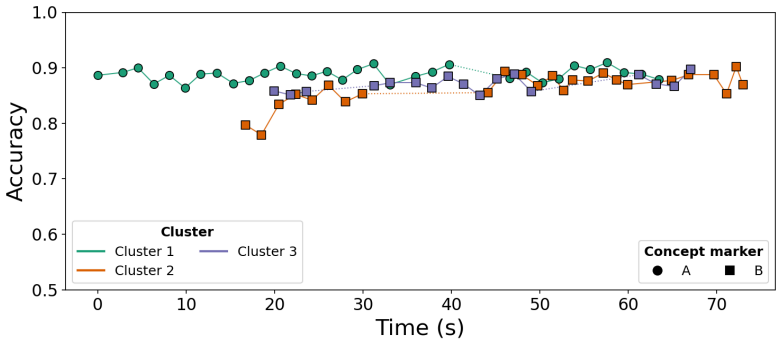


Figure 7.7: Cluster accuracy plot for Experiment 3 with MDDM G detector.

Table 7.10: Cluster membership table for Figure 7.7.

Cluster 1

Time	Best	True	Acc
0.00	A	A (15)	0.8865
13.48	A	A (14)	0.8906
15.33	A	A (13)	0.8719
17.13	A	A (12)	0.8771
18.84	A	A (11)	0.8906
20.62	A	A (9), B (1)	0.9031
22.43	A	A (8), B (1)	0.8896
24.17	A	A (7), B (1)	0.8854
25.90	A	A (6), B (1)	0.8938
27.64	A	A (6)	0.8781
29.35	A	A (5)	0.8969
33.02	A	A (4)	0.8698
35.92	A	A (3)	0.8844
37.81	A	A (2), B (1)	0.8927
39.73	A	A (1)	0.9062
46.54	A	A (2)	0.8813
48.42	A	A (3)	0.8927
50.28	A	A (5)	0.8729
52.12	A	A (4)	0.8802
53.88	A	A (3)	0.9042
55.69	A	A (3), B (1)	0.8969
57.59	A	A (2), B (1)	0.9094
59.50	A	A (1), B (1)	0.8917
61.47	A	A (1)	0.8885
63.43	A	B (1)	0.8792

Cluster 2

Time	Best	True	Acc
16.70	B	B (1)	0.7969
22.36	B	B (2)	0.8521
27.98	B	B (1)	0.8385
46.03	B	B (2)	0.8938
47.96	B	B (3)	0.8875
49.76	B	B (4)	0.8677
53.67	B	B (6)	0.8781
64.91	B	B (7)	0.8771
66.81	B	B (8)	0.8875
72.94	B	B (8)	0.8698

Cluster 3

Time	Best	True	Acc
19.92	B	B (1)	0.8583
31.25	B	B (2)	0.8677
33.06	B	B (3)	0.8729
37.74	B	A (1), B (2)	0.8635
39.60	B	B (3)	0.8844
41.37	B	B (2)	0.8708
43.20	B	A (1), B (1)	0.8500
45.16	B	B (1)	0.8802
49.02	B	A (1)	0.8573
61.16	B	B (2)	0.8875
67.04	B	B (2)	0.8969

Compared to the correct number of clusters, two, produced by the FHDDM detector, three clusters were created with MDDM G. Both **Cluster 2** and **Cluster 3** represent concept **B**, which means that an additional pointless cluster was created.

7.4 Experiment 4 – CIRCLES Abrupt

Table 7.11: Concept drift detector performance on Experiment 4. TTC and delay are reported as mean ± std across clients; TP/FP/FN are totals across clients; precision, recall, and F1 are mean ± std of per-client rates.

Detector	TTC [ms]	Delay [Batches]	TP	FP	FN	Prec	Rec	F1
None	–	–	0	0	45	–	0.000 ± 0.000	–
DDM	0.0887 ± 0.0046	1189.8 ± 1088.5	13	17	32	0.478 ± 0.314	0.289 ± 0.117	0.397 ± 0.066
EDDM	0.0728 ± 0.0008	1007.1 ± 821.0	19	37	26	0.351 ± 0.241	0.422 ± 0.235	0.413 ± 0.120
MDDM A	0.0976 ± 0.0043	289.0	1	0	44	1.000 ± 0.000	0.022 ± 0.086	0.039 ± 0.000
MDDM E	0.0951 ± 0.0049	–	0	0	45	–	0.000 ± 0.000	0.000 ± 0.000
MDDM G	0.0927 ± 0.0043	–	0	0	45	–	0.000 ± 0.000	0.000 ± 0.000
HDDM A	0.0800 ± 0.0039	–	0	0	45	–	0.000 ± 0.000	0.000 ± 0.000
HDDM W	0.0828 ± 0.0050	–	0	0	45	–	0.000 ± 0.000	0.000 ± 0.000
FHDDM	0.0817 ± 0.0024	–	0	0	45	–	0.000 ± 0.000	0.000 ± 0.000
FHDDMS	0.1162 ± 0.0065	51.0	1	0	44	1.000 ± 0.000	0.022 ± 0.086	0.039 ± 0.000

Table 7.12: Adaptation performance per detector on Experiment 4 (mean ± std across all clients).

Detector	**Adapt TTC [ms]**	**Adaptor Mem [Batches]**
None	–	–
DDM	631.2 ± 505.7	624.0 ± 477.1
EDDM	21.8 ± 8.8	20.6 ± 8.9
MDDM A	47.9	25.0
MDDM E	–	25.0
MDDM G	–	25.0
HDDM A	–	–
HDDM W	–	–
FHDDM	–	25.0
FHDDMS	56.3	100.0

Experiment 4 exposes a fundamental limitation in the sensitivity of all tested detectors to subtle concept drift. The CIRCLES dataset, described in section 6.2, transitions between concepts by shifting and expanding a circular decision boundary across four small steps. A model trained on one concept can still achieve a reasonable accuracy on the next, because the decision regions partially overlap. This means that the prediction error does not rise sharply at the drift point, but instead degrades slowly and partially, as can be seen in Figure 7.8 for no detector.

The results in Table 7.11 confirm this: all HDDM and FHDDM variants detect zero drifts, and MDDM variants detect at most one drift across all clients combined. Only DDM and EDDM achieve any detection, but with far more false positives than true positives. Furthermore, the detection delays for DDM and EDDM where detection does occur are on the order of 1000 batches, compared to delays of 10–20 batches in Experiments 1 and 2. This reflects the incremental nature of the underlying signal, i.e., the error only accumulates above the detection threshold long after the drift has occurred.

7.4.1 None detector

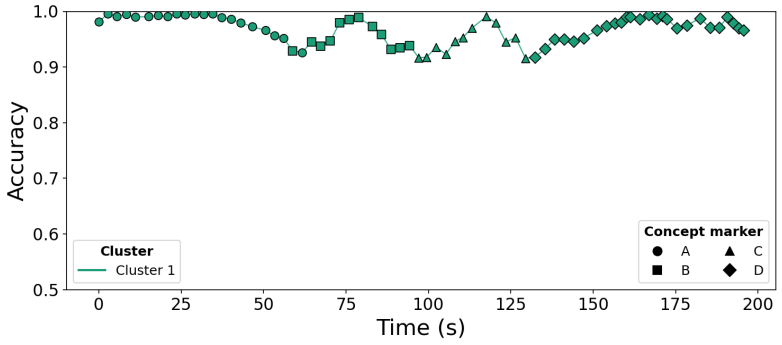


Figure 7.8: Cluster accuracy plot for Experiment 4 with no detector.

Table 7.13: Cluster membership table for Figure 7.8.

Cluster 1

Time	Best	True	Acc
0.00	A	A (15)	0.9812
34.55	A	A (14), B (1)	0.9958
40.21	A	A (13), B (2)	0.9865
43.01	A	A (12), B (3)	0.9792
46.54	A	A (11), B (4)	0.9729
50.51	A	A (10), B (5)	0.9656
53.32	A	A (9), B (6)	0.9563
56.04	A	A (8), B (7)	0.9521
58.81	B	A (7), B (8)	0.9292
61.65	A	A (6), B (9)	0.9260
64.52	B	A (5), B (10)	0.9448
67.33	B	A (4), B (11)	0.9375
73.05	B	A (3), B (12)	0.9792
76.02	B	A (2), B (12), C (1)	0.9854
78.84	B	A (1), B (12), C (2)	0.9885
83.01	B	B (12), C (3)	0.9729
85.81	B	B (11), C (4)	0.9583
88.64	B	B (9), C (6)	0.9312
91.38	B	B (8), C (7)	0.9344
94.25	B	B (7), C (8)	0.9385
97.08	C	B (6), C (9)	0.9167
102.37	C	B (5), C (10)	0.9354
105.29	C	B (4), C (11)	0.9229
110.44	C	B (3), C (12)	0.9531
113.20	C	B (3), C (11), D (1)	0.9698
117.53	C	B (2), C (11), D (2)	0.9906
120.51	C	B (1), C (11), D (3)	0.9792
123.49	C	C (12), D (3)	0.9448
126.45	C	C (11), D (4)	0.9531
129.45	C	C (9), D (6)	0.9156
132.29	D	C (8), D (7)	0.9177
135.34	D	C (7), D (8)	0.9323
138.32	D	C (6), D (9)	0.9500
141.12	D	C (5), D (10)	0.9500
147.05	D	C (4), D (11)	0.9521
154.01	D	C (3), D (12)	0.9740
156.58	D	C (2), D (13)	0.9781
161.32	D	C (1), D (14)	0.9896
164.16	D	D (15)	0.9865
195.58	D	D (15)	0.9656

An interesting observation is that despite up to 3 different concepts being aggregated together, the performance of the global model never dips below 90% accuracy. Looking at the CIRCLES dataset (Figure 6.3), this is for the same reason that

drifts were not detected: the different concepts are not different enough from one another, allowing for relatively high performance on all of them.

7.4.2 EDDM detector

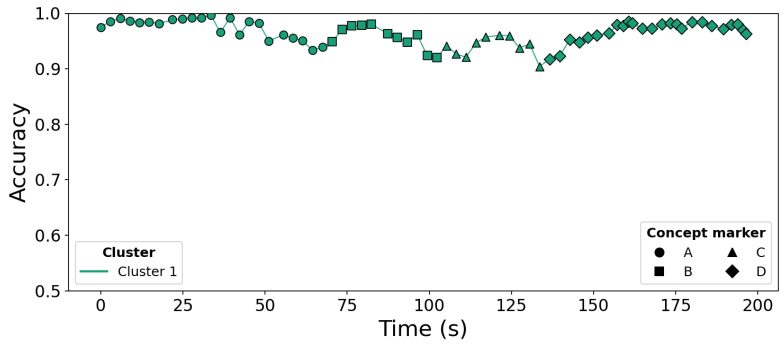


Figure 7.9: Cluster accuracy plot for Experiment 4 with EDDM detector.

7.4.3 MDDM A detector

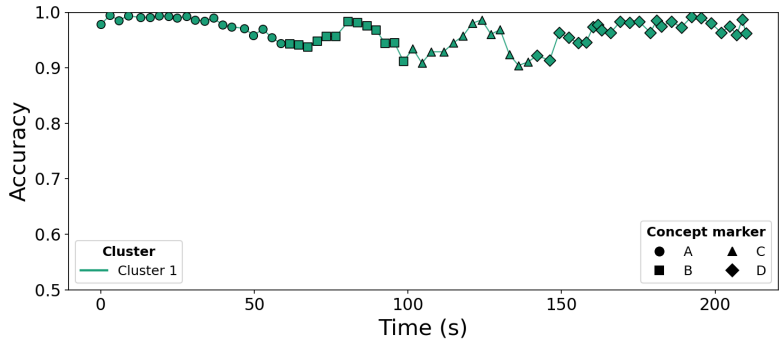


Figure 7.10: Cluster accuracy plot for Experiment 4 with MDDM A detector.

7.4.4 DDM detector

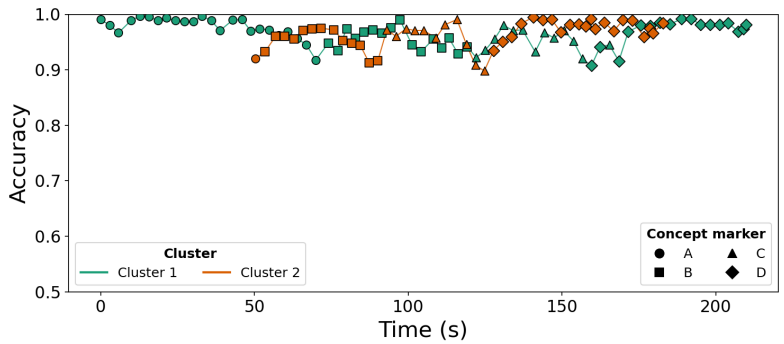


Figure 7.11: Cluster accuracy plot for Experiment 4 with DDM detector.

As a result of there not being any detections, the clustering mechanism is effectively bypassed, with most clients remaining aggregated in a single cluster throughout the experiment and no new clusters being created. Even for EDDM and MDDM A, which do trigger detections, no additional clusters emerge, while DDM is the only detector that causes a reclustering event. However, the additional cluster closely mirrors the original one, exhibiting a nearly identical accuracy pattern and concept representation. The cluster accuracy in Figure 7.8 further show that the single cluster model continuously shifts between the best-performing concepts as the underlying client concepts change. Rather than adapting through explicit drift handling, the model passively adapts to the changing data distribution through continued online training. This behaviour closely matches the passive learning approach described in **??**, where adaptation occurs implicitly without any meaningful separation between concepts.

7.5 Experiment 5 – Mixed

Experiment 5 is the most complex experiment in terms of client heterogeneity, with nine clients experiencing different ordered sequences of SINE1 and SINE2 concepts at different step lengths, as described in subsection 6.3.5. The drift severity differs due to both the complete decision boundary inversion of SINE concepts and the decision boundary between SINE1 and SINE2 being similar.

Table 7.14: Concept drift detector performance on Experiment 5. TTC and delay are reported as mean ± std across clients; TP/FP/FN are totals across clients; precision, recall, and F1 are mean ± std of per-client rates.

Detector	TTC [ms]	Delay [Batches]	TP	FP	FN	Prec	Rec	F1
None	–	–	0	0	55	–	0.000 ± 0.000	–
DDM	0.0905 ± 0.0029	159.4 ± 217.0	50	6	5	0.896 ± 0.087	0.921 ± 0.126	0.902 ± 0.078
EDDM	0.0715 ± 0.0031	452.2 ± 601.6	38	15	17	0.690 ± 0.204	0.667 ± 0.268	0.659 ± 0.219
MDDM A	0.0915 ± 0.0029	10.2 ± 3.3	55	36	0	0.663 ± 0.257	1.000 ± 0.000	0.773 ± 0.175
MDDM E	0.0888 ± 0.0050	11.1 ± 2.3	55	33	0	0.664 ± 0.255	1.000 ± 0.000	0.773 ± 0.186
MDDM G	0.0859 ± 0.0030	9.9 ± 2.5	55	27	0	0.697 ± 0.219	1.000 ± 0.000	0.804 ± 0.153
HDDM A	0.0834 ± 0.0036	18.5 ± 11.7	55	1	0	0.986 ± 0.042	1.000 ± 0.000	0.993 ± 0.022
HDDM W	0.0847 ± 0.0019	107.4 ± 285.2	55	5	0	0.917 ± 0.143	1.000 ± 0.000	0.951 ± 0.087
FHDDM	0.0828 ± 0.0058	10.3 ± 1.7	25.0	21	0	0.794 ± 0.228	1.000 ± 0.000	0.868 ± 0.152
FHDDMS	0.0936 ± 0.0036	11.2 ± 3.7	100.0	20	0	0.767 ± 0.193	1.000 ± 0.000	0.856 ± 0.123

Table 7.15: Adaptation performance per detector on Experiment 5 (mean ± std across all clients).

Detector	Adapt TTC [ms]	Adaptor Mem [Batches]
None	–	–
DDM	134.6 ± 225.3	129.2 ± 212.6
EDDM	18.7 ± 2.3	18.2 ± 2.3
MDDM A	26.4 ± 0.8	25.0
MDDM E	26.1 ± 1.2	25.0
MDDM G	26.2 ± 1.2	25.0
HDDM A	6.0 ± 2.7	4.9 ± 2.6
HDDM W	5.4 ± 2.4	4.5 ± 2.4
FHDDM	26.0 ± 1.0	25.0
FHDDMS	26.6 ± 1.6	25.0

As seen in Table 7.14, HDDM A achieves the best overall detection performance with 55 TP, 1 FP, and 0 FN (Table 7.14), corresponding to a mean F1 of 0.993 ± 0.022. DDM and EDDM perform the poorest with 5 and 17 FN.

7.5.1 None detector

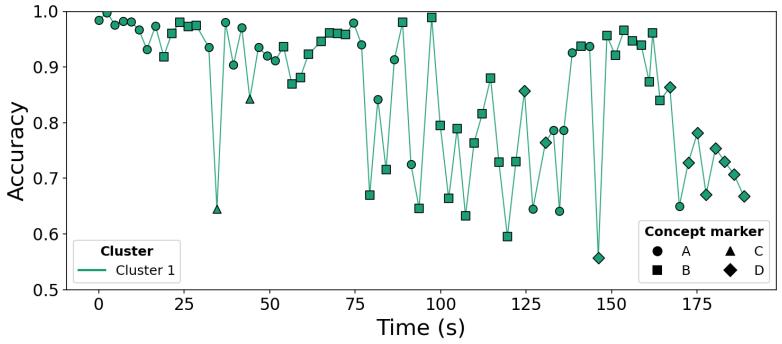


Figure 7.12: Cluster accuracy plot for Experiment 5 with no detector.

Table 7.16: Cluster membership table for Figure 7.12.

Cluster 1

Time	Best	True	Acc
0.00	A	A (9)	0.9844
11.76	A	A (7), B (2)	0.9667
18.98	B	A (3), B (6)	0.9187
21.34	B	A (2), B (6), D (1)	0.9604
28.47	B	A (2), B (5), C (1), D (1)	0.9750
32.16	A	A (4), B (2), C (2), D (1)	0.9354
34.55	C	A (4), B (2), C (2), D (1)	0.6448
37.03	A	A (4), B (1), C (3), D (1)	0.9802
41.80	A	A (5), C (3), D (1)	0.9708
44.18	C	A (4), B (1), C (4)	0.8427
46.68	A	A (4), B (1), C (3), D (1)	0.9354
49.19	A	A (4), B (1), C (2), D (2)	0.9198
54.06	B	A (2), B (3), C (1), D (3)	0.9365
61.33	B	A (2), B (4), C (1), D (2)	0.9229
72.19	B	A (3), B (4), C (1), D (1)	0.9583
74.53	A	A (5), B (2), C (1), D (1)	0.9792
79.22	B	A (4), B (3), C (1), D (1)	0.6698
81.62	A	A (4), B (3), C (1), D (1)	0.8417
84.03	B	A (4), B (3), C (1), D (1)	0.7156
86.45	A	A (4), B (3), C (1), D (1)	0.9135
88.85	B	A (3), B (4), C (1), D (1)	0.9802
91.30	A	A (3), B (4), C (1), D (1)	0.7250
93.69	B	A (4), B (4), C (1)	0.6458
97.36	B	A (2), B (5), C (2)	0.9885
102.34	B	A (2), B (4), C (3)	0.6635
104.82	B	A (1), B (5), C (3)	0.7896
107.27	B	A (1), B (4), C (4)	0.6323
114.60	B	A (1), B (4), C (3), D (1)	0.8802
117.03	B	A (2), B (2), C (3), D (2)	0.7292
119.52	B	A (2), B (2), C (2), D (3)	0.5948
121.99	B	A (2), B (2), C (1), D (4)	0.7302
124.50	D	A (2), B (2), C (1), D (4)	0.8573
126.95	A	A (3), B (1), C (1), D (4)	0.6448
130.66	D	A (3), B (1), C (1), D (4)	0.7646
132.92	A	A (3), B (1), C (1), D (4)	0.7865
141.05	B	A (1), B (3), C (1), D (4)	0.9375
143.55	A	A (3), B (2), D (4)	0.9375
146.10	D	A (3), B (2), D (4)	0.5573
148.62	B	A (2), B (3), D (4)	0.9563
167.03	D	A (1), B (3), D (5)	0.8635
169.81	A	A (1), B (3), D (5)	0.6500
172.42	D	A (1), B (3), D (5)	0.7281
188.67	D	A (1), B (3), D (5)	0.6677

The expected severe model poisoning is visible in both the cluster accuracy plot in Figure 7.12 and the cluster membership table (Table 7.16), where **Cluster 1** contains mixtures of all four or three concepts for nearly the entire experiment. Consequently, the cluster is unable to maintain a stable representation of any single concept, causing frequent changes in the best-performing concept and large fluctuations in validation accuracy.

7.5.2 DDM detector

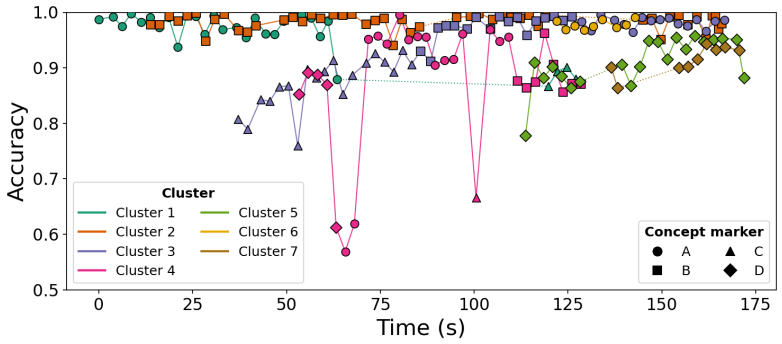


Figure 7.13: Cluster accuracy plot for Experiment 5 with DDM detector.

Table 7.17: Cluster membership table for Figure 7.13.

Cluster 1

Time	Best	True	Acc
0.00	A	A (9)	0.9875
3.73	A	A (8)	0.9917
6.19	A	A (9)	0.9750
13.73	A	A (7)	0.9906
18.62	A	A (6)	0.9906
21.06	A	A (2), B (1)	0.9375
23.46	A	A (2)	0.9937
36.70	A	A (3), C (1)	0.9729
39.16	A	A (4), C (1)	0.9542
44.40	A	A (2), B (1), C (1)	0.9615
46.86	A	A (2)	0.9604
49.27	A	A (4)	0.9844
56.59	A	A (2)	0.9896
63.50	A	A (1), D (1)	0.8792
119.74	C	C (1)	0.8667
127.11	C	C (1)	0.8802

Cluster 2

Time	Best	True	Acc
13.87	B	B (2)	0.9771
18.69	B	B (3)	0.9927
23.51	B	B (6)	0.9937
28.49	B	B (5)	0.9479
33.42	B	B (3)	0.9969
37.16	B	B (2)	0.9667
39.59	B	B (1)	0.9646
56.71	B	B (3)	0.9958
67.46	B	B (4)	0.9969
78.47	B	B (2)	0.9406
95.37	B	B (1)	0.9906
98.98	B	B (2)	0.9927
104.99	B	B (3)	0.9875
112.30	B	B (2)	0.9885
154.55	B	B (3)	0.9948
165.96	B	B (3)	0.9792

Cluster 3

Time	Best	True	Acc
37.14	C	C (1)	0.8073
52.99	C	C (2), D (1)	0.7594
55.52	C	C (2)	0.8969
57.99	C	C (1)	0.8823
85.72	B	B (1)	0.9302
128.57	A	A (1)	0.9833
131.22	A	A (3)	0.9667
151.93	A	A (1)	0.9896
166.69	A	A (1)	0.9865

Cluster 4

Time	Best	True	Acc
53.34	D	D (1)	0.8521
60.79	D	D (2)	0.8698
63.24	D	A (1), D (1)	0.6125
65.66	A	A (1), D (1)	0.5688
71.82	A	A (2)	0.9521
80.17	A	A (5)	0.9958
82.53	A	A (4), B (1)	0.9510
87.18	A	A (5), B (1)	0.9552
89.62	A	A (5), B (2), C (1)	0.9052
92.08	A	A (4), B (2), C (1)	0.9135
96.95	A	A (4), B (1), C (2)	0.9615
100.52	C	A (2), B (2), C (1)	0.6656
104.26	A	A (2), B (1), C (1)	0.9688
109.11	A	A (1), C (1)	0.9552
111.56	B	B (1), C (1)	0.8760
128.49	B	B (1), C (1)	0.8708

Cluster 5

Time	Best	True	Acc
113.69	D	D (1)	0.7781
171.87	D	D (1)	0.8823

Cluster 6

Time	Best	True	Acc
121.98	A	A (1)	0.9844
124.40	A	A (2)	0.9688
142.90	A	A (1)	0.9906

Cluster 7

Time	Best	True	Acc
136.54	D	D (1)	0.9010
154.58	D	D (2)	0.9000
170.64	D	D (2)	0.9312

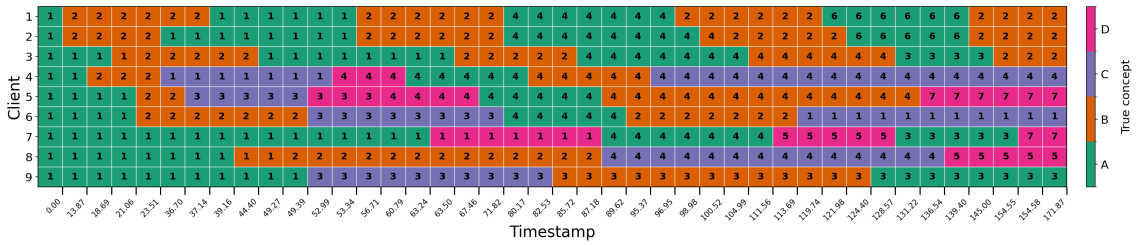


Figure 7.14: Concept progression grid for Experiment 5 with DDM detector.

DDM's high number of FN's leads to cluster models being unable to represent singular concepts, as seen in Figure 7.13. Eventually, a drift is correctly detected and reclustering is triggered. However, since all clusters have previously been poisoned, no cluster model exists to represent the new concept even if the concept has appeared previously, leading to new clusters being created which are subsequently poisoned again. This results in the chaotic graph shown in Figure 7.13, where clusters are created and then often switch their best concepts, with their accuracy dropping repeatedly. This can be seen in **Cluster 4** in the graph at around 60 seconds, where the accuracy nosedives, the concept switches and it quickly regains its accuracy in the new concept. This happened because, as seen in **Cluster 4** in the cluster membership Table 7.17 and concept progression grid in Figure 7.14, we have the following timeline of events:

1. **Cluster 4** has two clients (clients 4 and 5) with concept D and **Best** concept D at time step 60.79s.
2. At 63.24s, client 4 has drifted to concept A but remains in **Cluster 4**.
3. **Cluster 4** aggregates with both clients, leading to severe poisoning and poor validation accuracy for concept D.
4. Two more aggregations occur, this time with A being the **Best** performing concept but still low accuracy.
5. At 78.54s, client 5 has now also drifted to concept A, and **Cluster 4** now performs well in that concept.

7.5.3 MDDM A detector

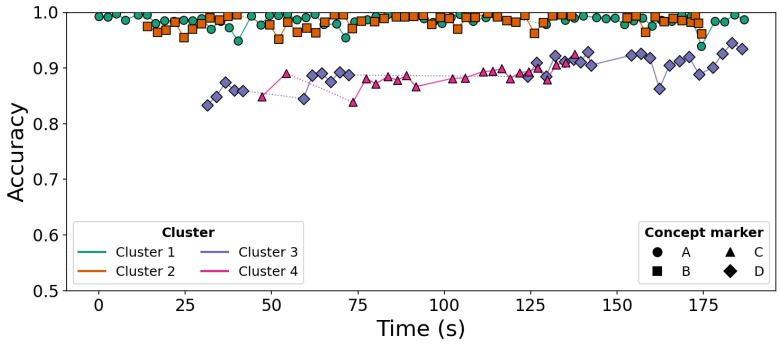


Figure 7.15: Cluster accuracy plot for Experiment 5 with MDDM A detector.

Table 7.18: Cluster membership table for Figure 7.15.

Cluster 1

Time	Best	True	Acc
0.00	A	A (9)	0.9927
13.87	A	A (7)	0.9958
21.78	A	A (2)	0.9844
32.51	A	A (2), C (1)	0.9698
35.18	A	A (4)	0.9844
57.24	A	A (2)	0.9875
68.73	A	A (1)	0.9792
71.43	A	A (2)	0.9542
77.70	A	A (3)	0.9875
83.62	A	A (5)	0.9969
86.24	A	A (4)	0.9958
104.56	A	A (2)	0.9969
114.72	A	A (1)	0.9958
129.54	A	A (2)	0.9781
140.25	A	A (4)	0.9937
144.09	A	A (5)	0.9906
152.09	A	A (3)	0.9781
160.15	A	A (2)	0.9760
165.82	A	A (1)	0.9844
186.71	A	A (1)	0.9875

Cluster 2

Time	Best	True	Acc
14.15	B	B (2)	0.9750
22.10	B	B (6)	0.9823
29.74	B	B (5)	0.9792
34.98	B	B (2)	0.9865
40.00	B	B (1)	0.9958
60.14	B	B (3)	0.9719
68.13	B	B (4)	0.9958
82.53	B	B (2)	0.9885
86.08	B	B (3)	0.9917
96.43	B	B (4)	0.9781
103.95	B	B (3)	0.9698
106.30	B	B (4)	0.9906
112.68	B	B (3)	0.9990
115.45	B	B (4)	0.9917
126.04	B	A (1), B (2)	0.9625
128.71	B	B (2)	0.9812
136.77	B	B (1)	0.9937
152.91	B	B (2)	0.9896
163.64	B	B (3)	0.9833
174.45	B	B (3)	0.9615

Cluster 3

Time	Best	True	Acc
31.34	D	D (1)	0.8333
59.23	D	D (2)	0.8448
61.81	D	D (3)	0.8865
67.05	D	D (2)	0.8750
69.70	D	D (1)	0.8927
126.74	D	D (2)	0.9094
134.77	D	D (3)	0.9115
139.48	D	D (2)	0.9104
141.56	D	D (3)	0.9292
154.12	D	D (4)	0.9229
177.79	D	D (5)	0.9010
186.09	D	D (5)	0.9344

Cluster 4

Time	Best	True	Acc
47.22	C	C (1)	0.8490
111.31	C	C (2)	0.8938
113.94	C	C (4)	0.8948
121.75	C	C (3)	0.8917
127.00	C	C (2)	0.9000
129.71	C	C (1)	0.8792
137.82	C	C (1)	0.9250

7.5.4 MDDM E detector

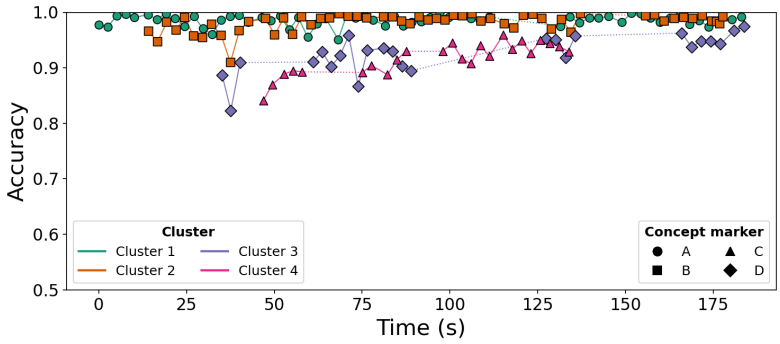


Figure 7.16: Cluster accuracy plot for Experiment 5 with MDDM E detector.

Table 7.19: Cluster membership table for Figure 7.16.

Cluster 1

Time	Best	True	Acc
0.00	A	A (9)	0.9771
13.96	A	A (7)	0.9958
21.78	A	A (2)	0.9885
34.84	A	A (3), C (1)	0.9865
37.41	A	A (4)	0.9927
42.55	A	A (2), C (1)	0.9823
46.45	A	A (4)	0.9906
59.48	A	A (2)	0.9552
68.20	A	A (1)	0.9510
70.73	A	A (2)	0.9979
75.65	A	A (3)	0.9927
84.13	A	A (4)	0.9927
89.13	A	A (3)	0.9823
91.80	A	A (4)	0.9833
105.94	A	A (2)	0.9885
139.70	A	A (5)	0.9896
154.43	A	A (3)	0.9979
162.65	A	A (1)	0.9896
183.09	A	A (1)	0.9917

Cluster 2

Time	Best	True	Acc
14.12	B	B (2)	0.9656
21.96	B	B (5)	0.9677
24.44	B	B (6)	0.9906
29.51	B	B (5)	0.9542
34.88	B	B (2)	0.9583
37.47	B	B (1), C (1)	0.9094
40.01	B	B (1)	0.9667
60.51	B	B (3)	0.9771
68.43	B	B (4)	0.9979
83.69	B	B (3)	0.9917
88.70	B	B (4)	0.9792
91.24	B	B (3)	0.9917
93.71	B	B (4)	0.9865
101.07	B	B (3)	0.9948
103.62	B	B (2)	0.9937
106.23	B	B (4)	0.9979
108.91	B	B (3)	0.9844
118.21	B	B (4)	0.9719
128.98	B	B (2)	0.9698
137.14	B	B (1)	0.9979
155.67	B	B (2)	0.9937
163.86	B	B (3)	0.9885
177.73	B	B (3)	0.9917

Cluster 3

Time	Best	True	Acc
35.13	D	D (1)	0.8865
68.69	D	D (3)	0.9219
71.21	D	D (2)	0.9583
73.88	D	D (1)	0.8667
135.66	D	D (2)	0.9573
171.58	D	D (3)	0.9479
183.72	D	D (3)	0.9750

Cluster 4

Time	Best	True	Acc
46.82	C	C (1)	0.8406
115.19	C	C (2)	0.9594
128.39	C	C (1)	0.9438
133.86	C	C (1)	0.9292

7.5.5 MDDM G detector

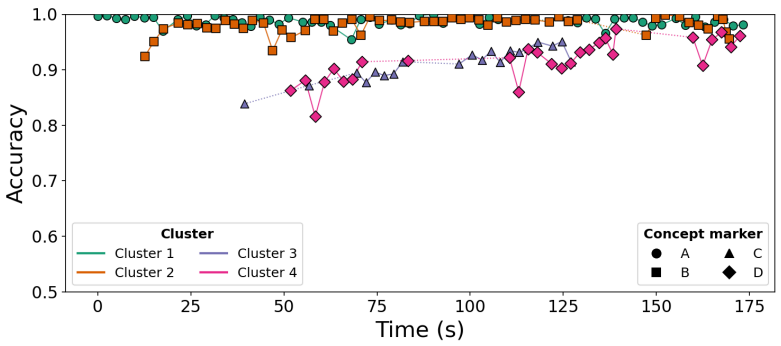


Figure 7.17: Cluster accuracy plot for Experiment 5 with MDDM G detector.

Table 7.20: Cluster membership table for Figure 7.17.

Cluster 1

Time	Best	True	Acc
0.00	A	A (9)	0.9969
12.45	A	A (7)	0.9937
21.50	A	A (2)	0.9906
36.20	A	A (4)	0.9906
41.06	A	A (3)	0.9781
43.54	A	A (4)	0.9958
57.50	A	A (2)	0.9865
68.08	A	A (1)	0.9542
70.60	A	A (2)	0.9906
75.44	A	A (3)	0.9823
78.83	A	A (5)	0.9896
83.75	A	A (4)	0.9833
90.10	A	A (5)	1.0000
92.65	A	A (4)	0.9844
102.54	A	A (2)	0.9823
107.53	A	A (1)	0.9906
126.27	A	A (2)	0.9885
131.25	A	A (4)	0.9937
133.68	A	A (5)	0.9917
146.34	A	A (3)	0.9865
151.36	A	A (2)	0.9812
155.26	A	A (1)	0.9927
173.24	A	A (1)	0.9812

Cluster 2

Time	Best	True	Acc
12.62	B	B (2)	0.9240
21.60	B	B (6)	0.9844
29.23	B	B (5)	0.9760
34.28	B	B (2)	0.9885
39.09	B	B (1)	0.9750
58.25	B	B (3)	0.9906
65.70	B	B (4)	0.9844
78.90	B	B (2)	0.9896
83.80	B	B (3)	0.9854
87.65	B	B (2)	0.9875
90.25	B	B (3)	0.9875
92.72	B	B (4)	0.9875
99.83	B	B (3)	0.9937
102.29	B	B (5)	0.9896
104.75	B	B (4)	0.9802
123.81	B	B (2)	0.9958
156.31	B	B (3)	0.9958
169.62	B	B (3)	0.9552

Cluster 3

Time	Best	True	Acc
39.31	C	C (1)	0.8385
105.59	C	C (2)	0.9333
110.69	C	C (3)	0.9344
124.63	C	C (2)	0.9510
127.13	C	C (1)	0.9146

Cluster 4

Time	Best	True	Acc
51.77	D	D (1)	0.8625
58.39	D	D (2)	0.8156
63.41	D	D (1)	0.9021
124.47	D	D (2)	0.9031
129.47	D	A (1), D (1)	0.9312
131.94	D	D (1)	0.9365
134.56	D	D (2)	0.9490
138.32	D	D (3)	0.9281
159.83	D	D (4)	0.9583
162.44	D	D (5)	0.9073
172.50	D	D (5)	0.9615

The cluster membership table (Table 7.20) for the MDDM G detector shows an almost perfect result. The clusters correctly represent the same concept over time, and the concepts of each client in their respective clusters reflect the best performing concept. There is only one instance of model poisoning in **Cluster 4** at time step 129.47, which is a consequence of a delayed detection.

The MDDM variants in Experiment 5 detect all 55 drift events (zero false negatives) but produce substantially more false positives (27–36 across variants). This is potentially problematic in the clustering context: each false positive triggers the full stabilisation and cluster reassignment pipeline, consuming adaptor memory, halting aggregation for the affected client, and potentially creating spurious new clusters. However, the concept cluster grids for MDDM A, E, and G in Figure A.10 do not show a higher number of clusters than there are concepts. This means that although MDDM detectors underperformed from a detector perspective because they are over-sensitive (large numbers of FPs), the number of clusters created is unaffected. Furthermore, the clusters are much more well-formed. As shown in the respective cluster membership tables (Table 7.18, Table 7.19, and Table 7.20), MDDM A, E, and G exhibit only 2, 3 and 1 incorrectly clustered clients. Additionally, by looking at the concept cluster grids in Figure A.10, the troublesome clients are clients 3 and 4 for MDDM A, clients 5, 6 and 9 for MDDM E, and client 7 for MDDM G.

7.5.6 HDDM A detector

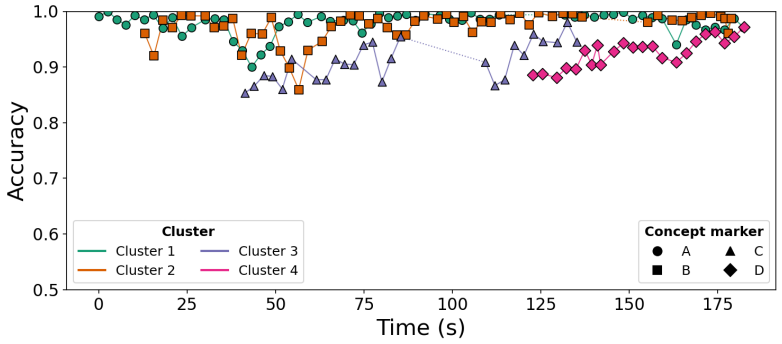


Figure 7.18: Cluster accuracy plot for Experiment 5 with HDDM A detector.

Table 7.21: Cluster membership table for Figure 7.18.

Cluster 1

Time	Best	True	Acc
0.00	A	A (9)	0.9906
12.82	A	A (7)	0.9854
20.78	A	A (2)	0.9885
35.40	A	A (4), C (1)	0.9854
43.21	A	A (2), C (1)	0.9000
45.79	A	A (4), C (1)	0.9219
53.62	A	A (4)	0.9812
56.26	A	A (3)	0.9948
58.91	A	A (2)	0.9802
69.19	A	A (1)	0.9844
74.33	A	A (2)	0.9615
76.89	A	A (3)	0.9771
81.81	A	A (5)	0.9885
84.38	A	A (4)	0.9917
105.28	A	A (2)	0.9979
137.33	A	A (3)	0.9937
140.07	A	A (5)	0.9896
153.73	A	A (3)	0.9927
163.30	A	A (1)	0.9406
182.37	A	A (1)	0.9708

Cluster 2

Time	Best	True	Acc
12.98	B	B (2)	0.9604
20.82	B	B (5)	0.9708
23.38	B	B (6)	0.9927
30.03	B	B (5)	0.9917
35.36	B	B (2)	0.9740
40.38	B	B (1)	0.9208
48.75	B	B (1), D (1)	0.9885
53.93	B	B (1), D (2)	0.8979
59.15	B	B (3), D (2)	0.9302
65.76	B	B (3), D (1)	0.9729
68.42	B	B (4), D (1)	0.9823
71.09	B	B (4)	0.9937
76.29	B	B (4), D (1)	0.9771
78.94	B	B (4)	0.9875
81.62	B	B (2), D (1)	0.9708
84.22	B	B (3), D (1)	0.9573
89.39	B	B (3)	0.9823
95.79	B	B (4)	0.9865
103.11	B	B (3)	0.9906
105.68	B	B (4), C (1)	0.9625
108.27	B	B (4)	0.9812
110.89	B	B (3)	0.9802
116.31	B	B (4)	0.9854
128.36	B	B (2)	0.9906
136.71	B	B (1)	0.9896
155.29	B	B (2)	0.9802
164.96	B	B (3)	0.9833
178.87	B	B (3)	0.9875

Cluster 3

Time	Best	True	Acc
41.32	C	C (1)	0.8531
46.64	C	C (2)	0.8844
49.15	C	C (1)	0.8833
117.51	C	C (2)	0.9396
122.91	C	C (3)	0.9594
125.63	C	C (2)	0.9458
129.69	C	C (1)	0.9438
135.19	C	C (1)	0.9448

Cluster 4

Time	Best	True	Acc
122.72	D	D (1)	0.8854
132.23	D	D (2)	0.8979
134.90	D	D (3)	0.8958
137.47	D	A (1), D (2)	0.9302
139.37	D	D (2)	0.9042
145.61	D	D (3)	0.9281
168.91	D	D (4)	0.9458
182.42	D	D (4)	0.9719

HDDM A performed nearly perfectly from the detection point of view, yet it is instantiated **Cluster 4** for concept D much later than the other detectors. Investigating further in Table 7.21, it shows that a client with concept D was incorrectly assigned to **Cluster 2** at time step 48.75 which had the best concept B, instead of creating a new cluster. Consequently, unlike the results of the MDDM G detector in subsection 7.5.5, it creates the fourth cluster after 122.72 seconds, much later than the results of other detectors.

8

Discussion

This chapter discusses the results presented in chapter 7 in the context of the challenges and design decisions outlined in chapter 4 and chapter 5. The discussion is organized around the three main components of the framework: concept drift detection, drift adaptation and model stabilization, as well as the cluster selection and reassignment procedures.

8.1 Concept Drift Detection

8.1.1 Performance on SINE datasets

For Experiments 1–3, which use SINE1 and SINE2 with abrupt and gradual drift respectively, the majority of detectors achieve close to perfect or perfect detection with low false positive rates. As described in section 6.2, SINE1 and SINE2 feature a complete inversion of the decision boundary between concepts, meaning that a model trained on one concept will perform almost entirely incorrectly on the other. This severity of drift produces a sharp and immediate increase in the online prediction error, which is precisely the signal that error-rate-based detectors are designed to detect. The results in Table 7.1, Table 7.4 and Table 7.8 confirm this; all MDDM variants, HDDM A, HDDM W, FHDDM, and FHDDMS achieve perfect F1 scores across all clients on both Experiments 1 and 2. The introduction of a gradual transition from one concept to the next did not significantly affect the performance of most detectors, however the delay increased substantially. This does not necessarily mean that it was harder to detect, since in a transition period of 200, at 100 batches after the first instance of the new concept the mix is 50/50, meaning that as soon as a near-majority of the batches were from the next concept the detectors signalled a drift.

8.1.2 Performance on CIRCLES

The subtleness of the drift in the CIRCLES dataset tested in Experiment 4 highlighted a fundamental challenge discussed in subsection 4.1.2; correctness of detection depends on the severity of the drift relative to the detector's sensitivity. Error-rate-based detectors, which constitute the entire set of detectors evaluated in this thesis, are inherently limited when the drift does not produce a sufficient change in the prediction error of the current model. The CIRCLES experiment was included

specifically to stress-test this dimension of detector performance, and the results confirm that the chosen class of detectors is insufficient for subtle drift or at the very least require specific hyperparameter tuning. The detector hyperparameters were tuned for MDDM and FHDDM to increase detection sensitivity as noted in Table 6.2, but this was insufficient to achieve detection on CIRCLES. It is likely the case that changing the order of concepts such that e.g. concept 1 (A) is directly followed by concept 4 (D) (see Figure 6.3) would improve drift detection since the drift is more drastic.

In the papers presenting the different detectors [13], [19], [20], [22], [23], [24], they performed fine on the CIRCLES dataset. Since the implementation and hyperparameters are identical (except for MDDM and FHDDM) in this thesis as they were in their original papers, the only distinguishing factor is the learner itself. They all used different classifiers (Naive Bayes, Hoeffding Trees among others), except for the DDM paper [13] which did use a neural network but did not specify its architecture. All this points to an unexpected additional factor in the relationship between the different components discussed throughout the thesis, which is that the type of classifier used seems to be significant in the performance of the detectors and requires additional study.

8.2 Framework Performance

The cluster performance graphs for each experiment indicate that, in general, the clustering mechanism worked as expected. More often than not, the number of clusters matched the number of concepts in each experiment and the configuration lacking any clustering functionality consistently performed worse. An example of the framework working almost perfectly is Experiment 5 for MDDM G, as described in subsection 7.5.5, where despite the difficult drifts there is only one instance of model poisoning. However, there are some interesting cases which highlight the issues and limitations of the framework.

8.2.1 Stability and Cluster Model Selection

Two processes that have very strong influence on the performance of the clustering mechanism is the stability function, which determines *when* a client should find its new cluster, and the cluster selection algorithm which determines the new cluster. If any of these fail, the framework is unable to work properly since different concepts are assigned to the same cluster which leads to poisoning. Although the stability and cluster selection algorithms described in algorithm 4 and algorithm 5 were both loosely based on the existing concept drift literature, they are relatively novel algorithms. This made it difficult to predict the resulting behaviour when combined with more complex drifts and poor detection and adaptation. Additionally, it is very difficult to analyse exactly what went wrong outside of guesswork and speculation, but using the plots and tables one can identify where a mistake was likely the fault of these algorithms rather than that of the detectors. In general, however, they functioned surprisingly well, indicating that the general idea shows promise.

A clear example can be seen in the results for HDDM A in Experiment 5 in subsection 7.5.6, where a client was incorrectly assigned to an existing cluster when it should have created a new cluster. This is likely due to a combination of stability being determined too early after adaptation as well as reassignment not being strict enough to see that no cluster was compatible. The stability being accepted too early is evidenced by the lack of a dramatic drop in accuracy of the cluster model, meaning that the fraction of samples that were concept D was too low during aggregation to incur significant poisoning. In other words, the weighted factor of the drifted client's model weights was significantly smaller than that of the other client when FedAvg was performed(Equation 2.6). This means that the drifted model had not yet trained much on the new concept. Looking at the two concepts B (concept 2 in Figure 6.1b) and D (concept 2 in Figure 6.2b) from the datasets in section 6.2, there is some overlap in the decision boundary, which means that some mistakes are expected to be made in distinguishing them.

An example of incorrect cluster creation is seen in the MDDM G accuracy graph in Experiment 3 in subsection 7.3.3, where an additional cluster was created for concept B despite already being covered by **Cluster 2**. This situation arose because when the newly drifted client was considered stable, **Cluster 2** was evaluated on and compared to whatever the state of the stability mechanism was at that client and determined to be a poor match for some reason, even if the concepts were the same. The fact that the third point of **Cluster 2** and first point of **Cluster 3** being very close in accuracy strongly indicates that this is a result of the stability/cluster selection misfiring.

8.2.2 Impact of Detections

The propagation of errors presented in subsection 4.1.2 and subsection 3.2.3 shows that the performance of the detectors is significant for the correctness of the clustering aspect of the framework. There are many mistakes that a detector may make which have noticeable effects on the clustering.

The most impactful mistake a detector can make is completely missing the drift, which will lead to a drifted client being aggregated with clients of other concepts, leading to severe poisoning. As described in subsection 7.5.2 with the results of Experiment 5 with the DDM detector, this issue does occur. In step 2 of the timeline of events, the reason for the drifted client remaining in **Cluster 4** could be either the detector missing the drift meaning that reclustering is never triggered, or the drift was indeed detected but the client was reassigned to the same cluster. It is much more likely the former, since concept A is already represented by **Cluster 1** and, as stated previously, the clustering mechanism generally works very well. Thus, the effect of poisoning caused by a missed detection resulted in the cluster changing concept entirely on its own, and the old concept is forgotten. This issue then happens again multiple times throughout the entire run.

Experiment 4 in section 7.4 demonstrates an extreme example of what happens when the data makes it difficult to detect drifts, leading to essentially no clustering. Despite the absence of this core part of the framework, performance remains high,

likely because the client models are capable of learning the new concepts sufficiently quickly on their own through online learning. This suggests, for datasets with drifts which are small enough to allow for passive adaptation, that the proposed framework provides no benefit over a conventional FL system with a single global model and only forces the clients to perform unnecessary computations.

Although not as impactful as outright missing drifts, delays are still problematic if the round aggregations happen to occur between a drift and a detection. This will cause poisoning, which in turn may lead to the clustering mechanism to fail or create unnecessary clusters, which leads to higher computational load on the edge during cluster selection. Experiment 3 had some issues with long delays leading to aggregations with drifted clients causing poisoning in the cluster models (subsection 7.3.2), although the fact that this happens during a transition period makes it less impactful since the concept is still technically a mix of both, even if the tables only say if the concept is one or the other.

For false positives, the results show that they are much less impactful and in some cases may even improve the performance of the framework as discussed for the MDDM variants in Experiment 5 in section 7.5. It indicates that an increased number of false positives is beneficial for cluster assignments, because more initiations of the full pipeline results in a better probability of finding the correct cluster.

8.3 Computational Performance

In general, all detectors computed their drift signals in more or less the same amount of time with minimal variance. All detectors except for DDM and EDDM performed similarly across all experiments, despite HDDM (A, W) not using any detection memory. The detector performance results for Experiment 5 in Table 7.14 show that HDDM A had a significantly higher F1 score than the rest. Thus, from a memory perspective, HDDM (A, W) is the most efficient detector while having the same or better detection performance. It should be noted that none of the experiments included datasets with noise, which could mean that some of the strengths of the other detectors, such as FHDDMS, are not utilized and therefore leading to the results showing similar performance.

Although false positives do not degrade the accuracy performance of the clusters in the framework, as shown in subsection 8.2.2, there are other unwanted consequences. Each false positive triggers a full process of the framework of instantiating a new model, stabilizing, and reclustering. This induces unwanted computational steps on the clients and communication rounds with the server. Furthermore, each reclustering throws away the existing model on the client. Thus, the time and computational resources spent training on the data is wasted, since the client model is replaced by the cluster model. Therefore, although false positives are fine from a framework perspective, they are harmful in computation and loss of information. This strengthens the case for HDDM (A, W), as it had significantly fewer FP's in Experiment 5, while only having one more in the other experiments.

Regarding adaptation, the non-windowing detectors such as HDDM (A,W) barely

used any samples for retraining, yet the framework performed just as well with them as it did with other detectors which provided 25 samples or more. This indicates that a bunch of samples for retraining after a fresh model is created is unnecessary, and computational resources are wasted on adaptation outside of creating a fresh set of weights. In cases where the delay is lower than the window size or the time spent between warning and drift, this set of samples is likely hurtful for adaptation, since the window by definition will also include data from the old concept. This does not seem to be a significant issue here, but important to note nonetheless if window sizes would have to be increased in future experiments. It may be the case that this type of adaptation is reasonable in cases where the datasets have very long gradual drifts, since then the number of samples may become significant. This would, however, lead to an increase in computational cost.

One final note is that this thesis did not analyse the computational cost of the cluster evaluation algorithm (algorithm 5), which has the potential to become computationally intensive for the edge. When this process is executed, for every N cluster models in the framework, the client must perform a forward-pass on 25 batches to compute the error rate and standard deviations, which means that if a large number of clusters are created, the batch sizes increase and if the evaluation window size is increased this may become problematic on low-power hardware.

8.4 Future work

8.4.1 Rigorous Stability and Cluster Selection

The stability (algorithm 4) and the cluster selection (algorithm 5) designed for this thesis were only tested empirically, not as a consequence of rigorous mathematical argumentation. Their basis, however, DDM [13], has a foundation of mathematical statistics. The results show that they more often than not worked as intended and allowed for a successful framework; however, there is likely much room for improvement. The framework is designed such that it is agnostic to the implementation of these algorithms, as long as some functionality exists to determine when an adapted client is ready for cluster selection and how the cluster is chosen. This means that an "improvement" to this component may be to create a completely different set of algorithms altogether which use similar methods to other detectors to determine stability and select matching clusters. Another may be to rigorously test and find optimal thresholds for methods which simply train for some number of samples after adaptation or require some maximum error rate which are not dataset-specific.

8.4.2 Cluster Merging, Pruning and Versioning

The framework currently retains all clusters indefinitely. While this was sufficient for the scale of the experiments, it does not scale indefinitely and introduces additional computational overhead during cluster reassignment. As described in algorithm 5, a drifted client must evaluate every existing cluster model on its evaluation window when determining whether a matching concept already exists. Consequently, the cost

of reassignment grows linearly with the number of maintained clusters. Although the memory requirements remain relatively modest because only a single model is stored per cluster, a large number of clusters would increase both storage and reassignment time, which is undesirable in resource-constrained edge environments.

A natural extension is to implement a merging algorithm analogous to the one described in FedDrift [18], which compares cluster models pairwise and merges those whose models perform similarly on each other's data. This would collapse spurious clusters created by false positive detections and reduce the communication overhead of cluster reassignment over time. Complementarily, a pruning mechanism could deprecate clusters that have been empty for a configurable number of rounds. The trade-off between deprecating old cluster models and retaining them for potential recurrent concept reuse would need to be evaluated empirically and is domain specific, as mentioned in subsection 4.4.4 on meta learning.

However, pruning introduces a trade-off between resource usage and cluster reuse. Retaining clusters for longer increases the probability that future clients can reuse an existing model instead of creating a new cluster, but increases the number of cluster models that must be evaluated during reassignment. The appropriate balance is likely application dependent and would need to be evaluated empirically, particularly in domains where recurring concepts are common.

Furthermore, the framework stores a snapshot of every aggregated cluster model throughout the experiments for evaluation purposes, as described in subsection 6.4.2. Each snapshot consists of the cluster model parameters together with metadata. Although these snapshots are currently only used for post-experiment analysis, they form the basis of a model versioning mechanism. In this context, model versioning refers to maintaining a history of previous cluster states, rather than only the most recent model. If a cluster becomes poisoned due to missed or delayed drift detections, the framework could revert the cluster model to a previously recorded snapshot with better performance. Additionally, historical cluster models could be used to initialize new clients or newly created clusters instead of always starting from the same initial model. This may reduce adaptation time when a previously observed concept reappears.

8.4.3 Batch Size and its Effect on Detection

All experiments in this thesis use a batch size of one sample. Larger batch sizes would smooth the error signal and reduce variance in the error rate estimate, potentially reducing false positives. However, a larger batch also mixes samples from before and after a drift boundary within the batch that straddles the transition, artificially inducing a gradual-like error signal even for abrupt drifts. The interaction between batch size, detection delay, and false positive rate is an important practical consideration, particularly for deployment on real vehicle data where sensor readings may arrive at rates that necessitate mini-batching for computational efficiency.

8.4.4 Evaluation on Real-World Vehicle Data

All experiments in this thesis use synthetic benchmark datasets with well-defined concept boundaries. Evaluating the framework on real telemetry data from heavy-duty vehicles, where concept boundaries are unknown beforehand, the types of drift may be unknown, and noise levels are higher, is a necessary step toward any production deployment. Such an evaluation would require ground-truth labelling of drift events by domain experts (meta learning in subsection 4.4.4), which is itself a non-trivial task, and would provide a much stronger basis for assessing whether the framework is viable in the motivating use case described in chapter 1. It may be the case that this framework has potential in being an unsupervised method to identify concepts in data, however much more work would have to be done to verify this, especially when something like a poor detector lead to large numbers of clusters and model poisoning etc.

9

Conclusion

This thesis presented a framework for concept-drift-aware federated online learning, combining error-rate-based drift detection, active adaptation, and clustered federated aggregation to handle staggered concept drift across heterogeneous edge devices. The framework was evaluated across five experiments of increasing complexity using the SINE1, SINE2, and CIRCLES datasets, covering abrupt, gradual, and recurring drift under both global and local drifts. The conclusion is presented against the four operational requirements stated in chapter 3.

Requirement 1 states that drift must be detected quickly to avoid sustained prediction errors and cross-concept contamination. For SINE-type experiments, the best-performing detectors achieved detection delays of 8–22 batches, which is sufficiently fast to limit poisoning. HDDM A in particular achieved a mean delay of 11.7 ± 5.7 batches in Experiment 5 with zero false negatives, representing the best balance of speed and reliability among all evaluated detectors. The requirement is therefore met for severe concept drift, but not for the subtle incremental drift of CIRCLES, where most detectors failed to fire at all and delays exceeded 1000 batches where detection did occur.

Requirement 2 states that all procedures must operate within the computational budget of edge hardware. All detectors tested in this thesis process a batch in under 0.12 milliseconds and require saving at most 100 error rates (which are one float each, i.e. this is on the order of a few hundred bytes), with HDDM A and HDDM W requiring zero error rate storage beyond some running statistics. Adaptation in general did not provide any benefit outside of creating a fresh model, meaning that no batches would have to be saved at all outside of the 25 batches used for stability and cluster selection. These figures are consistent with deployment on low-power embedded hardware, and the requirement is considered met for the detectors that also satisfy Requirement 1. The exception to this is that the cluster selection algorithm may incur computational costs under certain circumstances and optimizations to the hyperparameters or algorithm itself may be necessary for practical use.

Requirement 3 states that the framework must handle both local and global drift without allowing different concepts to interfere during aggregation. The clustering mechanism addresses this directly by separating clients into concept-specific clusters and preventing cross-concept aggregation. For all experiments where detection was reliable (Experiments 1–3 and 5), the cluster models maintained high accuracy and each cluster consistently represented a single concept, including across recurring

drifts. When detection failed (Experiment 4), this requirement was not met, as all clients remained in a single cluster and concepts were mixed during aggregation. The requirement is therefore conditionally met, subject to the drift being detectable by the chosen detector. While the framework was designed specifically for asynchronous local drift, it also automatically solves global drift.

In summary, the framework represents a functioning proof of concept for drift-aware clustered federated online learning that meets all operational requirements under different types of distributed asynchronous concept drift. The primary limitation is the dependence on the quality of the detection algorithm, where poorly tuned detectors lead to big issues for the framework. In cases where the differences between concepts are subtle, the framework could be completely replaced by a single-global-model FL system with online passive adaptation. Further, the novel stability (algorithm 4) and cluster selection (algorithm 5) methods proposed facilitate a basis for automatic determination of when clients should be reassigned to a new cluster as well as how that cluster is determined, however there are likely better solutions. From the datasets used in these experiments, the window-less detectors performed on par or better than the detectors which needed to save error rates. Additionally, saving samples for retraining a fresh model during adaptation gave no real benefit, and can be reduced to just instantiating a new model.

While there is likely a significant amount of future work required to be able to implement a production-ready large-scale framework, this thesis lays the foundation for such a system. A robust and scalable drift-aware framework is able to work within the heterogeneous streaming setting, giving this area of machine learning research true potential.

Bibliography

[1] IBM, *What Is Federated Learning? | IBM*, en, Apr. 2025. Accessed: Nov. 28, 2025. [Online]. Available: https://www.ibm.com/think/topics/federated-learning.

[2] A. A. Awan. "What is online machine learning?" Blog post. [Online]. Available: https://www.datacamp.com/blog/what-is-online-machine-learning.

[3] J. Gama, I. liobait, A. Bifet, M. Pechenizkiy, and A. Bouchachia, "A survey on concept drift adaptation," *ACM Computing Surveys*, vol. 46, no. 4, pp. 1–37, Mar. 2014, ISSN: 1557-7341. DOI: 10.1145/2523813.

[4] D. Bergmann, *What is machine learning?* Nov. 2025. [Online]. Available: https://www.ibm.com/think/topics/machine-learning.

[5] Ibm, *What is gradient descent?* Nov. 2025. [Online]. Available: https://www.ibm.com/think/topics/gradient-descent.

[6] S. C. Hoi, D. Sahoo, J. Lu, and P. Zhao, "Online learning: A comprehensive survey," *Neurocomputing*, vol. 459, pp. 249–289, Oct. 2021, ISSN: 0925-2312. DOI: 10.1016/j.neucom.2021.04.112.

[7] A. Gepperth and B. Hammer, "Incremental learning algorithms and applications," in *European Symposium on Artificial Neural Networks (ESANN)*, Bruges, Belgium, 2016. [Online]. Available: https://hal.science/hal-01418129.

[8] M. Sulaiman, M. Farmanbar, S. Kagami, A. N. Belbachir, and C. Rong, "Online deep learnings role in conquering the challenges of streaming data: A survey," *Knowledge and Information Systems*, vol. 67, no. 4, pp. 3159–3203, Feb. 2025, ISSN: 0219-3116. DOI: 10.1007/s10115-025-02351-3.

[9] I. liobait, M. Pechenizkiy, and J. Gama, "An overview of concept drift applications," in *Big Data Analysis: New Algorithms for a New Society*. Springer International Publishing, Dec. 2015, pp. 91–114, ISBN: 9783319269894. DOI: 10.1007/978-3-319-26989-4_4.

[10] 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," *Multimedia Tools and Applications*, vol. 81, no. 3, pp. 3397–3419, Jul. 2021, ISSN: 1573-7721. DOI: 10.1007/s11042-021-11219-x.

[11] J. Lu, A. Liu, F. Dong, F. Gu, J. Gama, and G. Zhang, "Learning under concept drift: A review," *IEEE Transactions on Knowledge and Data Engineering*, pp. 1–1, 2018, ISSN: 2326-3865. DOI: 10.1109/tkde.2018.2876857.

[12] T. Dasu, S. Krishnan, S. Venkatasubramanian, and K. Yi, "An information-theoretic approach to detecting changes in multidimensional data streams," *Interfaces*, Jan. 2006.

[13] J. Gama, P. Medas, G. Castillo, and P. Rodrigues, "Learning with drift detection," in Sep. 2004, vol. 8, pp. 286–295, ISBN: 978-3-540-23237-7. DOI: 10.1007/978-3-540-28645-5_29.

[14] G. Ditzler, M. Roveri, C. Alippi, and R. Polikar, "Learning in nonstationary environments: A survey," *IEEE Computational Intelligence Magazine*, vol. 10, no. 4, pp. 12–25, Nov. 2015, ISSN: 1556-603X. DOI: 10.1109/mci.2015.2471196.

[15] G. Oliveira, L. L. Minku, and A. L. I. Oliveira, "Tackling virtual and real concept drifts: An adaptive gaussian mixture model approach," *IEEE Transactions on Knowledge and Data Engineering*, 2021, ISSN: 2326-3865. DOI: 10.1109/tkde.2021.3099690.

[16] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y. Arcas, "Communication-Efficient Learning of Deep Networks from Decentralized Data," in *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, A. Singh and J. Zhu, Eds., ser. Proceedings of Machine Learning Research, vol. 54, PMLR, 20–22 Apr 2017, pp. 1273–1282. [Online]. Available: https://proceedings.mlr.press/v54/mcmahan17a.html.

[17] C. Zhang, Y. Xie, H. Bai, B. Yu, W. Li, and Y. Gao, "A survey on federated learning," *Knowledge-Based Systems*, vol. 216, p. 106 775, Mar. 2021, ISSN: 0950-7051. DOI: 10.1016/j.knosys.2021.106775.

[18] E. Jothimurugesan, K. Hsieh, J. Wang, G. Joshi, and P. B. Gibbons, "Federated learning under distributed concept drift," in *Proceedings of The 26th International Conference on Artificial Intelligence and Statistics*, F. Ruiz, J. Dy, and J.-W. van de Meent, Eds., ser. Proceedings of Machine Learning Research, vol. 206, PMLR, Apr. 2023, pp. 5834–5853. [Online]. Available: https://proceedings.mlr.press/v206/jothimurugesan23a.html.

[19] M. Baena-García, J. Campo-Ávila, R. Fidalgo-Merino, A. Bifet, R. Gavald, and R. Morales-Bueno, "Early drift detection method," Jan. 2006.

[20] A. Pesaranghader, H. L. Viktor, and E. Paquet, "Mcdiarmid drift detection methods for evolving data streams," in *2018 International Joint Conference on Neural Networks (IJCNN)*, IEEE, 2018, pp. 1–9. DOI: 10.1109/ijcnn.2018.8489260.

[21] C. McDiarmid, "On the method of bounded differences," in *Surveys in Combinatorics, 1989*. Cambridge University Press, Aug. 1989, pp. 148–188, ISBN: 9781107359949. DOI: 10.1017/cbo9781107359949.008. [Online]. Available: http://dx.doi.org/10.1017/CBO9781107359949.008.

[22] I. Frías-Blanco, J. d. Campo-Ávila, G. Ramos-Jiménez, R. Morales-Bueno, A. Ortiz-Díaz, and Y. Caballero-Mota, "Online and non-parametric drift detection methods based on hoeffdings bounds," *IEEE Transactions on Knowledge and Data Engineering*, vol. 27, no. 3, pp. 810–823, 2015. DOI: 10.1109/TKDE.2014.2345382.

[23] I. Frías-Blanco, J. d. Campo-Ávila, G. Ramos-Jiménez, R. Morales-Bueno, A. Ortiz-Díaz, and Y. Caballero-Mota, "Online and non-parametric drift detec-

tion methods based on hoeffdings bounds," *IEEE Transactions on Knowledge and Data Engineering*, vol. 27, no. 3, pp. 810–823, 2015. DOI: 10.1109/TKDE.2014.2345382.

[24] A. Pesaranghader, H. Viktor, and E. Paquet, "Reservoir of diverse adaptive learners and stacking fast hoeffding drift detection methods for evolving data streams," *Machine Learning*, vol. 107, Nov. 2018. DOI: 10.1007/s10994-018-5719-z.

[25] *What Is Model Performance in Machine Learning? | IBM*, en, Jul. 2025. Accessed: Mar. 23, 2026. [Online]. Available: https://www.ibm.com/think/topics/model-performance.

[26] J. E. Hammond, T. A. Soderstrom, B. A. Korgel, and M. Baldea, "A selective kalman filtering approach to online neural network updating under system drift," *Scientific Reports*, vol. 15, no. 1, Dec. 2025, ISSN: 2045-2322. DOI: 10.1038/s41598-025-24558-8.

[27] M. Abu-Shaira and W. Shi, *Olr-waa: Adaptive and drift-resilient online regression with dynamic weighted averaging*, 2025. arXiv: 2512.12779 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2512.12779.

[28] J. Gama and P. Kosina, "Learning about the learning process," in *Advances in Intelligent Data Analysis X*. Springer Berlin Heidelberg, 2011, pp. 162–172, ISBN: 9783642248009. DOI: 10.1007/978-3-642-24800-9_17.

A

Appendix

A.1 Experiment 1

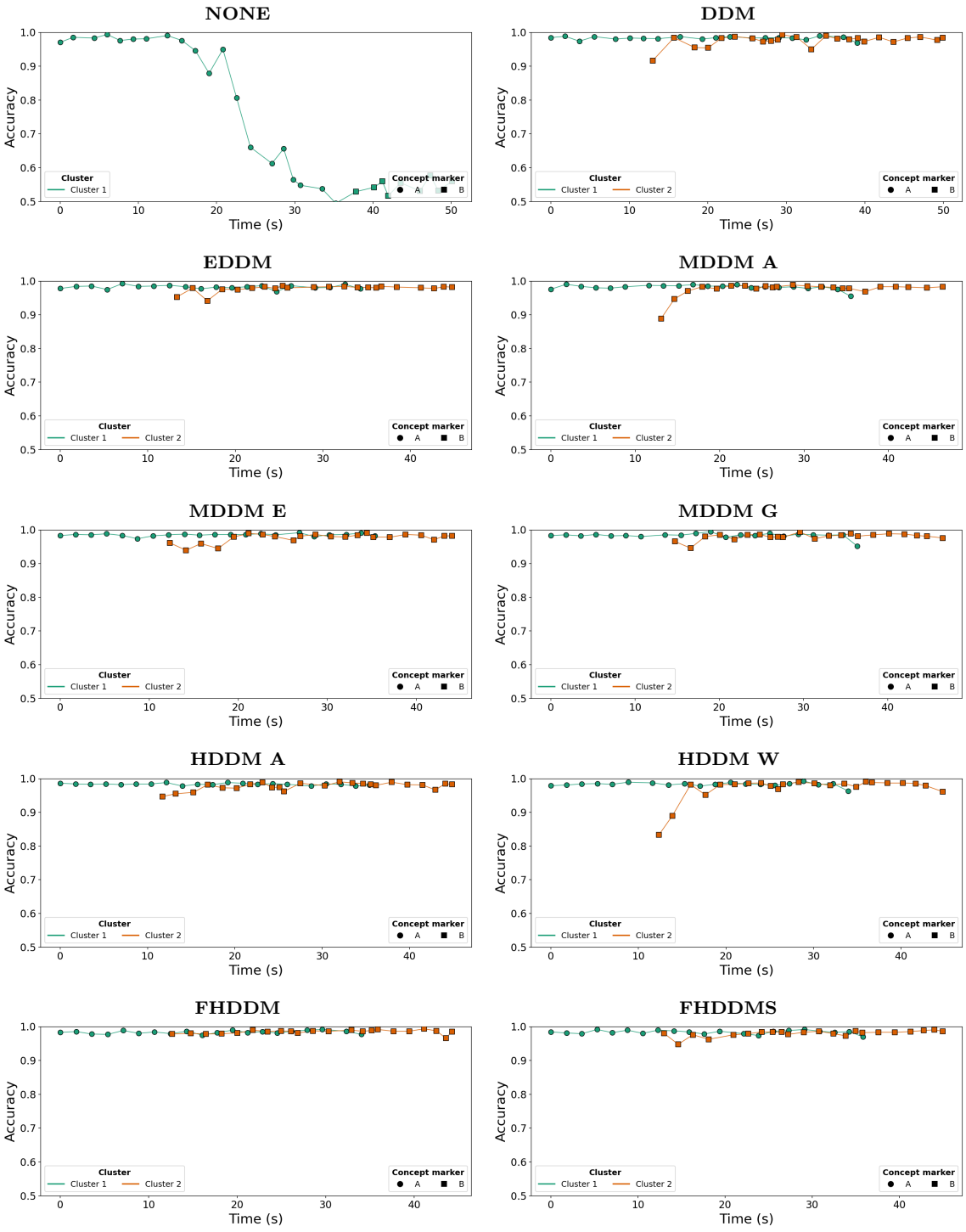


Figure A.1: Cluster accuracy plots for Experiment 1.

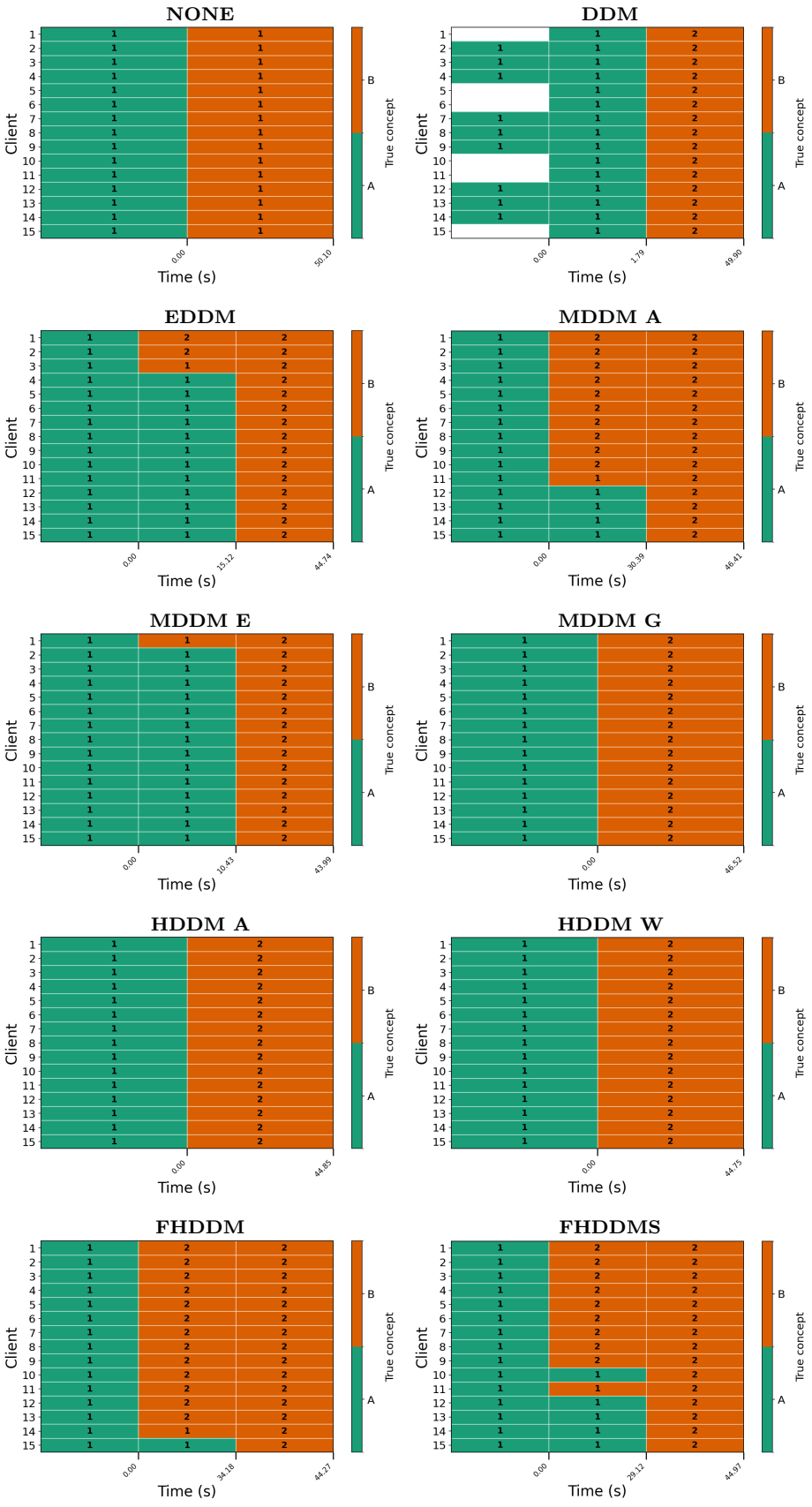


Figure A.2: Concept cluster grids for Experiment 1.

A.2 Experiment 2

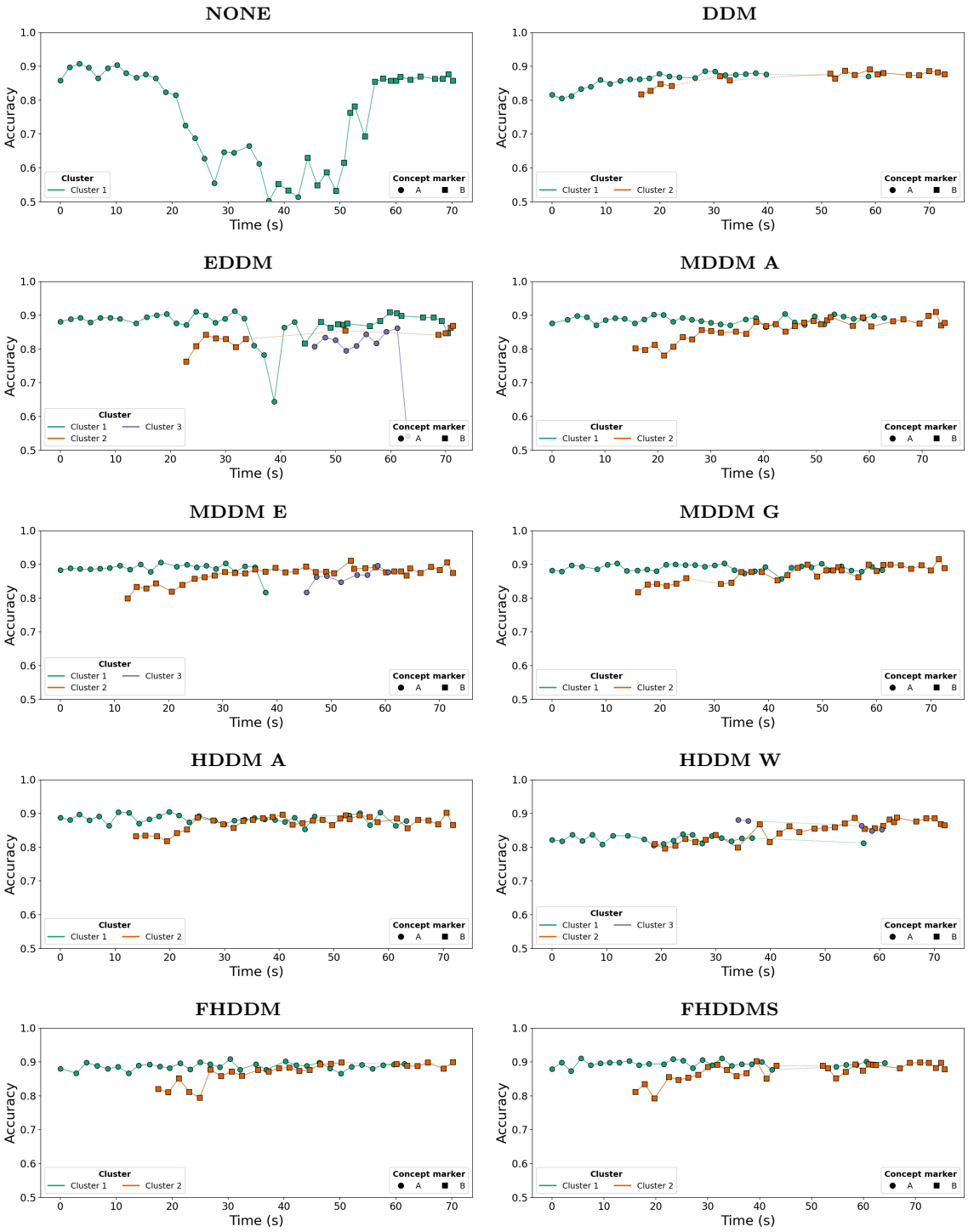


Figure A.3: Cluster accuracy plots for Experiment 2.

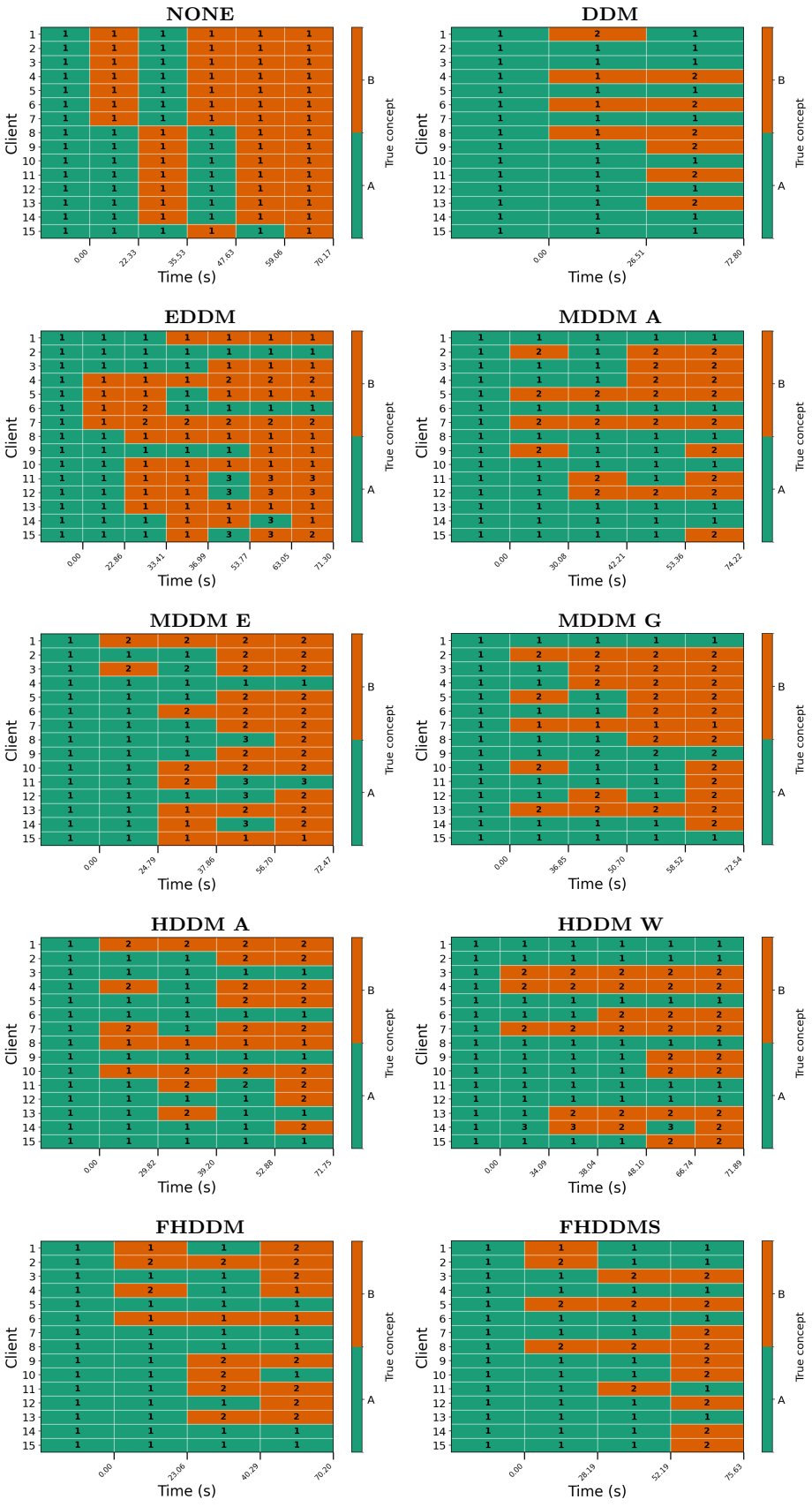


Figure A.4: Concept cluster grid for Experiment 2.

A.3 Experiment 3

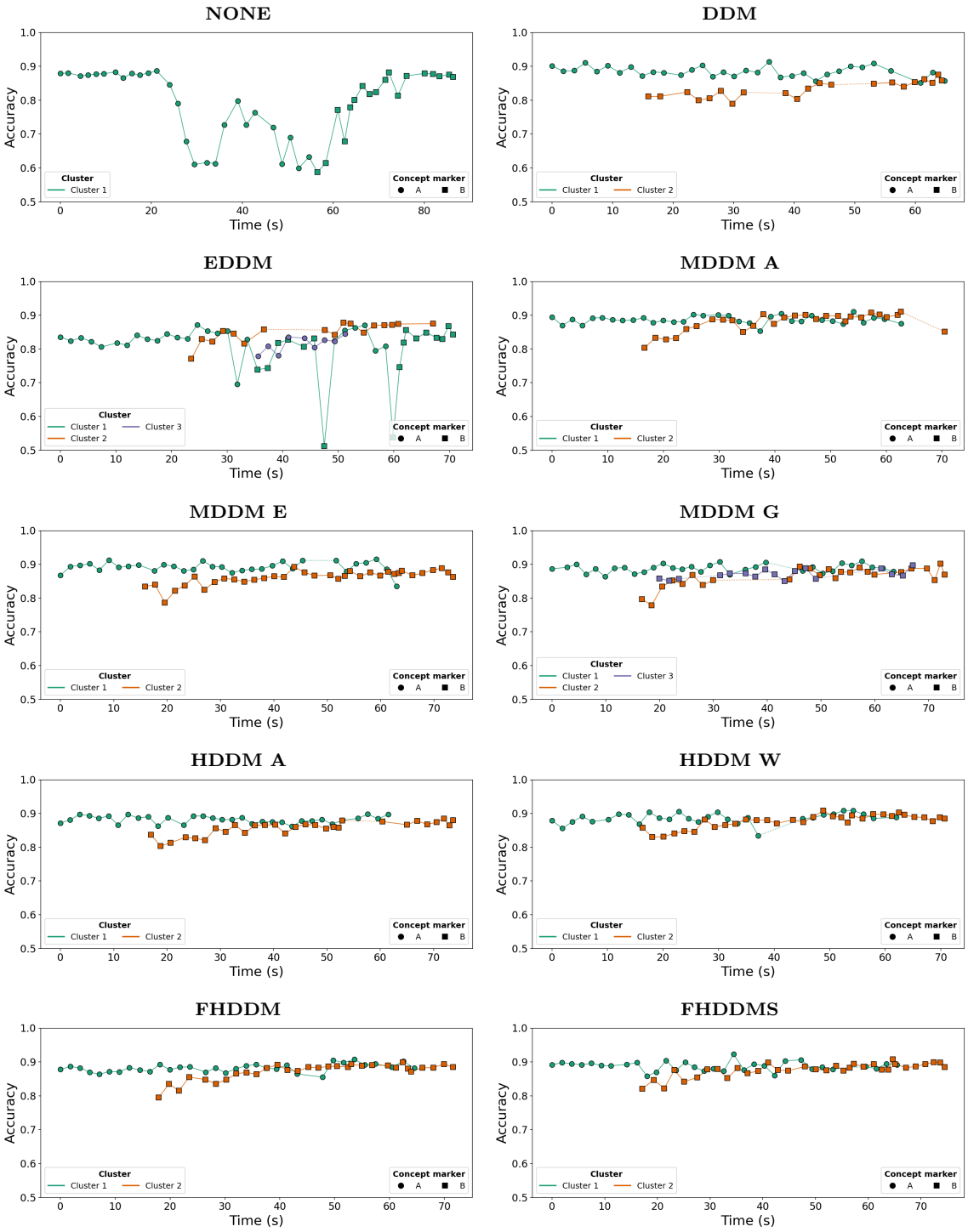


Figure A.5: Cluster accuracy plots for Experiment 3.

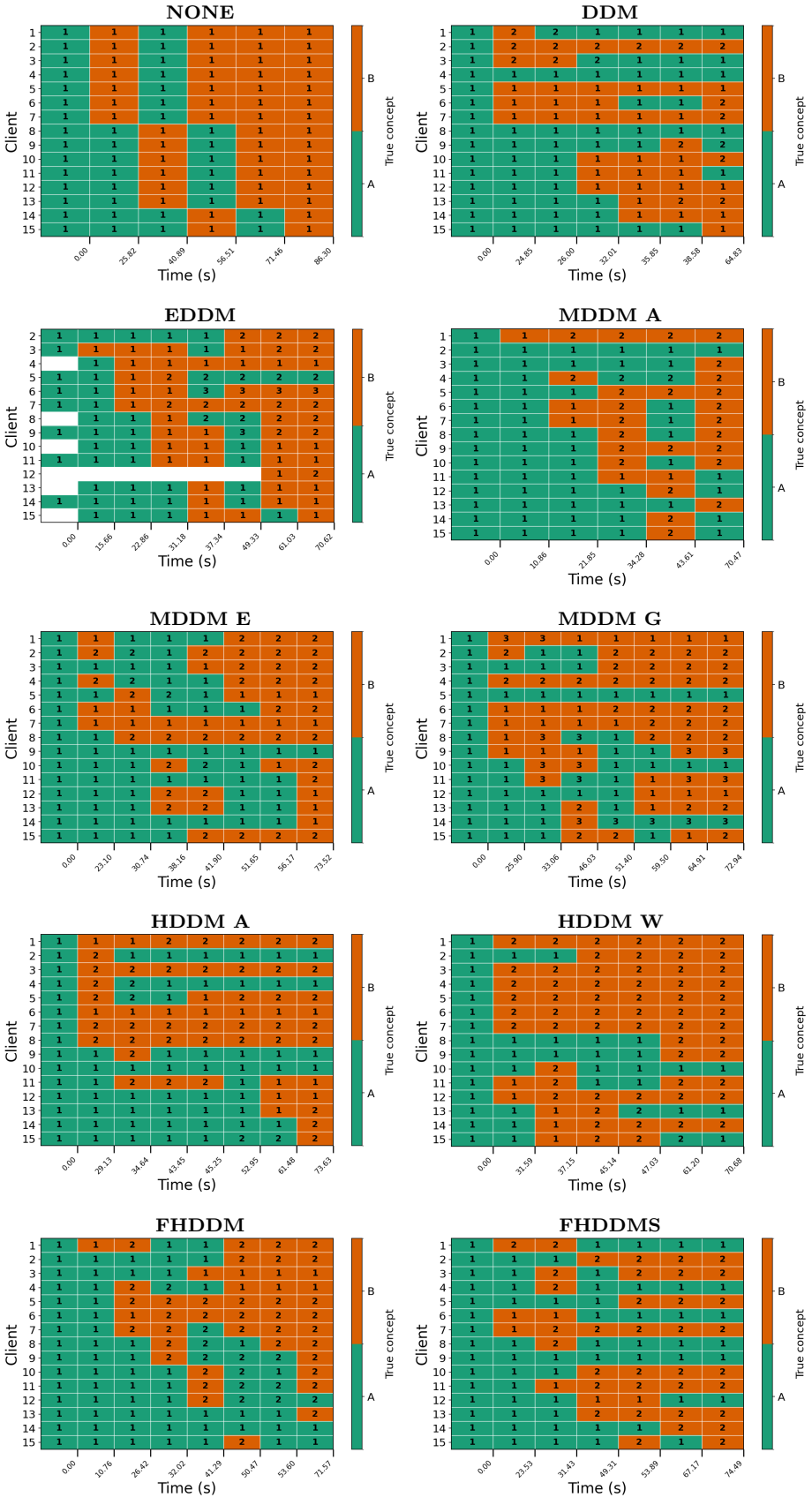


Figure A.6: Concept cluster grids for Experiment 3.

A.4 Experiment 4

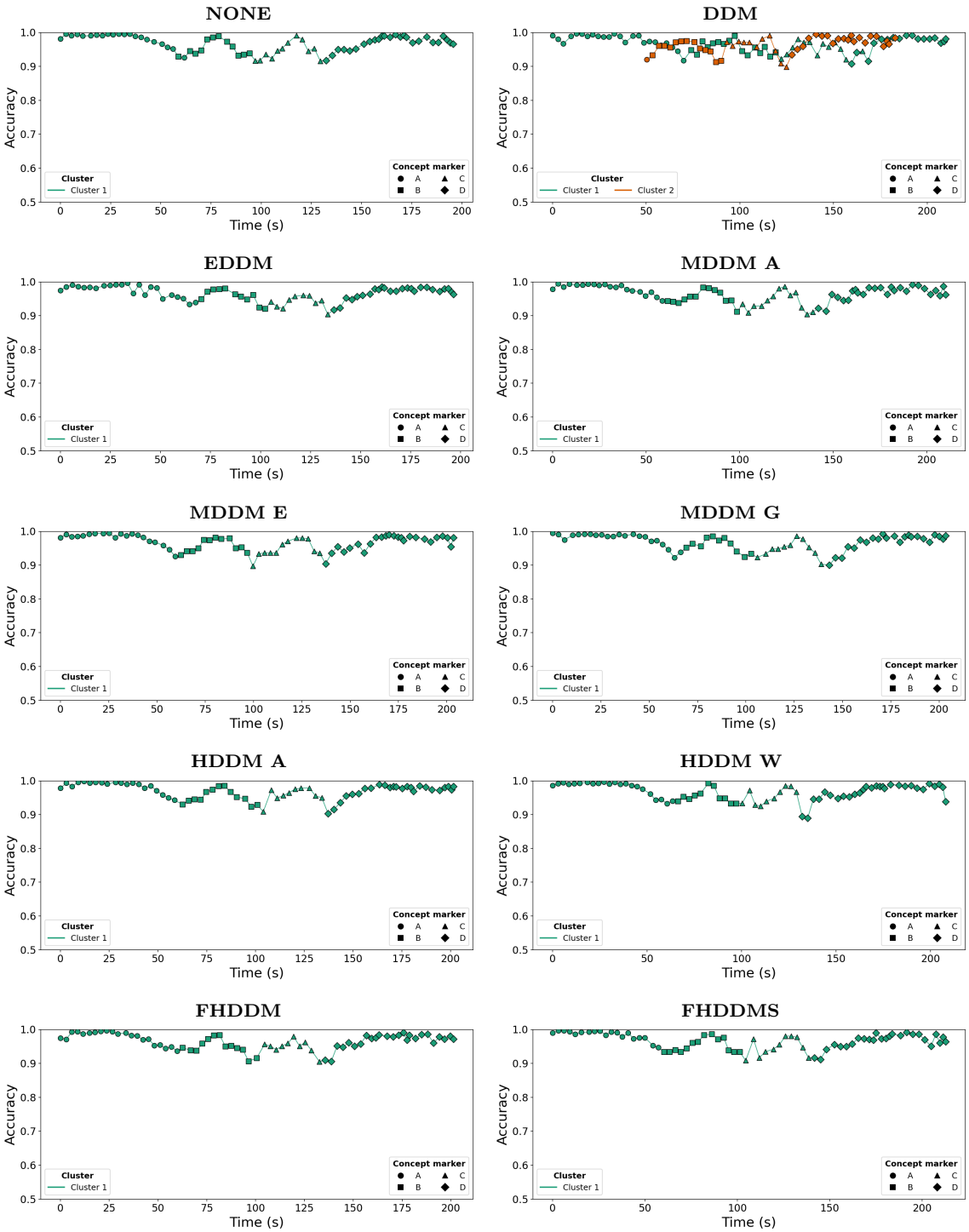


Figure A.7: Cluster accuracy plots for Experiment 4.

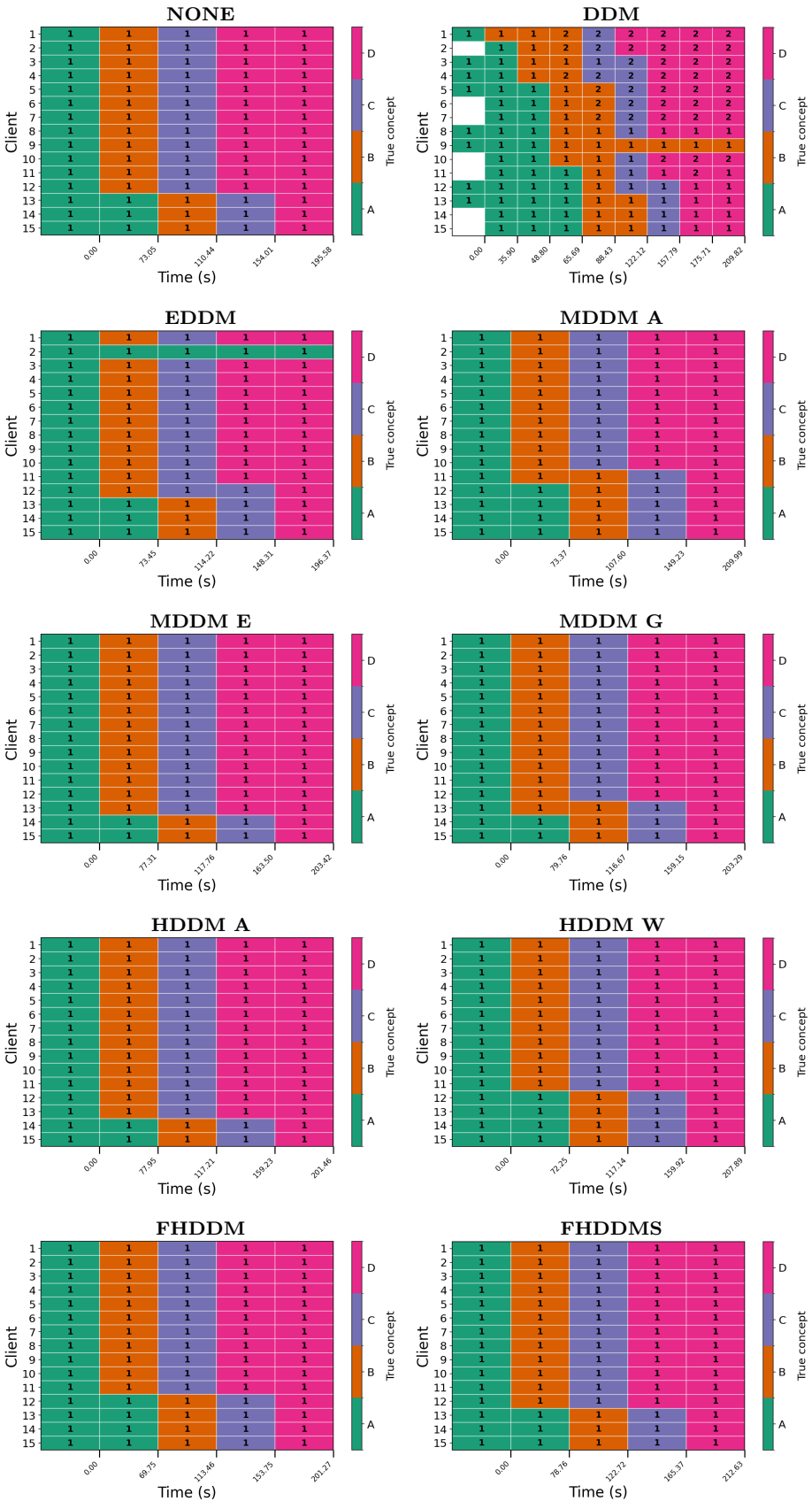


Figure A.8: Concept cluster grids for Experiment 4.

A.5 Experiment 5

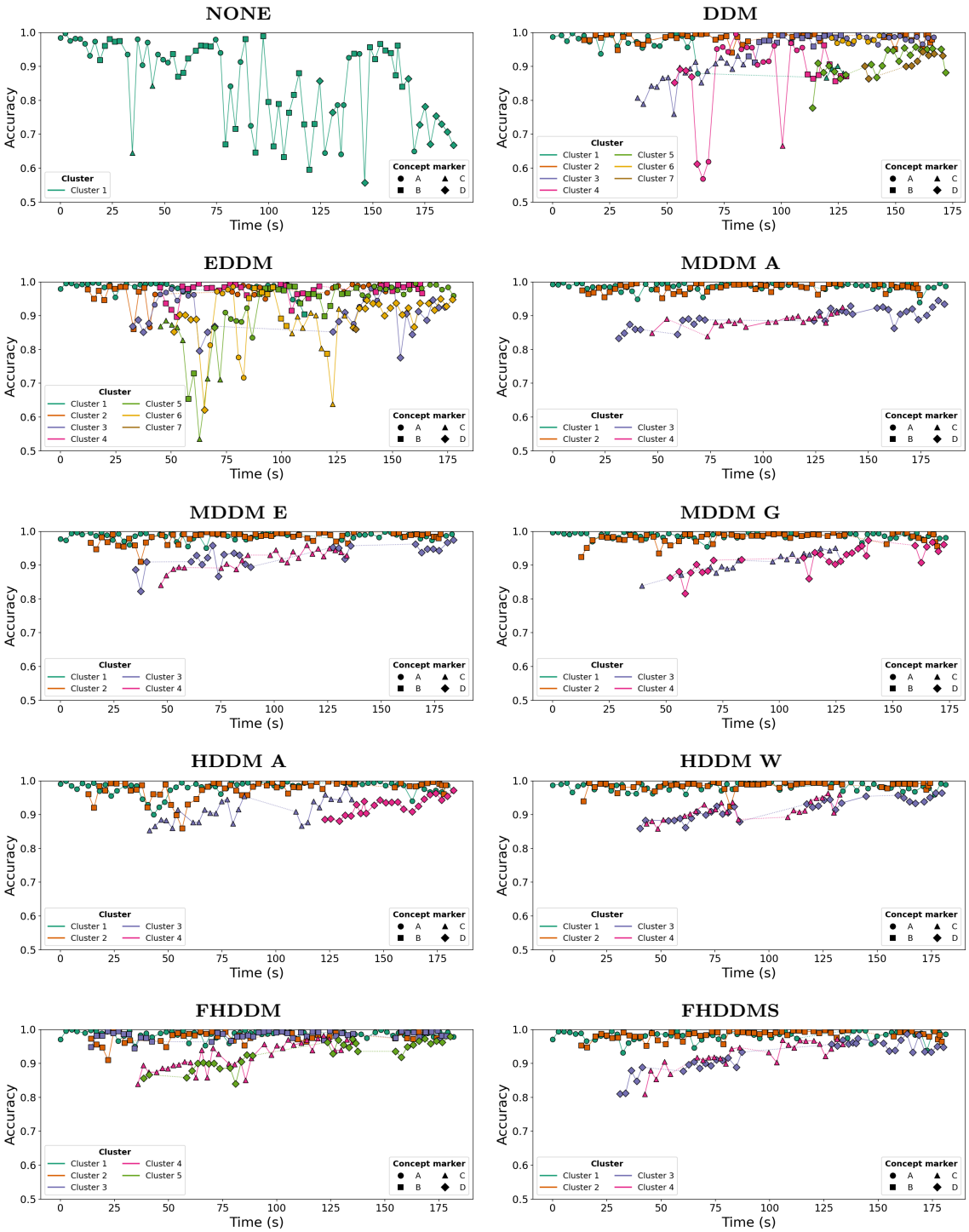


Figure A.9: Cluster accuracy plots for Experiment 5.

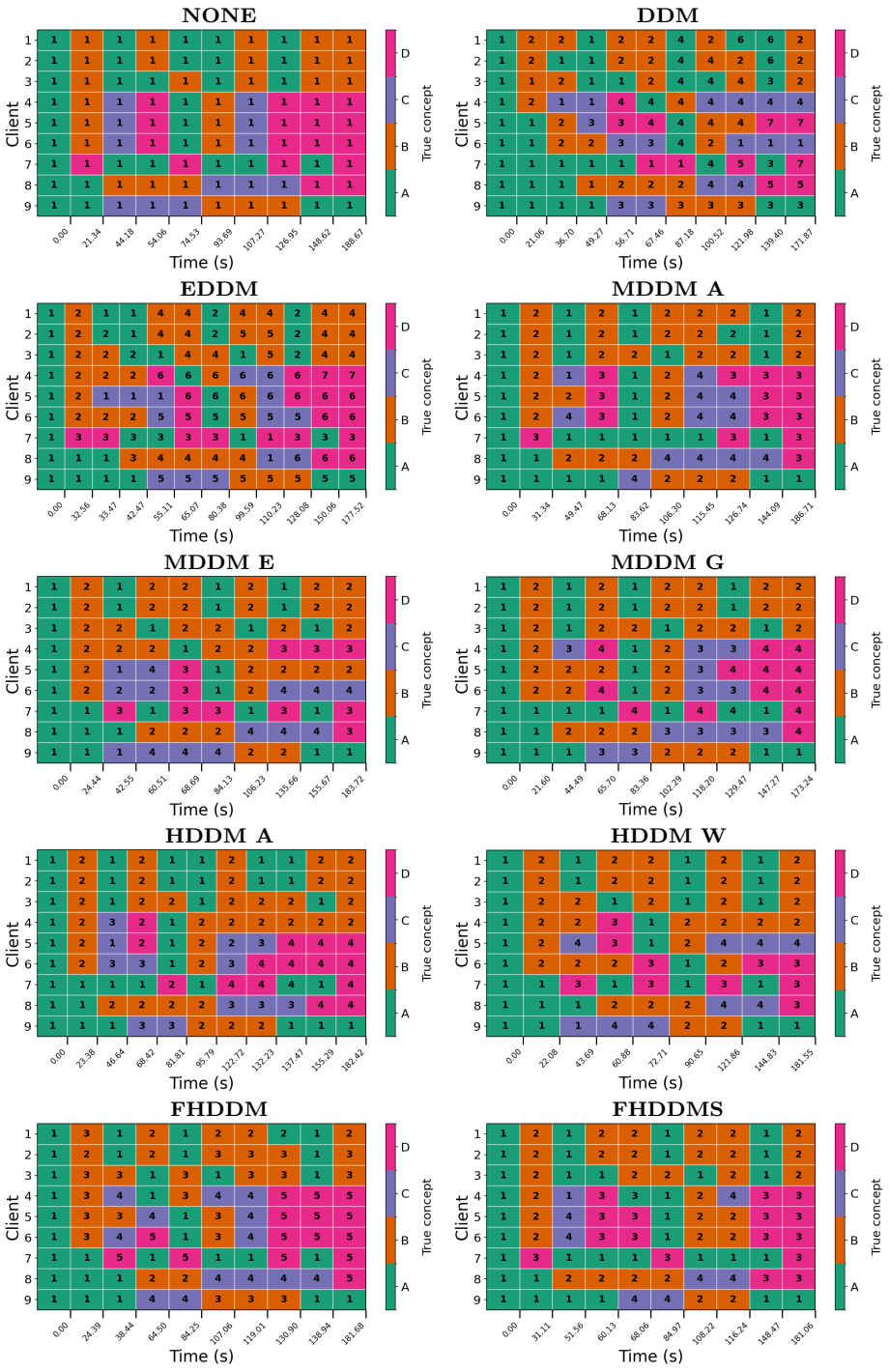


Figure A.10: Concept cluster grids for Experiment 5.