CHALMERS
UNIVERSITY OF TECHNOLOGY

Decentralized Deep Learning under Distributed Concept Drift

A Novel Approach to Dealing with Changes in Data Distributions Over Clients and Over Time

Master's thesis in Data Science and AI

EMILIE KLEFBOM

Master's thesis in Complex Adaptive Systems

MARCUS ÖRTENBERG TOFTÅS

DEPARTMENT OF Mathematical Sciences

CHALMERS UNIVERSITY OF TECHNOLOGY

Gothenburg, Sweden 2023

www.chalmers.se

Master's thesis 2023

Decentralized Deep Learning under Distributed Concept Drift

A Novel Approach to Dealing with Changes in Data Distributions Over Clients and Over Time

EMILIE KLEFBOM
MARCUS ÖRTENBERG TOFTÅS

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Mathematical Sciences
Division of Applied Mathematics and Statistics
Chalmers University of Technology
Gothenburg, Sweden 2023

Decentralized Deep Learning under Distributed Concept Drift
A Novel Approach to Dealing with Changes in Data Distributions Over Clients and Over Time
EMILIE KLEFBOM
MARCUS ÖRTENBERG TOFTÅS

Advisor: Edvin Listo Zec, RISE
Supervisor: Devdatt Dubhashi, Data Science and AI
Examiner: Klas Modin, Applied Mathematics and Statistics

Master's Thesis 2023
Department of Mathematical Sciences
Division of Applied Mathematics and Statistics
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: DALL·E 2 generated image for the prompt: 'Picture of a decentralized neural network with different clustered clients'

Typeset in LaTeX
Gothenburg, Sweden 2023

Decentralized Deep Learning under Distributed Concept Drift
A Novel Approach to Dealing with Changes in Data Distributions Over Clients and Over Time
Emilie Klefbom
Marcus Örtenberg Toftås
Department of Mathematical Sciences
Chalmers University of Technology

Abstract

In decentralized deep learning, clients train local models in a peer-to-peer fashion by sharing model parameters, rather than data. This allows collective model training in cases where data may be sensitive or for other reasons unable to be transferred. In this setting, variations in data distributions across clients have been extensively studied, however, variations over time have received no attention. This project proposes a solution to address decentralized learning where the data distributions vary both across clients and over time. We propose a novel algorithm that can adapt to the evolving concepts in the network without any prior knowledge or estimation of the number of concepts. Evaluation of the algorithm is done using standard benchmarks adapted to the temporal setting, where it outperforms previous methods for decentralized learning.

Keywords: Federated Learning, Decentralized Learning, Machine Learning, Data Heterogeneity, Non-IID, Personalization, Concept Drift

Acknowledgements

We would like to extend our gratitude to the deep learning team at RISE. Edvin, Olof, Martin, John, and Aleksis, thank you for welcoming us into your team and for all your patience and advice. You have all helped us grow as engineers and pushed us to take this project to the next level. We also want to thank our friends from TBK who ensured there was always something to look forward to and someone to have a beer with when we needed a break. Lastly, we will include a quote that helped us through hours of debugging and became our own inside joke for the project.

"It just works" - Todd Howard

Emilie Klefbom, Marcus Örtenberg Toftås, Gothenburg, June 2023

List of Acronyms

ANN Artificial Neural Network.

CNN Convolutional Neural Network.

`DAC` Decentralized Adaptive Clustering.

FL Federated learning.

`HAST` Hierarchichal Aggregation with Similarity based Tuning.

IID Identically and independently distributed.

Glossary

`FedAvg` In this paper, we use the term `FedAvg` to refer specifically to the aggregation method used in Federated Averaging [1], rather than the whole algorithm. For more information see section 2.2.

batch size The number of training examples used in a single iteration of backpropagation of a machine learning algorithm.

catastrophic forgetting When a machine learning model forgets knowledge about previous tasks i.e. performance on previous tasks drops as models continue to train on new data.

client Used to denote one node/user in a distributed network.

continual learning Refers to setting in Machine Learning where models have to learn from a continuous stream of data.

curse of dimensionality Refers to the phenomenon in machine learning where increasing the number of features or dimensions in a dataset can lead to computational challenges and deteriorating model performance due to the sparsity of data.

decentralized learning Denotes the machine learning setting where both models and model training is distributed across multiple clients and communication is not facilitated by one central point.

deep learning Refers to the field of machine learning that deals with layered models such as artificial neural networks, convolutional neural networks, transformers, etc.

drift See section 2.4.

empirical risk minimization (ERM) is a machine learning principle that minimizes average loss/error over a dataset.

federated learning Denotes the machine learning setting where both models and model training is distributed across multiple clients and communication is facilitated by one central node.

hyperparameters Are user-set parameters in machine learning that control the model/algorithm behavior, training, and performance. Examples include learning rate and regularization strength.

perceptron A basic unit of artificial neural networks. A single perceptron can be seen as a binary classifier.

shift See section 2.4.

spatial In this paper used to refer to variations in data distributions occurring across clients in the decentralized network.

temporal In this paper used to refer to variations in data distributions over time for one or several clients in a decentralized network.

transfer learning A machine learning technique that leverages knowledge from one task to improve performance on a related task, allowing pre-trained models to be adapted and fine-tuned for different but related tasks.

Contents

1 Introduction

1.1 Background

With the drastic spread of smartphones and other devices capable of continuously collecting and transmitting data, the amount of available data for machine learning has increased immensely. However, in some of these cases, data cannot be shared explicitly. First, users and businesses may not want to share private or business-sensitive data due to privacy concerns. Second, regulations like GDPR [2] or other data protection acts may restrict data sharing. Lastly, practical limitations such as large distributed datasets or low network bandwidth may make transmitting data to a centralized location infeasible.

In such scenarios, Federated learning (FL) algorithm can be a viable solution as its scalability and applicability in various domains has already been demonstrated in, hospitals [3], retail stores [4], and companies such as Google [5]. The FL algorithm, allows collaboration between multiple clients to train one shared global model without exchanging their local data. However, FL often relies on a central node to coordinate communication among the clients, which can cause communication bottlenecks and single points of failure. To overcome these limitations, Decentralized learning proposes a peer-to-peer communication protocol that eliminates the need for a central node and thereby reduces the vulnerability of the system. However, decentralized learning also poses new challenges, such as how to optimize the client models in a decentralized manner.

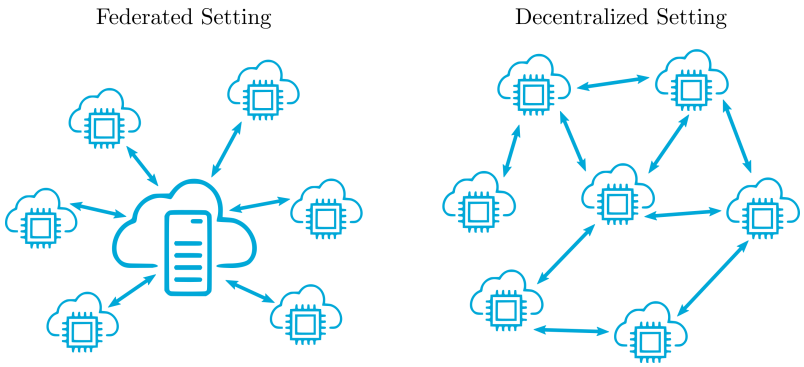


Figure 1.1: The left image symbolizes a federated setting in which a central server coordinates all communication on the network, while the right image presents the decentralized approach with peer-to-peer communication.

Traditionally, decentralized machine learning is based on consensus optimization, where clients converge to a common model using a gossip learning approach [6]–[8]. This works well when the data distributions across clients are similar, however, when the data distributions or tasks differ significantly across clients, consensus optimization can be detrimental. Therefore, recent work has suggested viewing decentralized learning as a clustering problem, where clients try to find suitable collaborators in a network of peers and avoid merging their models with dissimilar ones [9], [10]. All of these works consider Non-IID data in decentralized deep learning, however, they still assume that the data distributions are stationary in time. In real-world scenarios, this is often not the case for edge devices that continuously collect new data.

1.2 Contributions

This work presents the first study of decentralized deep learning with distributed concept drift, which accounts for the dynamic nature of data distributions over time. We propose a novel algorithm that allows clients to learn personalized models in Non-IID settings where their concepts may evolve over time. Our problem setting is related to the recent work of [11], which investigates Non-IID data across clients and time in a federated learning setting. Additionally, we provide an extensive overview of Non-IID settings for temporal and spatial heterogeneity for decentralized algorithms and suggest a coherent naming convention.

1.3 Problem Formulation

We assume a fully connected network $\mathcal{N}$ onto which we apply a distributed learning scenario. On it, we have K clients: $\mathcal{C} = \{C_1, ..., C_K\}$, each with its own deep neural network f_k with model parameters $w_k \in \mathbb{R}^d$. There exist N different data distributions, which we denote through a cluster identity $n \in \{1, ..., N\}$. We can describe these distributions as $\mathcal{P}(x, y) = \{P_1(x, y), ..., P_N(x, y)\}$ where x are the input features and y the data labels.

Throughout all scenarios, model training is split into T time steps, each consisting of some number of training rounds. At which each client k may draw a new local training set $D_k^t(x, y)$ from one of the distributions, ensuring $D_k^t(x, y) \in \mathcal{P}(x, y) \mid \forall k, t$. During each round of training, all clients sample m other clients based on some function and receive the sampled model parameters. The client then aggregates these with its local parameters before training locally for a set number of epochs. After completion by all clients, the next training round begins.

Distributed concept drift occurs when $D_k^t(x, y)$ and $D_k^{t+1}(x, y)$ are not drawn from the same distribution $P_n(x, y)$. The problem at hand is to define a decentralized learning algorithm that utilizes the local data and model of each client to train the weights w_k in order to solve the optimization problem described in equation 1.1. Formally it becomes a statistical learning problem in which we want to map a feature space X to a label space Y. This mapping $h : X \longrightarrow Y$ is called the hypothesis and it represents the ideal solution for the model, as it correctly maps all features to their corresponding labels. Throughout training the weights are updated to more closely resemble the hypothesis, and during validation, the predicted labels $\hat{Y}$ are compared with the expected outputs Y to calculate a loss value. Here the statistical learning attempts to minimize the risk, i.e. maximize the probability distribution that $\hat{Y} \equiv Y$, as it would indicate that the model has achieved perfect performance. As calculating the true distribution is difficult, we instead use empirical risk minimization to approximate it, so that the models could improve over time.

$$\min_{w_k \in \mathbb{R}^d} \mathcal{L}(f_k(w_k)) := \min_{w_k \in \mathbb{R}^d} \mathbb{E}_{x_k \sim D_k}[\mathcal{L}(f(x_k; w_k); y_k)] \tag{1.1}$$

1.4 Limitations

With time as a limiting factor in this research-oriented study, some areas were left unexplored. The decentralized algorithm should in theory be problem agnostic and be able to work efficiently for multiple domains but during this project, testing has only been performed on supervised image classification, therefore the performance on other domains is unknown. Further, we use datasets and data augmentations that are common in the research field. This includes modifying the datasets to simulate Non-IID distributions. These datasets however do not accurately represent the nuances of real-world data [12].

We use a small CNN model for each client in all experiments. The model is not designed to achieve state-of-the-art performance on the supervised tasks, but rather to have sufficient capacity to solve the tasks while exploring decentralized temporal drifts. Therefore we do not know how the performance is affected by more complex models. Lastly, several hyperparameters have been set to estimated values without extensive hyperparameter tuning.

Simplifications were also made on a system level. All clients were simulated as objects on a single device. Different hardware capabilities, network latencies, and other system variations that influence performance have not been explored. In all simulations, clients were also considered to be fully connected. These simplifications would in a real-world application be highly unrealistic, however, this is common practice in decentralized research, and simulating the system falls outside the scope of the project.

2
Theory

In this chapter, we will cover the theory required for a deeper understanding of the proposed algorithm and its different components. To begin with, we will give a brief introduction to deep learning, federated learning, and decentralized learning. Secondly a more in-depth description of data heterogeneity and what Non-IID strategies exist in a federated setting. Finally, a short description of two papers that have been highly influential for our work.

2.1 Deep Learning

Deep learning refers to layered machine learning models based on Artificial Neural Networks (ANN) that learns deep representations based on large amounts of data. ANN's consist of multiple layers of perceptrons, which allows data to be transformed into multi-dimensional spaces, wherein it can be more easily separated, allowing for solutions to complex issues such as self-driving vehicles, speech recognition, and image classification.

2.1.1 Artificial Neural Networks

In a standard ANN, as illustrated in figure 2.1, an input vector $\mathrm{X} = \{x_1, ..., x_n\}$ with n number of inputs, is propagated through the network. Each perceptron utilizes its own weights $\mathrm{W} = \{w_1, ..., w_n\}$ and bias b to calculate an output $\mathcal{O}$ through equation 2.1. The outputs from all perceptrons in a layer are then sent through a non-linear activation function and are then used as the input for the following layer.

$$\mathcal{O} = b + \sum_{i=1}^{n} w_i \cdot x_i \tag{2.1}$$

This continues until the last layer where a specific activation function is used based on the model's task to get the network's final predictions $\hat{Y}$. In this project, we looked at multi-class image classification, in which the most commonly used function is the softmax function. It works by normalizing all outputs and converting them into a probability distribution vector, from which the output with the highest value is chosen as the predicted class.

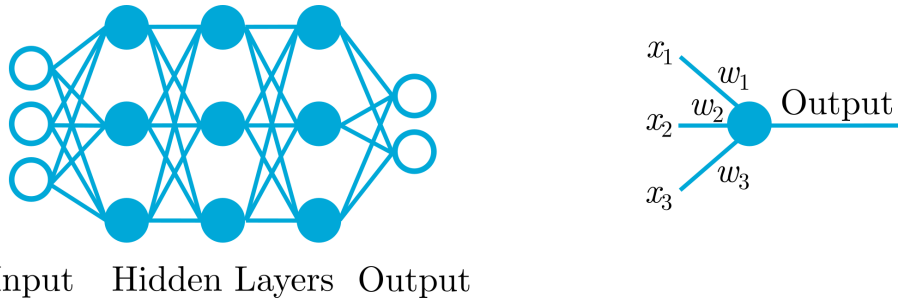


Figure 2.1: To the left is an illustration of a fully connected ANN with three hidden layers made of perceptrons. To the right, is an illustration a perceptron receiving three different inputs x_1, x_2, and x_3 their corresponding weights w_1, w_2 and w_3.

For image classification, the number of original inputs n is typically based on the image size so that n is the total amount of pixels multiplied by the number of color channels. The final layer has m perceptrons correlating to the total number of predicted classes. Meaning that the index of the output with the highest probability directly correlates to the predicted class.

2.1.2 Backpropagation

Artificial neural networks are typically initialized at random, which means that the likelihood that the network properly predicts correct results at initialization is near zero, thus, training is required. One of the most common methods for this is stochastic gradient descent, in which the individual perceptrons' weights and biases get updated through *backpropagation*. During this, the neural network iteratively adjusts its parameters so that the difference between predicted output and true output approaches zero. This update is often done using a *loss function* where correct predictions are assigned lower values and incorrect predictions are assigned higher values.

The goal of backpropagation then becomes an optimization problem where the goal is minimization of the total loss. In our case, we have a multiclass classification and therefore use the Negative Log Likelihood Loss described in equation 2.2. Here x is the input, y is the target label, w is the weight, and N is the batch size.

$$\mathcal{L}(x, y) = \sum_{n=1}^{N} \frac{1}{\sum_{n=1}^{N} w_{y_n}} l_n, \qquad l_n = -w_{y_n} x_{n,y_n} \tag{2.2}$$

By rewriting this loss function on the more common form $\mathcal{L}(y, \hat{y})$, the naming of the function becomes more intuitive, see equation 2.3.

$$\mathcal{L}(y, \hat{y}) = -\sum_{n=1}^{N} y_n \log(\hat{y}_n) \tag{2.3}$$

With the rewritten loss function, the gradient with respect to the model parameters can then be computed as seen in equation 2.4.

$$\nabla_w \mathcal{L}(y, \hat{y}) = \frac{\partial \mathcal{L}(y, \hat{y})}{\partial w} \tag{2.4}$$

Finally, the weights are updated in the direction of the negative gradient, see equation 2.5. Here, w_{t+1} corresponds to the new weights, w_t the previous values, and η the learning rate, a hyperparameter that determines how much the values can be changed in each backpropagation.

$$w_{t+1} = w_t - \eta \nabla_w \mathcal{L}(y, \hat{y}) \tag{2.5}$$

2.1.3 Convolutional Neural Networks

Although ANNs perform well in many areas, they don't scale well to some domains. These include image- and video analysis and natural language processing[13]. In the case of image analysis, the issue stems from that ANNs input typically requires at least one perceptron per pixel and channel, meaning that if a training set has m color images, each with dimensions n by n-pixels, the ANN would require $m \cdot 3 \cdot n \cdot n$ weight updates via backpropagation per training round, which quickly becomes costly to calculate.

For these tasks, Convolutional Neural Networks (CNN) are most commonly used. These use convolutional layers which apply a kernel to the input data to extract its main features, see figure 2.2. The kernel works as a moving window that looks at different sets of pixels throughout the image and converts them into a feature space matrix. The new matrix is commonly smaller than the original and contains spatial information of the image features, effectively reducing the number of weights needed. By stacking convolutional layers, we can extract more complex features such as faces, numbers, or animals.

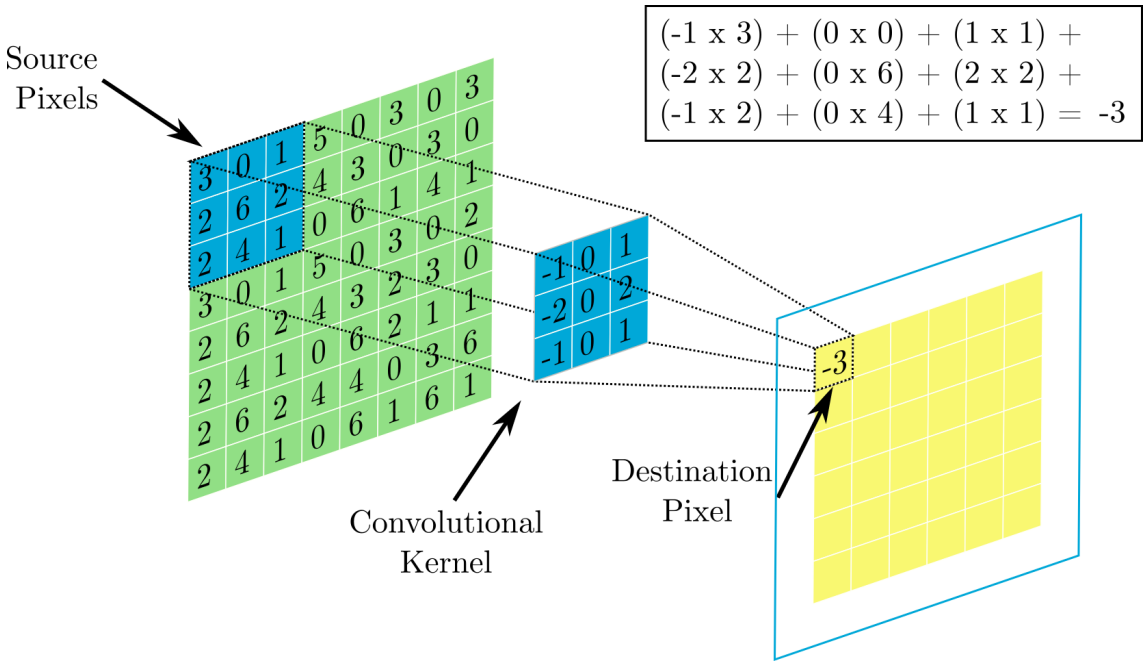


Figure 2.2: Illustration of convolutional kernel. The kernel is applied to the source pixels and the equation in the top right describes the computations performed to get the resulting value of the destination pixel.

2.2 Federated Learning

The concept of Federated learning (FL) stems from a 2016 publication [1]. It is an algorithm for training a decentralized network without explicitly sharing any data. In it, a central node coordinates communication by sampling subsets of clients and aggregating their model parameters to form new global models. These are then distributed back to all clients to use as their new local models. The algorithm they proposed, `FederatedAveraging`, can be seen in algorithm 1. In it, we can see that the algorithm calculates the weighted model average based on the client dataset size in order to create a new model. Giving preference to models coming from clients with larger local datasets. Though the algorithm itself is not used in our study, the aggregation step will be reused under the name `FedAvg`.

Since 2016, additional algorithms which utilize different client sampling- and aggregation methods have been proposed. Some of these will be covered later in this text, but the fundamental structure of the FL algorithm still stands in most publications.

Algorithm 1: `FederatedAveraging` [1]

w_k: model parameters on client k
D_k: local data on client k
m: number of sampled clients at each round
d_k: number of datapoints on client k
$\mathcal{C}$: set of all clients $\{C_1, ..., C_K\}$

`FederatedAveraging()`:
- initialize w_{base}
- **for** each client $C_k \in \mathcal{C}$ **do**
  - $w_k \leftarrow w_{base}$
- **for** each round $r = 1, 2, ...$ **do**
  - $S \leftarrow$ Random subset of m clients $\in \mathcal{C}$
  - **for** each client $C_j \in S$ **do**
    - $w_j \leftarrow$ `ClientTraining`(j, w_j)
  - $w_{avg} \leftarrow$ `FedAvg`(S) //Aggregation Step
  - **for** each client $C_k \in \mathcal{C}$ **do**
    - $w_k \leftarrow w_{avg}$

`FedAvg`(S):
- $\vec{w} \leftarrow \{w_j \mid w_j \in S\}, \quad \vec{d} \leftarrow \{d_j \mid d_j \in S\}$
- $\vec{d} \leftarrow \vec{d} / \sum \vec{d}$
- $w_{avg} \leftarrow \sum_{j}^{m} \vec{w}_j \cdot \vec{d_j}$
- **return** w_{avg}

`ClientTraining`(j, w):
- $\mathcal{B} \leftarrow$ (split D_j into batches of size B)
- **for** each local epoch e $= 1, 2, ...$ **do**
  - **for** batch $b \in \mathcal{B}$ **do**
    - $w \leftarrow w - \eta \nabla_w \mathcal{L}(w; b)$
- **return** w

2.3 Decentralized learning

As mentioned in the introduction, FL has some limitations. To counter these, decentralized learning, instead, removes the need for a central server in favor of peer-to-peer communication. Decentralized learning also caters naturally to personalization problems as each client trains their own local model instead of sharing one or several global models. However, as there is no longer any central node facilitating client communication, new communication schemes are needed. The trivial implementation is referred to as **gossip learning** [6], in which the individual clients sample a subset of other clients to communicate with at random.

Our version of gossip learning, which we use as a baseline, is called `Random` [9] and is described in detail in section 3.3. Even though the decentralized peer-to-peer idea is newer than federated learning, different algorithms have already been proposed that improve on the trivial solution. One of these methods is decentralized adaptive clustering `DAC`, which will be covered later in this chapter.

2.4 Proposition of Non-IID Terminology

This project deals with Non-IID data, data that is heterogeneous across clients or time, which we will refer to as the spatial and the temporal settings. This is equivalent to different clients sourcing their local data from different distributions or the distribution of our training data changing over time.

As this is an emerging field of research, the existing terminology overlaps and contradicts itself, making it difficult to efficiently refer to specific cases. Thus, we propose the following terminology as a comprehensive and overlooking summary of the different Non-IID settings. We have attempted to reuse as much of the existing terminology as possible but included additional terms to clarify differences between spatial and temporal Non-IID settings. For reference purposes, we have included other commonly used terminology in parentheses.

In this nomenclature, the same type of heterogeneity will get the same name in both spatial and temporal settings but will have separate suffixes. When the data distribution varies among clients, the suffix *shift* will be used, and when it varies over time *drift* will be used. For example, when observed labels vary it is referred to as a *covariate* heterogeneity and would be called *covariate shift* if it varies among clients, and *covariate drift* if it varies over time. In settings where both *drift* and *shift* occur simultaneously we will use the term *Distributed Drift* as is done in [11]. In the case of e.g. the *Covariate* case this would then be denoted as *Distibuted Covariate Drift*. Lastly, to refer to a setting where more than one type of Non-IID heterogeneity occurs we propose using the terminology *concept*. For example, if both labels and observed data distributions vary across clients this would be a *Concept Shift*.

From here the following mathematical notation will be used:

- x : observed data. For example, the observed images from a smartphone may depend on the user and the phone model, age, quality, etc.
- y : observed labels. Labels here refer to the labels assigned to the data for our deep-learning models. In short, it can be described as "what the data depicts".
- $P^i(), P^j()$: Here we refer to the data distributions that clients i and j pull their data from.
- $P^t(), P^{t+k}()$: Here we refer to data distribution used by all clients, at some time t and t+k.
- P_i, P_j: Refers to two distinct data distributions.
- P_t, P_{t+k}: Refers to some data distribution at time t and t+k.

In addition to describing each setting mathematically, we have also included some real-world examples. For each setting, only the behavior specified by the mathematical formulas is guaranteed.

- **Covariate shift** (Feature distribution skew):
  $P^i(x) \neq P^j(x), P^i(y|x) = P^j(y|x)$
  Difference in observed data across clients. For example, in a handwriting recognition domain, users who write the same words might still have different stroke widths, slants, etc.
- **Covariate drift** (Virtual concept drift):
  $P^t(x) \neq P^{t+k}(x), P^t(y|x) = P^{t+k}(y|x)$
  Difference in observed data across time. Consider that you are training an autonomous car, this situation happens, for instance, when clients move into places or regions previously unseen to them.
- **Label shift** (Label distribution skew, prior probability shift, pathological heterogeneous):
  $P^i(y) \neq P^j(y), P^i(x|y) = P^j(x|y)$
  Difference in observed labels across clients. For example, when clients are tied to particular geo-regions, the distribution of labels varies across clients. Kangaroos are typically only seen in Australia or at zoos; a person's face is only seen in a few locations worldwide and for mobile device keyboards, certain emojis or Unicode symbols are used by some demographics but not others.
- **Label drift** (not previously mentioned):
  $P^t(y) \neq P^{t+k}(y), P^t(x|y) = P^{t+k}(x|y)$
  Difference in observed labels over time. For example, as time passes new labels may appear. E.g, looking at genres for movies, books, etc., new genres emerge over time, but previous works do not get reclassified.
- **Sentiment shift** (same features different labels, concept shift):
  $P^i(x) = P^j(x), P^i(y|x) \neq P^j(y|x)$
  Same feature vectors in a training data item have different labels. For example, labels that reflect sentiment or next-word predictors have personal and regional variations.
- **Sentiment drift** (Real concept drift):
  $P^t(x) = P^{t+k}(x), P^t(y|x) \neq P^{t+k}(y|x)$
  Same feature vector change label over time. An example of this could be how someone's food preference changes over time and what was previously disliked might become enjoyed in the future.
- **Feature shift** (same label, different features, concept drift):
  $P^i(y) = P^j(y), P^i(x|y) \neq P^j(x|y)$
  Same label y has different features x for different clients, e.g. due to cultural differences, weather effects, and standards of living. For example, images of homes can vary dramatically around the world and items of clothing vary widely. Even within the U.S., images of parked cars in the winter will be snow-covered only in certain parts of the country. The same label can also look very different at different times and at different time scales: day vs. night, seasonal effects, natural disasters, fashion and design trends, etc.

- **Feature drift** (not previously mentioned):
  $P^t(y) = P^{t+k}(y), P^t(x|y) \neq P^{t+k}(x|y)$
  Same label y changes its corresponding features x over time. Building on the previous example, you could consider the effects global warming has on our climate. What we perceive as normal changes, and what temperature and weather we expect in summer or winter changes.
- **Quantity shift** (unbalancedness, quantity skew):
  $|P_i| \neq |P_j|$
  The amount of data in different distributions may vary spatially. E.g. local dataset size or the number of clients per cluster may vary.
- **Quantity drift** (not previously mentioned):
  $|P_t| \neq |P_{t+k}|$
  The amount of data in different distributions may vary temporally. E.g. the number of news articles about the American elections increases significantly during the later stages of the election period.
- **System heterogeneity**:
  Storage, computational, and communication capabilities of each device in federated networks may differ due to variability e.g. in hardware, network connectivity, and power. These system-level characteristics dramatically exacerbate challenges such as straggler mitigation and fault tolerance.

2.5 Non-IID Strategies in Federated Settings

In a federated setting, one typically attempts to generate a global model or distinct local models for each individual client. These clients are devices with local datasets, typically only operated by one or a small group of users, each with individual preferences. Due to this, the local data stored on any individual device typically reflect the needs of the user or group. This results in many different types of data distributions existing on all devices, making it more likely than not, that the training data is Non-IID.

As previously mentioned there are several types of heterogeneous distributions. This section summarizes the findings of Criado et al.s paper *Non-IID data and Continual Learning processes in Federated Learning: A long road ahead* [12]. In it, they give an overview of which areas in FL have been studied for different Non-IID distributions, and where work is still needed. It highlights that while multiple studies have looked at spatial heterogeneity, very little is done when the data is both spatially and temporally heterogeneous. To better understand this area we will first only look at cases in which we have heterogeneity in one dimension.

2.5.1 Data Shifts

Since client distributions typically vary, algorithms handling data shifts require some degree of personalization. Criado et al. note how there are at least two different approaches to solving this issue, the first being **client-level personalization**, wherein each client has and trains its own model, typically in a federated setting, this means that the individual devices optimize their received model to perform well on the local data. The second approach is **group-level personalization**, in which clients with similar datasets are clustered into groups, wherein they share a single model. The focus of these types of algorithms, therefore becomes to determine which clients' data are more homogeneous and should be clustered together, allowing for them to train with one another, resulting in a common model within the cluster that is still personalized for their needs. Both methods have advantages but are more computationally demanding than standard federated learning.

In addition to this, different types of data will need different approaches toward optimizing their performance. When looking at for example covariate heterogeneity, there are two main approaches found by Criado et al: domain transformation and domain adaptation. **Domain transformation** attempts to detect the data heterogeneity by transforming all data into a shared input space, from which conclusions can be made. This method has seen some use but suffers from high computational costs due to the high-dimensional spaces and lower accuracy due to the curse of curse of dimensionality. **Domain adaptation** on the other hand attempts to leverage general knowledge from a broad range of data to hopefully generalize sufficiently to perform well on the specific use case which is what we will attempt.

Lastly, in the case of sentiment heterogeneity, the algorithm needs to accommodate that different clients may not necessarily correlate the same features with the same labels. In these cases, a single global model cannot exist to capture this behavior, and thus personalization is once again required. One possible solution to this is to treat the issue as a multi-task problem, where associating data with different conditional distributions correlates to unique tasks.

2.5.2 Data Drifts

If we instead look at the scenarios in which data is homogeneous among clients, but heterogeneous over time, we get what is commonly referred to as continual learning (CL). Typically, CL algorithms train an initial model that is then gradually adapted to the new data. During this time it attempts to preserve its previous knowledge to avoid what is known as catastrophic forgetting, the case in which it can no longer perform its previous task. In Criado et al's paper, it is shown that though this has to some extent been studied in federated settings, it has not been so in decentralized ones. Instead, they separate the problem into three categories as seen illustrated by figure 2.3.

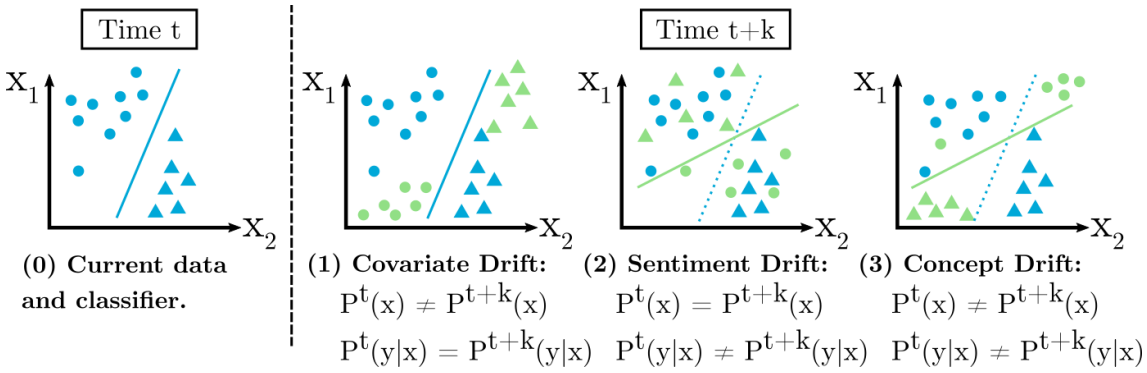


Figure 2.3: Illustration of concept drifts taken from [12]. Two-dimensional input space X with two possible labels $Y = \{\circ, \triangle\}$. On the left of the dotted line, we find the data samples received before time t, and on the right, the new data is illustrated in green. Here we see three different time-evolving situations that we will mention. (1): The new data samples observed are situated in new regions, previously unseen. However, the conditional probability still holds and a classifier trained on previous data may still produce the correct labels. (2): The new instances appear in already known regions of the input space, but they are incorrectly classified using the model from (0). (3): The two previous situations are combined.

For *covariate drift*, as it only relies on input data distributions, there are two main methods. First, **memory-based methods** keeps a record of data samples from previous concepts so that when a drift is detected, the model is trained on both the old and new data to avoid forgetting. This approach has shown quite effective at mitigating catastrophic forgetting. This has resulted in good results but is significantly more computationally costly. Second, **Regularization methods** restrict the weight updates to prevent forgetting old tasks. The downside is that this may scale poorly and become computationally inefficient when trying to learn several tasks.

For *sentiment drift*, different clients need to have models that can produce different outputs even though they get similar input. Here as well there are two main approaches, **contextual information methods** and **architecture-based methods**. Contextual information methods discuss the possibility of adding a piece of identifying information to separate tasks or domains. When new tasks are identified new layers or models can be added to learn the new task. While architecture-based methods focus on the model architecture, to keep important neurons correlating to specific tasks fixed.

Some of these methods require us to determine when the client distributions change over time. In other words, the clients need to detect when a drift occurs. This can mainly be done through two methods, **data distribution-based** and **error rate-based methods**. The first of which aims to detect a *covariate drift* by measuring the similarities among data features, grouping these into clusters, and evaluating the number of features from the new data sample in each cluster. The second detects *sentiment drift* through a sliding window that detects sharp changes in prediction error over time.

2.6 Previous Works

In this section, we will go deeper into two specific algorithms that we have used as inspiration for our algorithm and evaluation methods.

2.6.1 Decentralized Adaptive Clustering

Decentralized Adaptive Clustering (`DAC`) is an algorithm for decentralized learning that encourages similar clients to communicate more frequently with one another. This is done with the help of a similarity metric which is used to calculate the sampling probability between clients. As clients sample other clients in the network the similarity score between clients is calculated as the inverse training loss between the incoming clients' models and the local client's data. The following round's sampling probability vector is then calculated from the updated similarity vector via a softmax function, with a temperature scaling τ. A weighted sampling function is then used to determine which clients should be sampled. This promotes interactions between clients with similar data distributions, which in theory improves the convergence rate and prevents model poisoning. To further speed up convergence, clients also use a two-hop similarity methodology where client C_i, when sampling client C_j, can estimate similarity scores for clients that C_j has communicated with but C_i has yet to sample. When compared to other benchmark models, the `DAC` algorithm steadily performs better in *quantity shift* settings. For our experiments, we use a slightly modified `DAC` which is further described in section 3.3.

2.6.2 `FedDrift`

`FedDrift` is one of the few FL algorithms that is designed to work with both spatial and temporal heterogeneity simultaneously. Instead of using only one central model `FedDrift` attempt to store one central model for each task. In each training round the clients calculate the training loss of each of these models on their local data, and use the model with the lowest loss as their new local model. If the loss from any of the existing models does not lie below a certain threshold, the client creates a new model that is trained on the local data. Over time similar models are conservatively merged until a threshold is reached. To evaluate their algorithm they introduced two *drift patterns*, which intend to more realistically introduce new tasks over time. A more in-depth description of these patterns can be found in section 3.4.2. When compared to other benchmarks, the `FedDrift` algorithm outperformed other methods such as AUE [14], DriftSurf [15], and an oracle which always communicated within the correct cluster.

3

Methods

3.1 Datasets

Decentralized learning is problem-agnostic, meaning that it should function well on different types of data such as sound, images, numbers, etc. However, in literature, it has most commonly been tested in supervised image classification, thus, to more easily compare our results to other methods, this project will also focus on supervised image classification.

Figure 3.1: Illustration of a subset of the CIFAR-10 dataset. Each column contains images from one of the ten labels present in the dataset with the corresponding label written above.

3.1.1 CIFAR-10

CIFAR-10 (Canadian Institute For Advanced Research) [16] is an image dataset. It consists of ten labels: airplanes, automobiles, birds, cats, deers, dogs, frogs, horses, ships, and trucks, see figure 3.1. In it, there exists a wide range of features for each label, for example, the ship-set contains both cargo ships, speed boats, and canoes. Most images also contain varying backgrounds, adding complexity to the images. The set contains 50000 training images and 10000 testing images, divided evenly into the ten labels. Each image is 32x32 pixels large and in full color. Top-of-the-line non-decentralized models have achieved an accuracy of 99.50% [17].

3.1.2 PACS - Modified

PACS (Photo, Art Painting, Cartoon, and Sketch) [18] is an image dataset consisting of seven different labels: dogs, elephants, giraffes, guitars, horses, houses, and humans, across four different domains: photographs, paintings, cartoons, and sketches, see figure 3.2. It was initially created to study domain adaptation and generalization, where three of the domains were used for training and validation while the last domain was for testing. It consists of roughly 10000, 227x227 pixel, full-color images split unevenly among the different labels and domains, see appendix A.1. Previous works have reached a 99% accuracy in domain detection [19]. However, as we are using the dataset to simulate distributed covariate drift, a modified version of the PACS set was used in which the labels refer to the image labels rather than their domains [20]. In addition, as neither of these two sets contained a test set, one was created from between 22 and 45 images per label and domain, with unevenness correlating to the training set, see appendix A.1.

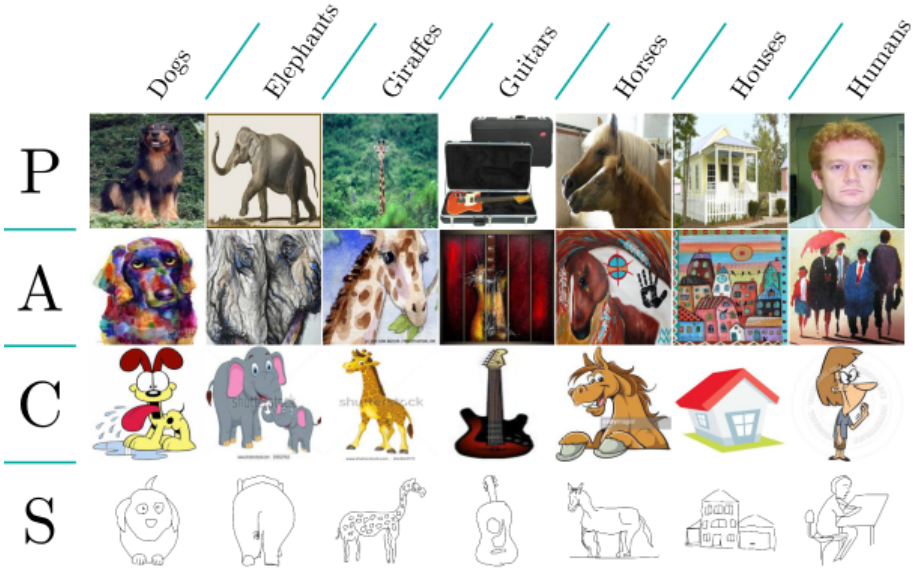


Figure 3.2: Four samples of each label in the PACS dataset. Each column contains the same label but one image is from each of the four domains. Each domain is denoted on the y-axis as (P)icture, (A)rt painting, (C)artoon, and (S)ketch. Each label is written out over each image column.

3.2 Client Model Architecture

For all experiments, a similar CNN was used. Consisting of two convolutional layers, each followed by a max pooling layer, and three fully connected layers. A full description of network parameters for the different experiments can be found in appendix A.2. This architecture was selected as it is a common benchmark in federated and decentralized research [10] [11] [1].

3.3 Algorithms

In this section we will cover the two algorithms used as baselines as well as our experiments, our novel algorithm `HAST`. Throughout this section, functions may be used by multiple algorithms, but they will only be presented at their first occurrence. Note that `FedAvg` used here, has already been presented in section 2.2. Further, as an algorithm is running in parallel on all clients at any point in time, the clients will refer to themselves as the *active client*. The specific hyperparameters used can be found in appendix A.3. The developed code can be found on: https://github.com/EmilieKar/HAST.

3.3.1 `Random`

The decentralized learning algorithm `Random`, proposed by [9], differs from the commonly used **Gossip Learning** [6], in that each client requests model parameters from a sampled subset, rather than sending its parameters to the sampled group. In short, the `Random` algorithm iterates over R rounds where it samples m other clients in the network each round, aggregates the sampled client models using `FedAvg`, described in section 2.2, to create a new base model, that it then trains using the client's local dataset for a set amount of local epochs. This new base model is used as the local model for sampling in the following round.

Algorithm 2: Adapted Gossip Learning

m: number of sampled clients at each round
C: active client
D: local dataset of active client
w: local model parameters of the active client
$\mathcal{C}$: set of all clients $\{C_1, ..., C_K\}$

```
Random():
    initialize w
    for each timestep t = 1, 2, ... do
        // Receive new data
        for each round r = 1, 2, ... do
            S ← Random subset of m clients ∈ {𝒞 \ C}
            w ← FedAvg(S ∪ C)
        w ← LocalTraining(w)

LocalTraining(w):
    ℬ ← (split D into batches of size B)
    for each local epoch e = 1, 2, ... do
        for batch b ∈ ℬ do
            w ← w − η∇_w ℒ(w; b)
    return w
```

As illustrated in algorithm 2 the `Random` implementation has also been extended to work in a temporal setting, by adding an outer loop for the time steps. Further, we use different methods of assigning new data to the clients at time steps which are described in detail in section 3.4.2.

3.3.2 DAC

The Decentralized Adaptive Clustering (`DAC`) algorithm here is identical to the one described in section 2.6.1, except for the addition of time steps. A full overview of the algorithm is seen in algorithm3. `DAC` represents an edge case to our proposed algorithm as it uses similarity aggregation for all layers of the model.

Algorithm 3: Decentralized Adaptive Clustering

m: number of sampled clients at each round
C: active client
w: model parameters on client
s: similarity vector containing similarity scores for all clients $C_j \in \{\mathcal{C} \setminus C\}$
$\mathcal{C}$: set of all clients $\{C_1, ..., C_K\}$

```
DAC():
    initialize w
    s ← Uniform(1/(K−1))
    for each timestep t = 1, 2, ... do
        // Receive new data
        for each round r = 1, 2, ... do
            S ← WeightedSampling(m, s)
            s ← UpdateSimilarities(S, w)
            w_new ← FedAvg(S ∪ C)
            w ← LocalTraining(w)

WeightedSampling(m, s):
    p ← SoftMax(s)
    S ← Weighted random subset of m clients ∈ {C \ C} with probability
      vector p
    return S

UpdateSimilarities(S, w):
    for each client C_j ∈ S do
        s_j ← 1/L(w_j, D)
    // Two-hop similarity update
    return s
```

3.3.3 HAST

Our algorithm Hierarchichal Aggregation with Similarity based Tuning (HAST), combines Random and DAC by using a hierarchical sampling and aggregation scheme. This allows it to adapt to different concepts that emerge at different times. At its core, it is based on the principle of transfer learning. Our hypothesis is that some domain knowledge is shared between tasks, therefore training a common feature extractor is possible. To better capture the nuances of the separate tasks we can then fine-tune a classifier based on similar clients, using similarity clustering taken from the DAC algorithm.

In more formal terms HAST works as follows. Each client k has a neural network consisting of a feature extractor f_θ^k and a classifier f_ϕ^k (the whole model being $f_\Omega^k = f_\phi^k \circ f_\theta^k$). HAST then consists of three steps. For each client k and for each communication round:

1. A subset of clients S_{rand} is sampled uniformly and at random. Client k updates all layers f_Ω^k using FedAvg.
2. A subset of clients S_{sim} is randomly sampled based on their similarity to client k. Client k updates *only* the classifier layers f_ϕ^k using FedAvg.
3. Client k performs local training on its own data, updating the whole model f_Ω^k. Afterward, it fine-tunes only the classifier f_ϕ^k.

The similarity function is heavily based on the similarity function used in DAC, i.e. the empirical training loss of client C_k's model w_k on client C_j's dataset D_j: $s_{kj} = 1/\mathcal{L}(w_k; D_j)$. This similarity score is transformed using a softmax with a temperature scaling τ in order to get a probability vector p_{kj} for each client and each communication round. A two-hop scheme is also used to allow clients to estimate similarities with clients that have not yet communicated which increases convergence speed.

Algorithm 4: Hierarchichal Aggregation with Similarity based Tuning

m: number of sampled clients at each round
C: active client
w_Ω, w_ϕ: model parameters corresponding to f_Ω and f_ϕ of the active client
s: similarity vector containing similarity scores for all clients $C_j \in \{\mathcal{C} \setminus C\}$
$\mathcal{C}$: set of all clients $\{C_1, ..., C_K\}$

```
HAST():
    initialize w_Ω
    s ← Uniform(1/(K−1))
    for each timestep t = 1, 2, ... do
        // Receive new data
        for each round r = 1, 2, ... do
            S_rand ← Random subset of m clients ∈ {C \ C}
            s ← UpdateSimilarities(S_rand, w_Ω)
            w_Ω ← FedAvg_Ω(S_rand ∪ C)

            S_sim ← WeightedSampling(m, s)
            w_φ ← FedAvg_φ(S_sim ∪ C)

            w_Ω ← LocalTraining(w_Ω)
            w_φ ← LocalTraining(w_φ)
```

3.4 Experiment Setup

As system simulation was not feasible in this project, all clients were simulated on a single GPU. To ensure parallel operations, the local training on each client only occurred after all clients had completed their sampling and aggregation. Otherwise, slower clients would benefit unfairly from communicating with faster clients on which local training had already been completed and for this project, all clients were considered to have the same hardware and network capacity.

The above algorithms were tested on multiple datasets with different distributional drifts including covariate, label, sentiment, and quantity. These datasets however are limited in size, and when distributing data over both clients and time, the amount of data was not enough. Therefore we were forced to reuse the training data, however, the test set that we evaluated the models on was kept completely separate and not reused. Each time the training set was reused the datasets was to minimize the risk of clients getting the same data multiple times. This may have affected our validation scores since data may have been used before, which is why we only present the test scores for our evaluation.

All experiments have been averaged over three random seeds to get more robust results. During training, we used early stopping locally on each client. Instead of stopping immediately if the validation loss stopped improving, we stopped when it had not improved for 50 rounds, as the sampling process can sometimes be suboptimal for the client and this seemed to give us better performance. At the beginning of each time step, we reset the early stopping so that all clients started training again. When stopped, the client could still be sampled by others, but it would not sample or train itself. In mosdt of our results we will only present the results of the early "best model" which is only updated as validation loss improved. Lastly, to speed up training, the clients were only tested on the test set every five rounds. This means that some resolution of our test scores is lower, however, testing each client each round was not computationally feasible.

3.4.1 Hyperparameter Tuning

As described, `HAST` performs two steps of sampling m clients per round. To ensure a fair comparison with our other baselines, `Random` and `DAC` were allowed to sample as many clients, as `HAST`, in their communication rounds. Further, `HAST` also performs two rounds of local training of E local epochs. However, when doubling the number of local epochs the performance dropped both for `Random` and `DAC`, see appendix A.4.1. From these results, we decided to use the best-performing version of the baselines in all comparisons with `HAST`, i.e. where the baselines samples $2 * m$ clients but train for E local epochs relative to the m and E used for `HAST`.

To further ensure that each algorithm performed optimally, a hyperparameter search was conducted. All parameters were not be tested, instead, only the learning rate was optimized. This was conducted by testing a range of learning rates surrounding an initial guess. This range covered $\pm\frac{1}{2}, 1, \frac{3}{2}$, and 2 magnitudes from the original value. When comparing these runs, the best result would typically correlate to the optimal learning rate, while the second and third best results were $\pm\frac{1}{2}$ magnitude from the optimum. When such results were achieved, it was determined that the best-performing learning rate was sufficiently close to the optimum to be used in the actual experiments. In some occasions, where the separation between the best-performing learning rates was not sufficient, an additional test was conducted in which the learning rate was changed by $\pm\frac{1}{4}$ magnitude to find the optimum through the same procedure as before.

Lastly, we explored the aggregation depth parameter for the `HAST` algorithm, which determines what layers belong to f_θ and f_ϕ respectively. For example, a five-layer model with aggregation depth three will have two layers in f_θ and three layers in f_ϕ. Here, setting the value to zero would emulate the `Random` algorithm, while a depth of five would correspond to `DAC`. The results of this tuning can be found in appendix A.4.2. From this, it was determined that `HAST` performed the best at depth three, i.e. when the feature extractor consisted of the two convolutional layers and the classifier consisted of the two fully connected hidden layers and the output layer.

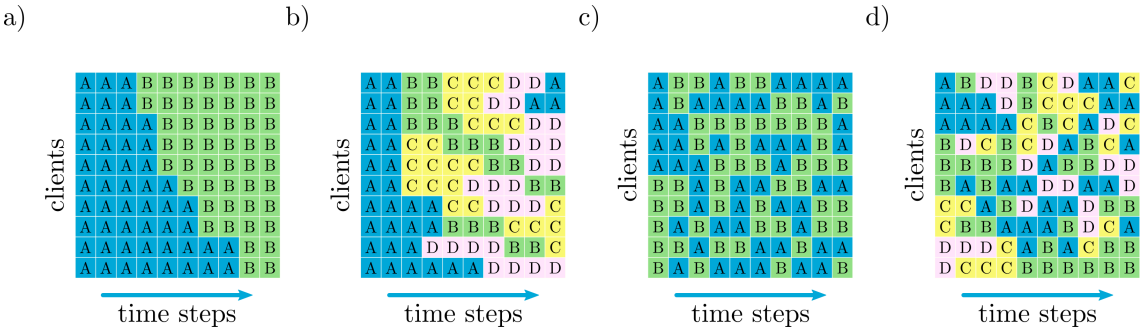


Figure 3.3: a) Staggered 2 clusters, b) Staggered 4 clusters, c) Random cluster assignment 2 clusters, d) Random cluster assignment 4 clusters. In images c) and d) an example pattern is shown. The respective letters A, B, C, and D correlate to the given cluster identity of the clients. In our experiments, each of these cluster identities is used to select which data distribution the client pulls its local data from.

3.4.2 Drift Patterns

Throughout all experiments, two main methods were utilized when assigning the cluster identities to the clients. The first one followed a predefined staggered pattern proposed by [11]. Rather than ten clients as in the original paper, we doubled the number by assigning two clients to each pattern, to increase the number of clients per cluster so that the similarity-based sampling has a chance to sample within the desired cluster. The second drift method assigns random cluster identities at each time step, maintaining the number of clients per cluster. In most experiments, clusters were assigned an equal number of clients, but some experiments were also done with uneven cluster sizes, see section 3.4.7. An example image of both drift patterns, in two and four cluster variants, can be seen in figure 3.3. Note, that the random patterns c) and d) are only one example of how the drifts could look.

In the experiments using the staggered drift pattern, clients only got new local data if their cluster identity changed, i.e. $D_k^t \equiv D_k^{t+1} \iff n_k^t \equiv n_k^{t+1}$. Further, new clusters may be introduced over time. Thus throughout training, the amount of data drawn from the different distributions varies. On the other hand, in experiments utilizing the random cluster assignment, the relative number of clients per cluster is maintained and therefore the relative amount of data from each distribution also remains constant. The random cluster assignment scheme also always assigns new data to all clients at all time steps, i.e. $D_k^t \neq D_k^{t+1}\ \forall t$. We included all of these variations to more clearly show the robustness of the algorithm regardless of the setting as there is yet to be any commonly agreed upon standard of how to perform the experiments in the distributed drift setting.

3.4.3 Distributed Covariate Drift

Previous experiments on covariate shifts have used rotated versions of the MNIST and Fashion MNIST dataset [10], however, we found that this task was too simple and we were unable to see any separation of the different algorithms. Instead, we designed a more complex and realistic experiment using the PACS dataset similar to the experiments proposed in [21]. Here, the different image domains simulated the variance in observed features for the same labels. In the experiment, all labels were present in all clusters, but each cluster drew its data from different domains.

For this setting, we only had the time to test the random cluster assignment method for four clusters as experiments on the PACS dataset were computationally expensive to run. Unlike the other experiments, we only ran these for four time steps because of time constraints. The model used is a version of the model described in section 3.2 adapted for 227x227 pixel images. A full list of the experiment parameters can be found in A.3.

3.4.4 Distributed Label Drift

For distributed label drift several experiments were conducted on the CIFAR-10 dataset. In these experiments, eight of the ten available labels were partitioned into n different clusters so that no overlap existed. We only used eight labels so that the same labels could be used in the two-cluster and four-cluster settings and so all clusters would have an equal number of labels assigned to them. The model used is the model described in section 3.2 for 32x32 pixel images. The experiment was conducted using all drift patterns described in 3.4.2 and with ten time steps. All experiment parameters can be found in appendix A.3

The labels were split into clusters that would seem correlated to humans. In the two-cluster case, four animals (cat, deer, dog, and horse) were in one cluster, and four vehicles (airplane, automobile, ship, and truck) were in the other. In the four clusters case, we instead split each of the previously mentioned clusters and ended up with the following: (airplane, automobile), (ship, truck), (cat, deer), (dog, horse).

3.4.5 Distributed Label Swap

For the two Non-IID variants that consider variations in conditional probabilities (sentiment and feature), the most common experiments in literature addresses both of these together through an experiment called label swap [22]. In this experiment, the data corresponding to two labels are ”swapped” i.e. if we have labels A and B we could swap these so that all images depicting A are labeled as B and vice versa. We tested all drift patterns described in section 3.4.2 and swapped only two labels in each of our clusters in the same fashion as the label swap experiments in [11].

The experiment was performed on CIFAR-10 using the same experimental parameters, labels, and model as previously mentioned for the *Distributed Label Drift*. In this experiment, however, all clusters have access to all 8 labels. Cluster A swaps no labels, cluster B swaps airplanes and automobiles, cluster C swaps cats and deer, and cluster D swaps dogs and horses.

3.4.6 Exploring Label Heterogeneity

To test the performance of the model's performance on different degrees of heterogenous data we performed two additional experiments in the *Distributed Label Drift* setting. In the first of these, we simulated a fully homogenous setting where all clients sampled from one distribution consisting of all available eight labels. In this experiment, all clients received new data at every time step similar to the random cluster assignment method. Since `HAST` was designed to work well in a heterogeneous setting, this experiment was included to show that the relative performance to the other baselines does not drop in a homogenous setting.

The second experiment is a variant of the *Distributed Label Drift* experiment, where instead of selecting labels according to human perception of similarity, the label groups were randomly assigned. The assignments were as in table 3.1:

Table 3.1: The three different heterogeneity mixes tested on distributed label drift. In each of the three experiments, the labels corresponding to each cluster were randomized to human perception of similarity.

Experiment 1		Experiment 2		Experiment 3	
Cluster 1	Cluster 2	Cluster 1	Cluster 2	Cluster 1	Cluster 2
Automobiles	Airplanes	Airplanes	Automobiles	Airplanes	Automobiles
Dogs	Birds	Birds	Cats	Birds	Cats
Frogs	Cats	Dogs	Deer	Frogs	Deer
Horses	Deer	Frogs	Ships	Ships	Dogs
Trucks	Ships	Horses	Trucks	Trucks	Horses

For this experiment we included the standard deviation of all cluster variations to attempts to illustrate how the degree of cluster heterogeneity affects the relative performance of our model compared to the baselines. However, there are many more experiments that could be done to more thoroughly test this. Due to time constraints, only this one was performed.

3.4.7 Quantity Shift

Lastly, we performed experiments similar to those in [10] where we looked at another version of the *Distributed Label Shift* experiment where the clients are unevenly assigned to each cluster. We only tested the experiment on random cluster assignment for two clusters. Here we try to simulate a constant disparity in cluster sizes where the smaller cluster typically gets disfavored by algorithms like `Random`. The two settings we explored were a 70-30 split and an 80-20 split, which were both compared to the previous 50-50 split.

3.5 Evaluation

Initially, we wanted to look at evaluation metrics typically used in continual learning problems such as knowledge retention and performance on previous tasks. However, we found these metrics to be unhelpful in experiments like label swap where conditional probabilities change. Instead, we focus on the algorithm's ability to quickly adapt to new concepts. We looked at three main metrics, performance, convergence rate, and robustness to task switching, and were calculated as follows:

- **Performance:** Average accuracy over time. The test accuracy is averaged across all clients at the end of each time step and then averaged across all time steps of the experiment.
- **Convergence rate:** The number of rounds required to reach 90 percent of the performance increase of a time step averaged across all clients and time steps.
- **Robustness:** Performance drop after concept switch. The difference in the previous time step's final accuracy compared to the new time step's worst accuracy averaged over all clients and time steps.

4

Results

Throughout this section, we will present the results of the experiments mentioned in section 3.4, using plots and the metrics presented in section 3.5. As the performance of HAST was studied on many different types of Non-IID data and outperformed the baselines in almost all settings, only some of those results will be highlighted here along with the setting where HAST did not outperform the baselines. Plots for all experiments mentioned in section 3.4 can be seen in appendix A.4.

4.1 Metric Results

First, we present the metrics from section 3.5 in table A.5, as it in a condensed fashion, illustrate the performance of our novel algorithm HAST compared to the two baselines DAC and Random. Due to its size, the table was moved to appendix. In the table, we see that our model achieves the highest accuracy in all experiments but two, in which our performance is only one-half percentage points lower. When looking at the convergence rate, our performance varies greatly from task to task. However, a high convergence rate alone is not sufficient to determine the model performance, if there is no increase in accuracy between time steps, the convergence rate will be naturally low. This can for example be seen in figure 4.4 where DACs accuracy decreases between several time steps but still achieves the highest convergence rate.

When we instead look at the resiliency of the algorithms we see that in most experiments, HAST and Random perform about equal, typically only seeing an initial drop of 1-2 percentage points directly after initiating a new time step. Further, we see that DAC usually has the largest drop in performance but also the highest convergence rate which we will discuss later.

As we can see the convergence rate and performance drop are heavily dependent on each other and cannot be viewed as independent metrics. Average performance also gives us a measurement of how the algorithm performs over time but in some cases the final performance may be more interesting and this metric may in some cases not reflect that. We have included these metric as our attempt at giving more insight into the plots we will present but we can also see that this fields need new and more informative metrics to be developed in the future.

4.2 Distributed Covariate Drift

Second, we present the performance on distributed covariate drift, based on the experiment described in section 3.4.3. Looking at table in appendix A.5 we see that `HAST`'s test accuracy outperforms the closest baseline with 5,39 percentage points while also achieving the fastest convergence rate. In both the table and figure 4.1 we see that `HAST`'s performance drop directly after initiating a new time step is rather large, however, its accuracy during these drops is still equivalent to or better than the baselines. Additionally, its recovery rate from the drops is fast and the algorithm manages to improve its performance through all time steps.

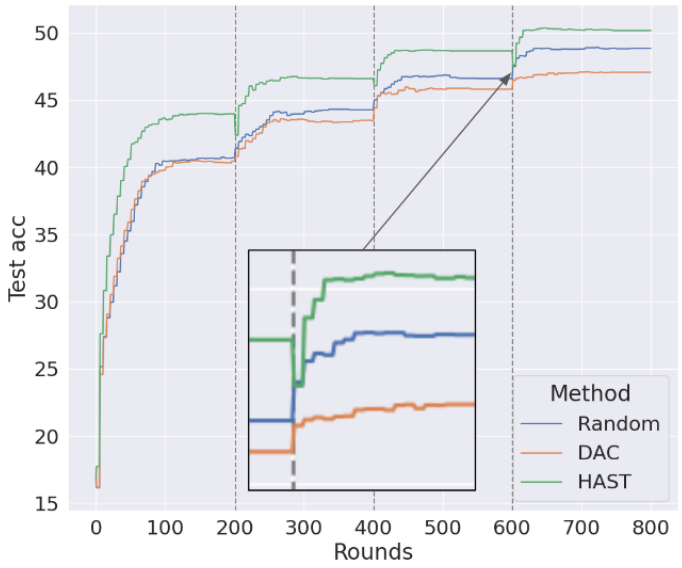
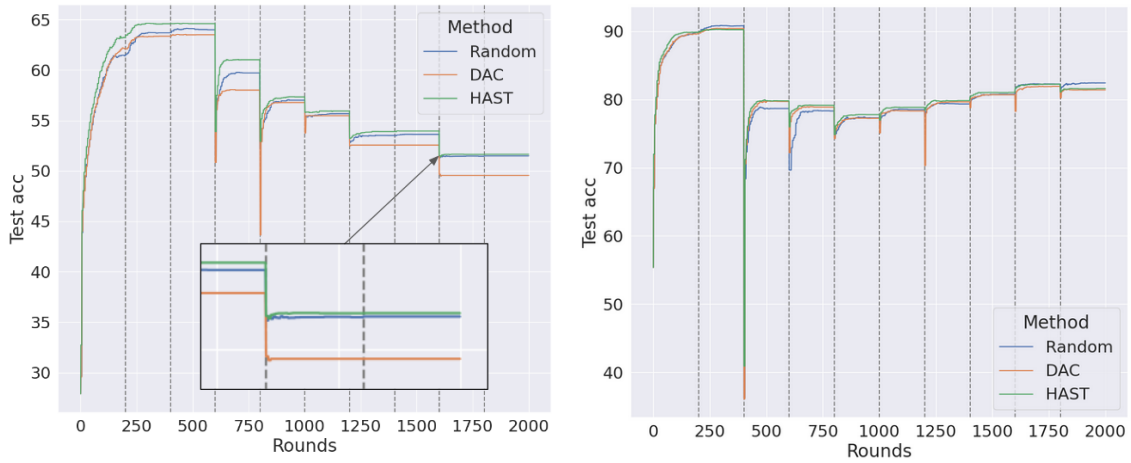


Figure 4.1: Distributed covariate drift test accuracy for all algorithms on the PACS dataset. In the experiment, four clusters exist and the clients are assigned random clusters at each time step.

4.3 Distributed Label Drift

Third, we show the performance on distributed label drift, based on the experiment described in section 3.4.4. In figure 4.2 we see that `HAST` achieves higher accuracy than the baseline, though with a smaller difference. From the table in appendix A.5, we see that `DAC` has the highest convergence rate, however, if we also look at the figure, we see that in most time steps, `DAC` stops improving within a few communication rounds, indicating that it can not take advantage of receiving new information as well as `HAST` and `Random`. We also see that as more clusters are introduced, the average performance of the algorithms decreases. In figure 4.2a where we only have two clusters, we would in an ideal situation expect the accuracy to return to roughly 65% as we reach the latter time steps, as all clusters eventually belong to the same cluster again. However, as this is not achieved we hypothesize that the new models consider all 4 labels, and this, in turn, is more difficult.



(a) $N = 2$ **(b)** $N = 4$

Figure 4.2: Distributed label drift test accuracy for all algorithms on the CIFAR-10 dataset. In the experiment, N clusters exist and the clients are assigned clusters based on the staggered drift patterns illustrated in figure 3.3.

If we instead look at the four-cluster case in figure 4.2b, we see that the initial performance drop is substantial. However, two new clusters are introduced at this point, tripling the number of labels. In this scenario, we do see that the model improves over time, even though the number of clusters does not decrease. We further see this behavior when randomly assigning cluster identities to the clusters, as in figure 4.3. Throughout this case `HAST` manages to increase its lead to the second-best algorithm as time goes on. As we in both of these settings introduce more new data this might be the main contributor to the performance increase, which would also explain why we do not see this in the staggered 2 cluster case.

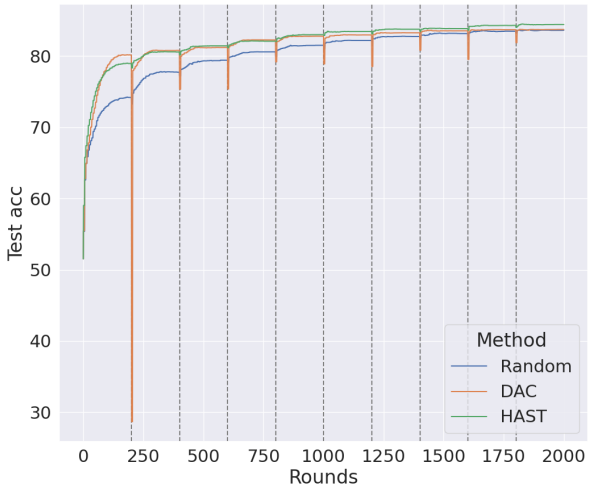


Figure 4.3: Distributed label drift test accuracy for all algorithms on the CIFAR-10 dataset. In the experiment, four clusters exist and the clients are assigned random clusters at each time step.

4.4 Label Swap

Fourth, we show the performance of the label swap experiment described in section 3.4.5. In table in appendix A.5 we see that `HAST` achieves the best accuracy in these settings and even with the performance drops `HAST`s performance in higher than the next closes baseline. If we look at figure 4.4, `HAST`'s performance between time steps remains mostly stable, while both `DAC` and `Random` see larger fluctuations to their accuracy.

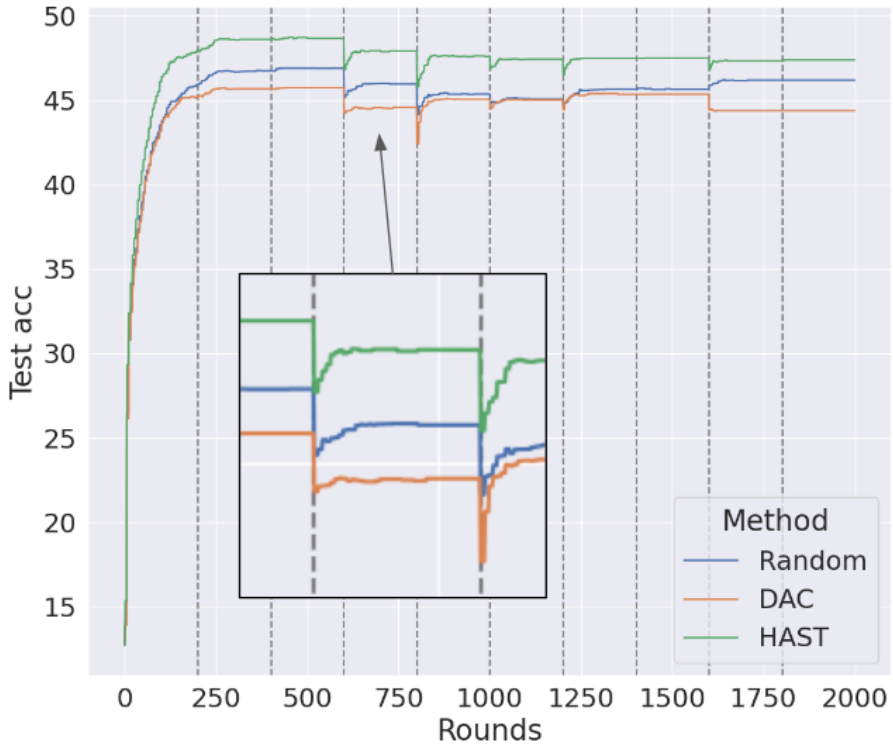


Figure 4.4: Label swap test accuracy for all algorithms on the CIFAR-10 dataset. In the experiment, two clusters exist and the clients are assigned clusters based on the drift pattern in figure 3.3.

4.5 Level of Heterogeneity

Fifth, we present the performance of `HAST` on different heterogeneities, as described in section 3.4.6. In figure 4.5 we see `HAST` manages to steadily outperform both baselines in a homogeneous setting, i.e. when the data is IID. This is interesting as both the local training set and test set contain data from the same distribution, thus, since all algorithms utilize the same neural network, we would assume them to perform equally. Despite this, our `HAST` achieves higher accuracy in all time steps with a faster convergence rate, see table in appendix A.5.

In the table, we also see that when testing different random label assignments for two clusters, we still achieve more than 1 to 2 percentage points higher accuracy than our baselines. Here we do see some variation in the final performance achieved by all models in the different cases, but the relative performance of `HAST` to the other models remains roughly the same, indicating that it can generalize more efficiently.

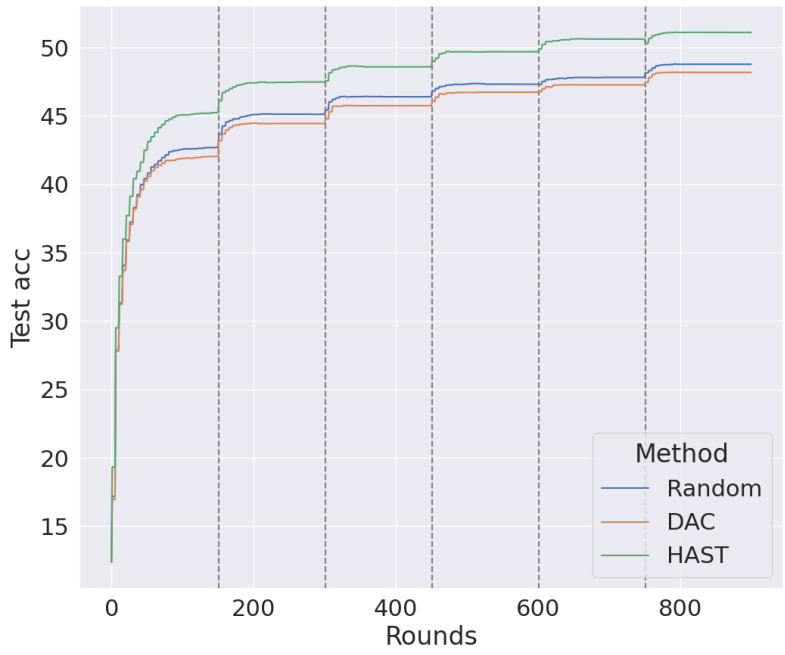
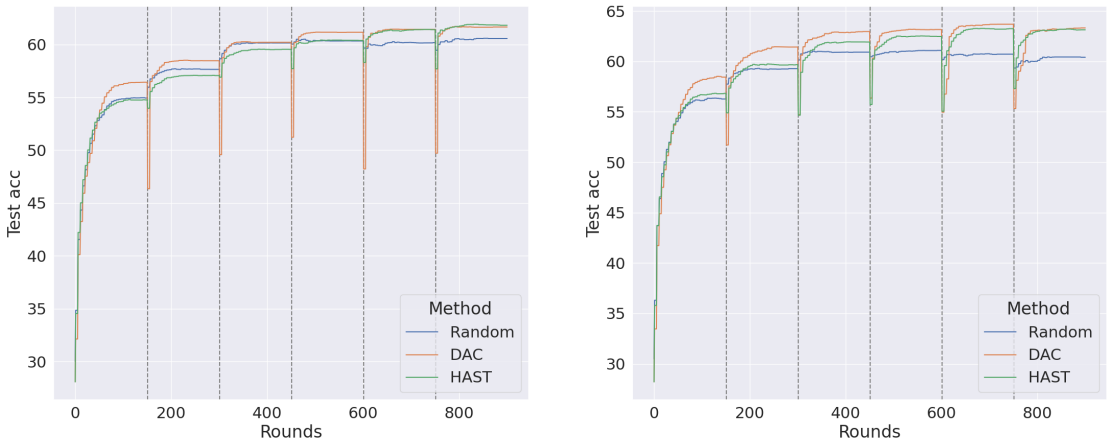


Figure 4.5: Test accuracy for all algorithms on the CIFAR-10 dataset. In the experiment, one cluster exists and the clients remain in it and they receive new data at each time step.

4.6 Distributed Quantity Shift

Lastly, we show the findings on distributed quantity shift, as described in section 3.4.7. In figure 4.6, we see that when the clusters become uneven in size, `DAC`'s performance starts improving, which correlates to findings in [10]. However, we also note that, over time, `HAST` seems to catch up and achieve the same level of performance, whereas `Random` stabilizes at a lower level. In table in appendix A.5, we see that the accuracy difference between `DAC` and `HAST` is about one-half percentage point. We further see that while `DAC` achieves good accuracy, its task resilience plummets as a result, falling between 8 and 14 % percentage points at each time step, while `HAST` achieves the same accuracy with less than 7 % accuracy drop between time steps. This clearly indicates that for this type of task, `DAC` does perform better than `HAST`, at the expense of task resiliency, whereas `HAST` is able to achieve similar performance over time but with a slower overall convergence rate.



(a) The first cluster contains 70% of the clients while the second only has 30%.

(b) The first cluster contains 80% of the clients while the second only has 20%.

Figure 4.6: Distributed label drift test accuracy for all algorithms on the CIFAR-10 dataset. In the experiment, two unevenly sized clusters exist and the clients are assigned random clusters at each time step.

5
Discussion

Our preliminary experiments suggest that our new method outperforms existing techniques in terms of accuracy under decentralization and exhibits robustness to different hyperparameter settings. In addition to improving the accuracy on all heterogeneities, we also see in appendix A.5, that our approach offers great resilience to task switching, where in most cases, the accuracy drop directly after a new time step is initiated is either the lowest or within one standard deviation of the best algorithm. Hinting that our algorithm can take advantage of common features that exist throughout all data distributions to both learn its current task and quickly adapt to new data.

If we instead look at convergence rate `HAST` seems to be lower than `DAC`s, however, at a time step, `HAST`'s accuracy drop is on average two percentage points compared to `DAC`'s ten points. This means, that if the algorithms' accuracy between two time steps increases by one percentage point, `HAST` would need to recover from its drop and improve the total accuracy with 0,7 points before converging, while `DAC` only needs to recover its dropped accuracy, in other words, since it drops so much in accuracy, it does not need to improve to reach 90% convergence. As we see, the metrics can be misleading if looked at individually, thus more informative metrics may need to be developed to achieve a more nuanced view of decentralized task switch performance.

All in all, our algorithm improves accuracy, offers higher resilience to task switching, and is robust to different heterogeneities, all while achieving a high convergence rate. All of these offer empirical support to our hypothesis, that learning from an underlying distribution is beneficial even in a decentralized setting where personalization of the models is the main goal.

Although we see many benefits with our algorithm, it comes with some drawbacks. First of all, our algorithm is more computationally heavy on the local device, as we perform both a similarity calculation on the incoming models as well as fine-tuning on parts of our model. Resulting in almost double the computational load as compared to `Random`. We considered ways of counteracting this downside by increasing the amount of local training for `Random` and `DAC`, however, as we see in figure A.1, both algorithms performed worse from this, likely due to overfitting on the local data. Indicating that the hierarchical approach may benefit from additional training compared to previous methods.

Even though HAST requires more computations for each individual client, the bottleneck of a distributed system is more often the communication cost. And here, HAST manages to achieve all the abovementioned results with less total data transferred, as it only transfers some of the model parameters in its second aggregation step, compared to Random and DAC which transfers all model parameters. If we look at the experiments conducted with three sampled clients per round, HAST would need to communicate three full models and three classifier models as compared to both Random and DAC which would transfer six full models.

Lastly, HAST requires tuning of an additional hyperparameter, namely the aggregation depth. As we used a simple model throughout all experiments we have not studied how the parameter is affected by the model complexity. Despite these drawbacks, HAST offers one immense benefit, namely that in all experiments it manages to outperform the baseline algorithms while communicating less data.

Further evaluation of HAST on separate datasets and with larger variation in clients, amount of sampling, and local training would help in ensuring the robustness of the algorithm while also shining more light on its possible uses of it. Specifically, it would be interesting to test the algorithms on more realistic datasets and real distributed devices which we did not have time for during this project. For future works, we would strongly recommend looking at one or more of the following datasets for distributed learning:

- **LEAF** [23] a curated suite of realistic federated datasets that focus on datasets where (1) the data has a natural keyed generation process (where each key refers to a particular device/user); (2) the data is generated from networks of thousands to millions of devices; and (3) the number of data points is skewed across devices. Currently, LEAF consists of six datasets: FEMNIST, Shakespear, Sentiment140, CelebA, Reddit, Synthetic.
- **MAPS** [24] dataset collected from keyloggers on smartphones of adolescents at risk of suicide. All participants gave consent for the data to be collected and shared for research purposes and scored their mood daily for a 6-month duration. MAPS is a realistic federated learning benchmark since it contains real-world data with privacy concerns and high device variance due to the highly personalized use of mobile phones.
- **WILDS** [25] is a curated benchmark of 10 datasets reflecting a diverse range of distribution shifts that naturally arise in real-world applications.

Lastly, we want to highlight the possible ethical issues with decentralized learning. While federated- and decentralized learning decouple model training from direct access to the training data, risks always exist when handling sensitive data such as hospital records, personal information, and other data that may be used in undesired and sometimes harmful ways to target individuals, institutions, or corporations. Even when the raw data is not shared, these methods still have several open issues regarding data privacy and security such as inference of model gradients to recover raw data, and possibly malicious clients poisoning models with foul data [26].

In addition to these risks, there is also the possibility that a malicious client join the training session and influence the models on other clients. And due to the black-box nature of many deep learning models, it becomes impossible to know what biases are picked up during training, which can lead to undesirable model behavior. The general consensus is that these methods alone are not sufficient to protect highly sensitive data, but that it needs to be used in tandem with additional methods such as Differential Privacy [27] [28], Homomorphic Encryption [29] or Secure Multi-Party Computation [30].

6
Conclusion

We have presented a novel algorithm, `HAST` for decentralized deep learning that can adapt to spatial and temporal data variations in a peer-to-peer network. It demonstrates the benefits of aggregating different layers of a neural network using `FedAvg` for robustness and generalization. Our proposed algorithm uses a similarity-based clustering technique to group clients with similar concepts and allows them to update their beliefs of potential collaborators over time. This enables clients to smoothly transition between different collaboration groups as their concepts change. Moreover, our algorithm employs a two-stage aggregation scheme that makes the personalized models robust to concept changes by leveraging the knowledge from other clients. `HAST` was tested on multiple types of Non-IID settings where it steadily outperformed other baselines, however, it was only tested on image classification, and further testing on other types of data is necessary.

References

[1] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y. Arcas, "Communication-efficient learning of deep networks from decentralized data," 2016. DOI: 10.48550/ARXIV.1602.05629. [Online]. Available: https://arxiv.org/abs/1602.05629.

[2] "General protection data regulation," European Commission. (May 25, 2018), [Online]. Available: https://gdprinfo.eu/ (visited on 01/24/2023).

[3] N. Rieke, J. Hancox, W. Li, *et al.*, "The future of digital health with federated learning," *CoRR*, vol. abs/2003.08119, 2020. arXiv: 2003.08119. [Online]. Available: https://arxiv.org/abs/2003.08119.

[4] Q. Yang, Y. Liu, T. Chen, and Y. Tong, "Federated machine learning: Concept and applications," *CoRR*, vol. abs/1902.04885, 2019. arXiv: 1902.04885. [Online]. Available: http://arxiv.org/abs/1902.04885.

[5] "Federated learning with formal differential privacy guarantees," Google. (), [Online]. Available: https://ai.googleblog.com/2022/02/federated-learning-with-formal.html.

[6] D. Kempe, A. Dobra, and J. Gehrke, "Gossip-based computation of aggregate information," in *44th Annual IEEE Symposium on Foundations of Computer Science, 2003. Proceedings.*, IEEE, 2003, pp. 482–491.

[7] S. Boyd, A. Ghosh, B. Prabhakar, and D. Shah, "Randomized gossip algorithms," *IEEE Transactions on Information Theory*, vol. 52, no. 6, pp. 2508–2530, 2006. DOI: 10.1109/TIT.2006.874516.

[8] M. Blot, D. Picard, M. Cord, and N. Thome, *Gossip training for deep learning*, 2016. DOI: 10.48550/ARXIV.1611.09726. [Online]. Available: https://arxiv.org/abs/1611.09726.

[9] N. Onoszko, G. Karlsson, O. Mogren, and E. L. Zec, "Decentralized federated learning of deep neural networks on non-iid data," *CoRR*, vol. abs/2107.08517, 2021. arXiv: 2107.08517. [Online]. Available: https://arxiv.org/abs/2107.08517.

[10] E. L. Zec, E. Ekblom, M. Willbo, O. Mogren, and S. Girdzijauskas, *Decentralized adaptive clustering of deep nets is beneficial for client collaboration*, 2022. DOI: 10.48550/ARXIV.2206.08839. [Online]. Available: https://arxiv.org/abs/2206.08839.

[11] E. Jothimurugesan, K. Hsieh, J. Wang, G. Joshi, and P. B. Gibbons, *Federated learning under distributed concept drift*, 2022. DOI: 10.48550/ARXIV.2206.00799. [Online]. Available: https://arxiv.org/abs/2206.00799.

[12] M. F. Criado, F. E. Casado, R. Iglesias, C. V. Regueiro, and S. Barro, "Non-iid data and continual learning processes in federated learning: A long road

ahead," *Information Fusion*, vol. 88, pp. 263–280, 2022, ISSN: 1566-2535. DOI: https://doi.org/10.1016/j.inffus.2022.07.024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1566253522000884.

[13] G. W. Lindsay, "Convolutional neural networks as a model of the visual system: Past, present, and future," en, *J. Cogn. Neurosci.*, vol. 33, no. 10, pp. 2017–2031, Sep. 2021.

[14] D. Brzezinski and J. Stefanowski, "Reacting to different types of concept drift: The accuracy updated ensemble algorithm," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 25, no. 1, pp. 81–94, 2014. DOI: 10.1109/TNNLS.2013.2251352.

[15] A. Tahmasbi, E. Jothimurugesan, S. Tirthapura, and P. B. Gibbons, "Driftsurf: Stable-state / reactive-state learning under concept drift," in *Proceedings of the 38th International Conference on Machine Learning*, M. Meila and T. Zhang, Eds., ser. Proceedings of Machine Learning Research, vol. 139, PMLR, Jul. 2021, pp. 10 054–10 064. [Online]. Available: https://proceedings.mlr.press/v139/tahmasbi21a.html.

[16] A. Krizhevsky, "Learning multiple layers of features from tiny images," 2009.

[17] A. Dosovitskiy, L. Beyer, A. Kolesnikov, *et al.*, *An image is worth 16x16 words: Transformers for image recognition at scale*, 2021. arXiv: 2010.11929 [cs.CV].

[18] D. Li, Y. Yang, Y.-Z. Song, and T. M. Hospedales, *Deeper, broader and artier domain generalization*, 2017. arXiv: 1710.03077 [cs.CV].

[19] Z. Li, K. Ren, X. Jiang, Y. Shen, H. Zhang, and D. Li, "Simple: Specialized model-sample matching for domain generalization," in *Proceedings of the International Conference on Learning Representations*, 2023. [Online]. Available: https://openreview.net/pdf?id=BqrPeZ_e5P.

[20] MachineLearning2020, *Homework3-pacs*, 2019. [Online]. Available: https://github.com/MachineLearning2020/Homework3-PACS/tree/master/PACS.

[21] S. Motiian, Q. Jones, S. M. Iranmanesh, and G. Doretto, "Few-shot adversarial domain adaptation," *CoRR*, vol. abs/1711.02536, 2017. arXiv: 1711.02536. [Online]. Available: http://arxiv.org/abs/1711.02536.

[22] F. Sattler, K. Müller, and W. Samek, "Clustered federated learning: Model-agnostic distributed multi-task optimization under privacy constraints," *CoRR*, vol. abs/1910.01991, 2019. arXiv: 1910.01991. [Online]. Available: http://arxiv.org/abs/1910.01991.

[23] S. Caldas, P. Wu, T. Li, *et al.*, "LEAF: A benchmark for federated settings," *CoRR*, vol. abs/1812.01097, 2018. arXiv: 1812.01097. [Online]. Available: http://arxiv.org/abs/1812.01097.

[24] P. P. Liang, T. Liu, Z. Liu, R. Salakhutdinov, and L. Morency, "Think locally, act globally: Federated learning with local and global representations," *CoRR*, vol. abs/2001.01523, 2020. arXiv: 2001.01523. [Online]. Available: http://arxiv.org/abs/2001.01523.

[25] P. W. Koh, S. Sagawa, H. Marklund, *et al.*, "Wilds: A benchmark of in-the-wild distribution shifts," in *Proceedings of the 38th International Conference on Machine Learning*, M. Meila and T. Zhang, Eds., ser. Proceedings of Ma-

chine Learning Research, vol. 139, PMLR, Jul. 2021, pp. 5637–5664. [Online]. Available: https://proceedings.mlr.press/v139/koh21a.html.

[26] N. Rodríguez, E. Martínez-Cámara, M. Luzon, G. Seco, M. Veganzones, and F. Herrera, *Dynamic federated learning model for identifying adversarial clients*, Jul. 2020.

[27] I. Mironov, O. Pandey, O. Reingold, and S. Vadhan, "Computational differential privacy," in *Advances in Cryptology - CRYPTO 2009*, S. Halevi, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 126–142, ISBN: 978-3-642-03356-8.

[28] K. Wei, J. Li, M. Ding, *et al.*, "Federated learning with differential privacy: Algorithms and performance analysis," *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 3454–3469, 2020. DOI: 10.1109/TIFS.2020.2988575.

[29] L. J. M. Aslett, P. M. Esperança, and C. C. Holmes, *A review of homomorphic encryption and software tools for encrypted statistical machine learning*, 2015. DOI: 10.48550/ARXIV.1508.06574. [Online]. Available: https://arxiv.org/abs/1508.06574.

[30] C. Zhao, S. Zhao, M. Zhao, *et al.*, "Secure multi-party computation: Theory, practice and applications," *Information Sciences*, vol. 476, pp. 357–372, 2019, ISSN: 0020-0255. DOI: https://doi.org/10.1016/j.ins.2018.10.024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0020025518308338.

A

Appendix 1

A.1 PACS Dataset

Table A.1: Number of images for each label and domain, in both the training- and test-set for the PACS dataset.

Labels	**Photographs**		**Art Paintings**		**Cartoons**		**Sketches**	
	Train	Test	Train	Test	Train	Test	Train	Test
Dogs	151	38	341	38	353	36	727	45
Elephants	163	38	218	37	421	36	695	45
Giraffes	143	39	247	38	308	37	708	45
Guitars	149	37	145	39	98	37	563	45
Horses	161	38	168	33	286	38	771	45
Houses	243	37	258	37	251	37	58	22
Humans	394	38	411	38	367	38	123	37

A.2 Network architectures

Throughout the experiments, a similar model was utilized however as the images in the different datasets vary in size, some changes were applied depending on the dataset to allow for faster computation. In table A.2 we will present the architecture used for the CIFAR-10 dataset, but it is also fully compatible with any images in the form of 32x32 pixels with 3 channels. Table A.3 presents the architecture used on the PACS dataset. As the images are much larger, 227x227 pixels, the computation time would be much larger, thus we introduced a stride of three pixels so that each convolutional layer significantly reduced the size of the incoming image.

Table A.2: Utilized network parameters for 32x32 pixel full color images. In both convolutional layers, a stride of 1 was used.

Layer	#Channels	Kernel Size	Activation	Output Size	#Parameters
Input Layer	-	-	-	32 x 32 x 3	-
Convolutional	6	5x5	ReLU	28 x 28 x 6	456
Max Pooling	-	2x2	-	14 x 14 x 6	-
Convolutional	16	5x5	ReLU	10 x 10 x 16	2416
Max Pooling	-	2x2	-	5 x 5 x 16	-
Dense Layer	1	-	Linear	120	48120
Dense Layer	-	-	Linear	84	10164
Dense Layer	-	-	SoftMax	10	850
Total					62006

Table A.3: Utilized network parameters for 227x227 pixel full color images. In both convolutional layers, a stride of 3 was used.

Layer	#Channels	Kernel Size	Activation	Output Size	#Parameters
Input Layer	-	-	-	227 x 227 x 3	-
Convolutional	6	5x5	ReLU	75 x 75 x 6	456
Max Pooling	-	2x2	-	37 x 37 x 6	-
Convolutional	16	5x5	ReLU	11 x 11 x 16	2416
Max Pooling	-	2x2	-	5 x 5 x 16	-
Dense Layer	1	-	Linear	120	48120
Dense Layer	-	-	Linear	84	10164
Dense Layer	-	-	SoftMax	7	595
Total					61751

A.3 Experiment Hyperparameters

To avoid duplicating many parameters throughout all experiments, we will present the parameters in a modular fashion where parameters that were always present when for example using a certain dataset or drift pattern will be presented together. More specifically, we will present the parameters correlating to each dataset in table A.4, algorithm in tableA.5, drift pattern in tableA.6, and experiment-specific parameters in table A.7. From here each conducted experiment will only specify which of these tables it drew its parameters. Note that the variations seen in different experiments are due in part to computational resources, in part to algorithm tuning, and lastly to ensure result robustness as experiment parameters vary.

Table A.4: Specific parameter for the different datasets used in all results seen in section 4. In all experiments using for example the CIFAR-10 dataset, unless otherwise specified, the values seen in this table were utilized.

Dataset	CIFAR-10	PACS
Number of Rounds	200	200
Number of Time Steps	10	4
Batch Size	8	8
Local Epochs	3	3
#Training Datapoints (on each client)	250	175
#Validation Datapoints (on each client)	75	75
Number of Labels	10	7

Table A.5: Specific parameter for the different algorithms used in all results seen in section 4. In all experiments using for example the `HAST` algorithm, unless otherwise specified, the values seen in this table were utilized.

Algorithm	HAST	Random	DAC
Sampled Clients	3	6	6
Finetune Epochs	3	-	-
Learning Rate	8e-5	8e-5	8e-5
Tau	30	-	30
Aggregation Depth	3	-	-

Table A.6: Specific parameter for the different datasets used in all results seen in section 4. In all experiments using, for example, the staggered drift pattern, unless otherwise specified, the correlating values for two or four clusters were used. †The number of clients per cluster in the staggered pattern depends on the current time step according to figure 3.3.

Drift Pattern		2 Cluster	4 Cluster
Random Pattern	Number of Clients	50	50
	Number of Clusters	2	4
	Clients Per Cluster	25	12 or 13
Staggered Pattern	Number of Clients	20	20
	Number of Clusters	2	4
	Clients Per Cluster	Varies†	Varies†

Table A.7: Specific parameter for the different datasets used in all results seen in section 4. In all experiments using for example the CIFAR-10 dataset and Label Swap, unless otherwise specified, the correlating values for two or four clusters were used. The labels are as follows: 0: airplanes, 1: automobiles, 3:cats, 4: deer, 5: dogs, 7: horses, 8: ships, 9: trucks.

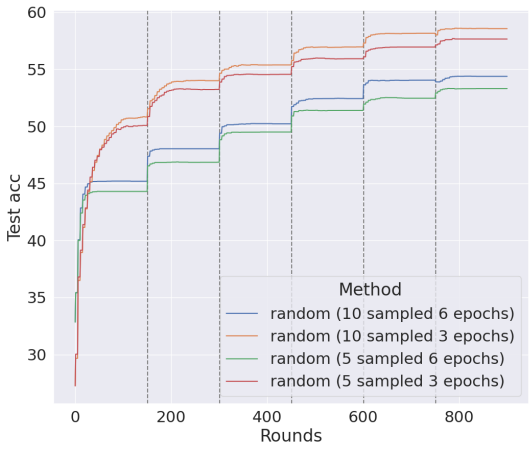
Experiments	2 Cluster	4 Cluster
CIFAR-10 Label Swaps	Cluster A: None	Cluster A: None
	Cluster B: 0 & 1	Cluster B: 0 & 1
		Cluster B: 0 & 1
		Cluster B: 0 & 1
CIFAR-10 Label Drift	Cluster A: 0, 1, 8, & 9	Cluster A: 0 & 1
	Cluster B: 3, 4, 5, & 7	Cluster B: 3 & 4
		Cluster C: 5 & 7
		Cluster D: 8 & 9

A.4 Additional Experiments

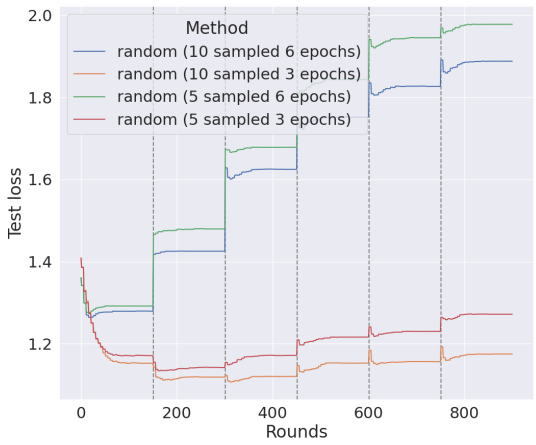
Below we present additional experimental results that did not fit in the main report.

A.4.1 Sampling and Training Impact on `Random` and `DAC`

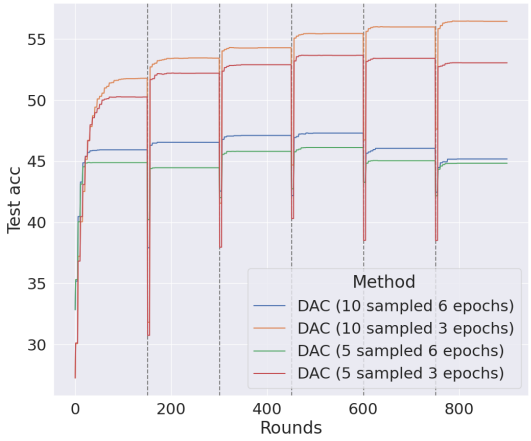
In figure A.1, we can see how the `Random` and `DAC` algorithms performance depends on the number of sampled clients as well as the number of local training epochs. The specific experiment utilized in these figures is a distributed label drift on the CIFAR-10 dataset with two clusters and random cluster identities assignment. In this experiment, both the number of sampled clients and the number of local training epochs were varied to ensure `HAST` did not have any inherent advantage. This is because, in essence, `HAST` completes two rounds of sampling and training per communication round.



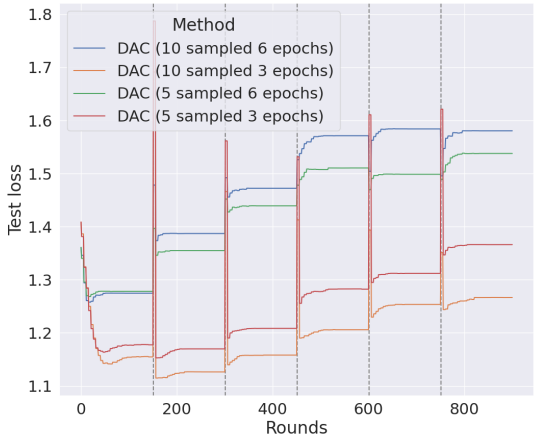
(a) Test accuracy averaged over all clients and clusters.



(b) Test loss averaged over all clients and clusters.



(c) Test accuracy averaged over all clients and clusters.

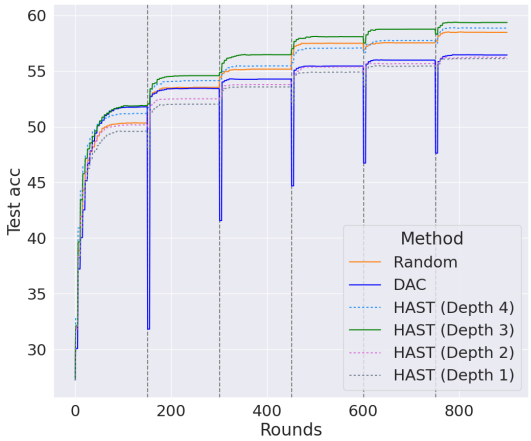


(d) Test loss averaged over all clients and clusters.

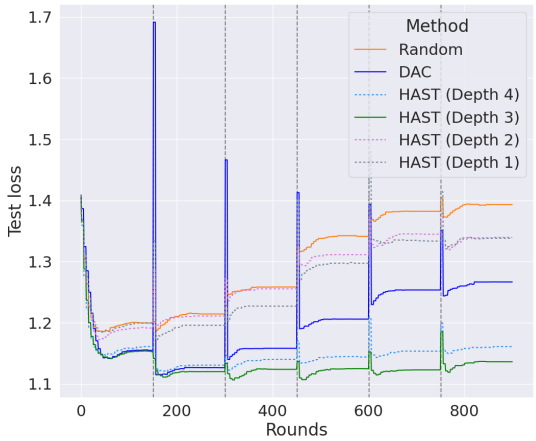
Figure A.1: In the figures we see that `Random`'s and `DAC`'s performance decreases as we double the number of training epochs, while its performance is slightly improved by allowing it to sample twice as many clients.

A.4.2 Aggregation Depth Impact on `HAST`

In figure A.2, we can see how the `HAST` algorithm's performance depends on the aggregation depth. The specific experiment utilized in these figures is a distributed label drift on the CIFAR-10 dataset with two clusters and random cluster identities assignment. In this experiment, aggregation depths zero and five were not tested as they correlate with the `Random` and `DAC` which are shown instead. To make the image more clear, the three mainly used algorithms are drawn fully, while the unused aggregation depths are dotted.



(a) Test accuracy averaged over all clients and clusters.

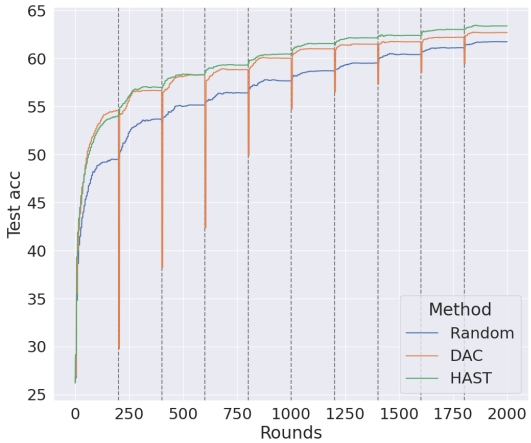


(b) Test loss averaged over all clients and clusters.

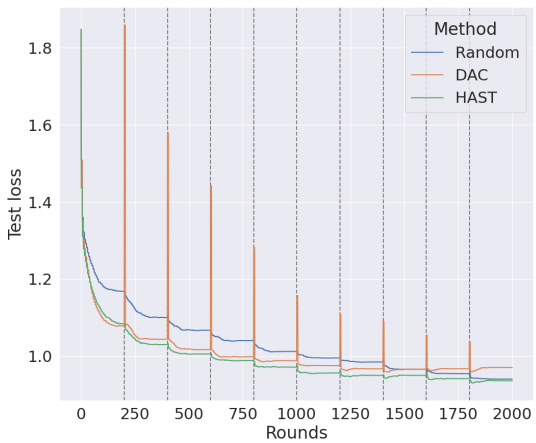
Figure A.2: In both figures we see how `HAST` with aggregation depth three outperforms all other algorithms. In both figures, `Random` correlates to aggregation depth zero, and `DAC` with aggregation depth five.

A.4.3 Distributed Label Drift - 2 Clusters

In figure A.3, we can see that the `HAST` algorithm achieves a higher accuracy and lower loss in all time steps after the initial one.



(a) Test accuracy averaged over all clients and clusters.

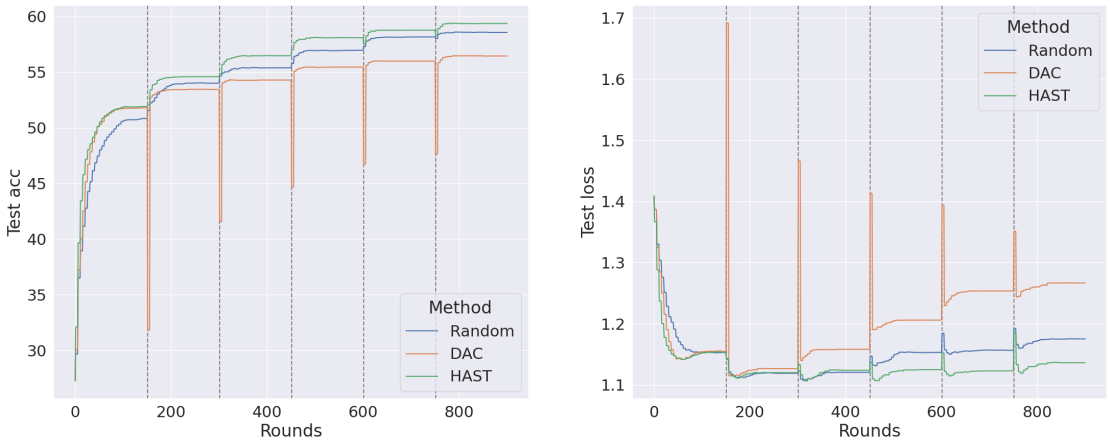


(b) Test loss averaged over all clients and clusters.

Figure A.3: Distributed label drift test accuracy and loss for all algorithms on the CIFAR10 dataset. In the experiment, two clusters exist and the clients are assigned random clusters at each time step.

We also experimented with changing the labels associated with each cluster to en-

sure that the results were consistent. In figure A.4, we see averaged results over three different random cluster label assignments. Here we once again see that our algorithm `HAST` manages to outperform the baseline models throughout all time steps.



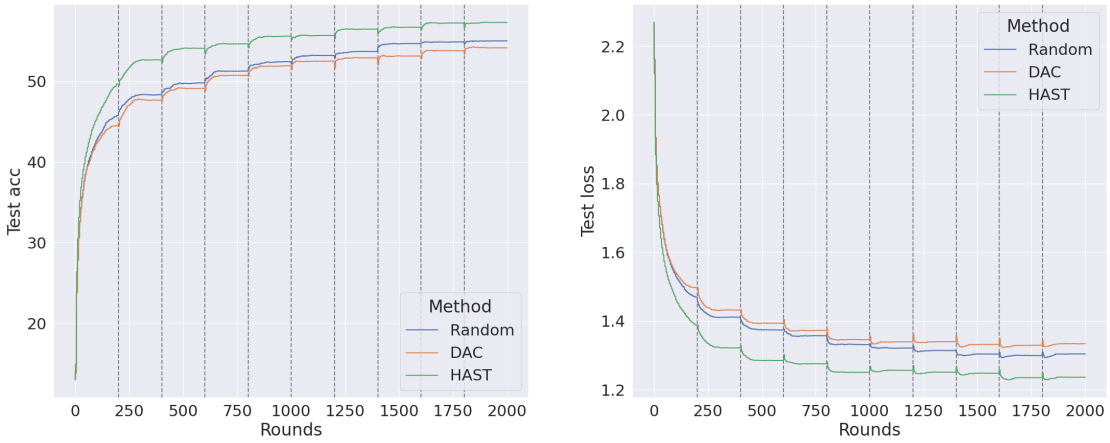
(a) Test accuracy averaged over all clients and clusters.

(b) Test loss averaged over all clients and clusters.

Figure A.4: Distributed label drift test accuracy and loss for all algorithms on the CIFAR10 dataset averaged over three different cluster label assignments. In the experiment, two clusters exist and the clients are assigned random clusters at each time step.

A.4.4 Distributed Label Swap - 2 Clusters

In figure A.5, we can see that the `HAST` algorithm achieves much higher accuracy and lower loss in all time steps.



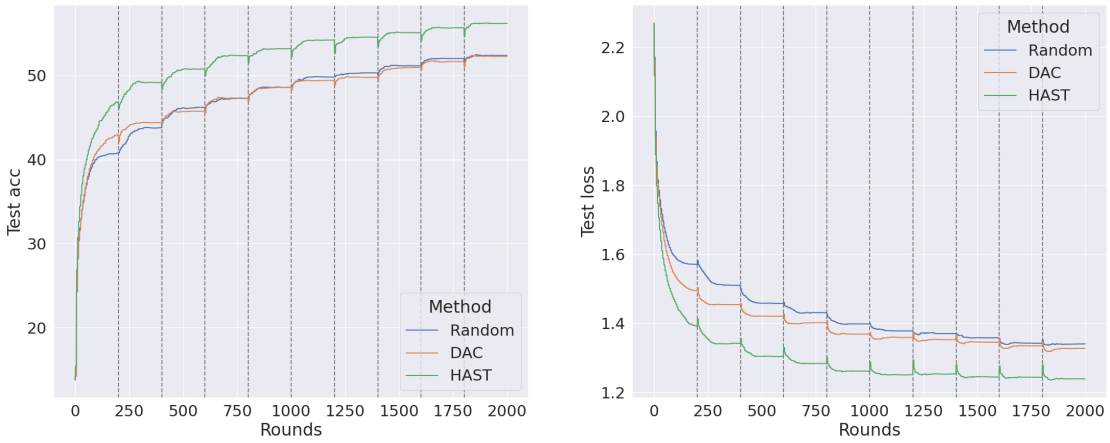
(a) Test accuracy averaged over all clients and clusters.

(b) Test loss averaged over all clients and clusters.

Figure A.5: Distributed label swap test accuracy and loss for all algorithms on the CIFAR10 dataset. In the experiment, two clusters exist and the clients are assigned random clusters at each time step.

A.4.5 Distributed Label Swap - 4 Clusters

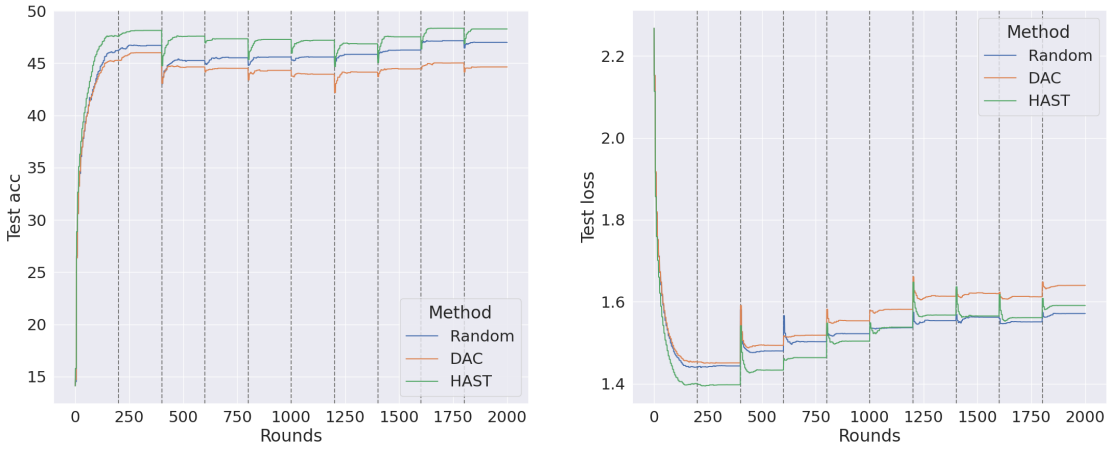
In figure A.6 and A.7, we can see that the `HAST` algorithm achieves much higher accuracy and lower loss in all time steps when using randomized clusters and that when using staggered patterns we still achieve the highest accuracy, but our loss becomes similar to the `Random` algorithm.



(a) Test accuracy averaged over all clients and clusters.

(b) Test loss averaged over all clients and clusters.

Figure A.6: Distributed label swap test accuracy and loss for all algorithms on the CIFAR10 dataset. In the experiment, two clusters exist and the clients are assigned random clusters at each time step.



(a) Test accuracy averaged over all clients and clusters.

(b) Test loss averaged over all clients and clusters.

Figure A.7: Distributed label swap test accuracy and loss for all algorithms on the CIFAR10 dataset. In the experiment, four clusters exist and the clients are assigned clusters based on the drift pattern in figure 3.3.

A.5 Metric Results

Table A.8: †Number of clusters, Notes: G_1: see experiment 1, G_2: see experiment 2, G_3: see experiment 3 in table 3.1. S_1: correlates to a 50-50 split, S_2: 70-30 split, S_3: 80-20 split

	#†	Drift Pattern	Average Accuracy			Convergence Rate (rounds to reach 90%)			Task-Switch Resilience (accuracy drop in %)			Note
			Random	DAC	HAST	Random	DAC	HAST	Random	DAC	HAST	
Distributed Covariate Drift	4	Random	58,98 ±4.49	58,37 ±4.86	64,37 ±5.78	24,66 ±17.97	24,33 ±16,50	21,33 ±8.18	1,98 ±0.98	1,99 ±1.01	4,94 ±0.78	
Distributed Label Drift	2	Staggered	57,85 ±3,85	56,13 ±5,92	59,1 ±4,78	22,10 ±15,37	8,44 ±9,41	13,22 ±8,03	2,64 ±1,99	2,94 ±4,12	1,89 ±1,43	
	4	Staggered	80,87 ±4,61	80,96 ±4,29	82,23 ±3,9	22.00 ±12,17	16,66 ±11,16	21,11 ±9,74	6,82 ±12,16	8,03 ±12,16	5,97 ±11,34	
	2	Random	58,61 ±4,48	61,02 ±3,27	62,26 ±3,35	32,77 ±20,13	9,77 ±7,50	26,66 ±22,91	1,62 ±0,95	11,01 ±7,94	1,26 ±0,82	
	4	Random	79,79 ±3,57	82,57 ±1,67	83,15 ±2,09	23,77 ±17,22	15.00 ±13,22	16.00 ±18,46	1,49 ±1,06	10,03 ±14,76	1,02 ±0,96	
Distributed Label Drift Mixed Labels	2	Random	61,86 ±2,61	60,8 ±1,74	62,82 ±2,34	14,6 ±12,53	2,00 ±0,00	17,00 ±9,67	1,23 ±0,74	27,63 ±0,88	1,54 ±0,83	G_1
	2	Random	64,08 ±2,86	63,55 ±2,05	66,16 ±2,72	15,40 ±12,61	2,00 ±0,00	22,2 ±15,19	1,23 ±0,81	28,30 ±1,70	1,67 ±0,85	G_2
	2	Random	62,28 ±2,77	60,59 ±1,69	62,91 ±2,5	17,60 ±12,15	2,00 ±0,00	17,6 ±11,18	1,34 ±0,69	24,01 ±3,27	1,51 ±0,89	G_3
		Average	62,74 ±2,75	61,64 ±1,83	63,96 ±2,52	15,86 ±12,43	2,00 ±0,00	18,93 ±12,24	1,27 ±0,75	26,65 ±2,19	1,57 ±0,86	
Label Swap	2	Staggered	49,13 ±1,61	48,31 ±1,06	51,01 ±1,45	15,11 ±14,03	7,55 ±8,97	11,22 ±9,43	0,95 ±0,32	1,02 ±0,66	1,07 ±0,42	
	4	Staggered	47,72 ±1,2	45,89 ±1,04	49,88 ±0,98	13,66 ±8,00	7,44 ±6,44	25,33 ±19,69	1,24 ±0,54	1,03 ±0,40	2,09 ±1,19	
	2	Random	54,41 ±3,02	53,62 ±3,12	57,99 ±3,02	24,88 ±17,41	21,55 ±15,86	24,88 ±15,07	1,19 ±0,61	1,35 ±0,87	1,22 ±0,66	
	4	Random	49,27 ±3,91	49,77 ±3,19	55,06 ±3,33	33 ±16,69	26,44 ±12,49	40,33 ±14,59	1,55 ±0,81	1,37 ±0,47	2,26 ±0,66	
Homogeneous	1	Random	47,59 ±2,37	49,95 ±2,35	50,57 ±2,49	24,8 ±21,99	20,8 ±4,01	17,6 ±8,35	1,80 ±0,83	1,43 ±0,20	1,45 ±0,71	
Distributed Quantity Shift	2	Random	57,61 ±3,38	57,18 ±2,16	59,14 ±2,76	29,6 ±21	5,2 ±1,93	16,8 ±10,85	1,67 ±0,86	14,37 ±4,61	1,50 ±1,12	S_1
	2	Random	59,95 ±1,40	62,17 ±2,08	61,47 ±2,54	8,80 ±9,6	14,00 ±7,92	21,00 ±13,26	1,31 ±0,88	11,98 ±1,07	3,44 ±1,08	S_2
	2	Random	60,35 ±1,21	63,6 ±1,98	63,04 ±2,38	6,80 ±11,6	25,40 ±5,23	25,2 ±3,18	0,95 ±0,88	7,79 ±1,54	6,66 ±0,84	S_3

DEPARTMENT OF MATHEMATICAL SCIENCES
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY