CAEFL: Composable and Environment Aware Federated Learning Models

Ruomeng (Cocoa) Xu
r.xu.1@research.gla.ac.uk
School of Computing Science,
University of Glasgow
Glasgow, United Kingdom

Anna Lito Michala
annalito.michala@glasgow.ac.uk
School of Computing Science,
University of Glasgow
Glasgow, United Kingdom

Phil Trinder
Phil.Trinder@glasgow.ac.uk
School of Computing Science,
University of Glasgow
Glasgow, United Kingdom

Abstract

Federated Learning allows multiple distributed agents to contribute to a global machine learning model. Each agent trains locally and contributes to a global model by sending gradients to a central parameter server. The approach has some limitations: 1) some events may only occur in the local environment, so a global model may not perform as well as a specialized model; 2) changes in the local environment may require an agent to use some dedicated model, that is not available in a single global model; 3) a single global model approach is unable to derive new models from dealing with complex environments. This paper proposes a novel federated learning approach that is local environment aware and can compose new dedicated models for complicated environments. The approach is implemented in Elixir to exploit pattern matching and hot-code-swapping to maximize versatility. Our proposed approach outperforms the state of the art FL by an average of 7-10% for the MNIST dataset and 2% for the FashionMNIST dataset in specific and complex environments.

CCS Concepts: • **Computing methodologies** → **Supervised learning**; **Neural networks**.

Keywords: Federated learning, Neural networks, Distributed systems, Elixir, BEAM

ACM Reference Format:
Ruomeng (Cocoa) Xu, Anna Lito Michala, and Phil Trinder. 2022. CAEFL: Composable and Environment Aware Federated Learning Models. In *Proceedings of the 21st ACM SIGPLAN International Workshop on Erlang (Erlang '22).* ACM, New York, NY, USA, 12 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

Federated Learning (FL) addresses the challenge of machine learning when the training data is distributed between agents with poor network connections[20]. Conventional FL trains a single global model that is used for all environments. To obtain the global model, agents in different environments train their models locally. Then a central Parameter Server (PS) periodically randomly selects a subset of agents to update the global model. It does so by collecting model gradients from the chosen agents and updating the global model with some average value. The new global model can be sent to, or requested by, agents to be used for training or inference. Over time the global model accuracy improves.

However, a global model may fail in some local environments because of the statistically heterogeneity in the data [23, 27, 32]. For example, an autonomous vehicle in the city faces driving challenges and events (pedestrians, rain, traffic lights). These may differ from those encountered in rural areas (uneven road, narrow roads, side winds). So while FL produces a global model that works in various environments, there are tasks where agents only operate in one specific environment. Moreover, the distribution of the sample space in a particular environment may be unique.

That is to say, some events are highly related to the local environment, especially when they are rare events. For example, an autonomous vehicle may encounter fallen trees on mountain roads, or experience vibration in earthquake-prone areas. These events can sometimes be safely ignored, e.g. wrongly predicting a bus as a truck. In other cases, there is no harm if the model gives a wrong prediction for an event, e.g. slowing down after wrongly predicting that a pedestrian is about to cross the road. For other events wrong predictions have serious consequences, e.g. failing to identify pedestrians, and not giving way.

A change in an agent's environment is likely to change the distribution of the sample space, requiring a new model. A single global model cannot provide this adaption, and while the global model may be effective in the new local environment, it may not perform as expected [27, 38]. For example, a global model trained for driving in sunny or rainy weather might not perform well on windy icy roads.

An agent's new environment may combine sets of events from two or more simple environments previously seen. For

Erlang '22, September 11–16, 2022, Ljubljana, Slovenia

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/XXXXXXX.XXXXXXX

example, driving in the rain with side winds. In such cases, a single specialized model might not perform as well as in the simple environment. Traditional methods for combining models do not take this approach. For example, the ''one-vs-all" (OvA), a.k.a ''one-vs-rest" (OvR), strategy trains N models (equal to the number of classes), where each model only has a binary output: whether this input belongs to a specific class i **??**. Then the class that has been predicted by most of the N models will be selected as the final prediction. In general, ensemble methods improve the predictive performance beyond a single model by training multiple models and combining their predictions**??**. Ensemble methods don't address the challenge of combining environments, as the agent has to feed the same input to all specialized models, and this increases the computation power needed and the inference time.

This paper proposes a novel federated learning approach that (1) enables the agent to be aware of its environment; (2) makes it possible to derive models specialized for specific environments that outperform the global model; and (3) makes it possible to compose models to create new models for complex environments. The paper contributions are:

1. A Composable and Environment Aware Federated Learning (CAEFL) approach that aggregates model updates calculated in similar local environments and derives multiple global specialized models each for a specific environment (Section 4).
2. A CAEFL Communication Protocol (CAEFL-CP) for agents to submit multiple model updates with environmental tags generated from local sensors. CAEFL-CP also allows agents to request specialized models based on the environmental tags (Section 5.2).
3. An *elite selection* process applied by CAEFL (Section. 5.3) on the PS when generating specialized models and new composite models appropriate for complex environments (Section 5.4).
4. An open-source Elixir library that implements the above methods[1].

The remainder of this paper is structured as follows. Section 2.1 reviews conventional FL technologies and how they train a global model using stochastic gradient descent. Then it outlines the non-i.i.d data challenge where the samples (random variables) do not have the same probability distribution as the others or are not mutually independent. Section 2.2 reviews similar approaches to solve this challenge and establishes the gap in the state-of-the-art that this paper addresses. Thereafter, a formal description of local environment aware FL is given, followed by the proposed method for the derivation, aggregation, and combination of local models in the PS for specialized models (Section 3). Next it describes the proposed method and the experimental settings used in Section,6. The experiment results are reported and analysed in Section 7. Conclusions are drawn in Section 9.

2 Background

2.1 Federated Learning

Typically FL coordinates multiple devices with a central parameter server (PS) to train a global model collaboratively. Each worker downloads the current model from the PS and computes its model update on its local dataset, e.g. using stochastic gradient descent (SGD). The local model is then uploaded to the PS, where all the received models are averaged periodically. The new global model may be sent back to, or requested by, agents. In the long run, the global model will be continuously updated and thus will gradually perform the task with better overall accuracy [28, 31, 35]. The general form of SGD using mini-batches [10] can be written as follows in Eq. 1 2 and defined in [16]:

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta_t \mathbf{g}_t \tag{1}$$

$$\mathbf{g}_t = \nabla f(\mathbf{w}_t; \mathbf{x}_t) \tag{2}$$

where $\mathbf{w}_t$ and $\mathbf{w}_{t+1}$ are the parameters of the model at iteration t and $t+1$ respectively, $\eta_t > 0$ is the learning rate, and $\mathbf{g}_t$ is the gradient of the error function f evaluated at $\mathbf{w}_t$ given input mini-batch $\mathbf{x}_t$. When $\mathbf{w}_t$ in Eq.(1) is fixed, we can partition the input samples $\mathbf{x}_t$ and compute the gradient for each subset as follows:

$$\mathbf{g}_t = \sum_{i=1}^{n} \nabla f(\mathbf{w}_t; \mathbf{x}_t^i) \tag{3}$$

$$\text{where } \begin{cases} \mathbf{x}_t^i \cap \mathbf{x}_t^j = \emptyset, & \forall i \neq j \\ \cup_{i=1}^{n} \mathbf{x}_t^i = \mathbf{x}_t \end{cases} \tag{4}$$

The n subsets of the training data will be distributed to available agents. Agents feed the training data as the input to the neural network and apply the SGD method to calculate the corresponding gradients in a parallel manner as long as the model parameters $\mathbf{w}_t$ are the same across these agents [1, 5, 11, 22]. After calculating the gradients, the PS first gathers all gradients from all agents, sums them and finally takes the average value as the gradient for the global model. This approach produces the same value as processing the input $\mathbf{x}_t$ on a single agent. That brings us to the most straightforward approach, synchronous SGD (Sync-SGD).

In Sync-SGD, the master node of the cluster splits the workload and dispatches work to all agents in each iteration. Meanwhile, the master node must ensure that each worker's model parameters are identical when calculating the gradients. Then all nodes will wait for the slowest worker to report its gradient back to the master node before starting the next iteration [33]. The authors in [7] implement such a mechanism using strict synchronization for every iteration,

[1] https://github.com/cocoa-xu/caefl

demonstrating a better convergence speed than training on a single node. The workers' and PS's algorithms are shown in Algo. 1 and 2 based on the process described in [6]. These implementations refer to a fixed η; where η is the learning rate ranging between 0 and 1. Later in [29, 42] adaptive learning rates were proposed to further improve accuracy.

Despite sync-SGD's simple form and ease of parallelization, further research [3, 25, 36, 39, 45] has observed and reported that synchronization and communication cost between workers is a major bottleneck when applying sync-SGD in a distributed system because we have to (i) wait for the slowest agent to complete and (ii) dispatch the new model parameters to all workers at each round. Those drawbacks hinder the scalability and deteriorate runtime performance.

Algorithm 1: Sync-SGD (agent) based on [6]

```
for t=1:T do
    wait for $\mathbf{w}_t$ to be available;
    $\mathbf{w}_t \leftarrow$ fetch $\mathbf{w}_t$ from PS;
    $\mathbf{x}_t^i \leftarrow$ current mini-batch of data on worker $i$;
    $\mathbf{g}_t^i \leftarrow \nabla f(\mathbf{w}_t; \mathbf{x}_t^i)$ /* calculate gradient */ ;
    acquire lock on PS;
    send $\mathbf{g}_t^i$ to PS;
    release lock;
```

Algorithm 2: Sync-SGD (PS) based on [6]

```
initialise and signal $\mathbf{w}_1$ is available;
for t=1:T do
    while the slowest worker is not done do
        wait for a gradient $\mathbf{g}_t^i$ to arrive;
        $\mathbf{w}_t \leftarrow \mathbf{w}_t - \eta \mathbf{g}_t^i$ /* update global model */ ;
    $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t$;
    signal $\mathbf{w}_{t+1}$ is available;
```

To allow asynchronous gradient updates, asynchronous SGD (Async-SGD) has been introduced and further improved in several research publications [8, 9, 17, 18, 34, 43] as it does not require strict synchronization prior to the training stage which reduces training time when compared to Sync-SGD.

A simplified algorithm of the Async-SGD agent and its PS are depicted in Algo.3 and 4 based on the description and algorithm of Async-SGD in [4]. The authors in [9] find that although Async-SGD is rarely applied to non-convex problems, it is a very good approach when training deep networks, particularly when combined with Adagrad [14] adaptive learning rates.

On the other hand, the asynchronous characteristics of Async-SGD also inevitably introduce the issue of stale gradients; as any worker may calculate gradients based on outdated model parameters. This situation happens when a node gives $\mathbf{g}_t = \nabla f(\mathbf{w}_t; \mathbf{x}_t^i)$ while the global model $\mathbf{w}_{t+u}, u \geq 1$ is u-iterations ahead.

Algorithm 3: Async-SGD (agent) based on [4]

```
wait for $\mathbf{w}_1$ to be available;
for t=1:T do
    $\mathbf{w}_t \leftarrow$ fetch latest $\mathbf{w}$ from PS;
    $\mathbf{x}_t^i \leftarrow$ current mini-batch of data on worker $i$;
    $\mathbf{g}_t^i \leftarrow \nabla f(\mathbf{w}_t; \mathbf{x}_t^i)$ /* calculate gradient */ ;
    send $\mathbf{g}_t^i$ to PS;
```

Algorithm 4: Async-SGD (PS) based on [4]

```
initialise and signal $\mathbf{w}_1$ is available;
$\mathbf{w} \leftarrow \mathbf{w}_1$;
while termination criteria not satisfied do
    wait for a gradient $\mathbf{g}$ to arrive;
    $\mathbf{w} \leftarrow \mathbf{w} - \eta \mathbf{g}$ /* update global model */ ;
```

Although Async-SGD can act as a pain reliever and alleviate the FL system from long waiting time in Sync-SGD, the main challenge in both Sync- and Async-SGD lies in that the gradient is evaluated at a specific point (model with specific parameter values) and the gradient is, technically, only valid at that point. In both practice and research [46], the stale gradient can be accepted with some trade-offs like calculation time or model performance, if it allows gradients that are t' iterations behind the latest global model. That is to say, gradient $\mathbf{g}_t^i = \nabla f(\mathbf{w}_t; \mathbf{x}_t^i)$ on agent i evaluated at $\mathbf{w}_t$ with local training data $\mathbf{x}_t^i$ is only valid if it is used to update the global model at that point $\mathbf{w}_t$ as in Eq. 5.

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \eta \sum_{i=1}^{n} \mathbf{g}_t^i \tag{5}$$

The PS can only choose between whether to apply it to the global model $\mathbf{w}_t$ or reject it because of staleness.

In this paper, we propose an adaptation of Async-SGD to solve this challenge and formally define our SGD approach in Section 3. In the following subsections, CAEFL is differentiated from other distributed ML approaches beyond SGD.

2.2 The Non-IID Data Challenge

Much research [26, 37] has been undertaken in non-i.i.d data, focusing on non-i.i.d data caused by the user. Our proposed method concentrates on the environment. These are two seemingly similar but different challenges:

1. **User-centric research question**: How should we train a personalized model in FL? Or in other words: How should the vehicle drive to suit the driver's preference?

2. **Environment-centric research question**: How to obtain a specialized model that adapts to a specific environment? Or in other words: How should the vehicle drive in different weather and road conditions?

In the former, the local model bonds to a specific user and does not necessarily contribute to the training of a global PS model. Whereas the local model bonds to a specific environment in the latter one and could contribute to a global model for this environment or even a general global model.

Another active research domain in ML discusses ensemble models [40?]. Ensemble models are often used to improve the accuracy of a problem by training identical or heterogeneous models on the same dataset and then comparing or merging the outputs to arrive at a final prediction. The main difference between our proposed combined models and ensemble models is that in an ensemble model, the sub-components (estimators) remain autonomous. For example, the random forest model will store all k decision trees separately. Although the sub-components (estimators) in an ensemble model can be extracted as a standalone predictor, they cannot be merged. Even if we can somehow merge these sub-trees into a single decision tree, the total number of parameters will be the sum of them in all sub-trees. Whereas in our proposed tagged model updates, once we obtain the gradient set we requested, we can merge them back into a single neural network model, regardless of the set size. This merging capability only depends on the architecture of the neural network being identical.

3 Defining a Composable Environment Aware Federated Learning Model

Let $\mathbf{x}_t \in \mathbb{R}^d$ denote d-dimensional training samples collected at epoch t. An epoch is completed when all training data is used exactly once to update the model from $\mathbf{w}_t$ to $\mathbf{w}_{t+1}$, where $\mathbf{w}_t$ is the neural network model at epoch t. Let $f(\mathbf{w};\mathbf{x}) : \mathbb{R}^d \times \mathbf{w} \rightarrow \mathbb{R}^+$ be the error function that measures the prediction error of model $\mathbf{w}$ when given inputs $\mathbf{x}$. K is the total number of mini-batches of training data $\mathbf{x}_t = \bigcup \mathbf{x}_{tk}, k = 1 \cdots K$. The goal is to find a model $\mathbf{w}^* = \mathbf{w}_{T+1}$ among all feasible $\mathbf{w}$ in the parameter space that gives minimal error after T epochs as shown in Eq. 6 based on the description in [2, 13].

$$\mathbf{w}^* = \mathbf{w}_{T+1} = \underset{\textit{all feasible } \mathbf{w}}{\arg\min} \sum_{t=1}^{T} \frac{1}{K} \sum_{k=1}^{K} f(\mathbf{w}_t; \mathbf{x}_{tk}) \tag{6}$$

To parallelize the training process, the training samples are firstly subdivided into n subsets $\mathbf{x}_t = \bigcup \mathbf{x}_t^i, i = 1, \cdots n$ obeying the conditions in Eq. 4. This is equivalent to update $\mathbf{w}_t$ using Eq. 7.

$$\begin{aligned}
\mathbf{w}_{t+1} &= \mathbf{w}_t - \eta \sum_{i=1}^{n} \nabla f(\mathbf{w}_{\mathbf{t}}; \mathbf{x}_t^i) \\
&= \mathbf{w}_t - \eta \sum_{i=1}^{n} \frac{1}{K} \sum_{k=1}^{K} \nabla f(\mathbf{w}_{\mathbf{t}}; \mathbf{x}_{tk}^i) \\
&= \mathbf{w}_t - \eta \sum_{i=1}^{n} \frac{1}{K} \sum_{k=1}^{K} \mathbf{g}_{tk}^i = \mathbf{w}_t - \eta \sum_{i=1}^{n} \mathbf{g}_t^i \\
&= \mathbf{w}_t - \eta \overline{\mathbf{g}_t}
\end{aligned} \tag{7}$$

where η is the learning rate and $\mathbf{g}_t^i$ is the observed gradient at point $\mathbf{w}_t$ on agent i when given training samples $\mathbf{x}_t^i$. The expected value $E(\overline{\mathbf{g}_t})$ should be equal to the true gradient $\mathbf{g}_t$ at point $\mathbf{w}_t$ when sufficient samples are presented in $\mathbf{x}_t$, i.e., $\overline{\mathbf{g}_t}$ only gives the estimated gradient at $\mathbf{w}_t$ based on the observed input samples.

Let $C = \{c_l | l = 1, 2, \cdots\}$ be a set of criteria. A subset C_k identifies an environment e_k if all environment conditions match those criterion declared in C_k. Hence the training process for specialized model $\mathbf{w}_{e_k}$ can be defined as in Eq. 8.

$$\begin{aligned}
\mathbf{w}_{t+1e_k} &= \mathbf{w}_{te_k} - \eta \sum_{i=1}^{n} \nabla f(\mathbf{w}_{te_k}; \mathbf{x}_{te_k}^i) \\
&= \mathbf{w}_{te_k} - \eta \sum_{i=1}^{n} \mathbf{g}_{te_k}^i
\end{aligned} \tag{8}$$

where $\mathbf{x}_{te_k}^i$ represents the samples observed in environment e_k on agent i at epoch t, $\mathbf{w}_{te_k}$ is the specialized model for the specific environment e_k at t-th epoch, and $\mathbf{g}_{te_k}^i$ is the corresponding gradient given by the specialized model $\mathbf{w}_{te_k}$ and samples $\mathbf{x}_{te_k}^i$.

A key novelty of our approach is the $c(\cdot)$ function that composes a set of specialized models $M = \{\mathbf{w}_{e_k}\}$ into a composite model $\mathbf{w}_c$ for the given environment set $\{e_k\}$. $\mathbf{w}_c$ aims to classify events in environment set $\{e_k\}$. For instance, $M = \{\mathbf{w}_{e_1}, \mathbf{w}_{e_2}\}$, and class 1, 2 are present in e_1, e_2 respectively, then using $\mathbf{w}_c = c(M)$ should be able to predict both classes and give better accuracy rate for class 2 than using $\mathbf{w}_{e_1}$, the same goes for class 1 and $\mathbf{w}_{e_2}$.

4 Environment Aware FL

Traditional FL transfers the samples from each edge device to a central PS and gains knowledge from these samples by transforming them into an optimization problem and applying appropriate machine learning techniques. More recently, to preserve privacy and minimize communication, edge devices contribute to the global model by uploading the gradients calculated based on observed samples $\mathbf{x}^i$. However, gradients contributed from samples in different environments are usually averaged and added to the global model. The environmental information is not taken into consideration.

To exploit local environment information and to enable environment aware models, an environment e_k is identified with a subset C_k of a set of criteria $C = \{c_l | l = 1, 2, \cdots\}$ if e_k match those criteria in C_k. In practice $\mathbf{x}^i$ collected on agent i may be subdivided to multiple smaller sets because they may be observed under different environment conditions e_k.

Each C_k is a set that contains one or more semantic or numerical criteria. For the autonomous vehicle example, the environmental tags can be the *time, temperature, humidity, sea level, and wind speed.* One of the benefits of doing so is that PS can store these tagged model updates and generate a new specialized model with a subset of them on-demand. The autonomous vehicle example might have the following c_l criteria set or subsets thereof:

$$c_1 = \{\textit{snow: true}\} \qquad c_2 = \{\textit{fog: true}\}$$
$$c_3 = \{\textit{rain: true}\} \qquad c_4 = \{\textit{sidewinds: true}\}$$

So c_1 and c_2 denote an environment e_1 that is snowy and foggy, c_3 denotes an environment with rain (e_2), and c_4 denotes an environment with sidewinds (e_3) (Fig. 1).

After obtaining the set of environments $E = \{e_k\}$, the training set $\mathbf{x}^i$ on agent i can be subdivided into $\{\mathbf{x}^i_{e_k}\}$. Then the local model is trained with $\{\mathbf{x}^i_{e_k}\}$ and the corresponding tagged model updates $\{\mathbf{d}^i_{e_k}\}$ will be stored. Next, tagged model updates will be uploaded to a PS (not necessarily a central server) and stored there. For instance, if the agent requests a dedicated model for snowy weather, then tagged model updates that match c_1 will select and the average value for each parameter will be used to generate a specialized model for that agent. If an agent requests a composite model for rainy and foggy weather, then tagged model updates that either match c_1 or c_2 will be selected and averaged to generate a new composite model. Fig.2 depicts this process. Moreover, if all tagged model updates are selected and merged the resulting model is the global model $\mathbf{w}_g$ produced by conventional federated learning.

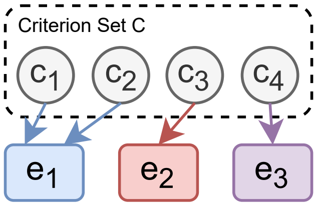

Figure 1. Three environment classes based on different criterion/criteria subsets.

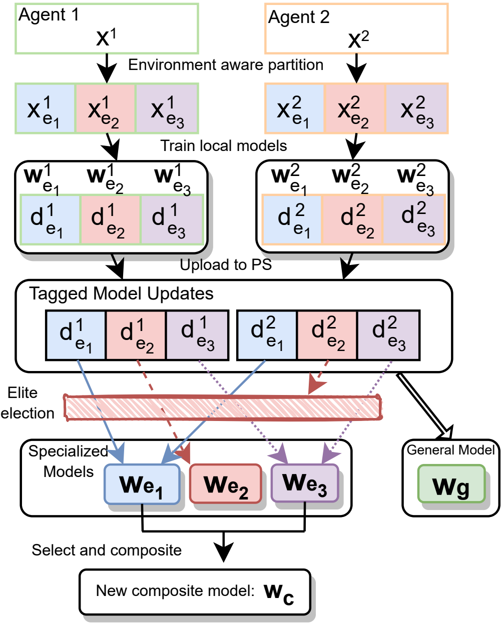

Figure 2. A demonstration of tagged model updates, specialized model generation and deriving new composite models. Tagged model updates will first be rebuilt back to a model and go through the *elite selection* process. *elite selection* will choose the top *r%* of the rebuilt models and their corresponding tagged updates will be merged into the specialized models. However, all tagged model updates will be stored for future use when composing a new composite model for complex environments.

5 Composing Environment Aware FL Models

5.1 Model Updates

In this work, we first rearange Eq. 5 to Eq. 9, then substitute $\mathbf{w}_{t+1}$ with $\mathbf{w}^i_{t+1}$, η with η^i in Eq. 10

$$\mathbf{w}_{t+1} - \mathbf{w}_t = \eta \sum_{i=1}^{n} \mathbf{g}^i_t \tag{9}$$

$$\forall_{i=1,\cdots,n} \quad \mathbf{w}^i_{t+1} - \mathbf{w}_t = \eta^i \mathbf{g}^i_t = \mathbf{d}^i_t \tag{10}$$

where $\mathbf{w}_t$ is still the global model with specific parameter values, but $\mathbf{w}^i_{t+1}$ and η^i is the local updated model and local learning rate on agent i. $\mathbf{d}^i_t$ denotes the difference between local updated model $\mathbf{w}^i_{t+1}$ and the global model $\mathbf{w}_t$. Since all parameter values are fixed in the pulled global model, we can use a unique identifier u_t to reference this fixed model.

The key change in our work is that CAEFL sends $\mathbf{d}^i_t$ and u_t to the PS instead of the gradient $\mathbf{g}^i_t$. This first allows the agent to locally update the global model $\mathbf{w}_t$ multiple times before reporting back to the PS. This delay can happen when there is no internet connection over a period of time or the agent computes slowly. The update is stale but not necessarily useless as the agent could have trained the model with valuable training samples. For example, an autonomous vehicle driving in a mountain area, plateau or desert for a week where it has poor to no internet coverage. Secondly, the PS can now rebuild the updated model using $\mathbf{d}_t^i$ and u_t instead of considering whether to apply the gradients to the global model or discard it. Lastly, this gives the agent the flexibility to apply adaptive learning rate methods [19, 44] to update the local model $\mathbf{w}_{t+1}^i$.

5.2 Tagged Model Updates

In our proposed CAEFL approach, for each input sample $\mathbf{x}^{ij} \in \mathbf{x}^i$ on agent i, there is an environment feature $\mathbf{e}^{ij}$ associated with it where $\mathbf{e}^{ij}$ is a *key-value* dictionary that stores the corresponding environment readings when $\mathbf{x}^{ij}$ is collected. We use $E = \{e_1, e_2, \cdots, e_p\}$ to denote p classes of environments. If an $\mathbf{e}^{ij}$ matches an environment e_k, then the corresponding training sample $\mathbf{x}^{ij}$ will be collected into a set $\mathbf{X}_{e_k}$.

There are two potential ways to define a match between an $\mathbf{e}^{ij}$ and an environment e_k. The first one is *partial match* where an $\mathbf{e}^{ij}$ matches an environment e_k as long as it satisfies all criteria defined in C_k. For example, $\mathbf{e}^{11} = \{$*rain: true, fog: true*$\}$ matches e_1 if $C_1 = \{$*rain: true*$\}$ when using partial match. This is useful because rain is presented in the sample $\mathbf{x}^{11}$, that is exactly what is required to satisfy the criteria in C_1. The fog presented in $\mathbf{x}^{11}$ can be seen as an extension of the rain environment. Using partial match will add the sample $\mathbf{x}^{11}$ into set $\mathbf{X}_{e_1}$. The second approach is a *full match*, where all features must be present in both $\mathbf{e}^{ij}$ and C_k, and features in $\mathbf{e}^{ij}$ should match the corresponding criteria in C_k. A full match can help when a strict and finer level of the environment set E is demanded. The optimization problem using SGD algorithm on a single node can be described in Algo. 5.

5.3 Specialized Models

After each tagged model updates $\{\mathbf{d}_{e_k}^i\}$ is calculated, they will go through an *elite selection* process. Firstly, they will be sent to a central PS along with corresponding unique identifiers $\{u_{e_k}\}$. Then the PS can find the model $\mathbf{w}_{e_k}$ using the unique identifier u_{e_k} and generate temporary models $\mathbf{w}_{e_k}^i = \mathbf{w}_{e_k} + \mathbf{d}_{e_k}^i$. After generating temporary models using tagged model updates from all agents, PS will calculate the accuracy of those temporary models using test samples from environment e_k. For each environment e_k, temporary models will be sorted by their test accuracy and only the top $r\%$ of them will be kept in a set S_k. Finally, for temporary models in the set S_k, they will be merged into a single specialized model $\mathbf{w}'_{e_k}$ by taking the average parameter value across them. $\mathbf{w}'_{e_k}$ will be the updated model for environment e_k. Algo. 6 describes this process.

Algorithm 5: CAEFL: Calculate Tagged Model Updates using SGD with Mini-batch (agent i)

```
Parameters: Batch size $b$, env. set $E = \{e_1, \cdots, e_p\}$;
Input: sample $\mathbf{x}^{i1} \cdots \mathbf{x}^{im}$, env. feature $\mathbf{e}^{i1}, \cdots \mathbf{e}^{im}$;
for $e_k \in E$ do
    initialise model $\mathbf{w}_{e_k}^i = \mathbf{w}_{e_k}$ and corresponding
      unique identifier $u_{e_k}$ from central PS;
    $\mathbf{X}_{e_k} = \{\}$;
    $g_{e_k}^i = \mathbf{0}$;
    for $\mathbf{x}^{ij} \in \mathbf{x}^i$ do
        $\mathbf{X}_{e_k} \leftarrow \mathbf{X}_{e_k} \cup \mathbf{x}^{ij}$ if $\mathbf{e}^{ij}$ matches $e_k$;
    for $m = 1$ to $s = size(\mathbf{X}_{e_k})/b$ do
        $\mathbf{w}_{e_k}^i \leftarrow \mathbf{w}_{e_k}^i - \eta^i \frac{1}{b} \sum_{d=b(m-1)+1}^{bm} \nabla f(\mathbf{w}_{ie_k}; \mathbf{X}_{de_k})$;
    $d_{e_k}^i \leftarrow \mathbf{w}_{e_k}^i - \mathbf{w}_{e_k}$;
Output: $\{u_{e_k}\}, \{d_{e_k}^i\}$;
```

Algorithm 6: CAEFL: Generate Specialized Models with Tagged Model Updates

```
Parameters: env. sets $E = \{e_k\}$;
Input: tagged model updates $\{\mathbf{d}_{e_k}^i\}$ from all agents,
  unique identifiers $u_{e_k}$;
for $e_k \in E$ do
    find base model $\mathbf{w}_{e_k}$ by $u_{e_k}$;
    $\mathbf{w}_{e_k}^i \leftarrow \mathbf{w}_{e_k} + \mathbf{d}_{e_k}^i$ for agent $i$, $i = 1, \cdots, n$;
    test model $\mathbf{w}_{e_k}^i$ using test samples for env. $e_k$;
    only keep top $r\%$ of them by accuracy in a set $\mathbf{N}$;
    $\mathbf{w}_{e_k} \leftarrow$ average of all models in $\mathbf{N}$;
Output: A set of specialized models $\{\mathbf{w}_{e_k}\}$;
```

Based on the design, it is a requirement that the architecture of the neural network model on all agents should be the same, that means that the value of the neural network parameters can be different while they all have the same number of layers, same input and output shapes and activation methods at each layer.

5.4 Deriving Composite Models

Let C_c be the criteria set for the requested composite environment e_c. To derive the composite models $\mathbf{w}_c$ from the central PS, CAEFL uses tagged model updates stored on the PS. Firstly, the tag of existing tagged model updates will be used to match with the requested criteria set C_c using the *partial match* approach. If the tag partially matches with C_c, then the corresponding tagged model updates $\mathbf{d}_{e_k}^i$ will be used to rebuild back to a model and added to a candidate set N. Next, the accuracy of each model in the set N will be evaluated if there are corresponding test samples available for the composite environment e_c. The top $r\%$ of them will be selected and added to set N'. If there are no test samples corresponding to the composite environment e_c, then all models in N will be selected, i.e, $N' = N$. In the following step, models in set N' will be merged by averaging their parameter values at corresponding positions. Algo. 7 shows this process.

```
Algorithm 7: CAEFL: Derive Composite Models with Tagged Model Updates
  Parameters: tagged model updates $\{\mathbf{d}^i_{e_k}\}$ from all
    agents, unique identifiers $u_{e_k}$, env. criteria sets
    $C = \{C_k\}$;
  Input: requested criteria set $C_c$;
  $\mathbf{N} \leftarrow \{\mathbf{d}^i_{e_k} | C_c \textit{ partial matches } C_k, C_k \in C\}$;
  $\mathbf{N}' \leftarrow$
    $\begin{cases} \textit{top } r\% \textit{ of } \mathbf{N} \textit{ by acc.,} & \textit{if has test samples for } e_c \\ \mathbf{N}, & \textit{otherwise} \end{cases};$
  $\mathbf{w}_c \leftarrow \textit{averaged parameter value of all models in } \mathbf{N}'$;
  Output: $\mathbf{w}_c$;
```

5.5 Elixir Implementation

First, the implementation of CAEFL will be discussed and then the benefits of using Elixir will be highlighted in this section.

For the Elixir implementation, each type of sensor will have its dedicated module and implement callbacks defined in `Sensor` to report data.

An `Environment` module will be in charge of collecting all data from those sensors in `Environment.collect_data/1` and map data from each sensor into environment tags (either nominal or numerical) in `Environment.transform/2`.

Next, the agent will have an `Input` module that acts as a data source, e.g., a camera, for the `NeuralNetwork` module. This paper uses *numerical-elixir (nx)* [30] to achieve all neural network related functionalities. Each input sample observed by the `Input` module will be associated with an set of environment tags. Then the `sample` and the `tags` can be pattern matched when calling the `predict` function.

To train the model, we follow a similar process except that the tags will be provided by the training dataset. The `train` function will also perform pattern match for different environment tags so that correct tagged model updates can be calculated when submitting to the PS.

For the `ParameterServer` module there are functions to store the uploaded tagged model gradients from agents, generate global specialized models and new composite models with *elite selection*.

It is worth noting that Elixir supports *hot-code-swapping* [15]. This feature enables CAEFL to dynamically change the code logic on-the-fly. For example, adding finer tags for the temperature sensor, and adding/changing/removing environment tags to be pattern matched when predicting the input sample. This feature makes it more versatile not only for the agents but also for the PS. For both sides, there is an option to control which model to use, and which tagged model updates to send and merge without having any downtime.

6 Experiment Designs

6.1 Dataset and Pre-processings

The effectiveness of the CAEFL technique is evaluated using two standard datasets: MNIST[12] and FashionMNIST [41]. Each data set has 10 classes, and the sample size is 28x28x1.

The architecture of the neural network used is the AlexNet [21], which has three hidden layers with 300, 100 and 10 neurons respectively.

Six types of pre-processing techniques are applied to the input images to simulate different environments. Three of them are compositions of the basic ones. The three basic pre-processing techniques are distortion, blurring, and salt-and-pepper noise. They are used to simulate rainy, foggy and snowy environments respectively. For the blur effect, a Gaussian filter is applied. The radius of the Gaussian filter is a random integer in $[4, 8]$. For salt-and-pepper noise, each pixel has a 10% of probability being covered by noise. For the distortion effect, the camera matrix is

$$\begin{bmatrix} 12+r & 0 & 12+r \\ 0 & 12+r & 12+r \\ 0 & 0 & 1 \end{bmatrix}$$

where $r \sim U(0, 12)$, and the distortion coefficients are:

$$\begin{bmatrix} -0.340671 & 0.110427 & -0.000868 & 0.000190 & -0.016005 \end{bmatrix}$$

Some training samples from different environments are presented in Fig. 3.

6.2 Sample Allocation

As a baseline, we generate and train a generic FL model that we will refer to as type FL. Training data for FL are evenly sampled from eight different environments, i.e., 12.5% of training data from each environment regardless of the complexity of the environment. This is to simulate that in standard FL the differences in agents' environments are ignored because standard FL updates the global model by randomly selecting a group of agents to avoid biases towards some specific classes.

For specialized models, two types (A and B) of sample allocation are explored and evaluated to reflect different real-life situations. For type A, samples from the same basic environment will make up 64% of the total samples, and the rest 36% will be sampled randomly from complex environments. This type is to simulate when different environments are mainly geographically isolated. For example, autonomous vehicles driving in forests, cities or deserts. Although they are geographically isolated, there are many forests, cities and

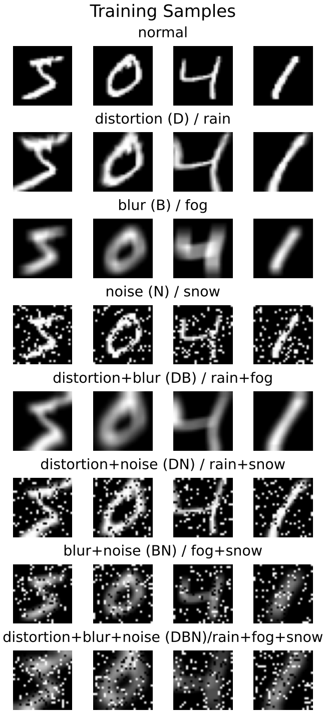
Training Samples
normal
distortion (D) / rain
blur (B) / fog
noise (N) / snow
distortion+blur (DB) / rain+fog
distortion+noise (DN) / rain+snow
blur+noise (BN) / fog+snow
distortion+blur+noise (DBN)/rain+fog+snow

Figure 3. Pre-processed training images of digits 5, 0, 4 and 1 in seven different environments. In the following figures, D, B, N will be used as abbreviations for distortion, blur effect, and salt-and-pepper noise respectively. For images with two or more pre-processing techniques, two or more corresponding letters will be used, e.g, DB will be used for images with distortion and blur effects.

deserts in the world. Thus, the resulting specialized models will mainly focus on improving driving in areas that display the same environmental conditions. Within this type, we consider the four sample allocations (A1-A4) (Table. 1).

For type B, the portion of samples from occasional and rare environments are increased. For the example use case of autonomous vehicles, an intuitive explanation for this type is that not all forests have the same amount of rainy or foggy days all over the year.

6.3 Evaluation

There are three main evaluation experiments for type A. The first evaluation for type A will show the specificity of the *specialized* models. The comparison is between the performance of the generated specialized models, local models (before updating local models from the PS), and the standard FL global models in different environments. The second evaluation will assess the impact on model accuracy when choosing different values for r during the *elite selection* process. The last one will show the performance of the *composite models* in complex environments. The experiment will test composite models for the complex environments DB, DN and BN using specialized models and tagged model updates.

There is only one evaluation for type B, that is to compare the specialized models generated by A2 and B. This evaluation is to assess how the specialized model performs when the agent observes more occasional events in an environment.

There will be 30 agents for types FL, B and each subtype in type A. For all agents, their local models will be trained for 100 epochs. Those experiments will be run 50 times with random initializations.

All experiments were performed on a PC with a Ryzen 9 3900XT (12 Core 24 Threads) CPU, 128 GB DDR4 RAM and an RTX 3090 GPU. All agents were spawned on the same processor.

7 Results

7.1 Accuracy of the Generated Specialized Models

Fig. 4 shows how each model performs in different environments when using the MNIST dataset. Models shown in Fig. 4 include (1) local models on agents (before pulling specialized models from the PS), (2) specialized models when the selected top $r = 25\%$ elite models are rebuilt from tagged model updates and (3) standard FL global models.

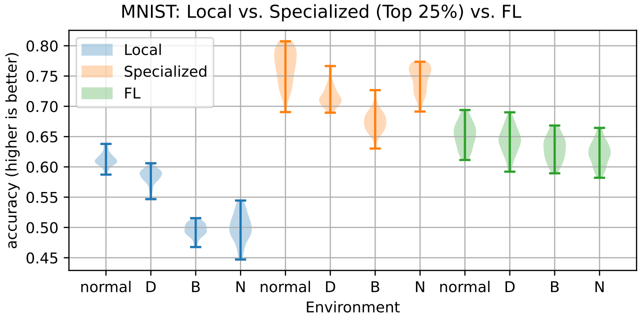

Figure 4. Model accuracy comparison (MNIST). Specialized models are generated with parameter $r = 25\%$.

For all environments, the highest accuracy of those specialized models is approximately 10% higher than the ones given by standard FL global models, and the accuracy distribution of specialized models also proves specialized models generally perform better than standard FL models. It is worth noting that for the normal, D, and N environments, the lowest accuracy of those specialized models is about the same as or even higher than the best accuracy of the corresponding standard FL global models.

When switching to the FashionMNIST datset, the specialized models perform similarly as the FL global models (Fig. 5). The reason for this is because samples in FashionMNIST are harder to identify in order to train a relatively good model to start with. The local models have significantly worse accuracy (20-30%) compared to the MNIST dataset (44-64%).

Table 1. Sample allocations for different subtypes. The total number of samples assigned to each agent is the same.

Frequency	Frequent				Occasional			Rare	
Subtype	normal	D	B	N	DB	DN	BN	DBN	Simulation Description
FL	12.5%	12.5%	12.5%	12.5%	12.5%	12.5%	12.5%	12.5%	Baseline FL.
A1	65%	7%	7%	7%	4%	4%	4%	2%	Sunny. Occasional rain / fog / snow.
A2	—	64%	—	—	16%	16%	—	4%	Forest. Occasional rain / fog / snow.
A3	—	—	64%	—	16%	—	16%	4%	City. Occasional rain / fog / snow.
A4	—	—	—	64%	—	16%	16%	4%	Desert. Occasional rain / fog / snow.
B	—	35%	—	—	27%	27%	—	11%	Forest. But more rain / fog / snow.

The specialized models tend to perform better than FL global models if the local models they based on have relatively good accuracy (with an average of 30% in this case). This is verified again when looking at the performance in the normal environment: the local models have 25-35% accuracy and their corresponding specialized model shows better accuracy than its FL counterpart. This observation indicates that the performance of CAEFL models could suffer from poorly performing local models.

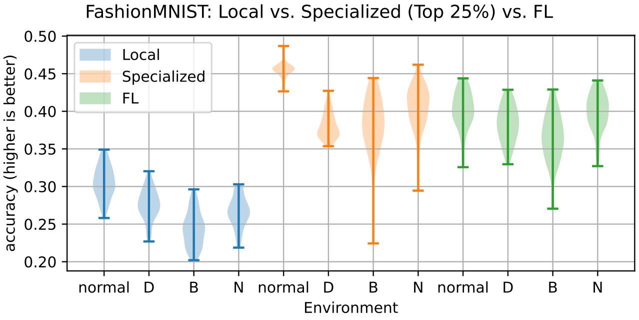

Figure 5. Model accuracy comparison (FshionMNIST). Specialized models are generated with parameter $r = 25\%$.

Fig. 6 shows the accuracy of specialized models when agents observe more occasional events in an environment. The distribution shows that the specialized model generated by type B tends to have better accuracy. This result proves that the specialized model can capture more ‘'knowledge" if more occasional events are observed.

7.2 Fine-tuned Elite Selection Hyper-Parameter r

To find out an optimal value for the hyper-parameter r during the elite selection process we follow the standard grid search approach [24]. This paper tested values of r in [10%, 15%, 20%, 25%, 30%, 35%]. Fig. 7 presents how the model accuracy changes when different r values are used in the elite selection process for each environment.

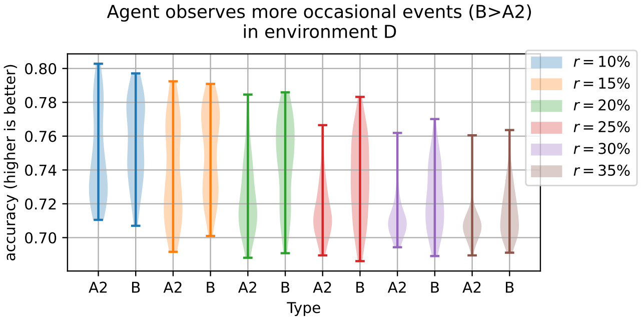

Figure 6. Type B agents can observe more occasional events than type A2 (27% vs 16%).

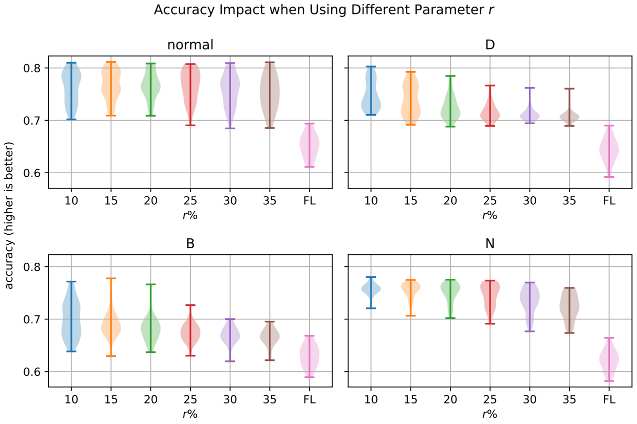

Figure 7. Changes in accuracy when selecting different values for parameter r in each environment. A smaller $r (\leq 20\%)$ tends to have higher accuracy while the highest $r (\geq 30\%)$ demonstrates the lowest accuracy without necessarily improving the standard deviation of the distribution bell for the violin plot.

In general, Fig. 7 shows that using any r between $10 - 20\%$ could be a good starting point for fine-tuning this parameter in different environments because using r in this range can generate specialized models that achieve higher accuracy. Evidently, r does not have a significant impact in terms of the highest accuracy and the general distribution in the normal environment while it can lower the highest accuracy in D and B, with the worst impact on the lowest accuracy for environment N.

7.3 Accuracy of Composite Models

To validate the proposed composite function $c(\cdot)$ in complex environments, we evaluate the two specialized models in the complex environment before the composition and then evaluate the performance of the composite model. Moreover, to examine the impact of the hyper-parameter r on the composite function, a set of r values is also considered. Furthermore, the performance of the standard FL global model is also shown as a baseline.

Fig. 8 shows the accuracy before and after the model composition in complex environment BN. In Fig. 8, the accuracy of the specialized model B gradually decreases as the value of r increases. When r is set to 35%, the performance of the specialized model B is quite close to the FL baseline model. Meanwhile, the specialized model N shows relatively good accuracy around 71%, and its standard deviation is smaller than specialized model B. Although its lowest accuracy also decreases when r increases.

When r is set to values between 10% to 20%, the composite model of B and N can achieve higher accuracy while model's B accuracy distribution is similar to the specialized model N. When r is set to 25%, the accuracy distribution begins to spread out more evenly, and when choosing a larger value for r, despite the highest accuracy (the composite is still higher than either specialized model) the distribution is more concentrated at the lower end. This indicates a small value of r should be used, and in this case, choosing r between $10 - 20\%$ seems to be optimal.

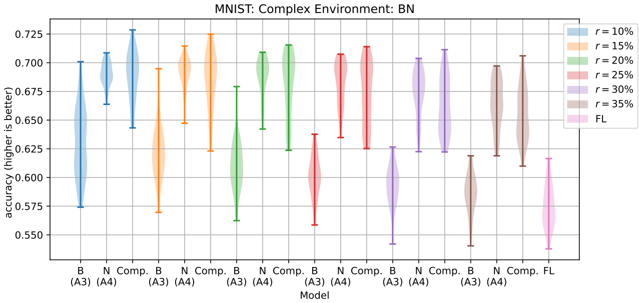

Figure 8. Model accuracy in complex environment BN.

The results for complex environment DB are plotted in Fig. 9. The results show that while the composite model DB can achieve similar or higher accuracy than model D or B, the relative range of the accuracy distribution is generally wider than what we see in complex environment BN. However, the 25 to 75 percentile of the accuracy of the composite model follows closely the specialized model D that has better accuracy than specialized model B. This shows that while the composite model may be affected by the worse specialized model resulting in a wider range of accuracy, it can maintain similar accuracy as the better performing specialized model in the complex environment.

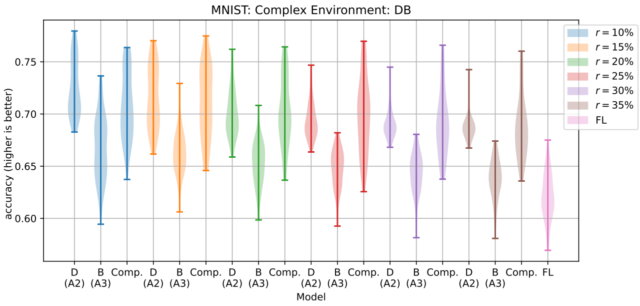

Figure 9. Model accuracy in complex environment DB.

Fig. 10 presents the results for complex environment DN. Though both specialized models can already perform relatively well in the complex environment, the composite model pushes the highest accuracy and distribution higher.

In summary, for all three complex environments, the composite model can generally achieve higher accuracy than the specialized models used in the composition. Additionally, in every case the composite model performs better than the baseline FL global model. Also, in the case that one of the specialized models performs much worse than the other in the complex environment, the composite model still achieves similar accuracy or outperforms the better one.

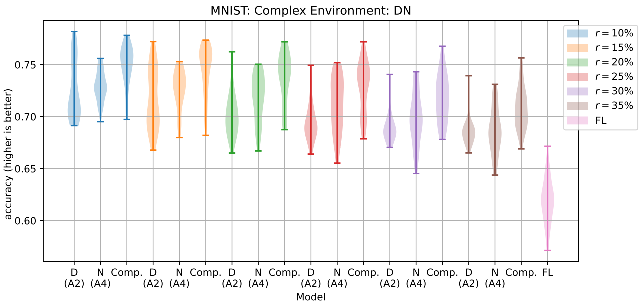

Figure 10. Model accuracy in complex environment DN.

8 Limitations

A limitation of the proposed approach is that the model size is fixed. Even if we can design some mechanism so that the agents can update to a newer model with a different architecture, the previous calculated and stored model updates will all be invalidated. As a result, CAEFL is only applicable as long as the neural network architecture remains static for all agents contributing to the federated learning model. However, this is a limitation that is true for all federated learning approaches proposed so far in the literature.

Another limitation is that sending tagged model updates adds the communication overhead. The communication overheads can be reduced by increasing the gap time before submitting to the PS.

9 Conclusion & Future Work

In summary CAEFL sends tagged model updates to the PS where the conventional approach sends overall gradients. Sending tagged model updates enables CAEFL to be versatile and to adapt to deal with complex environments.

The *elite selection* improves the accuracy of generated models and avoids the impact of model updates with poor accuracy. In experiments, CAEFL models are observed to be more accurate than a single global model by 7-10% for the MNIST dataset and 2% for the FashionMNIST dataset.

There are several avenues for future work. Different environments, $\{e_k\}$, can be identified with the help of an expert or obtained from other machine learning approaches such as a decision tree where we treat each leaf as an environment. The availability of environment tags may depend on what information that the agent can, and is allowed to, observe. Some tags can be automatically generated from the sensors on the agent, e.g., time, temperature and humidity because the model and the output of those sensors are known at the time of building the agent. It is relatively easy to define tags derived from such known data. Semantic tags, such as *near a river*, are more challenging. For geolocated semantic tags information from a GPS locator could be used. If the GPS component is not allowed or unavailable, it is possible to fallback to use cameras to infer some environment tags. Furthermore, if a complex environment arises frequently the PS can cache the combined model, and prompt the user to promote it to a dedicated environment.

Acknowledgments

This work is partially funded by the College (ECR) Scholarship 2021, College of Science & Engineering, School of Computing Science, University of Glasgow.

References

[1] Alekh Agarwal, Martin J. Wainwright, and John C. Duchi. 2010. Distributed Dual Averaging In Networks. *Advances in Neural Information Processing Systems* 23 (2010).

[2] S Agatonovic-Kustrin and R Beresford. 2000. Basic concepts of artificial neural network (ANN) modeling and its application in pharmaceutical research. *Journal of Pharmaceutical and Biomedical Analysis* 22, 5 (2000), 717–727. https://doi.org/10.1016/S0731-7085(99)00272-1

[3] Dan Alistarh, Demjan Grubic ETH Zurich, Jerry Z Li, Ryota Tomioka, and Milan Vojnovic. 2017. QSGD: Communication-Efficient SGD via Gradient Quantization and Encoding. *Advances in Neural Information Processing Systems* 30 (2017).

[4] Arda Aytekin, Hamid Reza Feyzmahdavian, and Mikael Johansson. 2016. Analysis and Implementation of an Asynchronous Optimization Algorithm for the Parameter Server. *CoRR* abs/1610.05507 (2016). arXiv:1610.05507 http://arxiv.org/abs/1610.05507

[5] Dimitri P. Bertsekas and John N. Tsitsiklis. 2003. Parallel and Distributed Computation:Numerical Methods. (2003). https://dspace.mit.edu/handle/1721.1/3719

[6] Jianmin Chen, Rajat Monga, Samy Bengio, and Rafal Józefowicz. 2016. Revisiting Distributed Synchronous SGD. *CoRR* abs/1604.00981 (2016). arXiv:1604.00981 http://arxiv.org/abs/1604.00981

[7] Jianmin Chen, Xinghao Pan, Rajat Monga, Samy Bengio, and Rafal Jozefowicz. 2016. Revisiting Distributed Synchronous SGD. (4 2016). https://arxiv.org/abs/1604.00981v3

[8] Trishul M. Chilimbi, Yutaka Suzue, Johnson Apacible, and Karthik Kalyanaraman. 2014. Project Adam: Building an Efficient and Scalable Deep Learning Training System. In *11th USENIX Symposium on Operating Systems Design and Implementation, OSDI '14, Broomfield, CO, USA, October 6-8, 2014*, Jason Flinn and Hank Levy (Eds.). USENIX Association, 571–582. https://www.usenix.org/conference/osdi14/technical-sessions/presentation/chilimbi

[9] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Quoc V. Le, Mark Z. Mao, Marc'Aurelio Ranzato, Andrew W. Senior, Paul A. Tucker, Ke Yang, and Andrew Y. Ng. 2012. Large Scale Distributed Deep Networks. In *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States*, Peter L. Bartlett, Fernando C. N. Pereira, Christopher J. C. Burges, Léon Bottou, and Kilian Q. Weinberger (Eds.). 1232–1240. https://proceedings.neurips.cc/paper/2012/hash/6aca97005c68f1206823815f66102863-Abstract.html

[10] Ofer Dekel, Ran Gilad-Bachrach, Ohad Shamir, and Lin Xiao. 2010. Optimal Distributed Online Prediction using Mini-Batches. *CoRR* abs/1012.1367 (2010). arXiv:1012.1367 http://arxiv.org/abs/1012.1367

[11] Ofer Dekel, Ran Gilad-Bachrach, Ohad Shamir, and Lin Xiao. 2012. Optimal Distributed Online Prediction Using Mini-Batches Ran Gilad-Bachrach Ohad Shamir. *Journal of Machine Learning Research* 13 (2012), 165–202.

[12] Li Deng. 2012. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine* 29, 6 (2012), 141–142.

[13] Stephan Dreiseitl and Lucila Ohno-Machado. 2002. Logistic regression and artificial neural network classification models: a methodology review. *Journal of Biomedical Informatics* 35, 5 (2002), 352–359. https://doi.org/10.1016/S1532-0464(03)00034-0

[14] John Duchi JDUCHI and Yoram Singer. 2011. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization * Elad Hazan. *Journal of Machine Learning Research* 12 (2011), 2121–2159.

[15] Geovane Fedrecheski, Laisa C. P. Costa, and Marcelo K. Zuffo. 2016. Elixir programming language evaluation for IoT. In *2016 IEEE International Symposium on Consumer Electronics (ISCE)*. 105–106. https://doi.org/10.1109/ISCE.2016.7797392

[16] W.A. Gardner. 1984. Learning characteristics of stochastic-gradient-descent algorithms: A general study, analysis, and critique. *Signal Processing* 6, 2 (1984), 113–133. https://doi.org/10.1016/0165-1684(84)90013-6

[17] Chenghao Hu, Jingyan Jiang, and Zhi Wang. 2019. Decentralized Federated Learning: A Segmented Gossip Approach. *CoRR* abs/1908.07782 (2019). arXiv:1908.07782 http://arxiv.org/abs/1908.07782

[18] Wenbin Jiang, Geyan Ye, Laurence T. Yang, Jian Zhu, Yang Ma, Xia Xie, and Hai Jin. 2019. A Novel Stochastic Gradient Descent Algorithm Based on Grouping over Heterogeneous Cluster Systems for Distributed Deep Learning. In *19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGRID 2019, Larnaca, Cyprus, May 14-17, 2019*. IEEE, 391–398. https://doi.org/10.1109/CCGRID.2019.00053

[19] Abdel Aziz Khater, Ahmad M. El-Nagar, Mohammad El-Bardini, and Nabila M. El-Rabaie. 2020. Online learning based on adaptive learning rate for a class of recurrent fuzzy neural network. *Neural Comput. Appl.*

32, 12 (2020), 8691–8710. https://doi.org/10.1007/s00521-019-04372-w

[20] Jakub Konečný, H. Brendan McMahan, Felix X. Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. 2016. Federated Learning: Strategies for Improving Communication Efficiency. *CoRR* abs/1610.05492 (2016). arXiv:1610.05492 http://arxiv.org/abs/1610.05492

[21] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems*, F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger (Eds.), Vol. 25. Curran Associates, Inc. https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf

[22] John Langford, Alexander J. Smola, and Martin Zinkevich. 2009. Slow Learners are Fast. *Advances in Neural Information Processing Systems 22 - Proceedings of the 2009 Conference* 1 (11 2009), 2331–2339. https://arxiv.org/abs/0911.0491v1

[23] Tian Li, Anit Kumar Sahu, Ameet Talwalkar, and Virginia Smith. 2020. Federated Learning: Challenges, Methods, and Future Directions. *IEEE Signal Process. Mag.* 37, 3 (2020), 50–60. https://doi.org/10.1109/MSP.2020.2975749

[24] Petro Liashchynskyi and Pavlo Liashchynskyi. 2019. Grid Search, Random Search, Genetic Algorithm: A Big Comparison for NAS. *CoRR* abs/1912.06059 (2019). arXiv:1912.06059 http://arxiv.org/abs/1912.06059

[25] Yujun Lin, Song Han, Huizi Mao, Yu Wang, and William J. Dally. 2017. Deep Gradient Compression: Reducing the Communication Bandwidth for Distributed Training. (12 2017). https://arxiv.org/abs/1712.01887v3

[26] Tian Liu, Jiahao Ding, Ting Wang, Miao Pan, and Mingsong Chen. 2022. Towards Fast and Accurate Federated Learning with non-IID Data for Cloud-Based IoT Applications. *CoRR* abs/2201.12515 (2022). arXiv:2201.12515 https://arxiv.org/abs/2201.12515

[27] Othmane Marfoq, Giovanni Neglia, Aurélien Bellet, Laetitia Kameni, and Richard Vidal. 2021. Federated Multi-Task Learning under a Mixture of Distributions. In *Advances in Neural Information Processing Systems*, M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan (Eds.), Vol. 34. Curran Associates, Inc., 15434–15447. https://proceedings.neurips.cc/paper/2021/file/82599a4ec94aca066873c99b4c741ed8-Paper.pdf

[28] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. 2017. Communication-Efficient Learning of Deep Networks from Decentralized Data. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics (Proceedings of Machine Learning Research, Vol. 54)*, Aarti Singh and Jerry Zhu (Eds.). PMLR, 1273–1282. https://proceedings.mlr.press/v54/mcmahan17a.html

[29] M. Moreira and E. Fiesler. 1995. Neural Networks with Adaptive Learning Rate and Momentum Terms. *IDIAP Technical Report* 95, 04 (1995). https://infoscience.epfl.ch/record/82307/files/95-04.pdf

[30] Sean Moriarity, José Valim, and Paulo O. Lenzi Valente. 2022. *Nx - Numerical Elixir*. https://github.com/elixir-nx/nx

[31] Dinh C. Nguyen, Ming Ding, Pubudu N. Pathirana, Aruna Seneviratne, Jun Li, and H. Vincent Poor. 2021. Federated Learning for Internet of Things: A Comprehensive Survey. *IEEE Communications Surveys and Tutorials* 23, 3 (4 2021), 1622–1658. https://doi.org/10.1109/comst.2021.3075439

[32] Solmaz Niknam, Harpreet S. Dhillon, and Jeffrey H. Reed. 2020. Federated Learning for Wireless Communications: Motivation, Opportunities, and Challenges. *IEEE Commun. Mag.* 58, 6 (2020), 46–51. https://doi.org/10.1109/MCOM.001.1900461

[33] Feng Niu, Benjamin Recht, Christopher Ré, and Stephen J Wright. 2011. Hogwild!: A Lock-Free Approach to Parallelizing Stochastic Gradient Descent. *Advances in Neural Information Processing Systems* 24 (2011).

[34] Shiva Raj Pokhrel and Jinho Choi. 2020. A Decentralized Federated Learning Approach for Connected Autonomous Vehicles. In *2020 IEEE Wireless Communications and Networking Conference Workshops, WCNC Workshops 2020, Seoul, Korea (South), April 6-9, 2020*. IEEE, 1–6. https://doi.org/10.1109/WCNCW48565.2020.9124733

[35] Amirhossein Reisizadeh, Aryan Mokhtari, Hamed Hassani, Ali Jadbabaie, and Ramtin Pedarsani. 2020. FedPAQ: A Communication-Efficient Federated Learning Method with Periodic Averaging and Quantization. In *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics (Proceedings of Machine Learning Research, Vol. 108)*, Silvia Chiappa and Roberto Calandra (Eds.). PMLR, 2021–2031. https://proceedings.mlr.press/v108/reisizadeh20a.html

[36] Frank Seide, Hao Fu, Jasha Droppo, Gang Li, and Dong Yu. 2014. 1-Bit Stochastic Gradient Descent and Application to Data-Parallel Distributed Training of Speech DNNs. https://www.microsoft.com/en-us/research/publication/1-bit-stochastic-gradient-descent-and-application-to-data-parallel-distributed-training-of-speech-dnns/

[37] Neta Shoham, Tomer Avidor, Aviv Keren, Nadav Israel, Daniel Benditkis, Liron Mor-Yosef, and Itai Zeitak. 2019. Overcoming Forgetting in Federated Learning on Non-IID Data. *CoRR* abs/1910.07796 (2019). arXiv:1910.07796 http://arxiv.org/abs/1910.07796

[38] Virginia Smith, Chao-Kai Chiang, Maziar Sanjabi, and Ameet S Talwalkar. 2017. Federated Multi-Task Learning. In *Advances in Neural Information Processing Systems*, I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.), Vol. 30. Curran Associates, Inc. https://proceedings.neurips.cc/paper/2017/file/6211080fa89981f66b1a0c9d55c61d0f-Paper.pdf

[39] Sebastian U. Stich. 2018. Local SGD Converges Fast and Communicates Little. *7th International Conference on Learning Representations, ICLR 2019* (5 2018). https://arxiv.org/abs/1805.09767v3

[40] Hao Wu and David Levinson. 2021. The ensemble approach to forecasting: A review and synthesis. *Transportation Research Part C: Emerging Technologies* 132 (2021), 103357. https://doi.org/10.1016/j.trc.2021.103357

[41] Han Xiao, Kashif Rasul, and Roland Vollgraf. 2017. Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms. *CoRR* abs/1708.07747 (2017). arXiv:1708.07747 http://arxiv.org/abs/1708.07747

[42] Chien Cheng Yu and Bin Da Liu. 2002. A backpropagation algorithm with adaptive learning rate and momentum coefficient. *Proceedings of the International Joint Conference on Neural Networks* 2 (2002), 1218–1223. https://doi.org/10.1109/IJCNN.2002.1007668

[43] Valentina Zantedeschi, Aurélien Bellet, and Marc Tommasi. 2020. Fully Decentralized Joint Learning of Personalized Models and Collaboration Graphs. In *The 23rd International Conference on Artificial Intelligence and Statistics, AISTATS 2020, 26-28 August 2020, Online [Palermo, Sicily, Italy] (Proceedings of Machine Learning Research, Vol. 108)*, Silvia Chiappa and Roberto Calandra (Eds.). PMLR, 864–874. http://proceedings.mlr.press/v108/zantedeschi20a.html

[44] Matthew D. Zeiler. 2012. ADADELTA: An Adaptive Learning Rate Method. *CoRR* abs/1212.5701 (2012). arXiv:1212.5701 http://arxiv.org/abs/1212.5701

[45] Hantian Zhang, Jerry Li, Kaan Kara, Dan Alistarh, Ji Liu, and Ce Zhang. 2017. ZipML: Training Linear Models with End-to-End Low Precision, and a Little Bit of Deep Learning. , 4035–4043 pages. https://proceedings.mlr.press/v70/zhang17e.html

[46] Wei Zhang, Suyog Gupta, Xiangru Lian, and Ji Liu. 2016. Staleness-Aware Async-SGD for Distributed Deep Learning. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016*, Subbarao Kambhampati (Ed.). IJCAI/AAAI Press, 2350–2356. http://www.ijcai.org/Abstract/16/335