Data-driven Synchronization Protocols for Data-parallel Neural Learning over Streaming Data

George Klioumis
School of ECE
Technical University of Crete
Chania, Greece
gklioumis@tuc.gr

Nikos Giatrakos
School of ECE
Technical University of Crete
Chania, Greece
ngiatrakos@tuc.gr

***Abstract*—We introduce EVENFLOW, a novel toolkit of synchronization protocols for data-parallel training of neural nets using the Parameter Server (PS) paradigm. EVENFLOW achieves both timely and accurate global model updates in streaming settings. Instead of leaving stragglers out of the global model to avoid delays (asynchronous protocol) or using laggy synchronizations of all learners (synchronous protocol), EVENFLOW establishes data-driven mechanisms that allow the PS paradigm to decide when a synchronization is necessary, i.e., the global model may have changed beyond an allowed tolerance value. EVENFLOW models this problem as a distributed, thresholded function monitoring task and decomposes it to local filters monitored independently by each learner. When a learner finds its local filter violated, only then a synchronization is triggered. Our experiments show that EVENFLOW combines the virtues of both the vanilla (synchronous, asynchronous) protocols. EVENFLOW offers the rapid training times of asynchronous, with mostly equal or even improved accuracy compared to synchronous.**

***Index Terms*—data-parallel learning, data streams, neural nets**

I. Background and Motivation

In neural learning over streaming data, applications receive unbounded, rapid streams of data in two pipelines, which operate on par with one another, as shown in Figure 1(a) [1]–[4]. The training pipeline (blue colored path in Figure 1(a)) continuously ingests training tuples from a stream source, such as an Apache Kafka `Training Topic`. These flow through a `Learner` into batches devised by a streaming window, for a number of epochs [5]. The model's weight are updated and the process continues as new tuples stream in.

In the prediction pipeline (red colored path in Figure 1(a)), there is a `Predictor`. The `Predictor` reads streaming tuples from a `Prediction Topic` and deploys an identical, to the training pipeline, neural network to use it for predictions. For instance, in case of a classification task, the `Predictor` uses the continuously updated neural network and deploys it to assign labels to unlabeled streaming tuples of the `Prediction Topic` of Figure 1(a).

As time passes and the neural network weights in the training pipeline get continuously updated, the `Predictor` must, also continuously, receive the updated weights of the neural network. To achieve that, as shown in Figure 1(a), there is a `Weights Topic`, via which new neural network weights are continuously transferred to the `Predictor`.

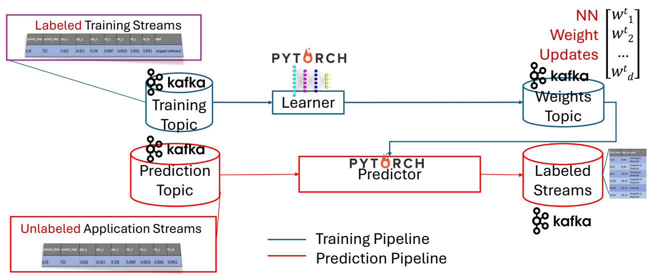
Let us exemplify this setup in some real application scenarios. Consider a robot navigation scenario in a smart factory terrain [1]. The goal is to classify segments of the robots movement operating on the field. Based on this classification, the application can then predict delays in the production lines. For instance, a robot which continuously stops due to obstacles, rotates frequently or collides with other robots, will probably delay the delivery of production particles between production lines. The terrain over which robots navigate is not static. It changes continuously as new obstacles (e.g. pallets) are added, people and pieces of equipment move and so on. Thus, at any given time, ROS simulations [6], [7] must produce classified segments of movement for simulated robots on the newly formed terrain, which are ingested in the `Training Topic` of Figure 1(a). The `Learner` trains a neural net which receives simulated robot streams, including features (position, velocity, direction, etc) and class of movement segment. As the neural net is trained on the simulated robots' streams, the most up-to-date model is passed to the `Predictor`, via the `Weights Topic`, to use it on streams of real robots operating on the field and classify segments of their movement.

Similarly, in physics, collisions at high-energy particle colliders are a traditionally fruitful source of exotic particle discoveries [8]. By looking for discrepancies between the simulations (training pipeline) and the actual collisions in the real experiment (prediction pipeline), in an online, real time fashion, scientists search for signs of disagreement which could lead to the discovery of new phenomena [9]. Based on that, they may choose to prolong or stop a running experiment.

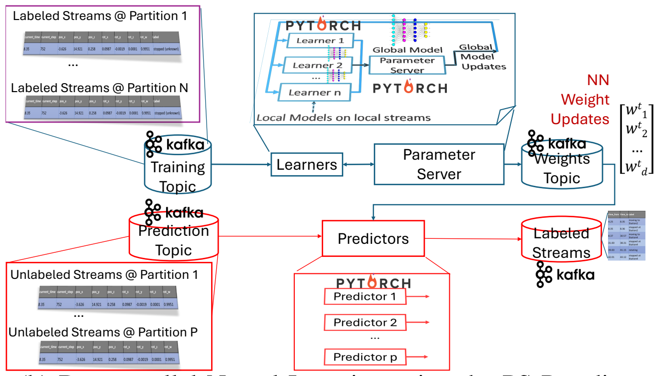
The training pipeline does not always involve streams from simulations. For instance, in Maritime Situation Awareness applications [4], [10], preceding vessel trajectory segments are classified based on the actual movement and are then used to predict future trajectories of vessels following up [4].

It is evident that operating over streaming data means that in the training pipeline, the neural network should be trained continuously in an online, real time fashion, while the prediction pipeline should continuously and immediately receive an up-to-date version of the neural model being trained, upon

This work was supported by the EU project EVENFLOW under Horizon Europe agreement No. 101070430.



(a) Non-parallel Neural Learning



(b) Data-parallel Neural Learning using the PS Paradigm

Fig. 1: Non-parallel and Parallel Training and Prediction Pipelines in Neural Learning over Streaming Data

weight updates. In that, the `Predictor` will continuously base its decisions on new, instead of obsolete, versions of the trained model. Given these, latency restrictions do apply and a major challenge comes from the volume and velocity of the input streams at both pipelines.

To handle these latency requirements, the first step is to come up with data-parallel versions of both pipelines. In Figure 1(b), we illustrate a data-parallel version of the architectural scheme of Figure 1(a). In the training pipeline, we have multiple learners operating in parallel on disjoint partitions of the training streams, while each of the parallel predictors receives the up-to-date neural model and performs predictions on disjoint partitions of streams from the `Prediction Topic`. The framework in Figure 1 has been used in [1]–[4]. But parallelization alone can only solve part of the problem [11], [12]. There is a major missing part that may render the entire data-parallel architectural scheme in Figure 1(b) useless. While the `Predictors` can operate independently on their partitions, the `Learners` must continuously synchronize their local models to a global one.

To accomplish this task, the prominent Parameter Server (PS) paradigm has been proposed in the literature [13]. The PS paradigm comes with two vanilla synchronization protocols, namely synchronous and asynchronous synchronization. In the synchronous protocol, each of the `Learners` trains on local data and periodically communicates a version of its locally trained model to the PS side. The PS, awaits to receive local models from all `Learners`, computes the global model by synthesizing local ones and then sends the current global model weights back to the `Learners`. In Figure 1(b), it also writes the new weights to the respective Kafka topic for the parallel predictors to update their own model copies. This process is periodically repeated in rounds. This protocol enhances the accuracy of the global model at any given time, but is prone to delays in training time, because the PS has to wait to listen from all learners, therefore the protocol is susceptible to stragglers. In our context, this means that it may sacrifice the real time, online character of the scheme. The asynchronous protocol allows every `Learner` independently communicate the local model and the PS sends a new global model, with each such update, back to the `Learners`, without waiting for all learners. Due to this, asynchronous synchronization reduces training time, but any number of `Learners` may be missing from the global model computation. Therefore, the new weights incorporate stale gradients, offering imprecise approximations of the true global model at any given time.

While a number of variations have been proposed to remedy the weaknesses of the vanilla protocols [12], [14]–[16], they have not been designed for streaming setups. For instance, in streaming setups, using a fully decentralized protocol [16] would mean that the whole set of learners should be continuously queried to deduce which one holds the global model at any given time, for the `Weights Topic` of Figure 1(b) to get updated with the new weights. Hence, the advantages of decentralized learning would be diminished by this barrier.

In this work, we introduce EVENFLOW, a novel toolkit of synchronization protocols for data-parallel training of neural nets under the PS paradigm. EVENFLOW achieves both timely and accurate global model updates in streaming settings. Instead of leaving stragglers out of the global model to avoid delays (asynchronous protocol) or using laggy synchronizations of all learners (synchronous protocol), EVENFLOW establishes data-driven mechanisms that allow the PS paradigm to decide when a synchronization is necessary, i.e., the global model may have changed beyond an allowed tolerance value. EVENFLOW models this problem as a distributed, thresholded function monitoring task and decomposes it to local filters monitored independently by each learner. When a learner finds its local filter violated, only then a synchronization is triggered. EVENFLOW offers the rapid training times of asynchronous, with mostly equal or even improved accuracy compared to synchronous.

II. Parameter Server and the Vanilla Protocols

A. *Preliminaries*

We first explicitly define some basic terms used throughout this paper adapted, when needed, in the streaming setup.

Learner, worker or client: In this work we use the term learner equivalently to the terms worker or client that are often used in the literature [5], [13]–[16].

Neural network weights or model parameters: In this work we use the terms neural network weights and model parameters, interchangeably. We further use explicit vector notation $\vec{\mathbf{w}}$ to emphasize that the lot of neural net parameters is a vector in the $\mathbb{R}^d$ space.

How deep is the neural network?: In this work, we do not assume that the neural network is trained offline and then used only for online inference. Instead, training and prediction occur simultaneously. The network's depth and complexity directly affect training time—larger networks with more layers increase latency, which is an issue in real-time streaming setups. To address this, one can apply transfer learning by adding new layers to a pre-trained, large model and only train these additional layers over the incoming streams, while keeping the original pre-trained network frozen. This allows the model to adapt to new data streams efficiently without retraining the entire network. Thus, the real-timeliness of the application is not compromised. This observations holds not only for the EVENFLOW, but for all protocols in this work.

Problem Definition: The objective of the training process over a neural network, even in a non-data parallel, setup, is to minimize a global loss function $\mathcal{L}(\vec{\mathbf{w}}^{(t)})$ over the stream:

$$\min_{\vec{\mathbf{w}}^{(t)}} \mathcal{L}(\vec{\mathbf{w}}^{(t)})$$

where $\mathcal{L}(\vec{\mathbf{w}}^{(t)})$ is the global loss function defined as:

$$\mathcal{L}(\vec{\mathbf{w}}^{(t)}) = \frac{1}{|S|}\sum_{j=1}^{|S|} \ell_j(\vec{\mathbf{w}}^{(t)})$$

- $\vec{\mathbf{w}}^{(t)}$ is the current (at time t) vector of the weights of the (global) neural model.
- $|S|$ is the number of the streaming tuples.
- $\ell_j(\vec{\mathbf{w}}^{(t)})$ is the loss associated with the j-th data sample.

Gradient Computation: The gradient of the global loss function with respect to the model parameters $\vec{\mathbf{w}}^{(t)}$ is:

$$\vec{\mathbf{g}}^{(t)} = \nabla_{\vec{\mathbf{w}}^{(t)}} \mathcal{L}(\vec{\mathbf{w}}^{(t)})$$

Update Rule: The model weights are updated using the computed gradient, for example, using Stochastic Gradient Descent (SGD) [17], where η is the learning rate:

$$\vec{\mathbf{w}}^{(t+1)} \leftarrow \vec{\mathbf{w}}^{(t)} - \eta \vec{\mathbf{g}}^{(t)}$$

Iteration: Repeat the gradient computation and update steps for an application defined number of times.

In the Parameter Server (PS) paradigm, the synchronization protocol determines how the local updates from the learners are aggregated to update the global model weights. When the data distribution is non-i.i.d., meaning that different learners may observe different distributions of streams, the synchronization protocols must account for this heterogeneity.

The notion of `epoch` in a streaming setting: In the traditional PS paradigm [13] which focuses on batch processing, local training at each learner involves passing each local batch through the local copy of the neural network. After processing each batch, the local model updates the global model based on the synchronization protocol (synchronous or asynchronous). Once the entire dataset has been processed by all learners, a new iteration, i.e., epoch begins, where training continues starting from the updated global model. In streaming settings, the training dataset is virtually unbounded, making traditional batch processing techniques unsuitable in their original form. Therefore, in all the protocols described in this work, when we refer to local training, we mean a process similar to the one outlined in [5]. Parallel learners perform local training by receiving a batch size, a streaming window size (composed of a number of batches), and a number of epochs at the beginning of the training process. They use this window to perform local training for the specified number of epochs. After completing this, a new window is formed, and local training continues, with or without synchronization, depending on the protocol.

The notion of a `round`: A single round in PS training consists of (a) local training and local model computation by the learners, (b) communication of gradients to the PS, (c) aggregation and updating of the global model by the server, and (d) communication of updated parameters back to the learners. The number of rounds in training can impact the training speed and the accuracy of the model. In the vanilla protocols described next, synchronous rounds might lead to more accurate training, while asynchronous rounds could allow for faster, though potentially less accurate, updates.

B. Parameter Server Paradigm - Synchronous Protocol

In the synchronous protocol, all learners update their local weights and then synchronize their updates with the PS at the same time. The process is sketched as follows:

Local Update: Each learner $i \in \{1, \cdots, n\}$ processes its local tuples at time t and computes local model parameters $\vec{\mathbf{w}}_i^{(t+1)}$:

$$\vec{\mathbf{w}}_i^{(t+1)} = \vec{\mathbf{w}}^{(t)} - \eta \nabla_{\vec{\mathbf{w}}^{(t)}} \mathcal{L}_i(\vec{\mathbf{w}}^{(t)}) = \vec{\mathbf{w}}^{(t)} - \eta \vec{\mathbf{g}}_i^{(t)} \qquad (1)$$

Here, $\mathcal{L}_i(\vec{\mathbf{w}}^{(t)})$ represents the local loss function and $\vec{\mathbf{g}}_i^{(t)}$ is the gradient computed at learner i, which may differ significantly across learners due to non-i.i.d. data distributions.

Synchronization: After all learners have completed their local updates, they send their local models' weights $\vec{\mathbf{w}}_i^{(t+1)} \in \mathbb{R}^d$ to the parameter server.

Global Update: The parameter server aggregates the local models to form the new global model. Given the not necessarily i.i.d. nature of the updates, the aggregation may involve weighted averaging due to differences in stream distributions:

$$\vec{\mathbf{w}}^{(t+1)} = \frac{\sum_{i=1}^{n} \gamma_i \vec{\mathbf{w}}_i^{(t+1)}}{\sum_{i=1}^{n} \gamma_i} \qquad (2)$$

where $\gamma_i > 0$ represents the importance assigned to the update from learner i, potentially based on factors like the size or importance of the local stream partitions.

Broadcast: The updated global model $\vec{\mathbf{w}}^{(t+1)}$ is then sent back to all learners, replacing the local weights/parameters $\vec{\mathbf{w}}_i^{(t+1)} \leftarrow \vec{\mathbf{w}}^{(t+1)}$.

C. Asynchronous Protocol

In the asynchronous protocol, learners do not wait for each other to complete their updates. Instead, they send their local updates to the parameter server as soon as they are computed, and the parameter server updates the global model immediately upon receiving an update from any learner. The process can be described as follows:

Local Update: Each learner i computes its local model parameters $\vec{\mathbf{w}}_i^{(t+1)}$ after processing its local data at time t. The local update computation is identical to Equation 1, though we repeat it here for clarity:

$$\vec{\mathbf{w}}_i^{(t+1)} = \vec{\mathbf{w}}^{(t)} - \eta \nabla_{\vec{\mathbf{w}}} \mathcal{L}_i(\vec{\mathbf{w}}^{(t)}) = \vec{\mathbf{w}}^{(t)} - \eta \vec{\mathbf{g}}_i^{(t)}$$

The local gradient $\vec{\mathbf{g}}_i^{(t)}$ again reflects the potentially diverse data distributions across learners.

Synchronization: As soon as a learner i computes its local model $\vec{\mathbf{w}}_i^{(t+1)}$, it sends the update to the parameter server.

Global Update: Upon receiving an update from any learner i, the PS immediately updates the global model. Due to the non-i.i.d. nature of updates, the global update function must account for the differing contributions of each learner's update:

$$\vec{\mathbf{w}}^{(t+1)} = \vec{\mathbf{w}}^{(t)} + \frac{\gamma_i \vec{\mathbf{w}}_i^{(t+1)} - \gamma_i \vec{\mathbf{w}}_i}{\sum_{i=1}^n \gamma_i}$$

$\vec{\mathbf{w}}_i$ is the local vector, learner i communicated the last time it contacted the PS. By subtracting $\vec{\mathbf{w}}_i$, we compensate for stale updates, removing the previous contribution of learner i.

Broadcast: The updated global model $\vec{\mathbf{w}}^{(t+1)}$ is sent back to learner i, leading to potentially different versions of the global model being used by different learners at any time.

III. EVENFLOW PROTOCOLS

The EVENFLOW synchronization protocol toolkit provides data-driven mechanisms which allow each learner to independently check, and inform the parameter server, if a full sync should take place. In order to do that, in all our protocols, the application must provide one, any, function $f(\vec{\mathbf{w}}^{(t+\varkappa+1)}, \vec{\mathbf{w}}^{(t)}) : \mathbb{R}^d \rightarrow \mathbb{R}$ along with a threshold T. The function together with the threshold express how much the global model weights can deviate before having the PS call for a synchronization. When $f(\vec{\mathbf{w}}^{(t+\varkappa+1)}, \vec{\mathbf{w}}^{(t)}) > T$ a synchronization must take place. Otherwise, if $f(\vec{\mathbf{w}}^{(t+\varkappa+1)}, \vec{\mathbf{w}}^{(t)}) \leq T$, learners can keep performing the training process only locally, i.e., without a synchronization. In that, EVENFLOW combines explicit accuracy guarantees based on $f(\vec{\mathbf{w}}^{(t+\varkappa+1)}, \vec{\mathbf{w}}^{(t)}) \leq T$, while avoiding the effect of stragglers due to synchronizing only at times when it is absolutely needed.

In the EVENFLOW protocols, consecutive local updates do not necessarily lead to the completion of a round, since no sync may take place if $f(\vec{\mathbf{w}}^{(t+\varkappa+1)}, \vec{\mathbf{w}}^{(t)}) \leq T$. Therefore, we use t to denote the last time a round was completed (i.e, a sync took place) and $t+\varkappa+1$, instead of simply $(t+1)$, to express current time. $\vec{\mathbf{w}}^{(t)}$ stays the same in between synchronizations.

But the function f receives as input the entire vector of neural model weights, i.e. the global model. The crucial question is on how one can monitor f, without having learners communicate all the time with the PS to compute $\vec{\mathbf{w}}^{(t+\varkappa+1)}$ and check if the function exceeds the posed threshold. The EVENFLOW protocols we present in this section provide mechanisms for decomposing $f(\vec{\mathbf{w}}^{(t+\varkappa+1)}, \vec{\mathbf{w}}^{(t)}) \leq T$ to local filters that each learner can check independently, in order to decide whether to request a synchronization or not.

A. The Basic EVENFLOW Protocol

We will first present the operation of the *Basic* protocol and then explain why it is correct to perform synchronization like this, given the application defined $f(\vec{\mathbf{w}}^{(t+\varkappa+1)}, \vec{\mathbf{w}}^{(t)}) \leq T$ tolerance on model weight vector change.

Local Update: Each learner $i \in \{1, 2, \cdots, n\}$ computes its local model parameters $\vec{\mathbf{w}}_i^{(t+\varkappa)}$ after processing its local stream at time $t+\varkappa$:

$$\vec{\mathbf{w}}_i^{(t+\varkappa+1)} = \vec{\mathbf{w}}_i^{(t+\varkappa)} - \eta \nabla_{\vec{\mathbf{w}}_\mathrm{i}} \mathcal{L}_i(\vec{\mathbf{w}}_i^{(t+\varkappa)}) = \vec{\mathbf{w}}_i^{(t+\varkappa)} - \eta \vec{\mathbf{g}}_i^{(t+\varkappa)} \quad (3)$$

Note that for $\varkappa = 0$, i.e., immediately after a synchronization and, thus, round completion, Equation 3 reduces to Equation 1. Equation 3 can then be rewritten using a recursive formula:

$$\vec{\mathbf{w}}_i^{(t+\varkappa+1)} = \vec{\mathbf{w}}_i^{(t)} - \eta \sum_{r=0}^{\varkappa} \nabla_{\vec{\mathbf{w}}_i} \mathcal{L}_i\left(\vec{\mathbf{w}}_i^{(t+r)}\right) = \vec{\mathbf{w}}_i^{(t)} - \eta \sum_{r=0}^{\varkappa} \vec{\mathbf{g}}_i^{(t+r)} \quad (4)$$

Distributively Determine if a Synchronization is needed: Each learner independently constructs a d-dimensional sphere $B\left(\vec{\mathbf{w}}^{(t)} - \frac{1}{2}\eta \sum_{r=0}^{\varkappa} \vec{\mathbf{g}}_i^{(t+r)}, \frac{1}{2}\|\eta \sum_{r=0}^{\varkappa} \vec{\mathbf{g}}_i^{(t+r)}\|\right)$, i.e. a sphere centered at $\vec{\mathbf{w}}^{(t)} - \frac{1}{2}\eta \sum_{r=0}^{\varkappa} \vec{\mathbf{g}}_i^{(t+r)}$ of $\frac{1}{2}\|\eta \sum_{r=0}^{\varkappa} \vec{\mathbf{g}}_i^{(t+r)}\|$ radius. Let $\mathbb{A}$ be the area of the d-dimensional input domain containing all possible weight vectors for which $f(\vec{\mathbf{w}}^{(t+\varkappa+1)}, \vec{\mathbf{w}}^{(t)}) > T$. If for the sphere of learner i $B\left(\vec{\mathbf{w}}^{(t)} - \frac{1}{2}\eta \sum_{r=0}^{\varkappa} \vec{\mathbf{g}}_i^{(t+r)}, \frac{1}{2}\|\eta \sum_{r=0}^{\varkappa} \vec{\mathbf{g}}_i^{(t+r)}\|\right) \bigcap \mathbb{A} = \emptyset$, the learner does not contact the PS. Otherwise, it contacts the PS asking for a synchronization. If no learner asks for a sync, no round completion and no communication takes place. Thus, all learners keep performing local updates as described above. This situation is illustrated in Figure 2 in 3 dimensions and $n = 5$ learners. Since no sphere intersects $\mathbb{A}$, where $f(\vec{\mathbf{w}}^{(t+\varkappa+1)}, \vec{\mathbf{w}}^{(t)}) > T$, no learner calls for a sync. If at least one learner finds its locally constructed sphere intersecting the $\mathbb{A}$ area, a synchronization takes place and a global update is performed (see below).

Global Update: In case the sphere of at least one learner i does intersect $\mathbb{A}$, the learner contacts the PS and a synchronization takes place. The parameter server aggregates the local models and the aggregation to a new global model gives ($\forall \gamma_i > 0$):

$$\vec{\mathbf{w}}^{(t+\varkappa+1)} = \frac{\sum_{i=1}^n \gamma_i \vec{\mathbf{w}}_i^{(t+\varkappa+1)}}{\sum_{i=1}^n \gamma_i} \quad (5)$$

Broadcast: The updated global model $\vec{\mathbf{w}}^{(t+1)}$ (setting $\varkappa = 0$ reflects this last synchronization) is then sent back to all learners, setting $\vec{\mathbf{w}}_i^{(t+1)} \leftarrow \vec{\mathbf{w}}^{(t+1)}$ as in the synchronous protocol. Note that, at the beginning of the training, the PS also broadcasts $f(.)$ and T. Furthermore, $\mathbb{A}$ does not need to get broadcast to the learners. The local learner filter check can be performed by checking if $\underset{\vec{x}}{argmin} f(\vec{x}), \underset{\vec{x}}{argmax} f(\vec{x})$ s.t. $\|\vec{x} - \vec{\mathbf{w}}^{(t)} + \frac{1}{2}\eta \sum_{r=0}^{\varkappa} \vec{\mathbf{g}}_i^{(t+r)}\| \le \frac{1}{2}\|\eta \sum_{r=0}^{\varkappa} \vec{\mathbf{g}}_i^{(t+r)}\|$ lie on different sides of T.

Why the Basic EVENFLOW Protocol is correct: The key point in accommodating any function $f(\vec{\mathbf{w}}^{(t+\varkappa+1)}, \vec{\mathbf{w}}^{(t)})$ in the Basic EVENFLOW synchronization protocol is to perform the monitoring in the input domain, rather than monitoring the function value. The question is, why the local filter $B\left(\vec{\mathbf{w}}^{(t)} - \frac{1}{2}\eta \sum_{r=0}^{\varkappa} \vec{\mathbf{g}}_i^{(t+r)}, \frac{1}{2}\|\eta \sum_{r=0}^{\varkappa} \vec{\mathbf{g}}_i^{(t+r)}\|\right) \bigcap \mathbb{A} = \emptyset$ performed by each learner ensures that $f(\vec{\mathbf{w}}^{(t+\varkappa+1)}, \vec{\mathbf{w}}^{(t)}) \le T$ and, therefore, if the local filter holds for all learners, no synchronization is needed. In other words, why the protocol is correct given the posed tolerance function and threshold. Starting from Equation 5, we add and subtract $\vec{\mathbf{w}}^{(t)}$:

$$\vec{\mathbf{w}}^{(t+\varkappa+1)} = \frac{\sum_{i=1}^{n} \gamma_i \vec{\mathbf{w}}_i^{(t+\varkappa+1)}}{\sum_{i=1}^{n} \gamma_i} =$$

$$= \frac{\sum_{i=1}^{n} \gamma_i \vec{\mathbf{w}}_i^{(t+\varkappa+1)}}{\sum_{i=1}^{n} \gamma_i} + \vec{\mathbf{w}}^{(t)} - \frac{\sum_{i=1}^{n} \gamma_i \vec{\mathbf{w}}_i^{(t)}}{\sum_{i=1}^{n} \gamma_i} =$$

$$= \vec{\mathbf{w}}^{(t)} + \frac{\sum_{i=1}^{n} \gamma_i (\vec{\mathbf{w}}_i^{(t+\varkappa+1)} - \vec{\mathbf{w}}_i^{(t)})}{\sum_{i=1}^{n} \gamma_i} \overset{\text{Equation 4}}{=}$$

$$= \vec{\mathbf{w}}^{(t)} + \frac{\sum_{i=1}^{n} \gamma_i (-\eta \sum_{r=0}^{\varkappa} \vec{\mathbf{g}}_i^{(t+r)})}{\sum_{i=1}^{n} \gamma_i} \Leftrightarrow$$

$$\vec{\mathbf{w}}^{(t+\varkappa+1)} = \frac{\sum_{i=1}^{n} \gamma_i (\vec{\mathbf{w}}^{(t)} - \eta \sum_{r=0}^{\varkappa} \vec{\mathbf{g}}_i^{(t+r)})}{\sum_{i=1}^{n} \gamma_i} \tag{6}$$

Given that $\forall \gamma_i > 0$ and $\sum_{i=1}^{n} \frac{\gamma_i}{\sum_{i=1}^{n} \gamma_i} = 1$, Equation 6 says that, at any given time $(t+\varkappa+1)$, the global vector of model weights, $\vec{\mathbf{w}}^{(t+\varkappa+1)}$ can be expressed as a convex combination of $\vec{\mathbf{w}}^{(t)} - \eta \sum_{r=0}^{\varkappa} \vec{\mathbf{g}}_i^{(t+r)}$ vectors. And as a convex combination of these vectors, it lies in their convex hull, i.e., $\vec{\mathbf{w}}^{(t+\varkappa+1)} \in Conv\left(\vec{\mathbf{w}}^{(t)} - \eta \sum_{r=0}^{\varkappa} \vec{\mathbf{g}}_1^{(t+r)}, \cdots, \vec{\mathbf{w}}^{(t)} - \eta \sum_{r=0}^{\varkappa} \vec{\mathbf{g}}_n^{(t+r)}\right)$

The convex hull formed by such vectors is illustrated in yellow in Figure 2. Now then, it has been proven [18]–[20] that for any convex hull of vectors having a common starting point (in our case the starting point is $\vec{\mathbf{w}}^{(t)}$), it holds:

$$Conv\left(\vec{\mathbf{w}}^{(t)} - \eta \sum_{r=0}^{\varkappa} \vec{\mathbf{g}}_1^{(t+r)}, \cdots, \vec{\mathbf{w}}^{(t)} - \eta \sum_{r=0}^{\varkappa} \vec{\mathbf{g}}_n^{(t+r)}\right) \subseteq$$

$$\bigcup_{i=1}^{n} B\left(\vec{\mathbf{w}}^{(t)} - \frac{1}{2}\eta \sum_{r=0}^{\varkappa} \vec{\mathbf{g}}_i^{(t+r)}, \frac{1}{2}\|\eta \sum_{r=0}^{\varkappa} \vec{\mathbf{g}}_i^{(t+r)}\|\right)$$

Summing up, because at any given time the global vector of model weights, $\vec{\mathbf{w}}^{(t+\varkappa+1)}$ lies in the said convex hull, it suffices to find a way to distributively monitor this convex hull. The above inclusion says that the convex hull is a subset of the union of the spheres constructed independently by each learner. If no such sphere intersects $\mathbb{A}$, as in Figure 2, no synchronization is necessary because $\vec{\mathbf{w}}^{(t+\varkappa+1)}$ cannot have entered the area of the input domain where $f(\vec{\mathbf{w}}^{(t+\varkappa+1)}, \vec{\mathbf{w}}^{(t)}) > T$. This explains the correctness of the protocol with respect to the posed tolerance constraint expressed in $f(\vec{\mathbf{w}}^{(t+\varkappa+1)}, \vec{\mathbf{w}}^{(t)}) > T$. The Basic EVENFLOW protocol, although it synchronizes all learners, keeps such syncs to only a need-to-happen basis. It, thus, achieves both model accuracy based on the monitored thresholded function and reduces the effect of stragglers.

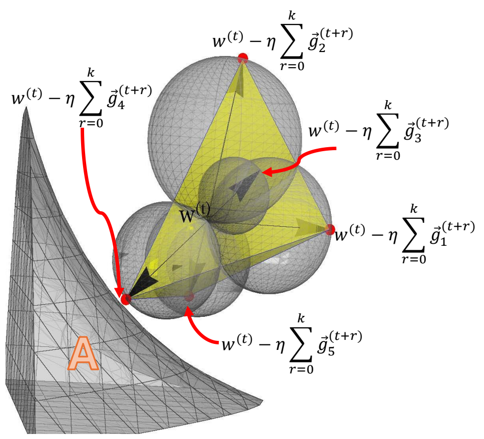

$w^{(t)} - \eta \sum_{r=0}^{k} \vec{g}_2^{(t+r)}$
$w^{(t)} - \eta \sum_{r=0}^{k} \vec{g}_4^{(t+r)}$
$w^{(t)} - \eta \sum_{r=0}^{k} \vec{g}_3^{(t+r)}$
$w^{(t)}$
$w^{(t)} - \eta \sum_{r=0}^{k} \vec{g}_1^{(t+r)}$
$w^{(t)} - \eta \sum_{r=0}^{k} \vec{g}_5^{(t+r)}$
A

Fig. 2: Basic EVENFLOW protocol. d-spheres constructed by $n = 5$ learners. $d = 3$ is used just for visualization purposes.

The Basic EVENFLOW protocol is useful for small neural networks like MultiLayer Perceptrons or, in the context of transfer learning, for already trained neural nets which we further refine by training them having added up few hundreds of neurons. When the dimensionality d increases, checking if $B\left(\vec{\mathbf{w}}^{(t)} - \frac{1}{2}\eta \sum_{r=0}^{\varkappa} \vec{\mathbf{g}}_i^{(t+r)}, \frac{1}{2}\|\eta \sum_{r=0}^{\varkappa} \vec{\mathbf{g}}_i^{(t+r)}\|\right) \bigcap \mathbb{A} = \emptyset$ becomes computationally intensive to an extent that outweighs the benefit of omitting unnecessary synchronizations and, thus, the effect of potential delays due to stragglers. Therefore, we introduce the Fast EVENFLOW protocol.

B. The Fast EVENFLOW Protocol

Compared to the Basic EVENFLOW, the Fast EVENFLOW protocol better tackles the computational complexity problem incurred by the increased dimensionality d of the weight vectors. It does so by applying a dimensionality reduction technique, namely Fast Fourier Transforms (FFT) for Discrete Fourier Transform (DFT) computation, and by providing deterministic bounds on the quality of the approximation of the monitoring problem $f(\vec{\mathbf{w}}^{(t+\varkappa+1)}, \vec{\mathbf{w}}^{(t)}) > T$, at hand. For this, we apply a transformation that subtracts $\vec{\mathbf{w}}^{(t)}$ (recall that this latter vector does not change until the next synchronization) from all the involved vectors. That is, we want to distributively monitor $f(\vec{\mathbf{w}}^{(t+\varkappa+1)} - \vec{\mathbf{w}}^{(t)}, \vec{0}) > T_{transformed} \Leftrightarrow f(\vec{\mathbf{w}}^{(t+\varkappa+1)}, \vec{\mathbf{w}}^{(t)}) > T$.

The phases of local update and, when needed, global update are exactly the same as in the Basic Protocol. What changes is the way the protocol distributively determines if a sync is needed, with vectors of reduced dimensionality, providing respective quality guarantees. Additionally, the content of the broadcast by the parameter server slightly differs, as we discuss shortly. We emphasize again that we do not aim at performing the training, local or global, itself using vectors of reduced dimensionality. Fast EVENFLOW uses such reduced vectors only for distributively monitoring $f(\vec{\mathbf{w}}^{(t+\varkappa+1)} - \vec{\mathbf{w}}^{(t)}, \vec{0}) > T_{transformed}$ using new versions of the local filters.

Distributively Determine if a Synchronization is needed:

In the transformed problem, having subtracted $\vec{\mathbf{w}}^{(t)}$, the local filter each learner i should check involves the sphere with a modified center, i.e., $B\left(-\frac{1}{2}\eta\sum_{r=0}^{\varkappa}\vec{\mathbf{g}}_i^{(t+r)}, \frac{1}{2}\|\eta\sum_{r=0}^{\varkappa}\vec{\mathbf{g}}_i^{(t+r)}\|\right)$. Note that subtracting $\vec{\mathbf{w}}^{(t)}$ does not affect the relative directions of the sphere centers or magnitudes of radii, but only their positions and this is the reason the threshold should become $T_{transformed}$ instead of T. However, this does not solve the dimensionality problem. To reduce the dimensionality of the problem at hand, each learner i utilizes Discrete Fourier Transforms (DFT) [21]. The DFT of the sum $\sum_{r=0}^{\varkappa}\vec{\mathbf{g}}_i^{(t+r)}$ vector can be computed exploiting the linearity of DFTs [22]:

$$\sum_{r=0}^{\varkappa}\widetilde{\mathbf{g}}_i^{(t+r)} = \sum_{r=0}^{\varkappa}\sum_{\lambda=0}^{d-1}\vec{\mathbf{g}}_i^{(t+r)}[\lambda]\cdot e^{-i\cdot 2\pi\cdot\xi\cdot\lambda/d}, \xi = 0, 1, \cdots, d-1$$

Here, $\widetilde{\mathbf{g}}_i^{(t+r)}$ represents the DFT of the vector $\vec{\mathbf{g}}_i^{(t+r)}$, and the formula sums these DFTs over all r from 0 to $\varkappa$. This computation can be performed in $O(n\ell ogn)$ time using FFT.

Learners can independently reduce the dimensionality of $\widetilde{\mathbf{g}}_i^{(t+r)}$ by sorting the coefficients in the $\widetilde{\mathbf{g}}_i^{(t+r)}$ in descending order and keep only the $m \ll d$ largest such Fourier coefficients. Now, by using the reduced dimensionality of $\widetilde{\mathbf{g}}_i^{(t+r)}{}_m$, the sphere that each learner should construct and check is $B\left(-\frac{1}{2}\eta\sum_{r=0}^{\varkappa}\widetilde{\mathbf{g}}_i^{(t+r)}{}_m, \frac{1}{2}\eta\|\sum_{r=0}^{\varkappa}\widetilde{\mathbf{g}}_i^{(t+r)}{}_m\|\right)$.

Broadcast: If a sync takes place, the parameter server sends back to the learners the $T_{transformed}$ threshold, the value of m (only in the first synchronization) and $\vec{\mathbf{w}}_i^{(t+1)} \leftarrow \vec{\mathbf{w}}^{(t+1)}$.

How good is the Fast EVENFLOW Protocol: But how good is this approximated sphere, compared to the original one? Let us begin by comparing the sizes of the radii. We stress that the learners just perform the monitoring the way described above; they do not perform the computations we present below. We do this computations to provide the quality guarantees, in the form of deterministic error bounds, of performing the local checks using the vectors of reduced dimensionality. The Inverse Discrete Fourier Transform (IDFT) of the gradient vectors is given by [22]: $\sum_{r=0}^{\varkappa}\vec{\mathbf{g}}_i^{(t+r)}[\lambda] = \frac{1}{d}\sum_{\lambda=0}^{d-1}\left(\sum_{r=0}^{\varkappa}\widetilde{\mathbf{g}}_i^{(t+r)}[\lambda]\right)\cdot e^{i\cdot 2\pi\cdot\xi\cdot\lambda/d}$. According to Parseval's Theorem [22], for each gradient: $\|\vec{\mathbf{g}}_i^{(t+r)}\|^2 = \frac{1}{d}\|\widetilde{\mathbf{g}}_i^{(t+r)}\|^2$. Then, the approximation of the initial gradients, by using only $m \ll d$ coefficients, is upper bounded by: $\left\|\vec{\mathbf{g}}_i^{(t+r)} - \widetilde{\mathbf{g}}_i^{(t+r)}{}_m\right\| \leq \sqrt{\frac{1}{d}\sum_{\lambda=m}^{d-1}|\widetilde{\mathbf{g}}_i^{(t+r)}[\lambda]|^2}$. The maximum value of each $|\widetilde{\mathbf{g}}_i^{(t+r)}[\lambda]|^2$ is $\|\vec{\mathbf{g}}_i^{(t+r)}\|^2$. Thus:

$$\left\|\vec{\mathbf{g}}_i^{(t+r)} - \widetilde{\mathbf{g}}_i^{(t+r)}{}_m\right\| \leq \sqrt{1 - \frac{m}{d}}\|\vec{\mathbf{g}}_i^{(t+r)}\| \tag{7}$$

Lemma 1. *Any learner $i \in \{1, 2, \cdots, n\}$, after $\varkappa$ local updates since the last synchronization, will construct a m-dimensional hypersphere in the DFT space ($m \ll d$) with a radius which approximates the $\left\|\frac{1}{2}\eta\sum_{r=0}^{\varkappa}\vec{\mathbf{g}}_i^{(t+r)}\right\|$ radius of the original d-dimensional space, with an error at most $\frac{1}{2}\eta\sqrt{(\varkappa+1)\cdot\left(1-\frac{m}{d}\right)}\|\vec{\mathbf{g}}_i^{(t+r)}\|$:*

$$\left|\left\|\frac{1}{2}\eta\sum_{r=0}^{\varkappa}\vec{\mathbf{g}}_i^{(t+r)}\right\| - \left\|\frac{1}{2}\eta\sum_{r=0}^{\varkappa}\widetilde{\mathbf{g}}_i^{(t+r)}{}_m\right\|\right| \leq$$
$$\leq \frac{1}{2}\eta\sqrt{(\varkappa+1)\cdot\left(1-\frac{m}{d}\right)}\|\vec{\mathbf{g}}_i^{(t+r)}\|$$

Proof. Consider the sum of differences between the original and DFT gradient vectors over all r from 0 to $\varkappa$, $\sum_{r=0}^{\varkappa}\left(\vec{\mathbf{g}}_i^{(t+r)} - \widetilde{\mathbf{g}}_i^{(t+r)}{}_m\right)$. We iniially want to find an upper bound on the norm of this sum $\left\|\sum_{r=0}^{\varkappa}\left(\vec{\mathbf{g}}_i^{(t+r)} - \widetilde{\mathbf{g}}_i^{(t+r)}{}_m\right)\right\|$. We apply Jensen's Inequality to the squared norm function, which is convex. Jensen's inequality tells us:

$$\left\|\frac{1}{\varkappa+1}\sum_{r=0}^{\varkappa}\left(\vec{\mathbf{g}}_i^{(t+r)} - \widetilde{\mathbf{g}}_i^{(t+r)}{}_m\right)\right\|^2 \leq$$
$$\leq \frac{1}{\varkappa+1}\sum_{r=0}^{\varkappa}\left\|\vec{\mathbf{g}}_i^{(t+r)} - \widetilde{\mathbf{g}}_i^{(t+r)}{}_m\right\|^2 \tag{8}$$

From Inequality 7 and Inequality 8 we derive:

$$\left\|\frac{1}{\varkappa+1}\sum_{r=0}^{\varkappa}\left(\vec{\mathbf{g}}_i^{(t+r)} - \widetilde{\mathbf{g}}_i^{(t+r)}{}_m\right)\right\|^2 \leq (1 - \frac{m}{d})\|\vec{\mathbf{g}}_i^{(t+r)}\|^2$$

Taking the square root and multiplying by $\sqrt{\varkappa+1}$:

$$\left\|\sum_{r=0}^{\varkappa}\left(\vec{\mathbf{g}}_i^{(t+r)} - \widetilde{\mathbf{g}}_i^{(t+r)}{}_m\right)\right\| \leq \sqrt{(\varkappa+1)\cdot\left(1-\frac{m}{d}\right)}\|\vec{\mathbf{g}}_i^{(t+r)}\| \tag{9}$$

Applying the Triangle inequality on the left-hand side of the above and multiplying by $\frac{1}{2}\eta$ concludes the proof. □

Example 1. *Assume we have ResNet-18 [23], with $d = 11.7\times 10^6$ (number of parameters/weights), $m = 50$ (number of DFT coefficients), $\eta = 0.001$ (typical learning rate for Adam) and relatively large gradient norm $\|\mathbf{g}_i^{(t+r)}\| \leq 10$: For $\varkappa = 1$: $\frac{1}{2}\eta\sqrt{(\varkappa+1)\cdot\left(1-\frac{m}{d}\right)}\|\mathbf{g}_i^{(t+r)}\| \approx 0.007071$. For $\varkappa = 10$: $\frac{1}{2}\eta\sqrt{(\varkappa+1)\cdot\left(1-\frac{m}{d}\right)}\|\mathbf{g}_i^{(t+r)}\| \approx 0.016583$. For $\varkappa = 100$: $\frac{1}{2}\eta\sqrt{(\varkappa+1)\cdot\left(1-\frac{m}{d}\right)}\|\mathbf{g}_i^{(t+r)}\| \approx 0.050249$*

The following lemma provides a deterministic upper bound on the position of the new, DFT-based sphere center as a direct consequence of the analysis in the proof of Lemma 1.

Corollary 1. *Any learner $i \in \{1, 2, \cdots, n\}$, after $\varkappa$ local updates since the last synchronization, will construct a m-dimensional hypersphere in the DFT space ($m \ll d$) which approximates the position of the sphere center $\left(-\frac{1}{2}\eta \sum_{r=0}^{\varkappa} \vec{\mathbf{g}}_i^{(t+r)}\right)$ of the original d-dimensional space, with an error at most $\frac{1}{2}\eta\sqrt{(\varkappa+1)\cdot\left(1-\frac{m}{d}\right)}\|\vec{\mathbf{g}}_i^{(t+r)}\|$:*

$$\left\|\frac{1}{2}\eta\sum_{r=0}^{\varkappa}\vec{\mathbf{g}}_i^{(t+r)} - \frac{1}{2}\eta\sum_{r=0}^{\varkappa}\widetilde{\mathbf{g}}_i^{(t+r)}{}_m\right\| \leq$$
$$\leq \frac{1}{2}\eta\sqrt{(\varkappa+1)\cdot\left(1-\frac{m}{d}\right)}\|\vec{\mathbf{g}}_i^{(t+r)}\|$$

Proof. A direct consequence of Inequality 9. □

IV. Experimental Evaluation

Experimental Setup: In our evaluation we utilize the physics scenario of Section I over the SUSY dataset [8], split into a 80-20% train-test data stream proportion. The dataset includes 5M tuples and each tuple is composed of 18 features. The aim is to timely distinguish signal processes which produce supersymmetric particles and background processes which do not. Please refer to [8], [24] for further details.

We train 4 different feed forward neural networks of 400 (`NN400`), 4K (`NN4K`), 400K (`NN400K`) and 4M (`NN4M`) weights, respectively, using the maximum stream ingestion rate supported by Kafka. The problem is modeled as a regression task, where each network outputs a single continuous value representing the estimated probability of a signal process occurring. The loss function used is Mean Squared Error (MSE). To assess model performance, we track the MSE on the test data stream during training and also measure the total training time for the entire stream.

We follow a fine tuned streaming setup where the streaming window size is set to 256 tuples, split into 4 batches and the number of epochs is set to 8, with a typical learning rate of 0.001. We report the accuracy and the training time of each neural net using a default value of 7 learners, but we also report on scalability experiments varying the number of learners between 3 and 15. To purely measure the effect of the EVENFLOW protocols, we do not include in our plots the time it takes to write the updated Kafka weights to the `Weights Topic` of Figure 1(b) before predictors can read it. This is a time lag common to all methods, but its effect is much less present in the EVENFLOW protocols due to less frequent (partial or full) synchronizations. Therefore, the reported times are actually a worst case scenario for our techniques.

For the Fast EVENFLOW protocol, we set the number of DFT coefficients to $m = 50$, but we also report performance results upon varying this parameter, as well. The monitored function which expresses the application tolerance value on weight changes before a synchronization, for the EVENFLOW protocols, is KL-divergence, while the threshold T is set to 0.3.

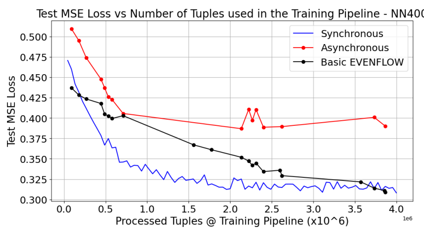
The EVENFLOW toolkit, as well as the vanilla synchronous and asynchronous protocols, are implemented in PyTorch 2.4.0+cu121. We use a Google Colab Pro+ subscription and run our experiments on A100 GPU setup with 83.5GB system RAM and 40GB GPU RAM.

Analysis on `NN400`: We begin our experimental evaluation and comparison of EVENFLOW against the vanilla protocols, with the `NN400` neural net. In Figure 3, we plot the Test MSE Loss across the training process (Figure 3a) and the Training Time to complete the training on the entire set of SUSY data streams (Figure 3b). The Basic and the Fast EVENFLOW protocols show negligible differences in their performance for `NN400`, therefore our plots include only Basic EVENFLOW. As shown in Figure 3a, the Basic EVENFLOW protocol achieves a Test MSE Loss that lies between the respective Asynchronous (red) and Synchronous (blue) plot lines. Basic EVENFLOW improves Asynchronous up to 35%, while being up to 15-18% worse compared to the Synchronous protocol. However, the first indication that the EVENFLOW protocol combines the virtues of both protocols comes when we combine these MSE Loss results with the Training Time in Figure 3b. There, the Basic EVENFLOW improves the Synchronous one by more than 2 times and the Asynchronous protocol by 33%. Thus, Basic EVENFLOW achieves the best combination of MSE Loss and Training Time.

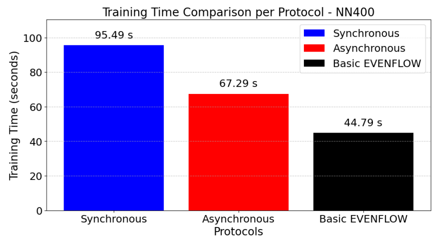
It is easy to see why Basic EVENFLOW improves the Synchronous protocol in terms of training time, i.e., by avoiding superfluous synchronizations; and the Asynchronous protocol in terms of MSE Loss by avoiding partial updates to some learners and stale gradients to others. But, the natural question that arises is how and why it improves the Asynchronous protocol, which does not introduce synchronization barriers between learners, in terms of training time. Nevertheless, the training time is measured as the time it takes for all learners to process their assigned streams in their entirety. In an Asynchronous protocol, while there are no synchronization barriers, the frequent partial updates by some learners engage unnecessary communication overhead and the frequent transmission of gradients that are only partially informative.

Next, for the `NN4K`, `NN400K` and `NN4M` neural networks, the Basic EVENFLOW protocol is not present. This is because the computational complexity of the local sphere intersection checks by the learners cannot abide by the posed 100ms RPC (Remote Procedure Calls) timeout, while communicating with the parameter server. Thus, we only use the Fast EVENFLOW protocol, which is the reason we invent it in the first place.

Analysis on `NN4K`: As explained above, we have switched to the Fast EVENFLOW protocol with $m = 50$ coefficients. In Figure 5, the situation does not change a lot, with the major difference being that, now, Fast EVENFLOW improves Synchronous in terms of training time by 4, instead of 2 times we observed in `NN400`. This is due to the increased dimensionality of the weight vectors which make the training process and the continuous syncs, of Synchronous, more time consuming. There are two more observations that we need to make here. First, in Figure 5b, the Training Time of Fast EVENFLOW is closer to the Asynchronous protocol. This is because, before the local check by the learners, there is also an overhead in computing the DFT coefficients. The



(a) NN400 Test MSE Loss



(b) NN400 Training Time

Fig. 3: Comparison of NN400 Test MSE Loss and Training Time

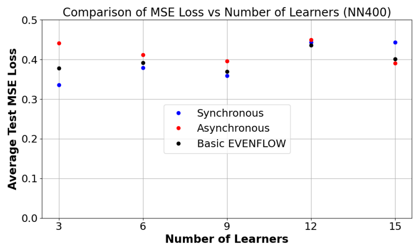
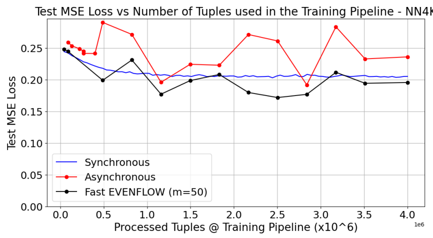
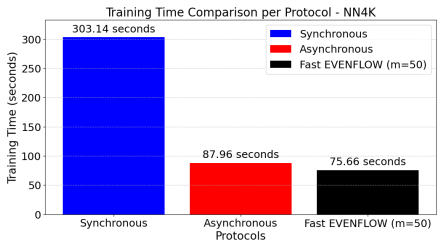


Fig. 4: LossVs#Learners NN400



(a) NN4K Test MSE Loss



(b) NN4K Training Time

Fig. 5: Comparison of NN4K Test MSE Loss and Training Time

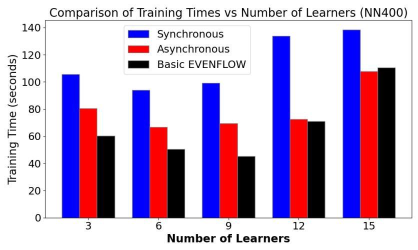
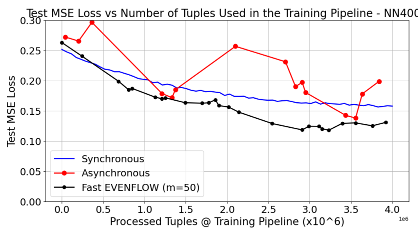
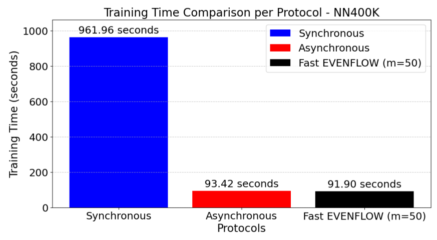


Fig. 6: TrainTimeVs#Learners NN400



(a) NN400K Test MSE Loss



(b) NN400K Training Time

Fig. 7: Comparison of NN400K Loss and Training Time

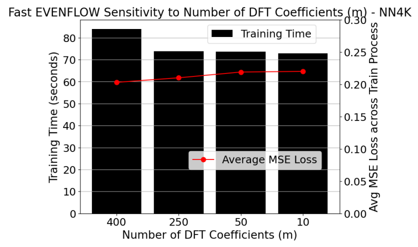
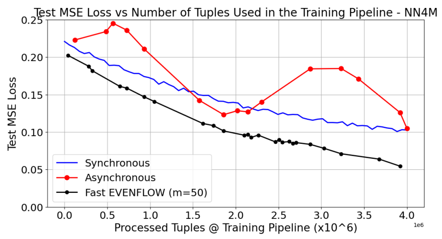
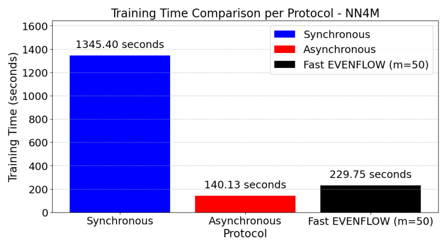


Fig. 8: #DFT Coefficients Sensitivity



(a) NN4M Test MSE Loss



(b) NN4M Training Time

Fig. 9: Comparison of NN4M Test MSE Loss and Training Time

second observation is that, in Figure 5a, the Fast EVENFLOW protocol occasionally shows lower Test MSE Loss than the Synchronous approach (up to 10% lower MSE for Fast EVENFLOW). This is due to the reduced frequency of synchronization points in the Fast EVENFLOW protocol. This reduction helps prevent the protocol from reacting too strongly to minor (potentially noisy) gradient fluctuations, unlike the Synchronous approach, where frequent synchronizations can cause the global weight vector to update too often in response to small, potentially insignificant changes.

Analysis on `NN400K` and `NN4M`: Figure 7 and Figure 9 present the respective results for the `NN400K` and `NN4M` neural nets. The Fast EVENFLOW protocol proves again that it combines the virtues of the two vanilla approaches offering rapid training time, while surpassing the best performance in terms of Test MSE Loss. In particular, in Figures 7a and 9a, we can see that Fast EVENFLOW clearly outperforms Asynchronous, while improving the Synchronous protocol up to 50% in terms of Test MSE Loss. Here, we have another, implicit advantage of the EVENFLOW toolkit we observe in practice. Fast EVENFLOW improves Synchronous due to the fact that, by synchronizing less frequently, it avoids temporary overfittings that occur during the streaming training process. On the other hand, in Figures 7b and 9b, we can observe that Fast EVENFLOW performs slightly worse than Asynchronous in term of training time. This is consistent with our previous observation that the time it takes to compute the DFT coefficients (besides the local sphere intersection check) increases with the dimensionality of the weight vectors.

Fast EVENFLOW Sensitivity to the Number of DFT Coefficients (m): According to Lemma 1 and Corollary 1, the upper bound on the accuracy of the Fast EVENFLOW depends on m and d, but the involved squared root $\sqrt{1-\frac{m}{d}}$ becomes 1 as d increases. Therefore, in Figure 8, we plot the way the Training Time and the Test MSE Loss of Fast EVENFLOW are affected by altering m from 400 to 250, 50 and finally 10 coefficients. The horizontal axis shows the values of m. The vertical axis on the left-hand side measures the Training Time, while the vertical axis on the right-hand side measures the Average Test MSE Loss during the training process. As Figure 8 illustrates, switching from 400 to 250 coefficients may improve the total training time of Fast EVENFLOW by 10 seconds. After that point, switching to 50 and 10 coefficients, gives a negligible improvement. Furthermore, the Average MSE Loss increases from 0.2 for $m = 400$ to 0.22 for $m = 50$ and $m = 10$. This explains the choice of a moderate $m = 50$ value in the experiments we previously discussed, to both decrease training time without favoring Fast EVENFLOW in terms of MSE loss. In all, for above 250 coefficients, the protocol shows little sensitivity to the chosen value of m.

Scaling with the Number of Learners (n): In Figure 4 and Figure 6, we plot the Average MSE Loss and the Training Time varying the number of learners in our setup. To purely judge the effect of parallelism in our experiments, we use the `NN400` neural network, because it is less affected by the computational cost of the training itself, compared to the more complex `NN4K`, `NN400K` and `NN4M` neural networks. This allows us to better understand how the system scales with the number of learners and how synchronization overhead impacts performance without the results being overshadowed by the computational complexity of the neural network itself. According to Figure 4, all protocols seem to simultaneously exhibit good MSE values upon being trained with 9 learners. On the other hand, Figure 6 shows that after 6 learners the training times tend to increase. For Synchronous and EVENFLOW this happens due to the fact that the synchronization process becomes more complex and time consuming as more learners are introduced beyond a certain point, outweighing the benefits of parallelism. On the other hand, for the Asynchronous protocol, with more learners communicating asynchronously, the network and system overhead can increase. The server must handle more incoming gradient updates, manage communication, and perform frequent updates to the model. This increased overhead can slow down the overall training process, as discussed earlier in our analysis in Figure 3b. Combining the above results for MSE Loss and Training Time, it appears that the sweet spot, where all protocols show good MSE and the parallelism favors decreased training times, is between 6 and 9 learners. This explains why we used a default number of 7 learners in our previous analysis.

V. Related Work

The PS paradigm was introduced in Li et al. [13]. Dean et al. [25] introduced DistBelief, facilitating asynchronous stochastic gradient descent (ASGD) to manage communication overheads. More recent efforts have focused on reducing the frequency and size of communications. For instance, Stich [26], [27] proposed a Local SGD method, where each learner performs multiple local updates before synchronizing with others. EVENFLOW differs by providing data-driven decision making mechanisms to distributively prescribe when a sync is needed. Moreover, these approaches are orthogonal to EVENFLOW. For instance, EVENFLOW can be used by learners to perform their local, sphere intersection checks having performed the local updates as per Stich [26].

Decentralized learning offers an alternative to the centralized PS paradigm [14]–[16], [28]. Lian et al. [28] demonstrated that decentralized SGD (DSGD) could achieve similar convergence rates to centralized methods while being more robust to training. As noted in the intro, using such a paradigm in streaming settings means that the whole set of learners should be continuously queried to deduce which learner holds the global model at any given time, for the `Weights Topic` of Figure 1(b) to get updated. Hence, the advantages of decentralized learning are diminished by this barrier.

For Federated Learning, the introduction of Federated Averaging (FedAvg) [5] was pivotal. To further improve the efficiency, various methods have been developed to address the issues of model divergence and communication overhead. Zhao et al. [29] and Reddi et al. [30] highlighted the challenge of model divergence, where local models drift towards their own minima, causing slow and unstable convergence. SCAFFOLD [31] was introduced to correct local updates using control variates, significantly speeding up convergence in federated settings. Similarly, FedProx [32] added regularization to the FedAvg method to keep local models closer to the global model, reducing divergence. These approaches can be used on par with EVENFLOW, since they prescribe when and how gradients or weights are computed or hyper-parameters are tuned. After that, EVENFLOW determines the need of a sync.

Compression techniques like sparsification and quantization have been used in neural learning. Alistarh et al. [33] explored sparsification, while Seide et al. [34] investigated quantization techniques, both transmitting gradients in a compressed form. These approaches have been combined with Local-SGD as well [35]. This is complementary to EVENFLOW. Having determined when a sync is needed, the actual gradients can be communicated using the above methods. However, such approaches cannot replace DFT in EVENFLOW, as they do not provide a priori known error bounds or may introduce approximation errors that affect the correctness of the data-driven sync decision. Sketching methods [36], [37] do preserve linearity and are suitable for similar tasks, but they provide probabilistic rather than deterministic error bounds, leading to monitoring fuzzy instead of fixed convex hulls.

Accelerating the convergence of distributed training is a crucial strategy for reducing training speed. Recent efforts include FedAdam [30], FedAvgM [38], Mime [39], FedDyn [40]. The sync decision making mechanisms of EVENFLOW are complementary to these techniques in reducing training time, simultaneously abiding by the posed weight change tolerance.

VI. Conclusion

We present EVENFLOW, a toolkit of data-driven synchronization protocols that achieve highly accurate data-parallel training with rapid training times under the PS paradigm. EVENFLOW models the synchronization decision problem as a distributed functional monitoring problem and solves this problem from any application-provided function. Our future work focuses on studying EVENFLOW on federated learning environments and across the cloud to edge continuum.

References

[1] N. Giatrakos, "SSTRESED: scalable semantic trajectory extraction for simple event detection over streaming movement data," in *TIME*, 2023.

[2] A. Deligiannakis, N. Giatrakos, Y. Kotidis, V. Samoladas, and A. Simitsis, "Extreme-scale interactive cross-platform streaming analytics - the INFORE approach," in *SEA-Data @ VLDB*, 2021.

[3] N. Giatrakos and et al, "Infore: Interactive cross-platform analytics for everyone," in *CIKM*, 2020.

[4] N. Giatrakos, A. Deligiannakis, K. Bereta, M. Vodas, D. Zissis, E. Alevizos, C. Akasiadis, and A. Artikis, "Processing big data in motion: Core components and system architectures with applications to the maritime domain," in *Technologies and Applications for Big Data Value*. Springer, 2022, pp. 497–518.

[5] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *AISTATS*, 2017.

[6] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall, "Robot operating system 2: Design, architecture, and uses in the wild," *Science Robotics*, vol. 7, no. 66, 2022.

[7] S. Macenski, A. Soragna, M. Carroll, and Z. Ge, "Impact of ros 2 node composition in robotic systems," *IEEE Robotics and Autonomous Letters (RA-L)*, 2023. [Online]. Available: https://arxiv.org/abs/2305.09933

[8] P. Baldi, P. Sadowski, and D. Whiteson, "Searching for exotic particles in high-energy physics with deep learning," *Nature communications*, vol. 5, p. 4308, 07 2014.

[9] "High energy physics simulations," https://lhcathome.web.cern.ch/projects/test4theory/high-energy-physics-simulations, accessed: 2024-10-29.

[10] M. Vodas, K. Bereta, D. Kladis, D. Zissis, E. Alevizos, E. Ntoulias, A. Artikis, A. Deligiannakis, A. Kontaxakis, N. Giatrakos, D. Arnu, E. Yaqub, F. Temme, M. Torok, and R. Klinkenberg, "Online distributed maritime event detection & forecasting over big vessel tracking data," in *IEEE Big Data* , 2021.

[11] Y. Huang, T. Jin, Y. Wu, Z. Cai, X. Yan, F. Yang, J. Li, Y. Guo, and J. Cheng, "Flexps: flexible parallelism control in parameter server architecture," *Proc. VLDB Endow.*, vol. 11, no. 5, p. 566–579, jan 2018.

[12] L. Mai, G. Li, M. Wagenländer, K. Fertakis, A.-O. Brabete, and P. Pietzuch, "Kungfu: making training in distributed machine learning adaptive," in *OSDI*, 2020.

[13] M. Li, D. G. Andersen, J. W. Park, A. J. Smola, A. Ahmed, V. Josifovski, J. Long, E. J. Shekita, and B.-Y. Su, "Scaling distributed machine learning with the parameter server," in *OSDI*, 2014.

[14] X. Lian, W. Zhang, C. Zhang, and J. Liu, "Asynchronous decentralized parallel stochastic gradient descent," in *ICML*, 2018.

[15] H. Tang, X. Lian, M. Yan, C. Zhang, and J. Liu, "D^2: Decentralized training over decentralized data," in *ICML*, 2018.

[16] I. Hegedüs, G. Danner, and M. Jelasity, "Decentralized learning works: An empirical comparison of gossip learning and federated learning," *J. Parallel Distributed Comput.*, vol. 148, pp. 109–124, 2021.

[17] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[18] I. Sharfman, A. Schuster, and D. Keren, "A geometric approach to monitoring threshold functions over distributed data streams," in *SIGMOD*, 2006.

[19] N. Giatrakos, A. Deligiannakis, M. N. Garofalakis, I. Sharfman, and A. Schuster, "Distributed geometric query monitoring using prediction models," *ACM Trans. Database Syst.*, vol. 39, no. 2, 2014.

[20] N. Giatrakos, A. Deligiannakis, M. N. Garofalakis, D. Keren, and V. Samoladas, "Scalable approximate query tracking over highly distributed data streams with tunable accuracy guarantees," *Inf. Syst.*, vol. 76, pp. 59–87, 2018.

[21] A. Kontaxakis, N. Giatrakos, and A. Deligiannakis, "A synopses data engine for interactive extreme-scale analytics," in *CIKM*, 2020.

[22] D. Rafiei and A. Mendelzon, "Similarity-based queries for time series data," *SIGMOD Rec.*, vol. 26, no. 2, p. 13–25, jun 1997.

[23] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[24] D. Whiteson, "SUSY," UCI Machine Learning Repository, 2014, DOI: https://doi.org/10.24432/C54606.

[25] J. Dean, G. S. Corrado, R. Monga, K. Chen, M. Devin, Q. V. Le, M. Z. Mao, M. Ranzato, A. Senior, P. Tucker *et al.*, "Large scale distributed deep networks," in *NIPS*, 2012.

[26] S. U. Stich, "Local sgd converges fast and communicates little," in *ICLR*, 2018.

[27] ——, "Communication-efficient distributed learning with deep networks on non-idd data," in *NeurIPS*, 2019.

[28] X. Lian, C. Zhang, H. Zhang, C.-J. Hsieh, W. Zhang, and J. Liu, "Can decentralized algorithms outperform centralized algorithms? a case study for decentralized parallel stochastic gradient descent," in *NIPS*, 2017.

[29] Y. Zhao, M. Li, L. Lai, N. Suda, D. Civin, and V. Chandra, "Federated learning with non-iid data," *CoRR*, vol. abs/1806.00582, 2018. [Online]. Available: http://arxiv.org/abs/1806.00582

[30] S. J. Reddi, Z. Charles, M. Zaheer, Z. Garrett, K. Rush, J. Konecný, S. Kumar, and H. B. McMahan, "Adaptive federated optimization," in *ICLR*, 2021.

[31] S. P. Karimireddy, S. Kale, M. Mohri, S. J. Reddi, S. U. Stich, and A. T. Suresh, "Scaffold: Stochastic controlled averaging for federated learning," in *ICML*, 2020.

[32] T. Li, A. K. Sahu, A. Talwalkar, and V. Smith, "Federated learning: Challenges, methods, and future directions," *IEEE Signal Processing Magazine*, vol. 37, no. 3, pp. 50–60, 2020.

[33] D. Alistarh, D. Grubic, J. Li, R. Tomioka, and M. Vojnovic, "Qsgd: Communication-efficient sgd via gradient quantization and encoding," in *NIPS*, 2017.

[34] F. Seide, H. Fu, J. Droppo, G. Li, and D. Yu, "1-bit stochastic gradient descent and its application to data-parallel distributed training of speech dnns," in *INTERSPEECH*, 2014.

[35] A. Reisizadeh, A. Mokhtari, H. Hassani, and R. Pedarsani, "Qsparse-local-sgd: Distributed sgd with quantization, sparsification, and local computations," in *NeurIPS*, 2020.

[36] D. P. Woodruff and Q. Zhang, "Sketching as a tool for numerical linear algebra," *Foundations and Trends in Theoretical Computer Science*, vol. 10, no. 1–2, pp. 1–157, 2014.

[37] N. Ivkin, D. Alistarh, T. Hoefler, P. W. Lo, S. Rouault, D. Tajarov, and M. Vojnovic, "Communication-efficient distributed sgd with sketching," in *NeurIPS*, 2019.

[38] T. C. Hsu, H. Qi, and M. Brown, "Measuring the effects of non-identical data distribution for federated visual classification," in *ICLR*, 2019.

[39] S. P. Karimireddy, S. Kale, M. Mohri, S. J. Reddi, S. U. Stich, and A. T. Suresh, "Mime: Mimicking centralized stochastic algorithms in federated learning," in *NeurIPS*, 2020.

[40] D. A. E. Acar, Y. Zhao, R. Matas, M. Mattina, P. Whatmough, and V. Saligrama, "Federated learning based on dynamic regularization," in *ICLR*, 2021.