# Privacy Preserving Distributed Training of Neural Networks

Spyridon Nikolaidis[1] and Ioannis Refanidis[1]

[1] University of Macedonia, Thessaloniki 54636, GREECE
{sp.nikola, yrefanid}@uom.edu.gr

**Abstract.** LEARNAE is a system aiming to achieve a fully distributed way of Neural Network training. It follows a "Vires in Numeris" approach, combining the resources of commodity personal computers. It has a full peer-to-peer model of operation; all participating nodes share the exact same privileges and obligations. Another significant feature of LEARNAE is its high degree of fault-tolerance. All training data and metadata are propagated through the network using resilient gossip protocols. This robust approach is essential in environments with unreliable connections and frequently changing set of nodes. It is based on a versatile working scheme and supports different roles, depending on processing power and training data availability of each peer. In this way it allows an expanded application scope, ranging from powerful workstations to online sensors. To maintain a decentralized architecture, all underlying tech should be fully distributed too. LEARNAE's coordinating algorithm is platform-agnostic, but for the purpose of this research two novel projects have been used: (i) IPFS, a decentralized filesystem, as a means to distribute data in a permissionless environment and (ii) IOTA, a decentralized network targeting the world of low energy "Internet of Things" devices. In our previous work, a first approach was attempted on the feasibility of using Distributed Ledger Technology to collaboratively train a Neural Network. Now, our research is extended by applying LEARNAE to a fully deployed computer network and drawing the first experimental results. This article focuses on use cases that require data privacy, thus there is only exchanging of model weights and not training data.

**Keywords:** Decentralized Neural Network Training, Data Privacy, Weight Averaging, Distributed Ledger Technology, IPFS, IOTA.

## 1 Introduction

During the past years Artificial Intelligence (AI) has experienced a significant boost because of exploding Neural Network (NN) advancements. Most approaches are based on a centralized architecture and there is only a minority of attempts that recognize the value of decentralization. In the classic centralized area, there is only a handful of private corporations that can fulfill the infrastructure standards to implement a global service that provides the enormous processing power needed for large-scale deep learning projects. This fact is increasingly raising concerns about privacy issues regarding the training data. There are more than a few cases where sensitive information was leaked, either due to lack of adequate security or due to misuse for financial gains. For moderate-scale projects, institutions can purchase high-end equipment which results to an extremely high setup cost.

Without doubt there is a significant gap: What about open communities of people who own commodity hardware and wish to join forces in order to pursue a common goal? Is there a way to create a network - in a non-binding way for the participants - while fulfilling the need for data privacy? LEARNAE is an attempt to fill this gap.

A purely distributed approach would enable the entity that created the information in the first place, to be the solely owner of the produced data throughout the whole process. While designing a permissionless system – where everyone can join and contribute to consensus mechanism – there are many aspects which can be dealt with in a decentralized way. Each implementation may also have a different level of decentralization, based on its intended use, as described in the following chapter.

Our previous work [1] presented a preliminary evaluation of LEARNAE's algorithm. All tests were contacted in a simulated environment running on a single computer. The virtual network was comprised of 10 workstations, each one running in a custom Docker container. Being a first study of the key characteristics of our algorithm, all sessions had short execution time and used only a fraction of the training dataset. Nevertheless, that first approached was a solid proof-of-concept; it proved that our algorithm, in combination with modern Distributed Ledger Technology (DLT), can be used as a novel type of ecosystem for supporting distributed NN training.

This article elaborates over our previous work in the following ways: (a) It provides experimental evidence on a significantly larger training dataset; (b) experiments are run over a real computer network instead of a virtualized environment; and, (c) the design and implementation of the experiments encompass privacy-oriented features.

The rest of the article is structured as follows: Section 2 presents previous related work, Section 3 presents the theory behind the decentralized concepts utilized by our system, whereas Section 4 presents the LEARNAE architecture. Section 5 presents extensive experimental results to evaluate the whole approach and, finally, Section 6 concludes the article and identifies future research challenges.

## 2 Related work

In most AI/NN cases the proposed solutions rely on a parameter server ("TensorFlow" [2], "Downpour SGD" and "Sandblaster L-BFGS" [3], "Parallel minibatch SGD" [4][5]) and require high performance infrastructure. Decentralized attempts, such as "Elastic Averaging SGD" [6], "HOGWILD!" [7] and "Selective SGD" [8] use local optimization and asynchronous model merging, reducing the communication demand, but the parameter server is still a bandwidth bottleneck which limits scalability. "One-shot averaging" [9][10] uses only one averaging step at the end of the training session and achieves solutions of good quality in many applications. Other distributed deep learning systems ("MXNet" [11], "FireCaffe" [12]) are able to overcome this bottleneck, but for doing so they need low latency networking, which results to very high setup cost and narrows down applicability. "Decentralized Parallel Stochastic Gradient Descent" (D-PSGD) [13][14] has no need for a parameter server. Instead, each worker exchanges their local gradients with its neighbors at every iteration, but it is a synchronous algorithm which requires syncing via a common clock. Unlike D-PSGD, in "Asynchronous Decentralized Parallel Stochastic Gradient Descent" (AD-PSGD) [15] workers don't have to wait for all others, but they communicate in a decentralized fashion. It relies on a ring-based network topology and after each iteration every worker selects a neighbor for averaging, where both workers replace their local models with the averaged one. "Parallel Restarted SGD" (PR-SGD) [16] attempts to reduce the needed communication between the peers. The whole training procedure is divided into epochs of predefined length, during which all workers are executing SGD in parallel. When an epoch ends, all individual solutions are averaged and the produced model is the initial point for the next epoch. "Asynchronous Parallel Stochastic Gradient Descent" (A-PSGD) [17][18][19][20] attempts to mitigate the need for synchronization by allowing workers to use stale weights. There are also other proposals which utilize second-layer methods for reducing the amount of exchanged data; these methods include passing low-bit quantized [21][22][23] or sparsified [24][25][26] gradients to the parameter server. In this way they optimize the network traffic by sacrificing the convergence rate to an acceptable extend.

The method of model averaging has the advantage of reducing privacy and security risks in federated learning [27] scenarios. This is because model averaging only passes deep learning models, which are shown to preserve good differential privacy and does not pass raw data or gradients owned by each individual worker.

Most of the previous approaches attempt to leverage distributed execution in order to decrease training time. It is of great importance to outline that LEARNAE follows a different path, focusing not only on cooperatively achieving improved models, but mainly on boosting

the major aspects of decentralization and democratization, among open and heterogeneous groups of participants who share a common goal. The prioritized features are:

**Peer-to-Peer.** With pure peer-to-peer architecture, all nodes have the same level of access regarding the produced neural models.

**Resilience.** The whole procedure is shielded against node downtime and large-scale network disruptions, having no single point of failure.

**Persistency.** All (meta)data can optionally remain available on the network, as long it is dictated by the participants' policy. This is vital in cases where new nodes may join at any phase of the training.

**Privacy.** Our proposal can be configured to operate in privacy mode, where peers collaborate without sharing their sensitive data.

**Polymorphism.** Participants can choose between four different roles, depending on the availability of processing power and training data.

**Heterogeneity.** Due to the completely asynchronous method, big differences in hardware are mitigated, thus there are no locks.

Table 1 presents a comparison on the above features, taking into consideration other approaches that implement asynchronous data parallelization.

**Table 1** Comparison of previous approaches

| | **Peer-to-peer** | **Resilience** | **Persistency** | **Privacy** | **Polymorphism** | **Heterogeneity** |
|---|---|---|---|---|---|---|
| LEARNAE | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| AD-PSGD | ✓ | ✓ | × | × | × | ✓ |
| Downpour SGD | × | ✓ | × | × | × | ✓ |
| Elastic Averaging SGD | × | × | × | × | × | × |

LEARNAE stretches decentralization and tolerance to maximum. Based on purely distributed peer-to-peer technology, it does not need servers or any kind of strict synchronization. Its indented use cases are environments with commodity-hardware nodes and networking infrastructure with moderate latency and connectivity. Our approach supports versatile acquiring of data from a variety of sources, including lightweight Internet of Things (IoT) devices, and uses novel DLTs as the data diffusion mechanism.

## 3 Underlying Distributed Technology

This section presents the related technological background which forms the basis of the LEARNAE system. The coordinating algorithm is platform-agnostic and can use any method of transferring the related data. For the current implementation, in order to maintain its decentralized nature, two novel technologies were leveraged, IPFS and IOTA.

### 3.1 IPFS

IPFS (InterPlanetary File System) is a distributed filesystem for peer-to-peer networks, where no node has special privileges compared to others. It does not make use of any centralized authority; it supports features like immutability, deduplication, versioning and content-based addressing [28]. IPFS may be seen as an intuitive combination of Git [29], the distributed source code version control system, and BitSwap[1], a novel data replication incentivizing algorithm inspired by BitTorrent [30], the decentralized block exchange protocol.

In IPFS every node is identified by a unique ID, which is the cryptographic hash of a public key. Unlike most filesystems, IPFS is not using an addressing method based on the location the data are stored in; instead, any file can be acquired by just using the cryptographic hash of its contents. Files larger than a specific size are sliced to chunks which are assigned with their corresponding hash. In order to route a request, each IPFS node utilizes a Distributed Hash Table (DHT), a ledger (based on [31][32][33]) which contains pairs of chunk/peer information. For a small chunk, DHT contains the actual block data, while for a larger one it contains the IDs of peers that can provide the specific chunk.

Regarding data exchange strategy, every IPFS node maintains a list of chunks it needs and one of chunks it already has. A major difference compared to BitTorrent's method [34] is that those lists are not limited to a specific torrent or a group of such, but it may include any chunk that has appeared on the network, no matter what file they are part of. Since there will be cases where two peers will not need a chunk stored in each other, or cases where a peer needs nothing, some kind of credit system had to be applied. Each node is incentivized to increase its credit with its peers, because by doing so it also increases the possibility others will help it to acquire the chunks it will need in the future. For a pair of nodes this credit is in fact the balance of verified bytes exchanged between the two.

Since the main concept of IPFS is decentralization, no dedicated data servers are needed to store the files. All nodes participating to the network must provide some local storage (similar to [35]). Every requested block is fetched from another peer and then saved to the local storage. If a peer wishes to permanently retain a file locally, it can "pin" it to avoid garbage collection. Considering all the above, IPFS should be able to be used as resilient distribution method of train-related data, ensuring load balancing between the nodes and a configurable data replication, in order to anticipate downtime on loosely connected topologies.

**PubSub.** IPFS supports a Publish-Subscribe scheme which allows fast messaging. Messages are separated in "topics", so if a peer needs access to a specific topic it subscribes to it and, by doing so, it can listen and broadcast data to that channel. As expected, PubSub has no need for a centralized authority and all its messages propagate using gossip protocol. LEARNAE utilizes IPFS PubSub feature to exchange limited-size training metadata among the peers.

### 3.2 IOTA

The IOTA (meaning "an extremely small amount") network[2] aims to create a decentralized infrastructure for all forms of data exchange in the upcoming IoT era. Although it supports a cryptocurrency token, machine-to-machine micropayments is just a portion of its use cases. It is based on a Directed Acyclic Graph (DAG), which within the IOTA community is referred to as the "Tangle"[3] . In general, IOTA attempts to achieve a consensus via permissionless procedures, using the Tangle as a DLT and a gossip protocol to propagate transactions throughout the network. In order to attach a new transaction, a node has to confirm two previous ones, so each addition contributes to the overall stability and performance of the network [36].

[1] Bitswap homepage, https://github.com/ipfs/specs/tree/master/bitswap

[2] IOTA Foundation homepage, https://www.iota.org

[3] Serguei Popov, "The Tangle", https://iota.org/IOTA_Whitepaper.pdf

**Masked Authenticated Messages.** The IOTA's feature-of-interest for this article is known as "Masked Authenticated Messages" (MAM). It allows lightweight IoT devices to attach zero-valued transactions to the Tangle by just performing a small amount of "Proof of Work" (PoW), which is solving a cryptographic puzzle. PoW is necessary in order to avoid spamming from bad actors. Each of these transactions may include a data portion, making MAM an IoT-oriented way to broadcast data streams that can support both encryption and authentication. MAM messaging is based on a publish-subscribe logic. A data stream is constructed as a singly linked list, where each transaction points to the next one. If someone knows the address (called "root") of a specific transaction, they can read all of the following stream, but they will have no access to previous data. Knowing the first root of a MAM chain grants access to all the messages it contains. A MAM stream may have one of three different accessibility modes: Public, Restricted and Private. In Public mode everyone can view the messages; in Restricted mode only those who know a "sidekey" can read the data; in private mode only the stream owner can access the data, since it requires the seed that created the MAM root sequence.

# 4 Implementation

## 4.1 Design decisions

**Parallelism type.** The first decision made was choosing between model parallelism [37] and data parallelism [38][39]. It is worth to mention that there are related efforts which propose hybrid systems. LEARNAE adopts data parallelism. According to this approach, each worker keeps the entire model locally and processes it using its own subset of the training data.

**Propagation.** After processing on workers, the produced models have to be combined. The two major methods for doing so are weight and update averaging, each of them having its own advantages and drawbacks. LEARNAE adopts weight averaging, thus after the training phase the actual values (not just the updates) of all parameters of the model are – under conditions which are described in a following chapter – averaged with the actual values of the correspondent parameters of a remote worker's model [40].

**Coordination.** Model merging can also be implemented with different levels of decentralization, as seen in Table 2. A parameter server has the task of receiving, combining and redistributing the averaged data. The use of a server in most cases results to an increase in training speed, but it also constitutes a single point of failure and – in large scale networks – a bandwidth bottleneck. This drawback can be mitigated by using more servers which cooperate with each other. For cases where the presence of a server is not feasible or desired, some of the participating peers are assigned special coordinating tasks, while also functioning as all other typical workers. At the edge of this spectrum there are implementations where no node has additional coordination duties, being a fully distributed environment, which is the design decision followed by our proposal.

**Table 2** Different approaches regarding training coordination

| Level of decentralization | Coordination entity |
|---|---|
| None | Parameter server |
| Low | Cluster of parameter servers |
| Medium | Some peers have elevated role |
| High | None |

**Synchronicity.** The training procedure may be synchronous or asynchronous. In synchronous designs the coordinating entity ensures that all results are only combined with others produced at the same training phase. In asynchronous designs there is no such need and the results of a worker can be embedded into the global model under more loose time-based rules. Each approach has its pros and cons. Synchronous training may converge faster since it prevents merging of very different models, but it may have locks from slow peers which can delay the whole process. Asynchronous training achieves higher worker utilization but suffers from gradient staleness, meaning that by the time a slow worker submits its results, they are already out of date compared to the global model. Many strategies have been proposed [41] to mitigate those downsides, resulting to a large number of variants, especially for Asynchronous Stochastic Gradient Decent. LEARNAE adopts an asynchronous design although it contains features which, if used in future implementations, may inject a configurable amount of synchronicity.

**Data privacy.** LEARNAE can operate in environments where participants do not wish to share their sensitive training data. In such cases the messages related to training data are deactivated and all exchanged data are solely models created by nodes after they finish each training cycle. In that way the network leverages the useful information contained in all training data, by averaging the models these data produced. All peers repeatedly average their local weights, so the models they publish are affected by an always-increasing number of remote models created by unknown neighbors. As a result, the changes in weights cannot be linked to their producer, thus there is no indirect leakage. In upcoming versions this protection could be enhanced, by implementing features like source-peer anonymizing and random noise data injection

### 4.2 Architecture

LEARNAE is based on a versatile working scheme and can adapt to different environments. Regarding data, it supports use cases where all training data are poured into the network during the initial phase, before any training. It also supports scenarios where data feeding is a continuous task and the enhancement of the neural model is an always-progressing procedure (online training).
In short, there is no limitation about the time training data may be injected into the network, which is a critical feature when it comes to sensor data streaming from IoT devices.

For operating versatility there are 4 different types of nodes (Table 3). What role a node has depends on whether it has access to training data, as well as on its computational capability. The first three roles in Table 3 require enough computational power to support participation to the IPFS swarm. The fourth role is indented for IoT domain, since data streaming through MAM messages may be broadcasted even by lightweight sensor devices. Fig. 1 demonstrates dataflow between nodes of different roles.

**Table 3** Supported node types and their features

| Node role | Platform | Model training | Data feeding |
|---|---|---|---|
| Full | IPFS + IOTA | Yes | Yes |
| Trainer | IPFS + IOTA | Yes | No |
| Feeder (fat) | IPFS | No | Yes |
| Feeder (thin) | IOTA | No | Yes |

As in all publish-subscribe implementations, the ID of the used channel (IPFS and IOTA) is the connecting link between peers. Knowing this ID allows a peer to participate to the network by listening and broadcasting messages. If needed, encryption and/or

authentication can be easily applied to any stage of the data exchange. Fig. 2 shows the workflow of a node's "Listening thread". There are 3 different types of messages:

***Slice hashlist.*** This message contains the hash of a file that a node made available on IPFS. The file contains a list of hashes of training dataslices that were also made available on IPFS by the same peer.

***Remote model.*** This message contains the hash of a file that a node made available on IPFS. The file contains a model trained on that peer, in HD5 format. Other metadata of the model are also included, like the achieved accuracy and the maturity of the model (the number of training cycles elapsed up to its creation).

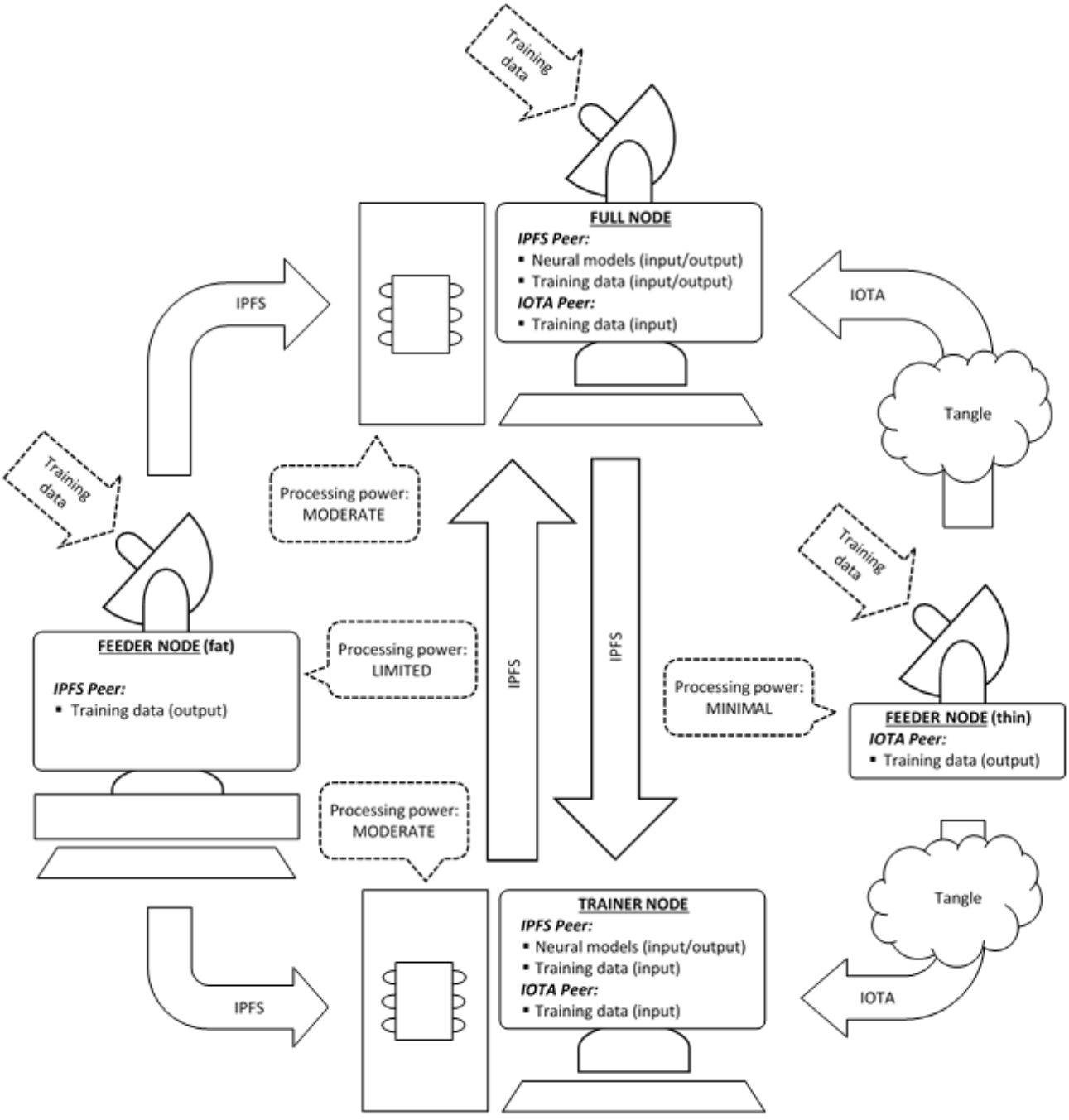


**Fig. 1** Dataflow between nodes of different roles

***Slice use stats.*** This message informs all participating peers that a specific dataslice has been used for training by a peer. This info supports the implementation of "overuse threshold" feature, which can – optionally – define a limit on how many times a dataslice may be used for training on different nodes.

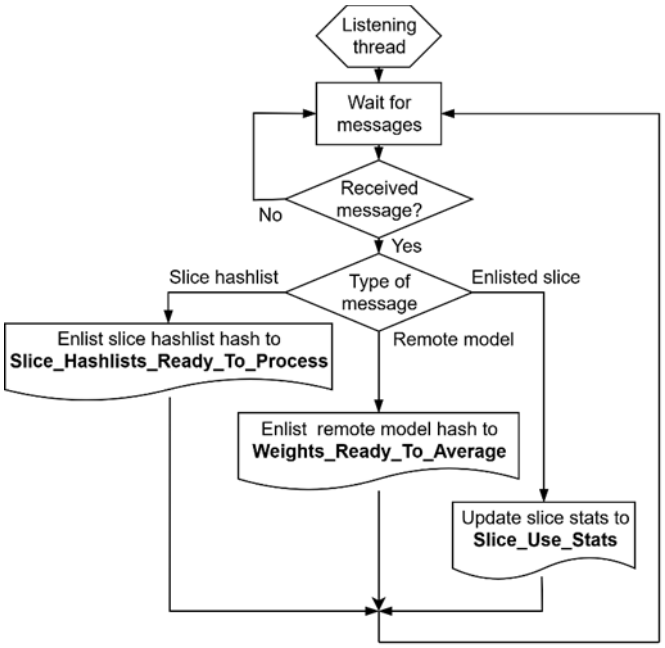


**Fig. 2** Workflow of a node's "Listening thread"

**Table 4** Node tasks

| Task | Description |
|---|---|
| Adding | Add new data to IPFS |
| Pinning | Make remote IPFS data available locally |
| Training | Train local model |
| Averaging | Average local and a remote model |

A peer may perform up to four tasks, as described in Table 4. In order to maximize node utilization, LEARNAE uses a different running thread for each task. All of them can be executed simultaneously, with the exception of the pair Training/Averaging, since both of those need read-write access to local model. Fig. 3 demonstrates the workflow of a full node.

Utilizing Keras framework, a basic Neural Network was used for evaluation, based on a sequential model with 16 hidden dense layers ranging from 30 to 100 neurons with ReLU activation function. The default SGD optimizer was used along with "Binary Cross-entropy" loss function, with a minibatch size of 32 instances.

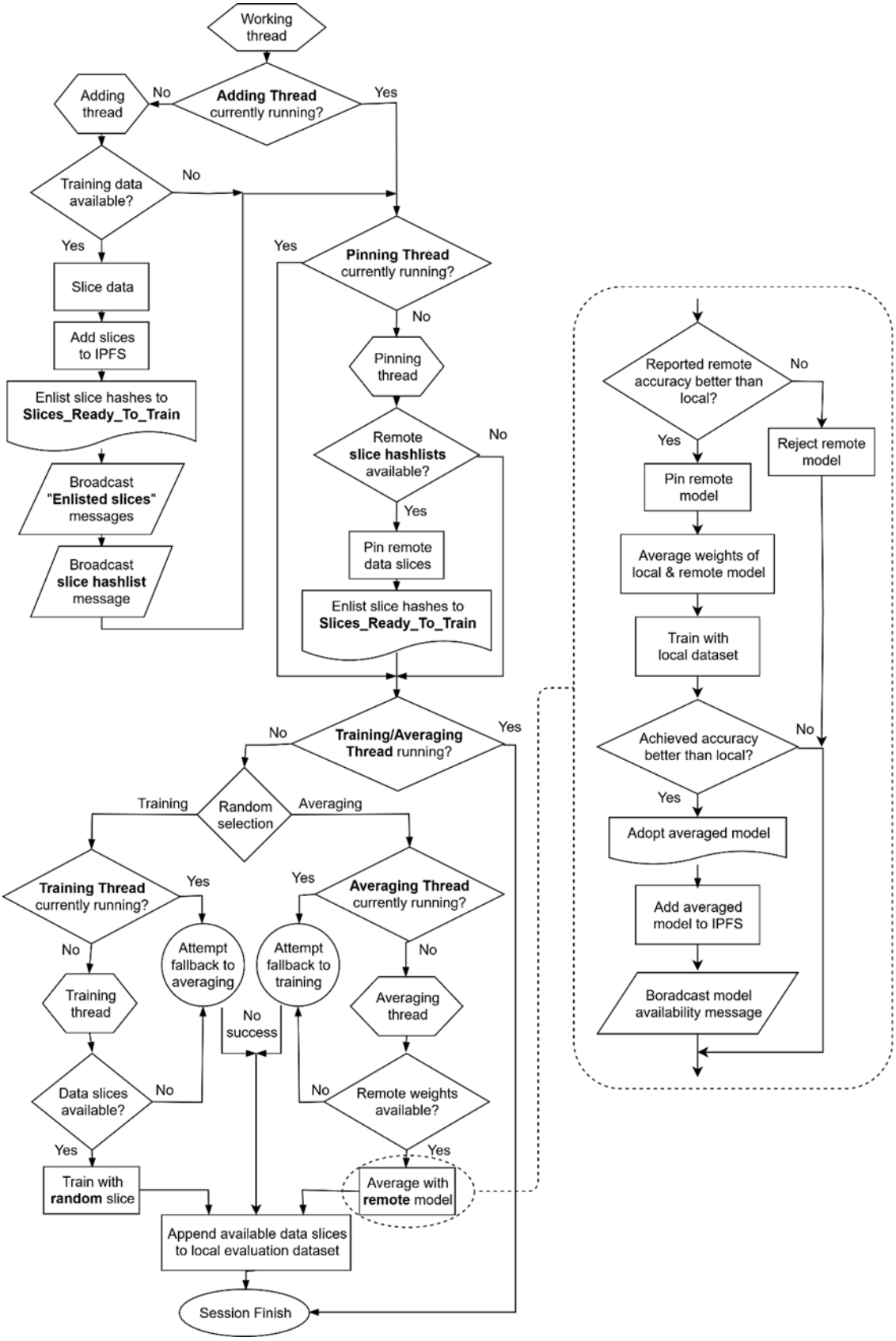


**Fig. 3** Workflow of a node's "Working thread"

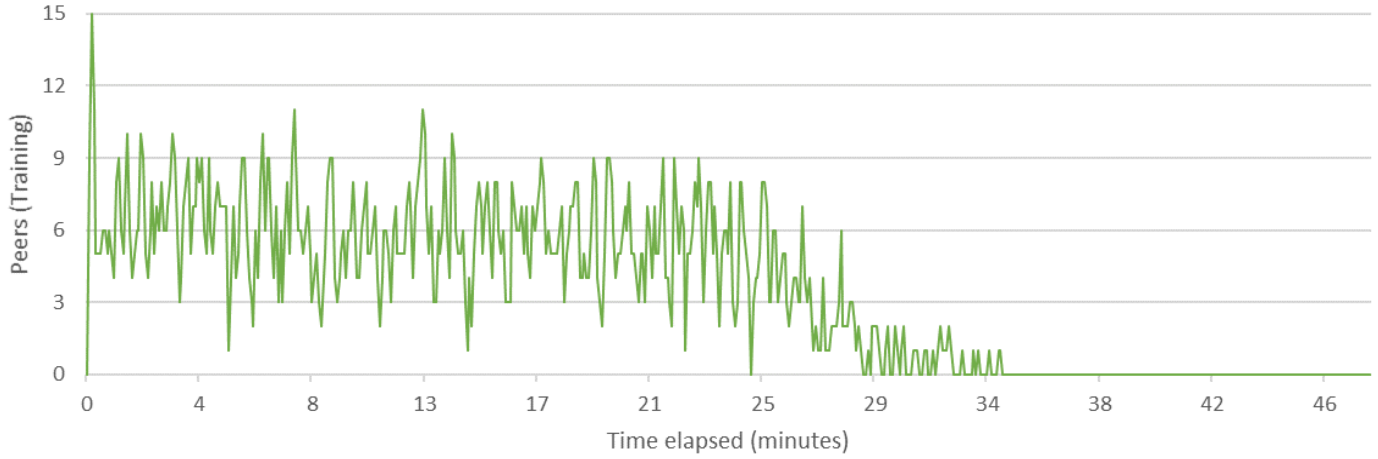


**Fig. 4a** Number of peers in *training* mode
(dataslices of 500 instances)

# 5 Experiments

## 5.1 Current scope

Continuing the work of [1], this article presents the first development of a real LEARNAE network, a typical LAN consisting of 15 commodity PCs. All computers have similar hardware and software specifications and they don't possess any sophisticated GPUs, since we focus on comparing "stand-alone vs collaborative" training and not on actual values. In general, the goal is to research the viability of using modern DLTs as data diffusion mechanism, aiming to gain benefits from this collaboration that were not possible with the stand-alone approach.

Additionally, the advantages of this approach in terms of privacy will be highlighted. The use cases are far from rare: A number of peers wish to collaboratively train a Neural Network. They all agree to participate, provided they will not have to share their sensitive data. We suggest a fault-tolerant way to accomplish this, by leveraging distributed protocols to exchange only models and not training data. The transmitted models are combined (averaged) by LEARNAE's coordinating algorithm, resulting to an improved model for all

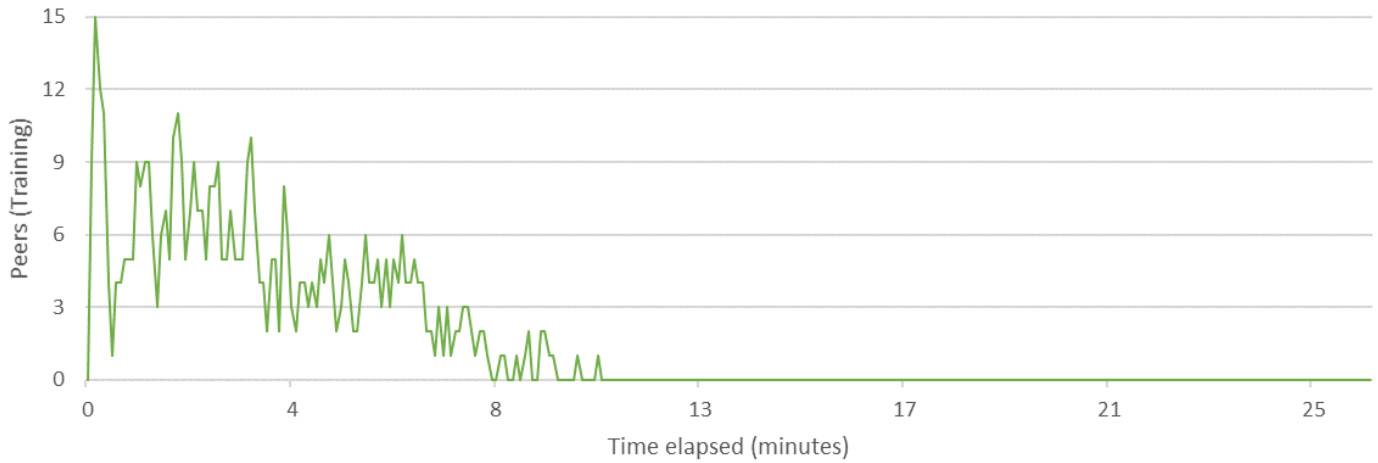


**Fig. 4b** Number of peers in *training* mode
(dataslices of 2000 instances)

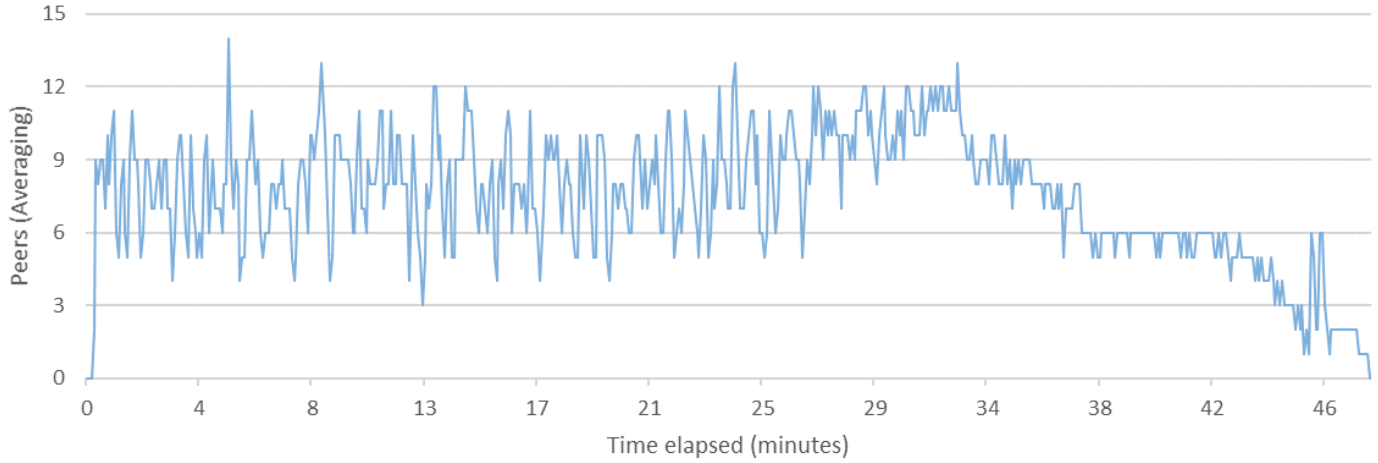


**Fig. 5a** Number of peers in *averaging* mode
(dataslices of 500 instances)

peers. The assumption that has to be made is that training data originating from different participants satisfy a level of uniformity, a fact that is valid in the vast majority of the use cases targeted by LEARNAE. Scenarios which also support sharing of training data will be thoroughly studied in a future work.

An experinental comparison to other systems is not possible, since (as also shown in Table 1) other systems have different characteristics and focus than LEARNAE.

### 5.2 Collaborative training sessions

The selected dataset (HEPMASS[4]) contains 10 million instances of 28 attributes. Our system was tested using dataslice sizes of 500, 1000, 2000, 5000, 10000 and 50000 instances, in order to demonstrate the impact of dataslice size on execution time, amount of exchanged model data and achieved accuracy. All training sessions were executed using Python and Keras framework. After training with one dataslice, each node notifies all others that the resulting model is available for sharing and weight averaging.

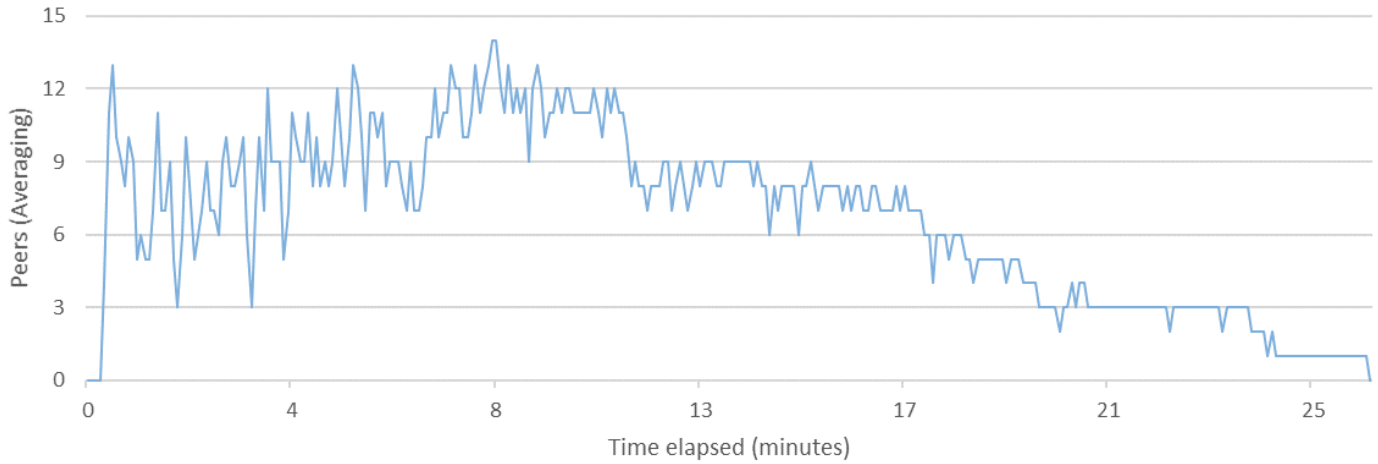


**Fig. 5b** Number of peers in *averaging* mode
(dataslices of 2000 instances)

[4] HEPMASS Dataset homepage, http://archive.ics.uci.edu/ml/datasets/hepmass

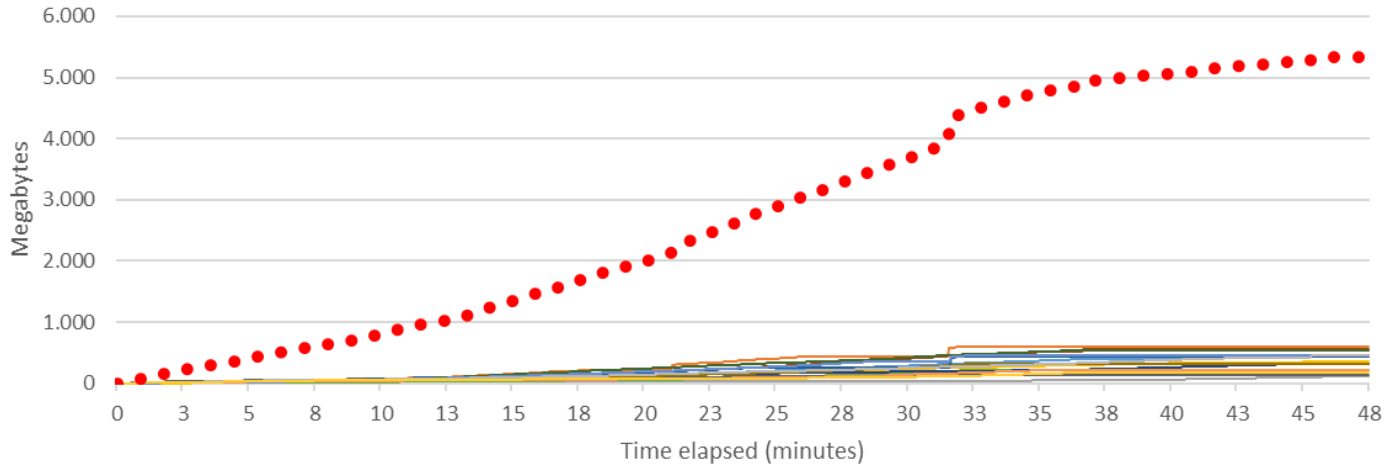


**Fig. 6a** Data sent – Peers & Total
(dataslices of 500 instances)

### 5.3 Sample graphs

In the rest of this section several key features of LEARNAE are presented, using sample graphs of sessions executed with dataslices of 500 and 2000 instances. As expected, sessions based on 500-instance dataslices require more time to complete, a fact which stands for all following figures. That is because a peer informs others for the availability of a new model right after it consumes a dataslice. Therefore, increasing the number of dataslices results to a larger number of available models on the network, thus increased work in queue for all nodes.

**Number of peers in *training* mode.** To be able to average their local model, all peers initially have to perform at least 1 training session (high values at graph start - Fig. 4a, 4b). As time passes, peer after peer consumes all local data and training phase ends. The required time depends on node's processing power and the random rate at which it selects training over averaging during each work cycle.

As stated above, 500-instance dataslices require more time to converge, due to the increased processing overhead caused by the higher number of models propagating the network.

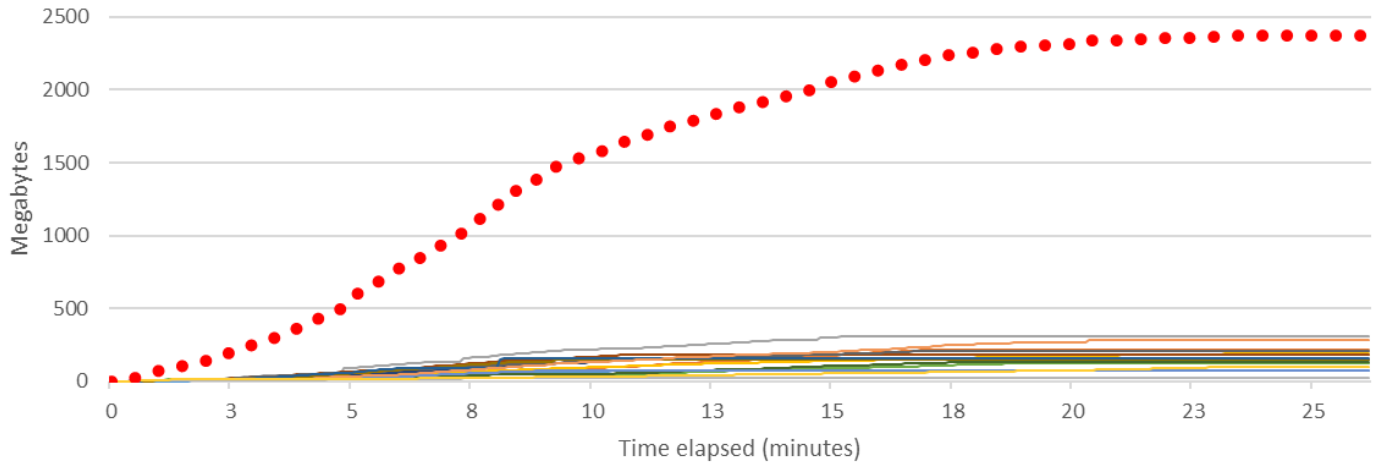


**Fig. 6b** Data sent – Peers & Total
(dataslices of 2000 instances)

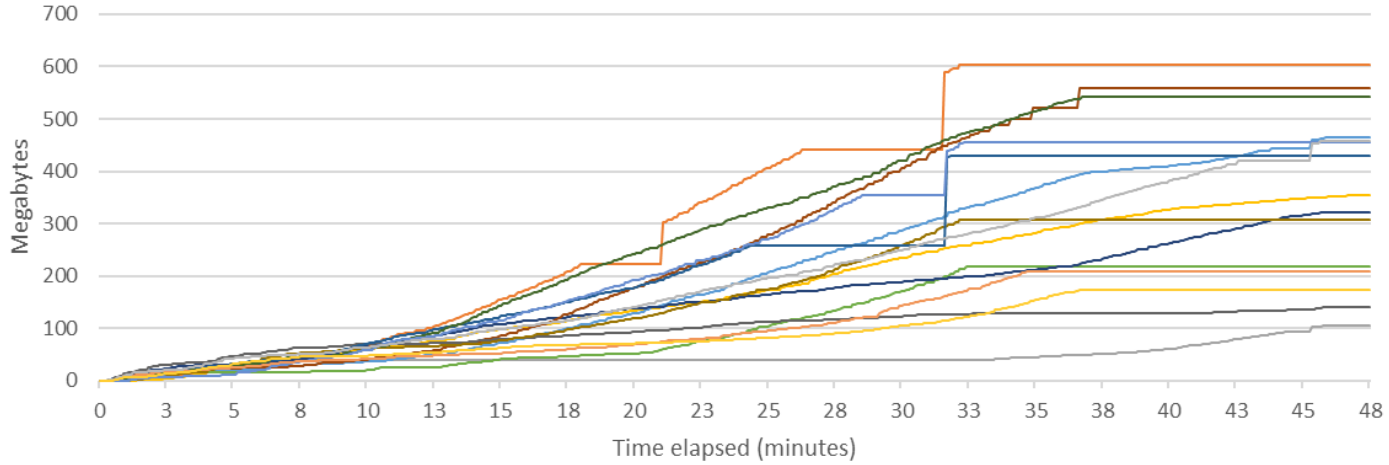


**Fig. 7a** Data sent – Peers
(dataslices of 500 instances)

**Number of peers in *averaging* mode.** Although averaging can start at the same phase as training, the former may continue even after all available training data are used on all peers. In averaging phase remote models are inspected via their broadcasted metadata and, if they fulfill specific requirements (described in the next paragraph), their chunks are fetched from multiple remote nodes and they are locally checked to decide whether they can contribute to enhancement of local model via weight averaging.

The method a peer uses to check a remote model in order to decide whether it is a candidate for averaging, is a point of high interest in our research, since an optimal way of identifying the most useful models will result to improved overall efficiency. Currently, a remote model has to pass two different checks. If its reported accuracy (contained in the metadata of its announcement message) is better than the local one, an averaging will be tried out. If, by using the locally stored evaluation dataset, the averaged model improves peer's accuracy, then this model is adopted by the peer.

As seen in Fig. 5a, 5b, peers eventually reach a high accuracy value, so the possibility of a successful averaging declines. This fact results to fewer new models broadcasted to network; thus, the peers have no more work to execute and the whole process achieves a final convergence.

A node cannot perform training and averaging at the same time, so these four graphs are complementary.

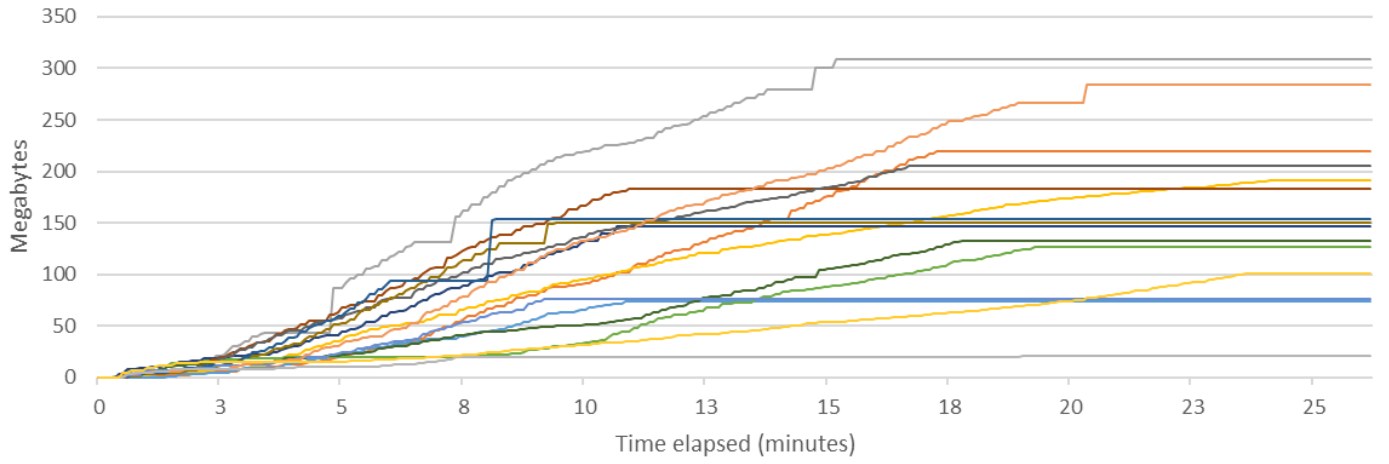


**Fig. 7b** Data sent – Peers
(dataslices of 2000 instances)

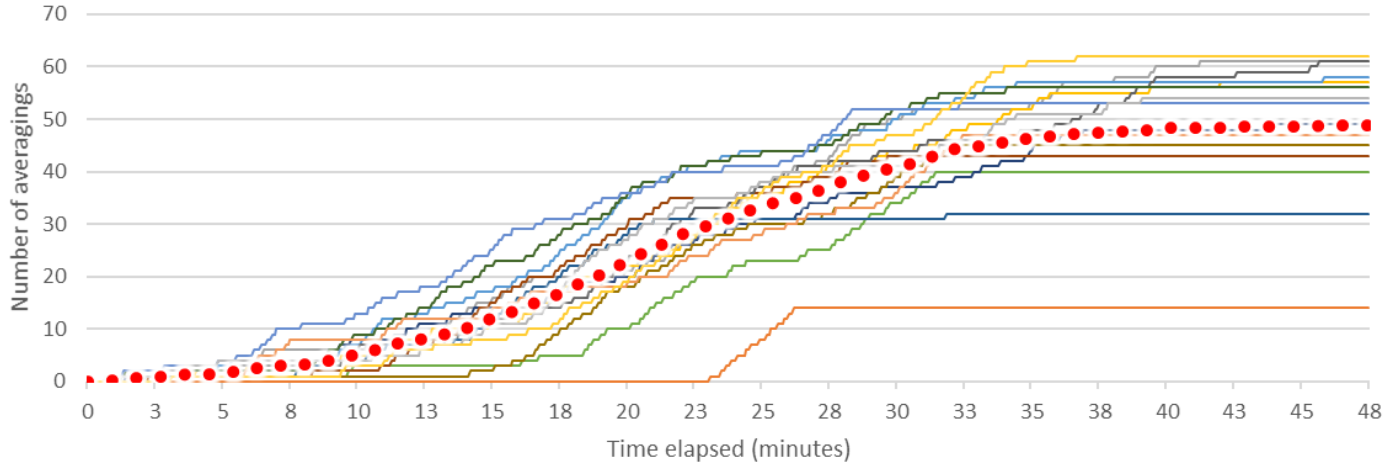


**Fig. 8a** Successful average operations
(dataslices of 500 instances)

**Data sent.** Figures 6a and 6b show the total amount of data sent by all nodes. All information exchange is carried out by IPFS, the distributed nature of which has a big impact on data availability and load balancing. When a model is becoming available to the network, it is split into multiple chunks. So, when a peer asks for this slice, it may receive the correspondent chunks from different neighbors. As expected, smaller size of dataslices results to higher network utilization, since there is a larger number of models announced to the participants, more averaging attempts, thus higher data transfer. Table 5 shows the exact values for the two sessions.

**Table 5** Data sent – Peers & Total

| **Dataslice size** ***(Instances)*** | **Total data sent** *(Gigabytes)* |
|---|---|
| 500 | 5.342 |
| 2000 | 2.375 |

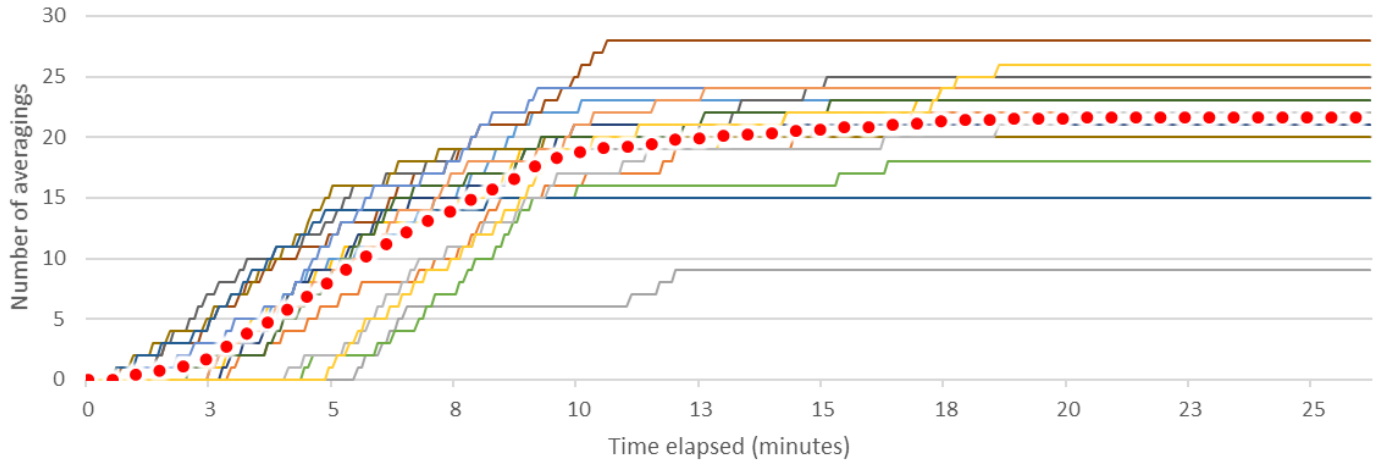


**Fig. 8b** Successful average operations
(dataslices of 2000 instances)

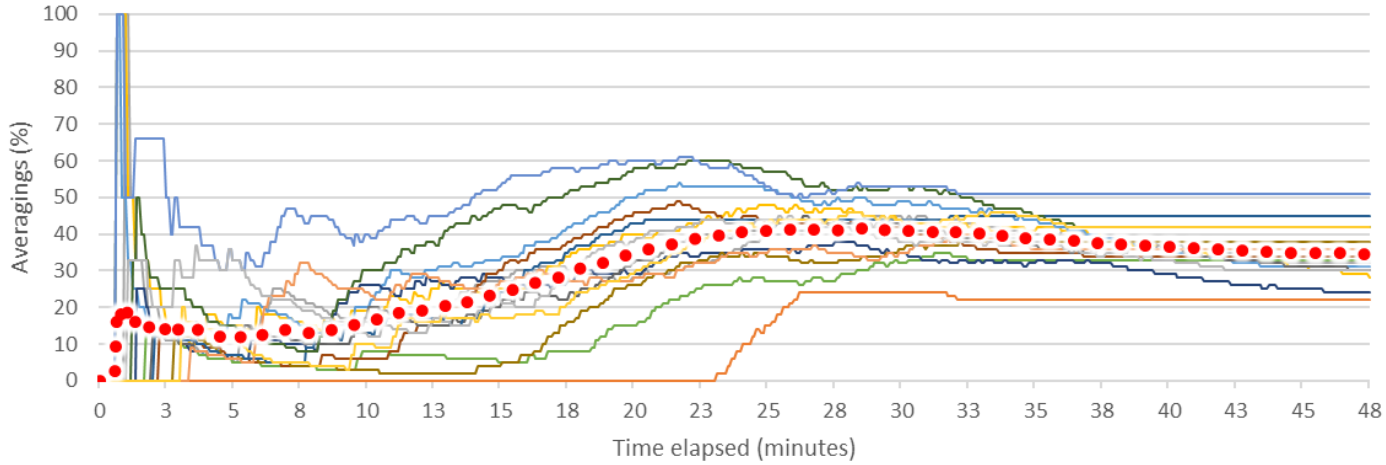


**Fig. 9a** Successful average operations %
(dataslices of 500 instances)

The training data possessed by peers will never be of precisely equivalent worth, so each node will initially achieve different accuracy. As seen in Fig. 7a and 7b, peers that reach high accuracy during the early stages will be asked more frequently to send their local models to others.

**Successful averagings.** When a peer trains a new model, it makes it available to the network and announces a packet of model's metadata. All participating nodes record these messages and use them in order to decide whether a remote model should be tried out to enhance their local one. If a remote model meets the requirements described previously, the node fetches it and attempts to average its own weights. This procedure may or may not produce a local model which is better than the previous one. If it does, that averaging is considered a successful one and the peer adopts the produced model. As expected, for smaller dataslices the number of successful averagings is higher, since the total number of attempts is higher too (Fig. 8a, 8b).

Fig. 9a, 9b show the typical curve of averaging success rate. At start there is a notable accuracy discrepancy among the peers, because the quality of training data on each of them may vary significantly. Initially there is a spike in the graph, because less accurate nodes leverage the insight provided to them by their more accurate neighbors. When the accuracy differences are minimized, the

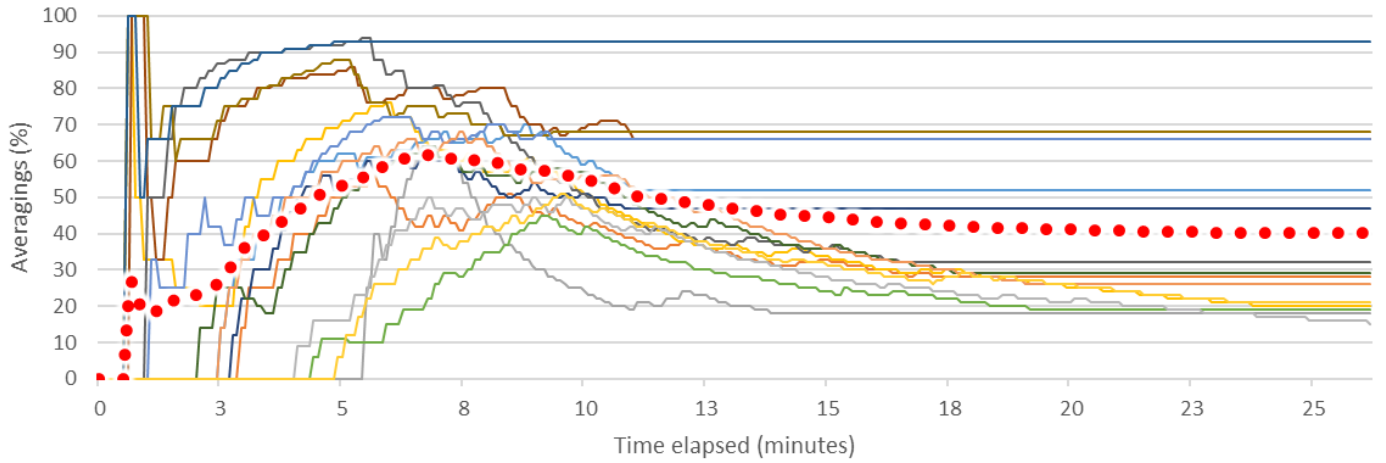


**Fig. 9b** Successful average operations %
(dataslices of 2000 instances)

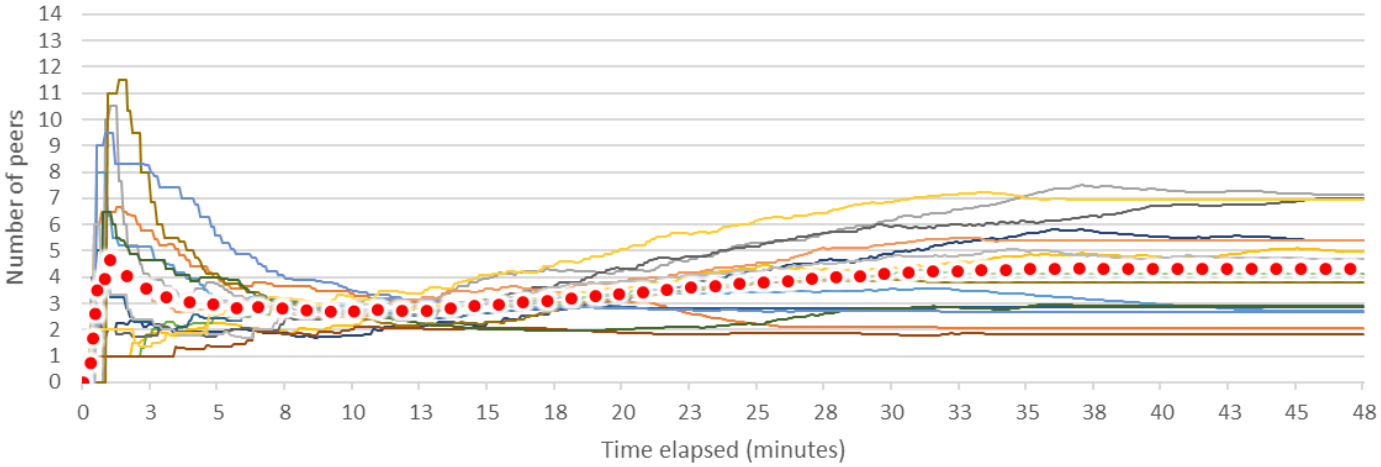


**Fig. 10a** Resilience

(dataslices of 500 instances)

successful averagings slightly decline for a small period of time. Then, the network starts to benefit from the newly consumed data and from the improved models which are created and shared during the training phase. The successful percentage is increasing until training phase ends. Right after that, averaging-only phase takes over and all participants attempt to further improve their models by utilizing the latest combined models that are made available on the network.

Finally, peers achieve a maximum level of convergence, all attempts cease and the successful averaging percentage is stabilized at a terminal value.

**Resilience.** Resilience is defined as the number of different remote peers that, at a given time, can provide a requested chunk of either a model or training dataslice. This article is focused on privacy-enabled configurations, so all network data are related to neural models.

The initial spike (Fig. 10a, 10b) is because at first all nodes have to perform a training phase, the models of which are announced and rapidly fetched by other peers. Averagings increase the number of broadcasted models so there is a temporary decrease in data availability. Finally, as averagings decline, new model production is decreased, thus the resilience value is stabilized.

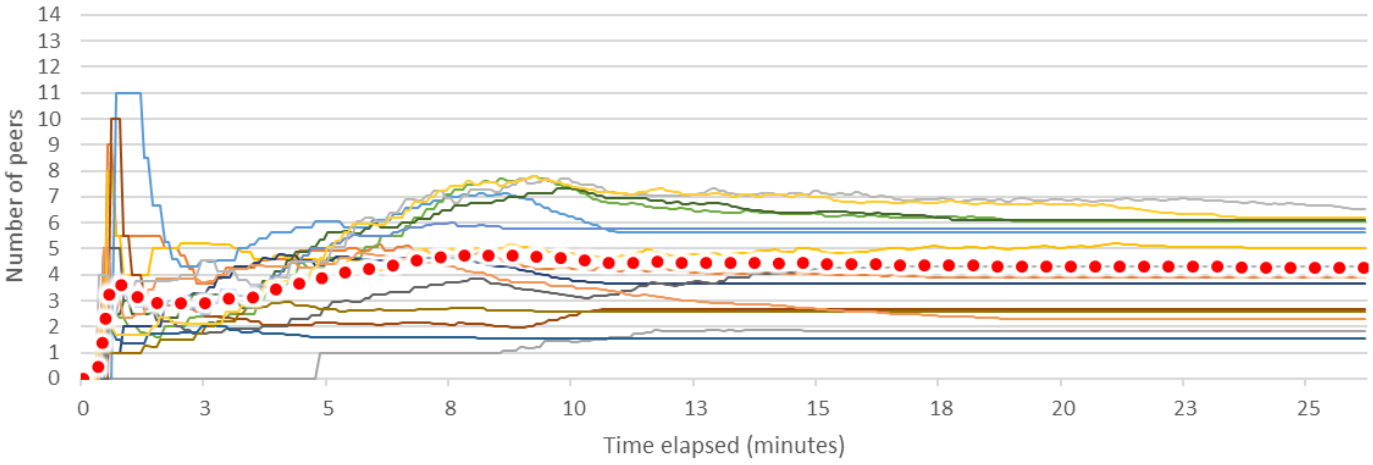


**Fig. 10b** Resilience

(dataslices of 2000 instances)

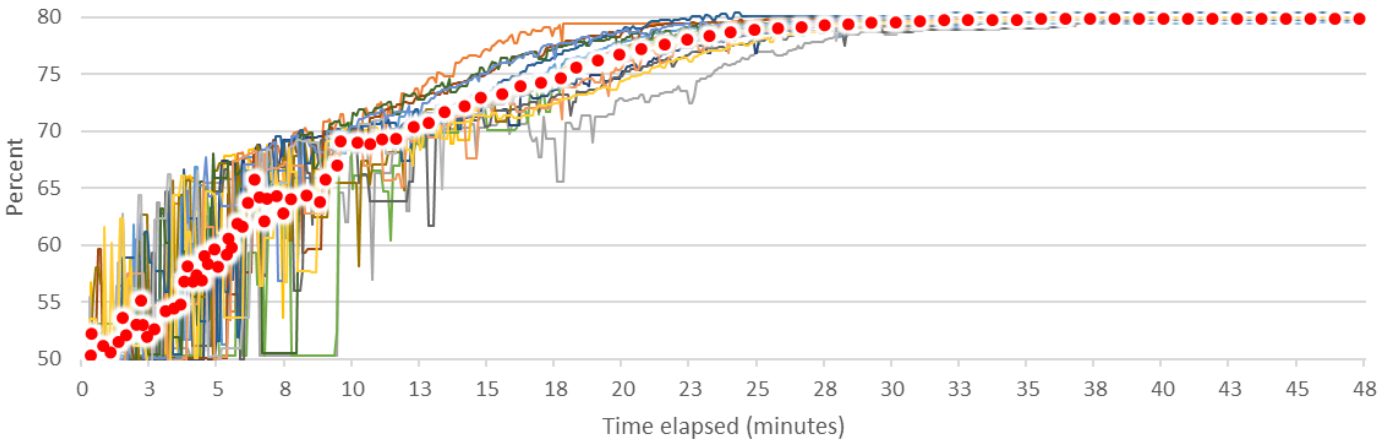


**Fig. 11a** Overall network accuracy
(dataslices of 500 instances)

On both configurations (50000-instance training data, 500 and 2000-instance dataslices) the terminal value of resilience was approximately 4.3, meaning that every chunk requested by any peer, was stored on average in 4.3 other peers. Fig. 12 shows the chunk availability dispersion (orange dots) and resilience value (gray area) of a random peer participating in a training session of 15 nodes using 2000-instance dataslices.

Resilience is a major aspect of this proposal, since one of LEARNAE's main goals is fault-tolerance and the ability to continue the collaborative training even after a major network disruption; a value of 4.3 resilience means that the procedure could continue with no loss, even if a significant part of the nodes was disconnected.

**Overall network accuracy.** As a metric for the achieved accuracy of the network, the mean value of the final accuracy of all participating nodes was used (Fig. 11a, 11b). Fig. 13 demonstrates the findings on how the selection of dataslice size affects the overall accuracy. From a theoretical point of view, during a LEARNAE collaborative training session there are two important opposing factors. When larger dataslices are selected, the portion of pure training in the whole procedure is increased. This results to faster training and more accurate models. But at the same time, larger dataslices have the drawback of reducing the possibility of successful averagings and the benefits they bring on stochasticity level and thus overall accuracy. As it can be seen in Fig. 13, there is a "sweet spot" where this tradeoff produces optimal neural models. In present configuration this was achieved for dataslices of 2000 instances.

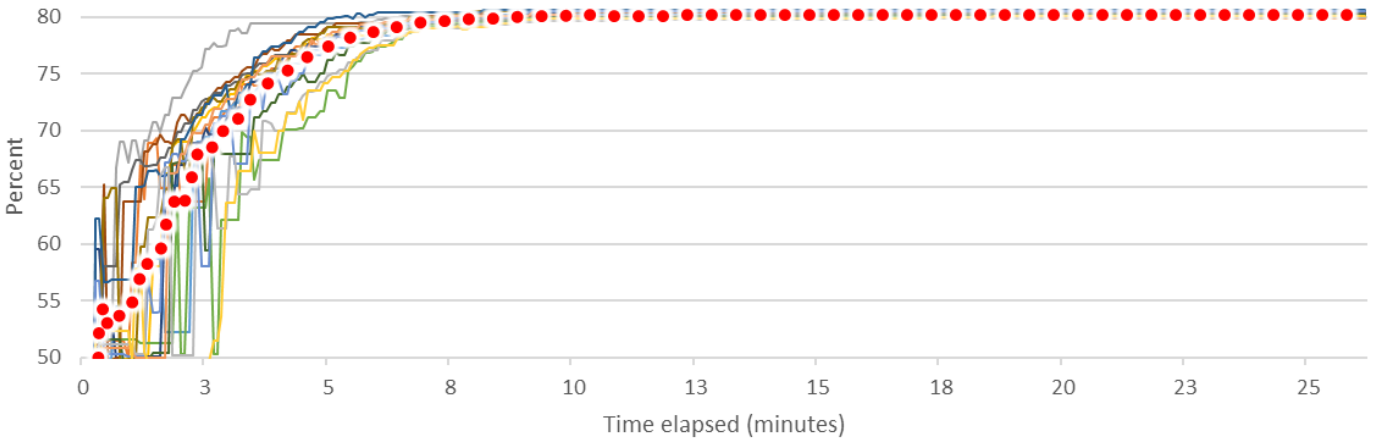


**Fig. 11b** Overall network accuracy
(dataslices of 2000 instances)

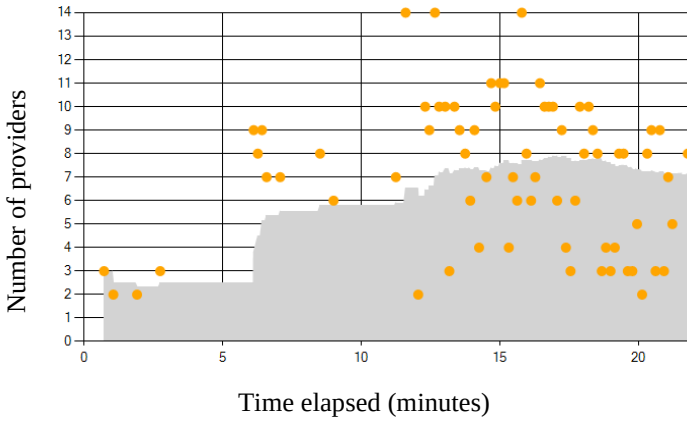


**Fig. 12** Sample snapshot of availability dispersion
(random peer)

The same figure presents the benefits of LEARNAE's coordinating algorithm, compared to "stand-alone" scenario in which all peers train their model using their own data, without communicating with others. The accuracy gain is maximized for the "sweet spot" (green diamond). Table 6 contains the actual numeric data. At the right edge of the graph, where all data are treated as just one dataslice, both methods give the same result because there can be no successful averaging, so the LEARNAE network is degenerated into a number of isolated workstations.

**Fig. 13** Collaborative overall network accuracy per dataslice size

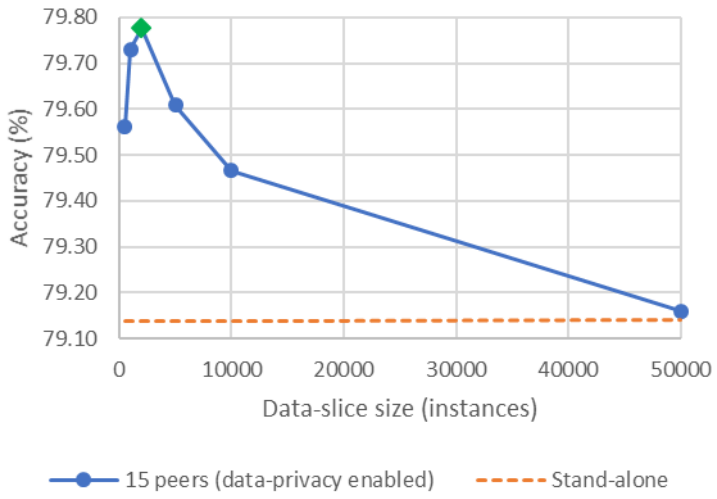

**Table 6** Collaborative overall network accuracy per dataslice size

| **Dataslice size** *(Instances)* | **15 peers with data-privacy enabled** *(%)* | **Stand-alone** *(%)* |
|---|---|---|
| 500 | 79.56 | 79.14 |
| 1000 | 79.73 | |
| 2000 | 79.78 | |
| 5000 | 79.61 | |
| 10000 | 79.47 | |
| 50000 | 79.16 | |

The important experimental conclusion is that it is possible for a number of participants (in this case 15) to co-exploit their sensitive training data without exposing them to others, in order to collaboratively achieve improved neural models. The gain in accuracy is expected to be positively affected when increasing the number of peers.

## 6 Conclusions and Future Work

Our previous work [1] was a first approach on leveraging Distributed Ledger Technology for purely decentralized NN training. Being a proof-of-concept, it had a limited applicability and it was tested solely on a single machine using virtualization containers.

In this work our research was expanded and applied to realistic conditions. The coordination algorithm was tested on a local network of PCs with moderate hardware specifications. Our methodology followed an exclusively comparative logic, since our concern was not actual performance, but possible benefits originating from this decentralized collaboration compared to single node training.

LEARNAE was evaluated with many different configurations, in order to determine how each parameter affects the final outcome. Each one of these configurations was repeatedly executed to ensure that every measurement was sufficiently consistent. The experiments showed that our proposal can offer tangible gain to collaborating peers, without compromising data privacy. Finally, this study pinpointed the optimal set of parameters for a maximized benefit regarding the achieved model accuracy.

Although our tests were conducted with merely 15 workstations, they showed a significant improvement compared to nodes working in isolation. Based on the knowledge acquired from these experiments, it is strongly concluded that this gain should be positively affected by increasing the total number of participating peers.

Many interesting questions remain to be addressed: What is the exact way such a system scale? What are the specifics of the performance/resilience tradeoff? What is the precise overhead of IPFS duplication in configurations that also support the sharing of training data? What will be the impact of including IoT nodes in a LEARNAE network and what is the maturity level of novel distributed concepts like the IOTA Tangle? All these questions will be the subject of our future work.

## 7 Acknowledgements

This research is funded by the University of Macedonia Research Committee as part of the “Principal Research 2019” funding program.

## 8 Conflict of Interest

The authors declare that they have no conflict of interest.

## 9 References

1. S. Nikolaidis and I. Refanidis, “Learnae: Distributed and Resilient Deep Neural Network Training for Heterogeneous Peer to Peer Topologies”, in Engineering Applications of Neural Networks, 2019, pp. 286-298.

2. M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng, “TensorFlow: A System for Large-Scale Machine Learning”, in Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation, pp. 265–283, 2016.

3. J. Dean, G. S. Corrado, R. Monga, K. Chen, M. Devin, Q. V. Le, M. Mao, M. Razato, A. Senior, P. Tucker, K. Yang, and A. Y. Ng, “Large Scale Distributed Deep Networks”, in Advances in Neural Information Processing Systems, 2012, pp. 1223–1231.

4. O. Dekel, R. Gilad-Bachrach, O. Shamir and L. Xiao, “Optimal distributed online prediction using mini-batches” in Journal of Machine Learning Research, 2012, pp. 165–202.

5. M. Li, D. G. Andersen, A. J. Smola and K. Yu, “Communication efficient distributed machine learning with the parameter server” in Advances in Neural Information Processing Systems (NIPS), 2014.

6. S. Zhang, A. Choromanska, and Y. LeCun, “Deep learning with Elastic Averaging SGD”, in Advances in Neural Information Processing Systems, pp. 685–693, 2015.

7. F. Niu, B. Recht, C. Re and S. J. Wright, “HOGWILD!: A Lock-Free Approach to Parallelizing Stochastic Gradient Descent”, arXiv:1106.5730v2, 2011.

8. R. Shokri and V. Shmatikov: “Privacy-Preserving Deep Learning”, in Proceedings of the 22nd ACM SIGSAC, 2015, pp. 1310-1321.

9. M. Zinkevich, M. Weimer, L. Li and A. J. Smola, “Parallelized stochastic gradient descent”, in Advances in Neural Information Processing Systems (NIPS), 2010.

10. R. McDonald, K. Hall and G. Mann, “Distributed training strategies for the structured perceptron”, in Proceedings of North American Chapter of the Association for Computational Linguistics (NAACL), 2010.

11. T. Chen, M. Li, Y. Li, M. Lin, N. Wang, M. Wang, T. Xiao, B. Xu, C. Zhang and Z. Zhang, “MXNet: A Flexible and Efficient Machine Learning Library for Heterogeneous Distributed Systems”, in Proceedings of LearningSys, 2015.

12. F. N. Iandola, K. Ashraf, M. W. Moskewicz, and K. Keutzer, “FireCaffe: near-linear acceleration of deep neural network training on compute clusters”, 2015.

13. 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 Advances in Neural Information Processing Systems (NIPS), 2017.

14. Z. Jiang, A. Balu, C. Hegde and S. Sarkar, “Collaborative deep learning in fixed topology networks” in Advances in Neural Information Processing Systems (NIPS), 2017.

15. X. Lian, W. Zhang, C. Zhang and J. Liu, “Asynchronous decentralized parallel stochastic gradient descent” in Proceedings of International Conference on Machine Learning (ICML), 2018.

16. H. Yu, S. Yang, S. Zhu, “Parallel Restarted SGD with Faster Convergence and Less Communication: Demystifying Why Model Averaging Works for Deep Learning”, arXiv:1807.06629, 2018.

17. A. Agarwal and J. C. Duchi, “Distributed delayed stochastic optimization” in NIPS, 2011.

18. H. R. Feyzmahdavian, A. Aytekin, and M. Johansson, “An asynchronous mini-batch algorithm for regularized stochastic optimization”, IEEE Transactions on Automatic Control, 2016.

19. T. Paine, H. Jin, J. Yang, Z. Lin and T. Huang, “Gpu asynchronous stochastic gradient descent to speed up neural network training”, arXiv:1312.6186, 2013.

20. B. Recht, C. Re, S. Wright and F. Niu, “Hogwild: A lock-free approach to parallelizing stochastic gradient descent”, in Advances in neural information processing systems, 2011.

21. 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 Annual Conference of the International Speech Communication Association (INTERSPEECH), 2014.

22. D. Alistarh, D. Grubic, J. Li, R. Tomioka and M. Vojnovic, “QSGD: Communication-efficient SGD via gradient quantization and encoding” in Advances in Neural Information Processing Systems (NIPS), 2017.

23. W. Wen, C. Xu, F. Yan, C. Wu, Y. Wang, Y. Chen and H. Li, “Terngrad: Ternary gradients to reduce communication in distributed deep learning”, in Advances in Neural Information Processing Systems (NIPS), 2017.

24. N. Strom, “Scalable distributed DNN training using commodity GPU cloud computing”, in Annual Conference of the International Speech Communication Association (INTERSPEECH), 2015.

25. N. Dryden, T. Moon, S. A. Jacobs and B. Van Essen, “Communication quantization for data-parallel training of deep neural networks”, in Workshop on Machine Learning in HPC Environments (MLHPC), 2016.

26. A. F. Aji and K. Heafield, “Sparse communication for distributed gradient descent”, in Conference on Empirical Methods in Natural Language Processing (EMNLP), 2017.

27. H. B. McMahan, E. Moore, D. Ramage, S. Hampson, et al, “Communication-efficient learning of deep networks from decentralized data”, in International Conference on Artificial Intelligence and Statistics (AISTATS), 2017.

28. J. Benet, “IPFS - Content Addressed, Versioned, P2P File System”, arXiv:1407.3561, 2014.

29. A. J. Mashtizadeh, A. Bittau, Y. F. Huang, and D. Mazieres, “Replication, history, and grafting in the ori file system”, in Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles, pages 151–166. ACM, 2013.

30. B. Cohen, “Incentives build robustness in bittorrent”, in Workshop on Economics of Peer-to-Peer systems, volume 6, pages 68–72, 2003.

31. I. Baumgart and S. Mies, “S/kademlia: A practicable approach towards secure key based routing”, in Parallel and Distributed Systems International Conference, 2007.

32. M. J. Freedman, E. Freudenthal, and D. Mazieres, “Democratizing content publication with coral”, in NSDI, volume 4, pages 18–18, 2004.

33. L. Wang and J. Kangasharju, “Measuring large-scale distributed systems: case of bittorrent mainline dht”, in 2013 IEEE Thirteenth International Conference, pages 1–10. IEEE, 2013.

34. D. Levin, K. LaCurts, N. Spring, and B. Bhattacharjee. “Bittorrent is an auction: analyzing and improving bittorrent’s incentives”, in ACM SIGCOMM Computer Communication Review, volume 38, pages 243–254. ACM, 2008.

35. J. Dean and S. Ghemawat, “leveldb–a fast and lightweight key/value database library by google”, 2011.

36. S. Popov, O. Saa, P. Finardi, “Equilibria in the Tangle”, arXiv:1712.05385, 2018.

37. A. Coates, Brody Huval, Tao Wang, David Wu, Bryan Catanzaro, and Ng Andrew, “Deep learning with cots hpc systems”, in Proceedings of The 30th International Conference on Machine Learning, 2013, pp. 1337–1345.

38. X. Zhang, J. Trmal, D. Povey, and S. Khudanpur, “Improving deep neural network acoustic models using generalized maxout networks” in Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference, 2014.

39. Y. Miao, H. Zhang, and F. Metze, “Distributed learning of multilingual dnn feature extractors using gpus”, 2014.

40. D. Povey, X. Zhang, and S. Khudanpur, “Parallel training of deep neural networks with natural gradient and parameter averaging”, 2014.

41. 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 Fifteenth Annual Conference of the International Speech Communication Association, 2014.