Using Distributed Ledger Technology to Democratize Neural Network Training

Spyridon Nikolaidis and Ioannis Refanidis
University of Macedonia, Department of Applied Informatics
Thessaloniki 54636, GREECE
E-mail: sp.nikola@uom.edu.gr, ORCID: https://orcid.org/0000-0001-6450-1005
E-mail: yrefanid@uom.edu.gr, ORCID: https://orcid.org/0000-0003-4697-4751

Postprint of the article published by Applied Intelligence Journal, Springer
https://link.springer.com/article/10.1007/s10489-021-02340-3

Abstract

Artificial Intelligence has regained research interest, primarily because of big data. Internet expansion, social networks and online sensors led to the generation of an enormous amount of information daily. This unprecedented data availability boosted Machine Learning. A research area that has greatly benefited from this fact is Deep Neural Networks. Nowadays many use cases require huge models with millions of parameters and big data are proven to be essential to their proper training. The scientific community has proposed several methods to generate more accurate models. Usually, these methods need high performance infrastructure, which limits their applicability to large organizations and institutions that have the required funds. Another source of concern is privacy; anyone using the leased processing power of a remote data center, must trust another entity with their data. Unfortunately, in many cases sensitive data were leaked, either for financial exploitation or due to security issues. However, there is a lack of research studies when it comes to open communities of individuals with commodity hardware, who wish to join forces in a way that is non-binding and without the need for a central authority. Our work on LEARNAE attempts to fill this gap, by creating a way of providing training in Artificial Neural Networks, featuring decentralization, data ownership and fault tolerance. This article adds some important pieces to the puzzle: It studies the resilience of LEARNAE when dealing with network disruptions and proposes a novel way of embedding low-energy sensors that reside in the Internet of Things domain, retaining at the same time the established distributed philosophy.

Keywords: Decentralized Neural Network Training; Data Ownership; Internet of Things; Distributed Ledger Technology; IPFS; IOTA

Funding
This research is not funded by any source.

Conflicts of interest/Competing interests
The authors declare that they have no conflict of interest.

Availability of data and material
For the experiments the following publicly available dataset was used: HEPMASS[1], dataset for training systems on exotic particle detection.

Code availability
Code and deployment instructions are available upon request by the authors.

[1] http://archive.ics.uci.edu/ml/datasets/hepmass

1 Introduction

Modern Deep Neural Networks (DNN) keep growing. If researchers or enthusiasts decide to practice novel ideas on DNN, they will most likely reach a critical point. To go beyond that point, they will require access to enormous amounts of training data and processing power. But both assets are usually available only in corporate data centers and large institutions. As a result, in most of the cases individual practitioners are excluded from this research domain. In many cases, data centers belong to multinational corporate entities. Unfortunately, more than a few times in the past, private-sector corporations offered inadequate privacy, either because of flawed security or due to intentional sharing of private data with third parties. Our proposal, LEARNAE [1][2] introduces a novel way to utilize modern distributed technology to address these significant problems. It creates a purely decentralized ecosystem, where individual Machine Learning (ML) researchers can collaborate with equitable roles and full access to results, without the need of expensive equipment. At the same time, they have the option to retain the ownership of their sensitive data, taking advantage of modern permissionless networks.

When dealing with Artificial Neural Network (ANN) training, there are many approaches that claim to be decentralized. But after a close study, one can realize that the concept of decentralization itself can reach many different levels. Scaling from low to high degree of decentralization, the literature contains solutions with a parameter server, a cluster of parameter servers, peers with an elevated role and, finally, pure peer-to-peer topologies. Our proposal is based on the last scheme; thus, all participating nodes have the same set of rights and none of them is essential to the training process.

LEARNAE utilizes novel Distributed Ledger Technology (DLT) as data diffusion mechanism. The coordinating algorithm is platform-agnostic; the current implementation adopts two novel technologies: (a) IPFS [3], a decentralized filesystem and (b) IOTA [4], a network infrastructure targeting the world of Internet of Things (IoT). Such a choice ensures high resilience, since all information is propagated using gossip[2] protocols, leaving no single point of failure. LEARNAE uses data parallelism [5][6], according to which each worker keeps the entire model locally and processes it utilizing its own training data. After processing on workers, the produced models need to be combined. So, during each averaging phase, all parameters of the local model are averaged with the corresponding parameters of a selected remote model [7]. This introduces additional stochasticity, improving the overall accuracy. In use cases where privacy is not a priority, nodes can also exchange training data applying the same decentralized mechanism. The collaborative training procedure is designed to work with loosely connected topologies. No kind of synchronization is needed, and all data remain on the network for the peers to use them at their own pace. Furthermore, there is no indirect leakage, since the broadcasted models are not produced only by the private data of each peer, but they are increasingly affected by the weights of remote models created by neighbors, making reverse engineering practically impossible.

This article builds on our previous work on LEARNAE in the following ways: (a) it proposes a way of embedding IoT data by utilizing the message streams of the IOTA network; (b) it measures the energy efficiency of the proposed method using sensors emulated by low-energy Single-board Computers (SBC); and (c) it studies the recovery abilities of the proposed architecture, by simulating network disruptions and peer downtime.

Our previous work focused on establishing the fundamentals of a novel permissionless network, where individuals with commodity hardware cooperate to create improved neural models. But LEARNAE's initial specifications include nodes with multiple roles, ranging from lightweight IoT sensors to fully equipped workstations. IoT domain was a priority from the beginning, since we acknowledge the critical role it will play in the upcoming years. Following this direction, this article contains our proposal on how to embed low-energy devices to the existing infrastructure, without compromising the holistic decentralized philosophy. Additionally, in this article we showcase and quantify one of the most critical features of our architecture, resilience. Resilience expresses the ability to overcome unavailable resources, via distributed data duplication. We implement a new subsystem that emulates disruptions, and we perform extensive experimental testing to evaluate the fault tolerance of our algorithm under unreliable network conditions.

The rest of the article is structured as follows: Section 2 presents related work, Section 3 presents the DLT platforms utilized by our system, whereas Section 4 presents the extended LEARNAE architecture. Section 5 presents the experimental results and, finally, Section 6 concludes the article and poses future research directions.

[2] https://en.wikipedia.org/wiki/Gossip protocol

2 Related work

Related studies can be grouped in four major categories, depending on how they deal with the two key characteristics, centralization and synchronicity. Centralized solutions demand an entity with management duties, like a parameter server or a node with increased rights. This usually creates a communication bottleneck and requires high performance networking; it also creates a single point of failure. On the other hand, synchronous approaches impose some kind of time-based coordination between the peers. There are solutions where all nodes must work in strictly concurrent phases and others where they must have access to a common clock. In many cases these solutions suffer from locks and stale updates originating from slow workers. The following paragraphs briefly present previous proposals which are (a) centralized synchronous, (b) centralized asynchronous, (c) decentralized synchronous and (d) decentralized asynchronous.

(a) Centralized synchronous. Sandblaster L-BFGS [8] and Parallel minibatch SGD [9] rely on a parameter server and require high performance infrastructure. Parameter Server [10] adds ways to mitigate the effect of slow nodes and introduces fault management by estimating redundancy. FireCaffe [11] is based on the Caffe [12] framework, uses a custom MapReduce protocol and it also requires high speed networking. CaffeOnSpark [13] follows the approach of DistBelief and it is built on top of Spark[3] framework. Each peer also serves as a parameter server for a part of the model. BigDL [14] mainly adopts the principles of CaffeOnSpark, differing only into the parameter exchange mechanism. It requires separated cycles of training and data exchanging, resulting to a strictly synchronous working scheme. MPCA SGD [15] splits the model into shards, which are reduced and shared separately and requires a driver node to coordinate the whole process.

(b) Centralized asynchronous HOGWILD [16] uses local optimization and asynchronous model merging via a parameter server, reducing the communication demand. DistBelief [17] is a popular approach which utilizes a cluster of parameter servers and their peers. Each server-worker group manages a portion of the model. After every cycle, the peers need to download the updated copy of the joint model. Project Adam [18] organizes the parameter servers in a cluster and attempts to minimize network traffic by taking a part of the computation from the workers and assigning it to the servers. Elastic Averaging SGD (EASGD) [19] executes the optimizers on the nodes, which in turn communicate independently with a parameter server every N work cycles. MXNet [20] introduces a hierarchical parameter server structure, where intermediate nodes can act as proxy servers. Petuum [21] proposes the idea of rules regarding staleness, to improve synchronicity and convergence speed. Selective SGD [22] uses local optimization and asynchronous model merging via a parameter server, reducing the communication demand. TensorFlow [23] can be considered as the successor to DistBelief, adding automatic computation graph optimization, which makes distributed model parallelism much more practical.

(c) Decentralized synchronous SparkNet [24] follows the approach of FireCaffe, while it attempts to adapt to low-bandwidth networks. It implements synchronous decentralized training. Thus, each worker runs a separate optimizer in isolation for N cycles. The resulting models are then reduced through averaging. Before the next computation cycle begins, the averaged model is broadcasted to all workers and replaces their local ones. In Decentralized Parallel Stochastic Gradient Descent (DPSGD) [25] the nodes are synced using a common clock and exchange parameters after every training cycle.

(d) Decentralized asynchronous GoSGD [26] implements the EASGD algorithm, with the peers being organized in a mesh topology. The pairs of workers that will exchange data are selected every Nth cycle, through a randomized algorithm [27]. DeepSpark [28] attempts to implement EASGD to commodity hardware on top of Spark framework. Asynchronous Decentralized Parallel Stochastic Gradient Descent (AD-PSGD) [29] 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.

More recent works focus on optimizing the usage of available resources, by either parallelizing computation and communication, or using intuitive scheduling. Dianne [30] is a distributed framework developed in Java, which uses the Torch backend. It splits the neural network into different building blocks and assigns each block to a specific node. It includes components for training and evaluating models using a parameter server. MXNet-MPI [31] uses a modified version of MXNet, attempting to combine aspects of both synchronous and asynchronous solutions. It proposes clustering the workers into groups, which independently execute SGD with AllReduce. This clustering enforces scalability and fault tolerance. Horovod [32] creates a new training layer of AllReduce via MPI and adds it to Tensorflow. Its primary focus is on GPUs, so it uses libraries that support multiple GPUs on a single worker. Since it prioritizes computing speed, it lacks scalability and fault tolerance. ByteScheduler [33] is based on the idea that partitioning and rearranging the tensor transmissions may offer improved performance in distributed environments. It proposes that the training speed can be increased and the impact of the communication overhead can be mitigated by changing the transmission

[3] Apache Spark website, https://spark.apache.org

order of the neural layers. BytePS [34] focuses on heterogeneous architectures with GPU clusters. It attempts leveraging their spare CPU and bandwidth resources, to optimize the distributed training tasks. [35] proposes a two-way (both to and from workers) compressed SGD with Nesterov's momentum. To reduce the communication cost, it partitions the gradient into blocks which are compressed and transmitted in 1-bit format using a scaling factor. In Independent Subnet Training [36] the neural network is partitioned into a set of subnetworks of the same depth. Each work cycle contains local training of the subnet and exchanging the results with other peers. The procedure does not require any parameter aggregation, because the subnets have no common parameters. This fact enhances independence among the subnets and reduces the need for communication. Computation and Communication Decoupled SGD (CoCoDSGD) [37] proposes an algorithm for running computation and communication in parallel. The workers don't exchange data after each iteration but only periodically. This results in better use of resources and reduced communication cost.

Geryon [38] is a proposal for accelerating CNN training by adding a scheduling mechanism at network level. It divides model parameters into groups of different urgency. Parameters that have crucial impact on model quality are prioritized and transferred before all others. Geryon speeds up the training procedure but has no effect on achieved model accuracy. Preemptive AllReduce Scheduling for Expediting Distributed DNN Training (PACE) [39] is another communication scheduler. It is based on preemptively scheduling AllReduce tensors based on the DAG of DNN training. It attempts to identify the optimal granularity of tensor communication, to achieve the maximum overlapping of communication and computation functions. The result of this scheduler is a minimized overhead and a maximized bandwidth utilization. Priority-based Parameter Propagation for Distributed DNN Training (P3) [40] consists of two major parts, parameter slicing and priority-based update. The first divides the model layers into smaller sublayers and synchronizes them independently. The second assigns a priority to each slice, based on whether it is required again in the subsequent iteration. P3 ensures that the most needed slices will consistently have the required network resources. SwitchML [41] is based on the following principles: First, parameter updates can be decomposed to chunks that can be individually processed. Second, SGD aggregation can be applied separately on different portions of the input data, disregarding order. Third, ML training is robust to modest approximations of its compute operations. The authors propose a method under which the created chunks are processed by the switch pipeline, where packet loss can be tolerated using a light-weight switch scoreboard mechanism and a retransmission mechanism driven solely by end hosts. TicTac [42] is built over TensorFlow and attempts to predict the order in which the parameters will be consumed by the underlying computational model. So, it arranges the network transfers accordingly, to reduce communication time. TonY [43] is an open-source orchestrator, which consists of a client and a scheduler. Users can use the client to submit ML jobs and the scheduler handles allocating resources, setting up configurations and launching the ML job in a distributed fashion. [44] attempts to address the scaling problem on public cloud clusters due to moderate inter-connection bandwidth. It proposes a top-k sparsification library for optimized computing and communication. It also implements a multi-level data caching mechanism to optimize I/O functions and introduces a novel parallel tensor operator to accelerate the update operation. The above features are utilized by a hierarchical communication algorithm which attempts to aggregate the sparsified gradients to achieve better utilization of the processing power.

In contrast to all other decentralized solutions, LEARNAE does not employ any notion of a shared model. Instead, every participating peer keeps its own model and attempts to leverage the knowledge of its neighbors to improve it. Most of the above-mentioned approaches attempt to distribute execution to decrease training time. It is of great importance to outline that LEARNAE follows a divergent path: It focuses not only on improved models, but also on implementing a fully democratized process, by prioritizing the following features:

Peer-to-Peer. With pure peer-to-peer architecture, all nodes enjoy the same level of access regarding the produced neural models. None of them carries out an either essential or privileged role.

Resilience. The whole procedure is shielded against node downtime and largescale network disruptions, allowing 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. The coordinating algorithm 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 scheme, vast differences in hardware are mitigated, thus there are no locks.

To the best of our knowledge, no other proposal sets and fulfils these priorities.

3 Distributed Ledger Technology

This section presents the concepts that serve as the basis for the LEARNAE system. The coordinating algorithm is platform-agnostic and can employ any method of data exchange. For the current implementation, to maintain its decentralized nature, two novel DLT projects were leveraged: IPFS and IOTA.

3.1 IPFS

Interplanetary File System (IPFS) combines features from Git [45] and BitSwap[4], which is an algorithm for incentivizing data replication (based on BitTorrent [46]). IPFS implements a decentralized filesystem for permissionless peer-to-peer topologies.

During IPFS setup, a node gets a unique hash string which is its identification when communicating with other peers. For every file that is added, the node assigns to it a cryptographic hash based solely on the file's contents. If the file is larger than a predefined size, it is divided into chunks that get their own hash and are stored independently. In contrast with most other filesystems, IPFS does not use an addressing method based on the storing location of the file, instead all files can be fetched just by knowing their unique hash. For this type of addressing to work, every node maintains a copy of a Distributed Hash Table (DHT), which is a ledger inspired by [47][48][49]. The DHT contains information about where data chunks can be retrieved from.

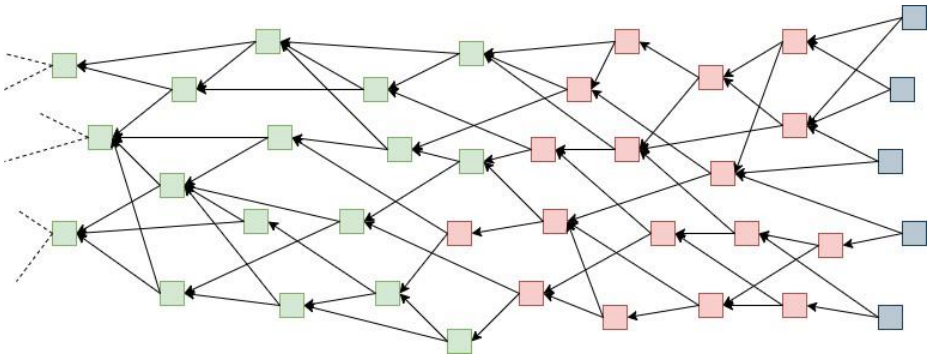

Fig. 1: Representation of transactions in the Tangle
(green: confirmed, red: unconfirmed, gray: tips)

IPFS follows a simple but effective incentivization mechanism. Each peer maintains a list of chunks stored locally and a list of chunks it needs. It also preserves a balance of verified bytes exchanged with every other peer. This balance acts as a credit system which indicates the level of participation to the swarm. Using gossip style communication, the nodes attempt to obtain the data they need, but at the same time they try to improve their reputation with their neighbors; because by doing so they improve the chances they will find chunks that they will need in the future.

IPFS is all about decentralization, so it has no need of any form of central entities. Since there is no server-based storage, all space is offered by the participating peers. When a node needs a file, it obtains it from another peer and then stores it locally. How long a file remains available on a specific node depends on the embedded garbage collection system. If a peer decides a file is important enough, it can "pin" the file and thus retain it indefinitely on local storage. Considering all the above, IPFS should be able to serve as a completely distributed way to share training-related information, ensuring load balancing and resilience via data replication.

PubSub. A collaborative training session among many participants requires exchanging a large amount of sidekick information. IPFS has a near-realtime messaging subsystem called PubSub, which follows a publish-subscribe approach. PubSub allows nodes to form communication groups by signing up to "topics", where all

[4] Bitswap webpage, https://github.com/ipfs/specs/tree/master/bitswap

messages are relayed using gossip protocols. LEARNAE leverages this feature to broadcast essential training metadata.

3.2 IOTA

The IOTA (meaning extremely small) project has as its goal to develop a completely decentralized network connecting all IoT devices. There are projections[5]predicting that the number of connected IoT devices will surge to 50 billion by 2030. It is obvious this will be accompanied by a huge increase of data transmitted by those devices. In many cases the devices are low-energy sensors that record data regarding environmental conditions, traffic, personal health, etc.

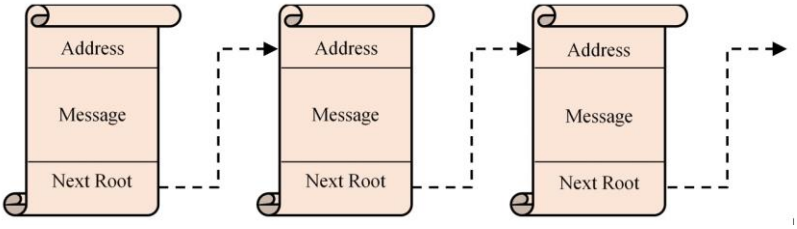

Fig. 2: The singly-linked list of a MAM stream

To overcome the scalability issues encountered by blockchains, IOTA is based on a Directed Acyclic Graph (DAG), which was named ”Tangle”[6]. The Tangle’s building blocks are ”transactions”. If a node needs to attach a new transaction, it must confirm two previous ones (Fig. 1), so -unlike with blockchains- increased activity is expected to result in better performance and security for the network. IOTA attempts to achieve consensus via a permissionless procedure, using the Tangle as DLT and propagating transactions throughout the network via a gossip protocol.

Masked Authenticated Messages. The Tangle supports feeless zero-value transactions that serve the purpose of data streams, called Masked Authenticated Messages (MAM). This article proposes a way of embedding these streams into the LEARNAE scheme, to be utilized as a completely distributed way of acquiring data from lightweight IoT devices.

Table 1: Encryption types of MAM streams

Mode	Address	Message decryption key
Public	Root	Root
Private	Hash (Root)	Root
Restricted	Hash (Root + Sidekey)	Root + Sidekey

MAM streams are based on a structure called singly-linked list. A message stream consists of several messages, where each message contains a pointer to the next one. Anyone who knows the address of a specific message, they can only access the stream that follows that message, a feature known as “forward secrecy” (Fig. 2).

Regarding privacy, there are three types of MAM streams. In Public mode there is no restriction, and everyone who knows a stream’s address can read the messages. In Restricted mode only those who know a sidekey can access the stream. In “Private” mode the stream can be accessed only by its creator because it is required to know the secret hash string (called seed) that created the stream.

MAM uses a signature scheme based on Merkle Trees [52] to sign the cipher digest of an encrypted message. The root of the Merkle tree is used as the ID of the channel and each message contains the root of the following tree. All messages are encrypted with a one-time pad that consists of the channel ID and the index of the key used to sign the message. An additional nonce may be used as a revocable encryption key. The resulting cipher hash is signed using the private key belonging to one of the leaves. The encrypted payload,

[5] Statistics webpage, https://www.statista.com/statistics/802690/worldwide-connecteddevices-by-access-technology

[6] Project whitepaper, https://iota.org/IOTA Whitepaper.pdf

the signature and the leaf's siblings are then published to the Tangle, where anyone knowing the symmetric key can find and decrypt it (Table 1).

4 LEARNAE system

This section presents how LEARNAE utilizes modern DLT to construct a network for decentralized DNN training. Paragraph 4.1 contains the workflow of fat nodes, that is peers with enough processing power to participate to IPFS swarm and conduct model training and parameter averaging. Paragraph 4.2 presents our proposal on a way to leverage IOTA's MAM messages, to embed IoT sensors that stream training data to "fat" nodes.

Table 2: Supported node types and their features

Node role	Platforms supported	Model training	Own training data
Full	IPFS + IOTA	Yes	Yes
Trainer	IPFS + IOTA	Yes	No
Feeder (fat)	IPFS	No	Yes
Feeder (thin)	IOTA	No	Yes

4.1 Coordinating algorithm

LEARNAE nodes can choose different roles (Table 2), depending on their available processing power. Nodes with adequate CPU specifications can participate to the IPFS swarm, to train and exchange neural models, hence contributing to the weight averaging process. Fig. 3 shows the data flow between different types of nodes.

Figure 4 depicts the principal parts of a Full Node's workflow. When a collaborative session starts, the peers that possess training data split them in "dataslices" of predefined size. These files are separately added to IPFS, and their hashes are grouped to a hashlist. This file is in turn added to the shared DHT and its hash is broadcasted to the network (purple area). Every time a peer consumes a dataslice, it broadcasts a message which informs the others that the specific dataslice has been used. This enables the "overuse threshold" feature of LEARNAE, according to which the participants can set a limit on how many times a specific dataslice can be used for training throughout the network.

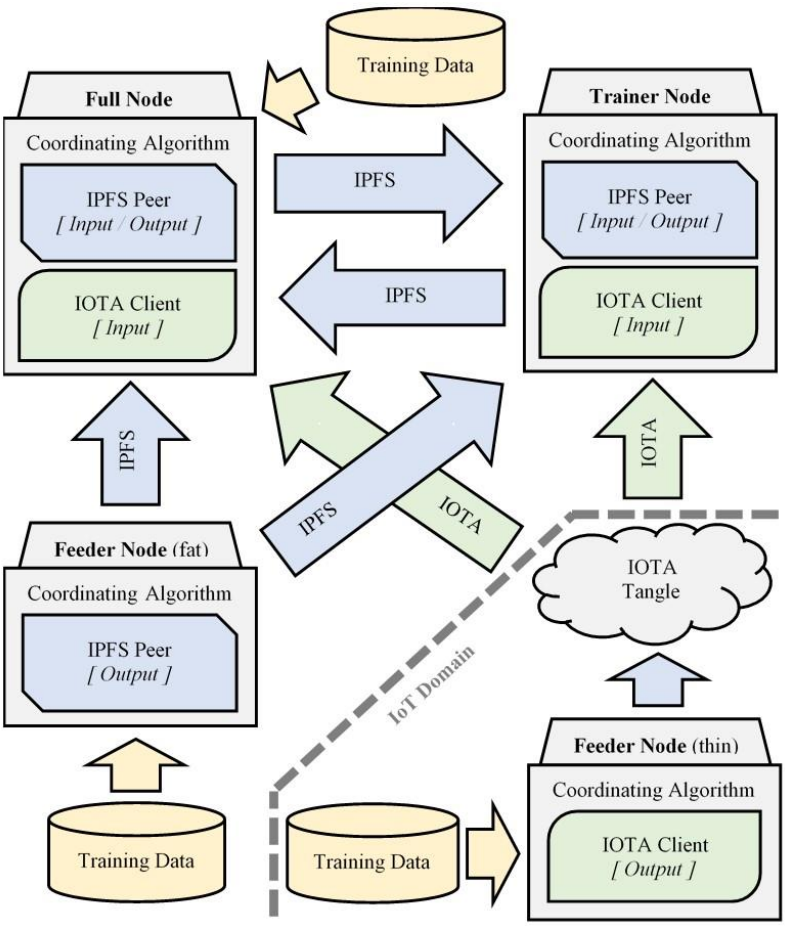

Fig. 3: Data flow between different node roles

When a peer receives a message about an available dataslice, it fetches its chunks from multiple neighbors and appends it to the queue containing the locally available dataslices; unless the slice has already reached the predefined overuse threshold, in which case the peer ignores it (red area).

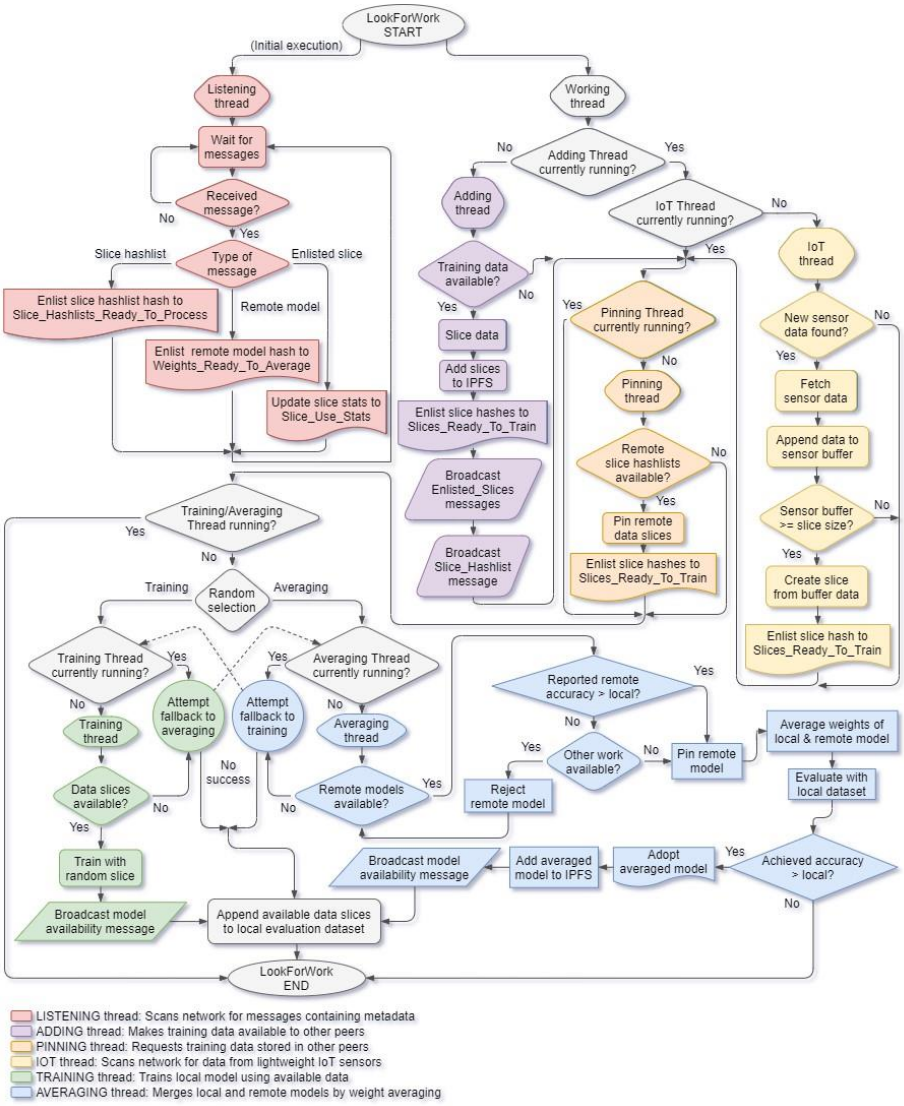

Fig. 4: The principal parts of a Full Node's workflow

In parallel with the above functions, and in every work cycle, peers randomly select between training and averaging, which enhances the overall stochasticity. After each training phase, the peers add the generated local model to IPFS and broadcast its availability (green area). When a peer selects averaging, it examines its local list of remote models. If it finds a model with a reported accuracy greater than the local, it obtains the remote model and averages it with the local one. If this process leads to an improved local accuracy, the peer adopts the averaged model, adds it to shared DHT and broadcasts its availability to the network. The message about available models contains metadata like the accuracy achieved by the broadcaster and the maturity of the model, e.g., how many work cycles preceded its creation (blue area). As a final optimization step, when all averaging attempts have been made and the network has reached the maximum convergence, every peer fetches the one remote model that achieved the highest accuracy. This model is evaluated against the local dataset and if it scores better, it is adopted as is by the peer.

4.2 IoT section

This article introduces a way of embedding data from lightweight IoT sensors using the IOTA Tangle (Fig. 5). The participating peers that can train models, periodically check the Tangle for new messages. If they find new data, they save them locally. Because of the IoT-oriented nature of the Tangle, it performs better when the transmitted packages are not too large. Specifics on the experimental metrics can be found in the next chapter. The saved sensor data are stored in a buffer and when their size reaches the preselected dataslice size,

they are appended to the list of dataslices that are ready to be used for training; subsequently the buffer is purged.

MAM Stream Configuration. In addition to the data portion, a MAM message may contain a tag field. Since MAM API supports search by tag, LEARNAE utilizes it as the connecting link between all sensors of a specific training session. Each sensor has its own MAM stream. The first message is tagged after a common pre-agreed codename (it could match the used IPFS PubSub topic name) and contains the sensor ID and the timestamp of its creation. This head message points to the first data message and so on. All information is in JSON format.

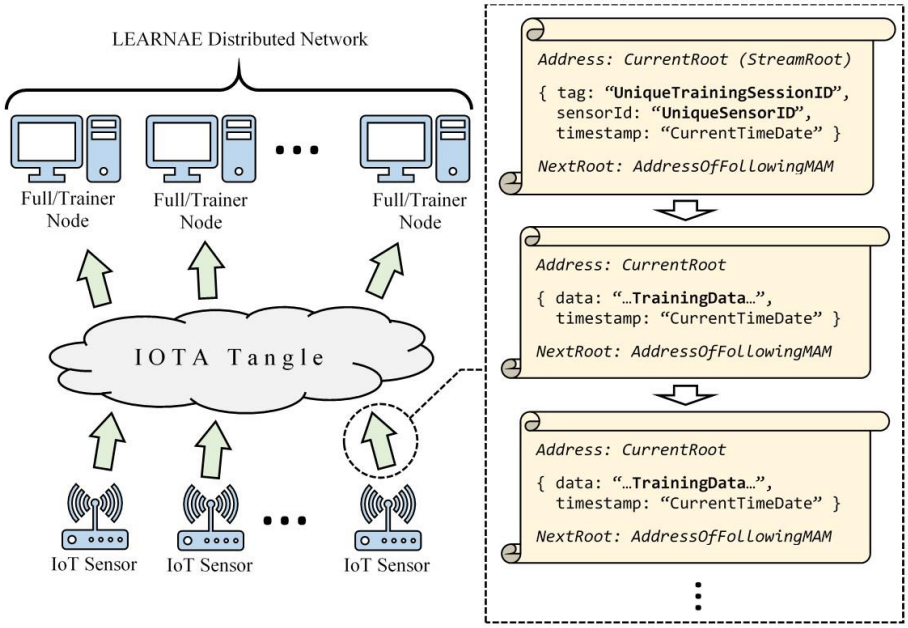

Fig. 5: The structure of MAM messages created by LEARNAE

Embedding an IoT sensor to a LEARNAE network is a two-step process: First, the inquiring peer queries the Tangle for MAM messages with the specified tag. If it finds a new message (e.g., with a first-seen address), attaches it to the list of known sensors. Then, each time the peer scans for new IoT data, it starts checking the linked list's tip of each known sensor. If it finds new data messages, it appends them to the local buffer and stores a pointer to that last message; hence, the next times it checks the specific sensor, the search will start from this pointer and not from the beginning of the whole stream.

Table 3: Configurations of VPS nodes

Metric	Value
CPU	QEMU Virtual CPU version 2.1.2 2.10 GHz (2 processors)
RAM	4 GB
Internet connection	50 Mbps

5 Experiments and results

LEARNAE's extensions presented in this article, were evaluated by conducting an extensive number of distributed training sessions. The network consisted of 20 workstations, which were set up on the cloud as Virtual Private Servers (VPS). Their configuration was selected to resemble that of a commodity PC, and it is shown in Table 3. A use case of distributed training with significant research interest, are environments where each peer only possesses a relatively small number of instances. In this case the collaboration can mitigate the low-data constraint and offer models with higher accuracy.

Table 4: Main hardware/software specifications of the emulated sensor

Specification	Value
CPU	Quad Core 1.2 GHz 64 bit
RAM	1 GB
Network	100 Base Ethernet
Storage	16 GB Micro SD Card (Class 10)
Internet connection	50 Mbps
OS	Raspbian
OS kernel version	4.19
OS image size	1136 MB
Framework of MAM-related code	NodeJS (ver: 11.10.1)

In contrast to our previous studies [1][2], the experiments of this article focus on such use cases, by setting the available training data of every peer to only 5,000 instances. All tests are configured for privacy protection, thus the participating peers exchange only models and not training data. The following sections quantify the benefits of the proposed architecture.

5.1 IoT evaluation

Sensor Setup. To emulate an IoT sensor, we adopted an SBC with moderate specifications and low energy consumption: The Raspberry Pi 3 Model B[7]. Table 4 shows its hardware specifications and the main software/development choices that were made to implement these experiments.

Dataset Characteristics. The dataset used was HEPMASS, which is intended for training systems on exotic particle detection using a large number of collisions. It includes 7 million instances (5 GB) for training and 3 million instances (2.5 GB) for testing. Each instance is approximately 750 bytes and is comprised (including metadata) of 30 floating point numbers. For these experiments, the data are transmitted as they are, with no compression/encryption. Each dataset instance emulates a new signal received by a sensor. Several instances are grouped to form a transmission packet. This number depends on the instance size, to create packets that will have optimal propagation through the Tangle.

Data Publishing. The training dataset was split to packets of 50 instances. Thus, every MAM message published to the Tangle was approximately 35 KB. This was found to be an optimal choice, since smaller values did not achieve maximum network utilization and larger values where occasionally rejected by the IOTA nodes due to congestion or abuse-related rules. Table 5 presents the metrics of the sensor data publishing procedure.

5.2 Fault tolerance evaluation

To evaluate the negative impact of network disruptions, a new subsystem was implemented in LEARNAE, simulating peers shutting down or facing connectivity problems. The intensity of the simulated issues is configured by three new testing parameters: (a) Offline Cycle, which determines how often the disconnection state will be updated; (b) Offline Duration, which is how long the disconnection will last; (c) Offline Probability, which sets how likely is the disconnection to happen.

Table 5: Metrics of data transmitted by the SBC

Metric	Value
Instance size (approx.)	750 bytes
Instances per packet	50
Packet size (approx.)	35 KB
Number of transmitted packets	1000
Total time required	3 h 52 m 18 s
Time required per packet	13.9 s

[7] Device webpage, https://www.raspberrypi.org/products/raspberry-pi-3-model-b

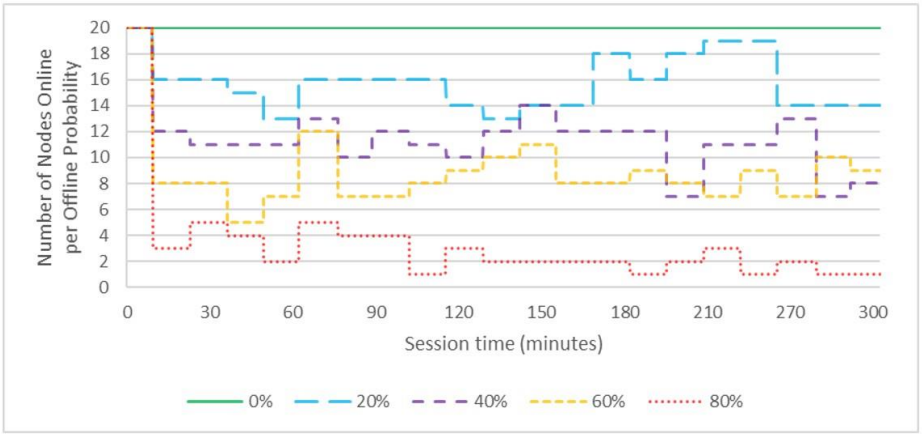

Fig. 6: The number of online peers for different Offline Probability

For these experiments, the first two values were set to 15, meaning that every 15 minutes the connectivity of peers was updated, and the new state lasted until the next update. In order to study the impact of these issues, all the experiments were conducted for 0%, 20%, 40%, 60% and 80% Offline Probability. Fig. 6 shows (a sample of) the fluctuation of the number of online peers during the conducted experiments.

Low Epoch Conditions. During a LEARNAE session there are two ways for a model to be improved: Training and averaging. In use cases with low number of epochs, the impact of training is reduced and averaging has an enhanced role. To demonstrate the maximum effect that network disruptions may have, initial experiments with a single training epoch were conducted. Fig. 7 depicts the achieved mean accuracy and spread for each offline probability, whereas Fig. 8 compares them. For reference purposes, these figures also show the results for stand-alone configuration, that is, the case when every peer trains its model in isolation using only its own data. As shown in these figures, the proposed algorithm managed to withstand all disruptions, outperforming the stand-alone configuration, except for 80% Offline Probability. The extended final spread (orange area) of the latter case (f) indicates that this was the only case the network failed to achieve proper convergence.

In LEARNAE's terminology the ability to overcome the problems caused by unavailable peers is expressed by resilience, an index defined as the total number of neighbors a data chunk can be fetched from at a specific time. To offer an insight on the achieved data replication, Fig. 9 shows how mean resilience and its spread are evolving during a training session with no simulated disconnections. The final value of 6 means that in average a requested chunk could be acquired from 6 different remote peers, out of total 20.

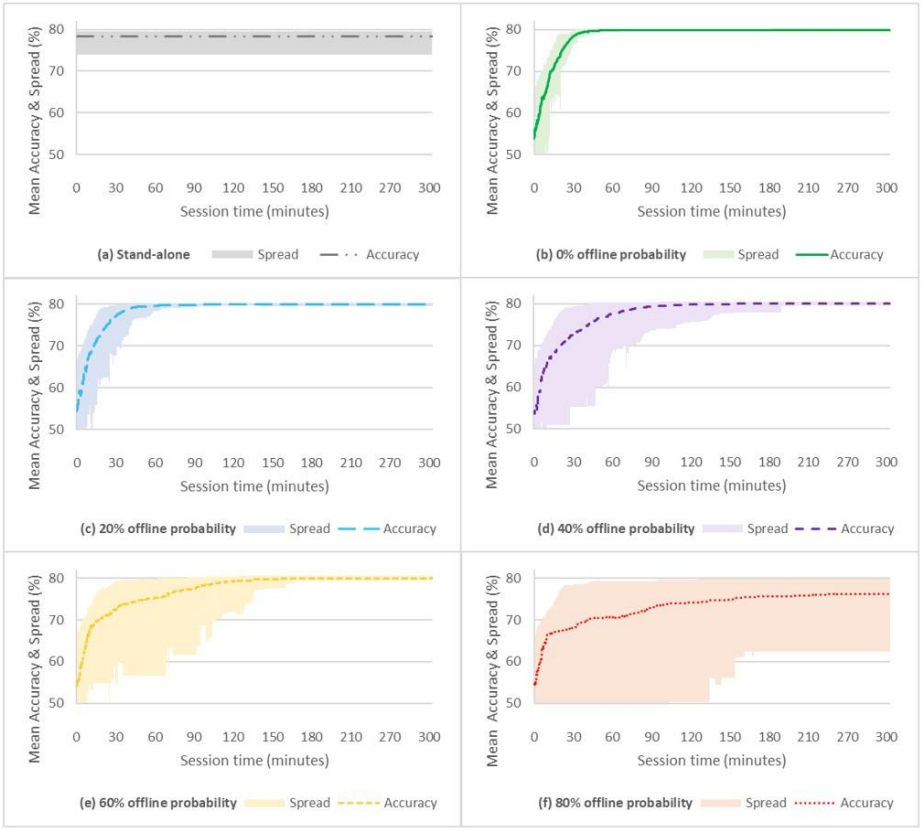

Fig. 7: Mean Accuracy and Spread per Offline Probability (Low Epochs)

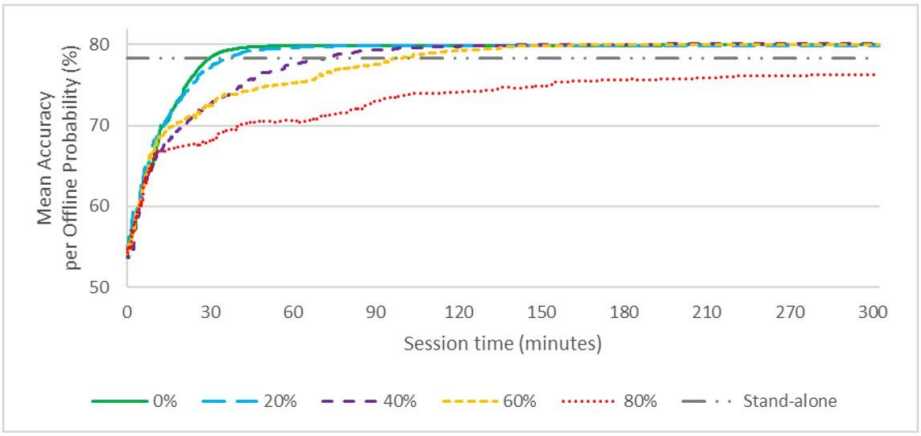

Fig. 8: Comparison of Mean Accuracy for different Offline Probability (Low Epochs)

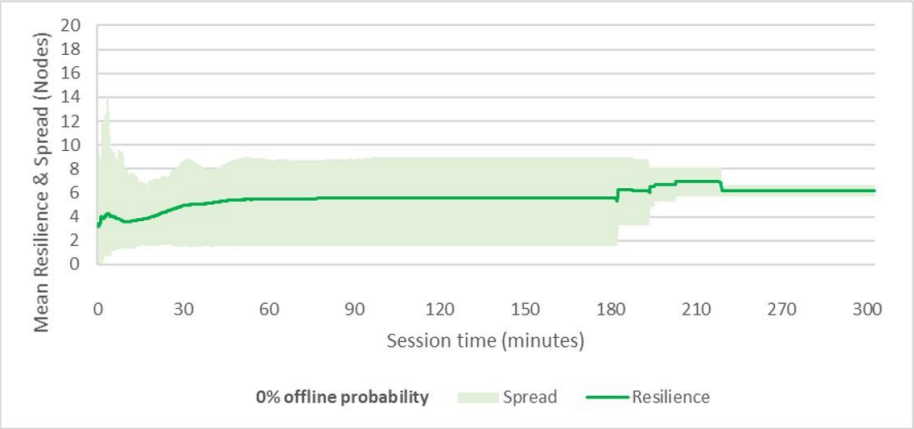

Fig. 9: Mean Resilience and Spread

Table 6: Metrics of data transmitted by the SBC

Offline Prob.	Acc. Spread *[Stand-alone]*	Acc. Spread *[Distributed]*	Spread Change
0	5.60	3.04	-2.56
20		2.39	-3.21
40		2.52	-3.08
60		2.36	-3.24
80		3.15	-2.45

Optimal Epoch Conditions. For the configuration of these experiments, the optimal number of epochs was found to be 10. The following tests were executed accordingly, so the positive effect of the training phases was maximized. As presented in Fig. 10, disruptions with Offline Probability up to 80% could not affect the overall performance. Fig. 11 shows the comparison of the achieved mean accuracy for different disconnection rates. The proposed averaging algorithm managed to reduce the accuracy spread among the peers (by up to 3.24%), even for 80% disconnection rate (Table 6).

5.3 Data balancing

Since the quality of the training data varies between the peers, the generated models will achieve different accuracy. This discrepancy is more intense at the start of the collaborative session and is reduced towards the end, when averaging is completed. The distributed nature of the proposed architecture ensures load balancing and the absence of congestion points. Fig. 12 depicts the amount of data sent by each participant. The peers that achieve more accurate models in the early stages must provide more data, since their model is massively fetched by others. But this surge is only temporary, and it lasts only while the replication factor of their data in the network is still low.

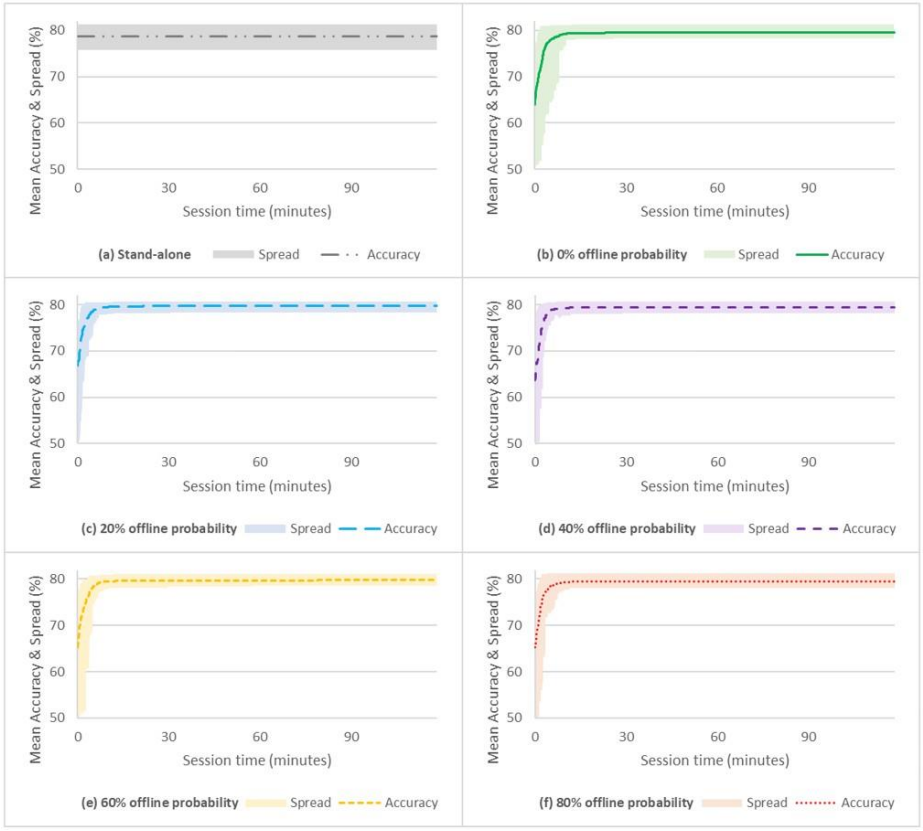

Fig. 10: Mean Accuracy and Spread per Offline Probability (Optimal Epochs)

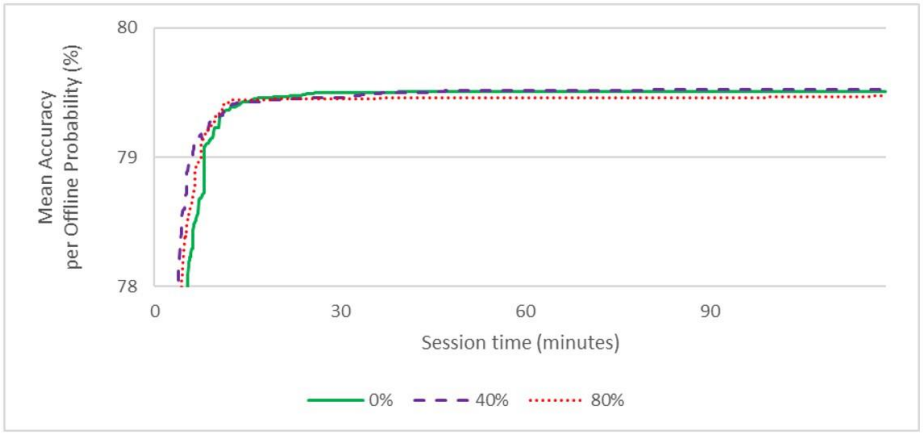

Fig. 11: Comparison of Mean Accuracy for different Offline Probability (Optimal Epochs)

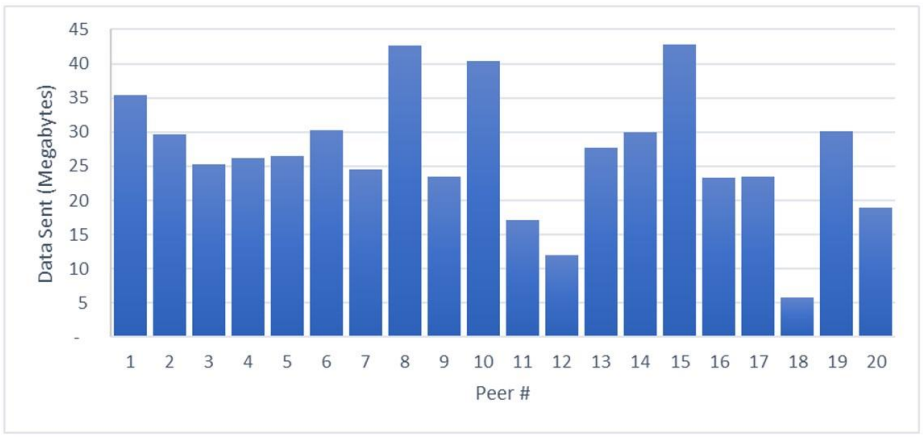

Fig. 12: Total amount of data sent by each peer

5.4 Benefits of proposed architecture

As shown in Fig. 13, during the first stage when data are poured into the network, peers are equally divided into two groups, for training (green) and averaging (blue). When all the available data are consumed, no training processes are invoked, and weight averaging becames dominant. This procedure ultimately leads to global convergence, where no peer can benefit from model merging and the averaging phase comes to an end. The rightmost spikes on the blue area are the final averaging attempts of peers that already have high-quality models. For better utilization when peers possess no other work in their queue, the attempt averaging with newly published remote models, even if their reported accuracy is lower than the local one.

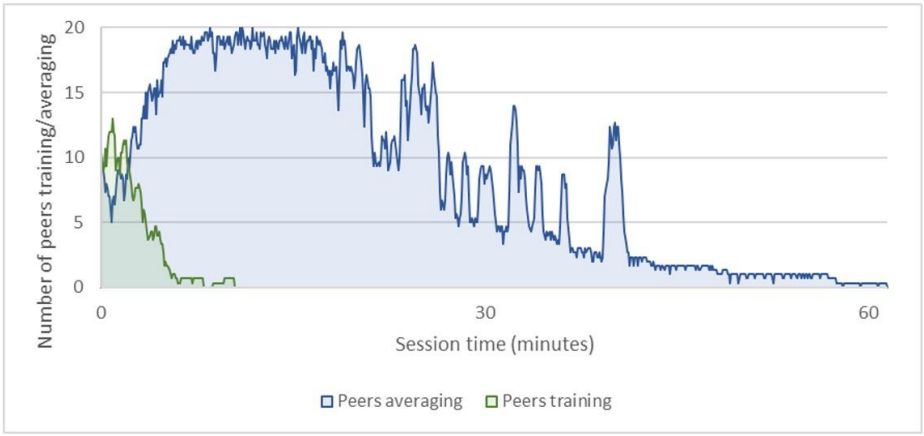

Fig. 13: Number of peers training/averaging

This generates further exploration of the parameter space and can lead to model improvements, without sacrificing valuable processing time. For the conducted experiments, the ratio of successful averaging attempts had a maximum value of 32%, while the mean value for all participants was 10%. These are the final values at the end of the session when the network had reached convergence. During the initial phase, the peers were improving their accuracy via model merging at a rate up to 70% (Fig. 14).

To evaluate the improvement in accuracy by the proposed architecture, two sessions were executed using optimal number of epochs. In the first, all peers trained their models in isolation. The second one was a LEARNAE session with privacy features enabled. As seen in Fig. 15 and Table 7, there is a significant improvement in the accuracy of the produced models by 1.12%, although there was no sharing of training data.

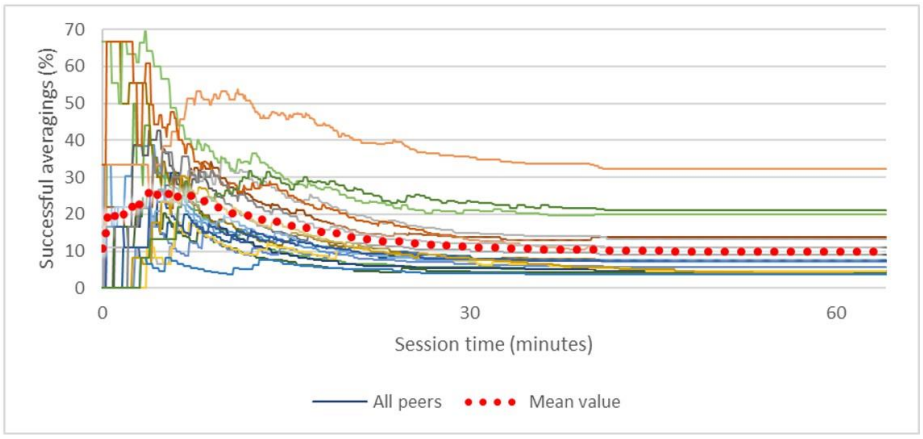

Fig. 14: Successful averaging attempts

Table 7: Mean model accuracy values (stand-alone vs distributed)

Method	Model Accuracy
Stand-alone	78.74 %
Distributed *(data privacy)*	79.86 %

6 Conclusions and future work

This article is an extension of our previous work [1][2] on utilizing modern DLT for enhancing decentralization, privacy, and fault-tolerance of ANN training. It proposes and evaluates a novel way to embed IoT sensors without undermining the distributed nature of the whole architecture. It also studies a key characteristic, resilience, and the impact of network disruptions and unreliable peers.

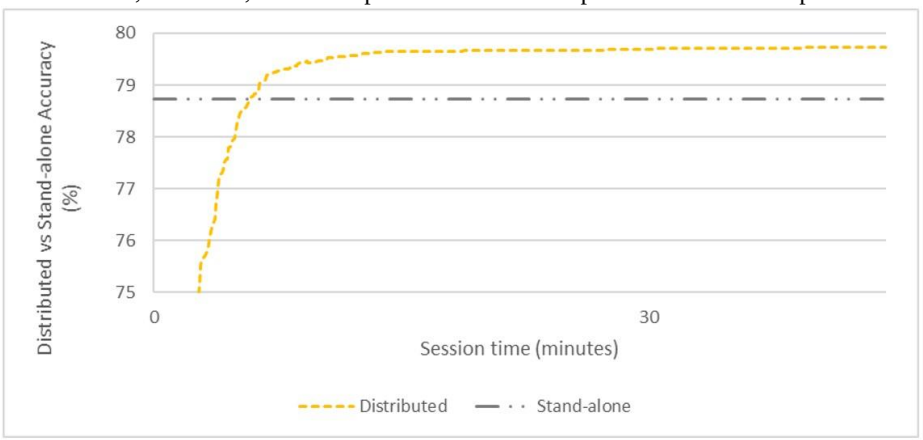

Fig. 15: Mean model accuracy (stand-alone vs distributed)

With these contributions, LEARNAE fulfills the following properties: (a) It does not require any kind of central coordinating entity or nodes with elevated rights, since it is based on a purely peer-to-peer topology. (b) It supports participants with heterogeneous hardware, while it prevents locks from slow workers. (c) It offers diverse roles, depending on processing power and training data availability. (d) It can operate in privacy mode, thus giving peers the option of retaining ownership of their sensitive data. (e) It can recover from intense network disruptions by using data replication. (f) It can keep all (meta)data available for any period, allowing

new peers to join at any time. (g) It realizes process democratization, since participants (i) enjoy full access to all information and produced models, (ii) do not need expensive and sophisticated hardware.

Beyond the typical training scenarios, such an approach could offer a solution to use cases where the training data are continuously accumulated, for example by IoT sensors. So, peers with common interest could form a community, to have the best possible model at any given time. As proven by the conducted experiments, while prioritizing the above features, LEARNAE collaboratively achieves models with an improved accuracy of up to 1.12% for the current configuration. That is with no sharing of training data, but simply by leveraging the knowledge of neighbors via a selective parameter averaging. According to the insight we gained from our experiments, the benefits from the proposed algorithm would be positively affected by increasing the total number of participants, which in this study were only 20 peers.

There are still many issues of research interest to be addressed. How much exactly can a large-scale deployment benefit the achieved results? How well can the implemented algorithm cooperate with methods like model sharding, for faster convergence? What additional metadata can be used for optimized selection of remote averaging candidates? Are there any experimental DLT platforms with a better fit, which can outperform the selected ones? Can this proposal be applied to other training methods like Random Forests, where each worker can train a tree or a set of trees with their sample of data? In what ways can peers be incentivized to participate to a LEARNAE network, even when they have no interest in the produced neural model? All the above-mentioned will be the subject of our future work.

Conflict of interest

The authors declare that they have no conflict of interest.

References

1. Nikolaidis S, Refanidis I (2019) Learnae: Distributed and Resilient Deep Neural Network Training for Heterogeneous Peer to Peer Topologies. International Conference on Engineering Applications of Neural Networks 286-298. https://doi.org/10.1007/978-3-030-20257-6 _24
2. Nikolaidis S, Refanidis I (2020) Privacy preserving distributed training of neural networks. Neural Comput & Applic. https://doi.org/10.1007/s00521-020-04880-0
3. Benet J (2014) IPFS - Content Addressed, Versioned, P2P File System. arXiv:1407.3561
4. Popov S, Saa O, Finardi P (2018) Equilibria in the Tangle. arXiv:1712.05385
5. Zhang X, Trmal J, Povey D, Khudanpur S (2014) Improving deep neural network acoustic models using generalized maxout networks in Acoustics, Speech and Signal Processing (ICASSP). IEEE International Conference
6. Miao Y, Zhang H, Metze F (2014) Distributed learning of multilingual dnn feature extractors using gpus
7. Povey D, Zhang X, Khudanpur S (2014) Parallel training of deep neural networks with natural gradient and parameter averaging
8. Dean J, Corrado GS, Monga R, Chen K, Devin M, Le QV, Mao M, Razato M, Senior A, Tucker P, Yang K, Ng AY (2012) Large Scale Distributed Deep Networks. Advances in Neural Information Processing Systems 1223-1231
9. Dekel O, Gilad-Bachrach R, Shamir 0, Xiao L (2012) Optimal distributed online prediction using mini-batches. Journal of Machine Learning Research 165-202
10. Li M, Andersen DG, Park JW, Smola AJ, Ahmed A, Josifovski V, Long J, Shekita EJ, Su BY (2014) Scaling Distributed Machine Learning with the Parameter Server. 11th USENIX Symposium on Operating Systems Design and Implementation 583-598
11. Iandola FN, Ashraf K, Moskewicz MW, Keutzer K (2015) FireCaffe: near-linear acceleration of deep neural network training on compute clusters. arXiv:1511.00175
12. Jia Y, Shelhamer E, Donahue J, Karayev S, Long J, Girshick RB, Guadarrama S, Darrell T (2014) Caffe: Convolutional Architecture for Fast Feature Embedding. ACM Intl. Conference on Multimedia 675-678
13. Feng A, Shi J, Jain M (2016) CaffeOnSpark Open Sourced for Distributed Deep Learning on Big Data Clusters
14. Wang Y, Zhang X, Wong I, Dai J, Zhang Y et al (2017) BigDL Programming Guide

15. Langer M, Hall A, He Z, Rahayu W (2018) MPCA SGD - A Method for Distributed Training of Deep Learning Models on Spark. IEEE Transactions on Parallel and Distributed Systems 29:2540-2556. https://doi.org/10.1109/TPDS.2018.2833074
16. Niu F, Recht B, Re C, Wright SJ (2011) HOGWILD!: A Lock-Free Approach to Parallelizing Stochastic Gradient Descent. arXiv:1106.5730v2
17. Dean J, Corrado GS, Monga R, Chen K, Devin M, Le QV, Mao M, Razato M, Senior A, Tucker P, Yang K, Ng AY (2012) Large Scale Distributed Deep Networks. Advances in Neural Information Processing Systems 1223-1231
18. Chilimbi T, Suzue Y, Apacible Y, Kalyanaraman K (2014) Project Adam: Building an Efficient and Scalable Deep Learning Training System. 11th USENIX Symposium on Operating Systems Design and Implementation 571-582
19. Zhang S, Choromanska A, LeCun Y (2015) Deep learning with Elastic Averaging SGD. Advances in Neural Information Processing Systems 685-693
20. Chen T, Li M, Li Y, Lin M, Wang N, Wang M, Xiao T, Xu B, Zhang C, Zhang Z (2015) MXNet: A Flexible and Efficient Machine Learning Library for Heterogeneous Distributed Systems. LearningSys
21. Xing EP, Ho Q, Dai W, Kim JK, Wei J, Lee S, Zheng X, Xie P, Kumar A, Yu Y (2015) Petuum: A New Platform for Distributed Machine Learning on Big Data. IEEE Transactions on Big Data 1:49-67
22. Shokri R, Shmatikov V (2015) Privacy-Preserving Deep Learning. 22nd ACM SIGSAC 1310-1321
23. Abadi M, Barham P, Chen J, Chen Z, Davis A, Dean J, Devin M, Ghemawat S, Irving G, Isard M, Kudlur M, Levenberg J, Monga R, Moore S, Murray DG, Steiner B, Tucker P, Vasudevan V, Warden P, Wicke M, Yu Y, Zheng X (2016) TensorFlow: A System for Large-Scale Machine Learning. 12th USENIX Symposium on Operating Systems Design and Implementation 265-283
24. Moritz P, Nishihara R, Stoica I, Jordan MI (2016) SparkNet: Training Deep Networks in Spark. Intl. Conference on Learning Representations
25. Lian X, Zhang C, Zhang H, Hsieh CJ, Zhang W, Liu J (2017) Can decentralized algorithms outperform centralized algorithms? A case study for decentralized parallel stochastic gradient descent. Advances in Neural Information Processing Systems (NIPS)
26. Blot M, Picard D, Cord M, Thome N (2016) Gossip training for deep learning. arXiv:1611.09726
27. Boyd S, Ghosh A, Prabhakar B, Shah D (2006) Randomized Gossip Algorithms. IEEE Transactions on Information Theory 52:2508-2530
28. Kim H, Park J, Jang J, Yoon S (2016) DeepSpark: Spark-Based Deep Learning Supporting Asynchronous Updates and Caffe Compatibility. arXiv:1602.08191
29. Lian X, Zhang W, Zhang C, Liu J (2018) Asynchronous decentralized parallel stochastic gradient descent International Conference on Machine Learning (ICML)
30. Coninck E, Bohez S, Leroux S, Verbelen T, Vankeirsbilck B, Simoens P, Dhoedt B (2018) DIANNE: a modular framework for designing, training and deploying deep neural networks on heterogeneous distributed infrastructure. Journal of Systems and Software 141:52-65
31. Mamidala AR, Kollias G, Ward C, Artico F (2018) MXNET-MPI: Embedding MPI parallelism in Parameter Server Task Model for scaling Deep Learning. arXiv:1801.03855
32. Sergeev A, Del Balso M (2018) Horovod: fast and easy distributed deep learning in TensorFlow. arXiv:1802.05799
33. Peng Y, Zhu Y, Chen Y, Bao Y, Yi B, Lan C, Wu C, Guo C (2019) A Generic Communication Scheduler for Distributed DNN Training Acceleration. Proceedings of the 27th ACM Symposium on Operating Systems Principles 16-29
34. Jiang Y, Zhu Y, Lan C, Yi B, Cui Y, Guo C (2020) A Unified Architecture for Accelerating Distributed DNN Training in Heterogeneous GPU/CPU Clusters. 14th USENIX Symposium on Operating Systems Design and Implementation
35. Zheng S, Huang Z, Kwok J (2019) Communication-Efficient Distributed Blockwise Momentum SGD with Error-Feedback. Advances in Neural Information Processing Systems
36. Yuan B, Wolfe CR, Dun C, Tang Y Kyrillidis A, Jermaine CM (2020) Distributed Learning of Deep Neural Networks using Independent Subnet Training. arXiv:1910.02120
37. Shen S, Xu L, Liu J, Liang X, Cheng Y (2019) Faster Distributed Deep Net Training: Computation and Communication Decoupled Stochastic Gradient Descent. arXiv:1906.12043
38. Wang S, Li D, Geng J (2020) Geryon: Accelerating Distributed CNN Training by NetworkLevel Flow Scheduling. IEEE Conference on Computer Communications 1678-1687
39. Bao Y, Peng Y, Chen Y, Wu C (2020) Preemptive All-reduce Scheduling for Expediting Distributed DNN Training. IEEE Conference on Computer Communications 626-635

40. Jayarajan A, Wei J, Gibson G, Fedorova A, Pekhimenko G (2019) Priority-based Parameter Propagation for Distributed DNN Training. arXiv:1905.03960
41. Sapio A, Canini M, Ho C, Nelson J, Kalnis P, Kim C, Krishnamurthy A, Moshref M, Ports DRK, Richtarik P (2019) Scaling Distributed Machine Learning with In-Network Aggregation. arXiv:1903.06701
42. Hashemi SH, Jyothi SA, Campbell RH (2018) Communication Scheduling as a First-Class Citizen in Distributed Machine Learning Systems. arXiv:1803.03288
43. Hsu A, Hu K, Hung J, Suresh A, Zhang Z (2019) TonY: An Orchestrator for Distributed Machine Learning Jobs. USENIX Conference on Operational Machine Learning
44. Shi S, Zhou X, Song S, Wang X, Zhu Z, Huang X, Jiang X, Zhou F, Guo Z, Xie L, Lan R, Ouyang X Zhang Y, Wei J, Gong J, Lin W, Gao P, Meng P, Xu X, Guo C, Yang B, Chen Z, Wu Y, Chu X (2020) Towards Scalable Distributed Training of Deep Learning on Public Cloud Clusters. arXiv:2010.10458
45. Mashtizadeh AJ, Bittau A, Huang YF, Mazieres D (2013) Replication, history, and grafting in the ori file system. Twenty-Fourth ACM Symposium on Operating Systems Principles 151-166
46. Cohen B (2003) Incentives build robustness in bittorrent, Workshop on Economics of Peer-to-Peer systems 6:68-72
47. Baumgart I, Mies S (2007) S/kademlia: A practicable approach towards secure key based routing. Parallel and Distributed Systems International Conference
48. Freedman MJ, Freudenthal E, Mazieres D (2004) Democratizing content publication with coral. NSDI 4:18-18
49. Wang L, Kangasharju J (2013) Measuring large-scale distributed systems: case of bittorrent mainline dht. IEEE Thirteenth International Conference 1-10
50. Levin D, LaCurts K, Spring N, Bhattacharjee B (2008) Bittorrent is an auction: analyzing and improving bittorrent's incentives. ACM SIGCOMM Computer Communication Review 38:243-254
51. Dean J, Ghemawat S (2011) leveldb-a fast and lightweight key/value database library by google
52. Merkle RC (1988) A Digital Signature Based on a Conventional Encryption Function. Advances in Cryptology - CRYPTO '87