University of Macedonia

Department of Applied Informatics

Decentralized Deep Neural Network Training via Distributed Ledger Technology

A Doctoral Thesis

by

Spyridon Nikolaidis

Supervisor:

Professor Ioannis Refanidis

Thessaloniki, Greece

2022

Devoted to my family

Disclaimer

Acknowledgements

I would like to thank my supervisor, Professor Ioannis Refanidis, for his unwavering encouragement and scientific backing. His suggestions and comments were immensely helpful during the development of this thesis.

Next, I would like to thank the other two members of my doctoral committee, Professor Nikolaos Samaras and Assistant Professor Ilias Sakellariou, for their assistance.

I would also like to thank my family for their patience and unconditional support.

Three-member committee

1. Professor Ioannis Refanidis

   Department of Applied Informatics, University of Macedonia
2. Professor Nikolaos Samaras

   Department of Applied Informatics, University of Macedonia
3. Assistant Professor Ilias Sakellariou

   Department of Applied Informatics, University of Macedonia

Seven-member committee

1. Professor Ioannis Refanidis

   Department of Applied Informatics, University of Macedonia
2. Professor Nikolaos Samaras

   Department of Applied Informatics, University of Macedonia
3. Assistant Professor Ilias Sakellariou

   Department of Applied Informatics, University of Macedonia
4. Professor Konstantinos Margaritis

   Department of Applied Informatics, University of Macedonia
5. Assistant Professor Panagiotis Papadimitriou

   Department of Applied Informatics, University of Macedonia
6. Professor Konstantinos Diamantaras

   Department of Information and Electronic Engineering, International Hellenic University
7. Professor Dimitrios Kalles

   School of Science and Technology, Hellenic Open University

Abstract

The amount of digital data is growing faster than ever, so the need for users' privacy has become critical. In today's world, everything is centralized; be it the Internet traffic, Social Media, Banks, E-commerce or Medical. Tech giants constantly collect the users' data and analyze it, to provide improved services to customers. However, the way in which these data are collected and processed is kept a secret in order to have an edge over competition. Unfortunately, there are more than few incidents where huge amounts of sensitive data were leaked, intentionally or due to lack of proper security. This excessive concentration of infrastructure, immensely affects Artificial Intelligence (AI) research. To properly train today's gigantic neural models, an individual needs access to corporate data centers for computing power and data storage.

The aforementioned factors highlight the necessity for decentralized systems in which there is no central authority to oversee node coordination. Such systems put a priority on data ownership and privacy, equal participation, and fault tolerance. Decentralization has the potential to benefit research as a whole. The greater the number of people who have access to the raw information and tools, the more novel ideas are bound to emerge. The democratization of AI removes entry barriers for individuals and organizations to begin experimenting. It lowers the total cost of developing AI solutions, as communities of programmers and users begin to utilize and enhance related technologies to create more powerful solutions. The openness of such systems, in which everything is made freely available to others, also aids in the development of required skills and, as a result, promotes innovation. However, democratization can help with another crucial component of AI: Even the most complex systems created by highly experienced engineers might be biased, introducing prejudice or serious flaws. Because of the higher variety of sources present in decentralized contexts, data diversity can mitigate biases that may be lurking in narrow datasets.

LEARNAE, the proposal presented in this thesis, attempts to address all of the aforementioned issues by utilizing novel Distributed Ledger Technology (DLT). By combining multiple DLT networks, it synthesizes an ecosystem in which individuals can train Deep Neural Networks collaboratively. There are no hardware or interconnection specifications that must be met, and everyone can contribute according to their capabilities. The process is completely asynchronous, with no locks caused by slow workers, and all nodes have equal rights to the produced models. When data privacy is a concern, LEARNAE can be configured to prevent training data from being shared. The generated swarm is based on purely peer-to-peer topology, with no need for central authorities. These

features, combined with data duplication, result in a network with no single-point-of-failure. The coordinating algorithm is platform agnostic, thus any DLT can be used as the underlying infrastructure. The final additions include an incentivization subsystem, which enables LEARNAE to attract participants who have no interest in the generated model, by offering them a reward proportional to their overall contribution.

Table of Contents

List of Tables

List of Figures

1 Introduction

Artificial intelligence has regained scientific interest, mostly due to the availability of big data. The growth of the Internet, social networks, and online sensors have resulted in the everyday production of a tremendous quantity of information. This unprecedented data availability propelled Machine Learning innovation.

Deep Neural Networks is one field of study that has benefitted significantly from this phenomenon. Numerous use cases now need massive models with millions of parameters, and big data has been shown to be critical for their successful training. Many approaches have been suggested by the scientific community for creating more accurate models. Typically, these techniques need high-performance infrastructure, which limits their applicability to big companies and institutions with the necessary financial resources.

Another issue is privacy; anybody who leases computing power from a remote data center must trust an organization with their data. Regrettably, sensitive information has been leaked in many instances, either for financial gain or due to security issues. However, there is a dearth of study on open communities of individuals using commodity hardware, who would like to cooperate in a non-binding and decentralized manner.

Our work on LEARNAE attempts to fill this gap, providing a way to train Artificial Neural Networks, featuring pure decentralization, data ownership and fault tolerance.

1.1 Synopsis of architecture

LEARNAE [1]–[5] introduces a novel approach to resolving the aforementioned issues via the use of novel distributed technologies. It establishes a completely decentralized environment in which individual Machine Learning (ML) researchers may cooperate with equal roles and full access to findings, all without the need for costly equipment. Meanwhile, they may maintain control of their sensitive data by using contemporary permissionless networks.

When it comes to Artificial Neural Network (ANN) training, there are many approaches that claim to be decentralized. However, upon closer examination, one may see that the idea of decentralization itself can take on a variety of forms. Scaling from low to high degrees of decentralization, the literature discusses methods that include a parameter server, a cluster of parameter servers, peers with elevated roles, and ultimately, pure peer-to-peer topologies.

Our approach is based on the last scheme; thus, all participating nodes have the same set of rights, and none of them is essential for the training process to take place. LEARNAE makes use of innovative Distributed Ledger Technology (DLT) to distribute data. The coordinating algorithm is platform-agnostic; the present implementation makes use of two novel technologies: (a) IPFS [6], a decentralized file system, and (b) IOTA [7], a network architecture focused on the Internet of Things (IoT). This approach provides great resilience, since all information is transmitted using gossip protocols, which eliminates the possibility of a single point of failure. LEARNAE uses data parallelism [8][9], where each worker stores and processes the whole model locally, using its own training data.

Following processing on workers, the generated models must be merged. All parameters of the local model are averaged with the corresponding parameters of a chosen remote model during each averaging phase [10]. This introduces additional stochasticity into the system, thus increasing its overall final accuracy. Nodes may also share training data using the same decentralized method in situations where privacy is not a concern.

The collaborative training procedure is designed to operate with topologies that are loosely connected. There is no need for synchronization, and all data remains on the network for peers to consume at their own pace. Additionally, there is no indirect leakage, since the broadcasted models are progressively influenced by the weights of remote models generated by neighbors, making reverse engineering practically impossible.

1.2 Design Decisions

LEARNAE does not require servers or any type of synchronization because it is totally based on distributed peer-to-peer technology. Its target use cases are environments with commodity-hardware nodes and networking infrastructure that may have significant latency and loose connection. Through the use of innovative Distributed Ledger Technology, we are able to collect data from a variety of sources, including lightweight Internet of Things (IoT) devices.

Our proposal employs data parallelism, which means that each worker keeps a local copy of the whole model and executes it using only a portion of the training data. LEARNAE uses weight averaging for model merging, which implies that after the training phase, all model parameters are averaged with the corresponding parameters of a remote worker's model.

Generally, to gather, integrate, and re-distribute averaged data, a central server can be employed. While using a server speeds up training in many cases, it also provides a single point of failure and a bandwidth bottleneck in big networks. This disadvantage can be overcome by increasing the number of servers that work together. When the presence of a server is neither practical or desired, some of the participants' peers are assigned specialized coordinating roles, while continuing to perform all other training activities. At the opposite end of the spectrum are systems in which no node takes on additional coordinating functions, resulting in a truly decentralized environment, as is the case in our design.

The training collaboration might be synchronous or asynchronous. In synchronous designs the coordinating entity guarantees that only results from the same training period are merged. In asynchronous architectures there is no such need, and the results of a worker can be incorporated into the global model using more flexible criteria. Synchronous training may converge faster because it avoids combining models that are less relative, but it may produce locks from sluggish peers, compromising the entire process. While asynchronous training optimizes worker utilization, it suffers from gradient staleness, which implies that by the time a sluggish worker submits their findings, the global model is already out of sync. Although LEARNAE is designed to be asynchronous, it does provide features that, when used in future implementations, may create a configurable level of synchronicity.

Our proposal can work in situations where participants are hesitant to share sensitive training information. Training-data related communications are inhibited in such cases, and all sent data consists only of models developed by nodes after their training or averaging sessions. In that way the network indirectly leverages the useful information contained in all training data, through the models these data produced.

1.3 Methodology

1.3.1 Proof of Concept

In [1], we proposed the fundamentals of a novel architecture (LEARNAE) that utilizes different types of Distributed Ledger Technology, to create an ecosystem for decentralized ANN training.

The concept made the assumption of loosely connected peer-to-peer topologies with unstable connections and unexpected downtimes. In that study, we specified four distinct roles from which nodes may choose, based on their processing power and data availability. The idea was evaluated by simulating a training swarm of 10 peers, using virtualization methods on a single computer.

1.3.2 Real-world deployment

In [2], we evaluated the proposed algorithm in real-world scenarios. LEARNAE was installed on a 15-computer local network using commodity hardware and networking. The experiments examined the averaging process and resulted to tangible gains in model accuracy.

1.3.3 Resilience Study & IoT Embedding

In [3], we extended the deployment to a group of 20 Virtual Private Servers (VPS) and assessed the resilience provided by data duplication. This was accomplished by implementing a new subsystem that simulated network disruptions and peer downtime. LEARNAE was able to withstand critical disconnections with no degradation to the produced model's performance. Additionally, that study introduced a novel way for embedding low-energy IoT sensors without compromising the overall decentralized philosophy.

1.3.4 Basic Peer Incentivization

In [4], we added another piece to the puzzle: a mechanism for incentivizing peers to join the training swarm, even if they have no interest in the neural network generated. This is accomplished via the incorporation of a reward subsystem within LEARNAE; as a result, peers who contribute to collaboration may earn a proportionate digital payout.

This first implementation served as a proof of concept, so it was rather simplistic: Every time a peer improved his local model by using the remote model of a neighbor, the peer sent a direct micropayment to that neighbor. For this scheme to work we made the assumption that all participants would acknowledge the help offered to them, and give the appropriate rewards.

Of course, this is not always the case, so in the next development phase we significantly enhanced the incentivization algorithm, to anticipate peers who attempt to evade sending rewards to others.

1.3.5 Advanced Peer Incentivization

Our work in [5] was a major paradigm-shift regarding incentivization. We proposed a way of utilizing a novel distributed concept, Decentralized Autonomous Organizations, to consolidate the rewarding algorithm. Prior to training session, all participants have to lock a specific amount of digital assets, declaring in that way their commitment to the process.

During the collaborative training, peers exchange metadata regarding the level of contribution for all participants. The result is the creation of a shared ledger which contains the quantified information about the help each node has offered to the swarm. The consistency of this ledger depends on the honesty of the peers, so the whole scheme works well when the majority is benevolent.

After model training, each peer is rewarded according to its reputation in this shared ledger. The conducted experiments showed that our algorithm managed to greatly mitigate reward-evading attempts by a high percentage of malicious actors.

1.4 Thesis Contributions

Unlike all previous decentralized systems, LEARNAE makes no reference to a common model. Rather than that, each participating peer maintains its own model and makes use of the knowledge of its neighbors to improve it. The majority of similar methods make an effort to spread execution in order to reduce training time. It is critical to emphasize that LEARNAE takes a unique path: It focuses not only on improved models, but also on completely democratizing the process, by prioritizing the following features:

- *Peer-to-peer*. With a true peer-to-peer architecture, all nodes have the same level of access to the neural models generated. None of them provides an essential or privileged role.
- *Resilience*. The whole procedure is immune to node failures and large-scale network disruptions, ensuring that there is no single point of failure.

- *Persistency*. All (meta)data may be optionally retained on the network, as long as the policy of the members permits. This is critical in situations when additional nodes may join the training at any point throughout the process.
- *Privacy*. The coordinating algorithm may be set to run in a privacy-preserving mode, in which peers cooperate without sharing sensitive data.
- *Polymorphism*. Participants may select from many distinct roles, based on the processing power and training data availability.
- *Heterogeneity*. Due to the fully asynchronous nature of the scheme, significant hardware differences are mitigated, thus there are no locks.

To the best of our knowledge, no other proposal sets and fulfils these priorities. The research conducted within this thesis resulted in the following publications:

- 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, Cham, 2019, pp. 286–298. doi: 10.1007/978-3-030-20257-6_24.
- S. Nikolaidis and I. Refanidis, "Privacy preserving distributed training of neural networks," Neural Comput & Applic, vol. 32, no. 23, pp. 17333–17350, Dec. 2020, doi: 10.1007/s00521-020-04880-0.
- S. Nikolaidis and I. Refanidis, "Using distributed ledger technology to democratize neural network training," Appl Intell, Mar. 2021, doi: 10.1007/s10489-021-02340-3.
- S. Nikolaidis and I. Refanidis, "Incentivizing Participation to Distributed Neural Network Training," in Proceedings of the 22nd Engineering Applications of Neural Networks Conference, Cham, 2021, pp. 364–374. doi: 10.1007/978-3-030-80568-5_30.

1.5 Thesis Structure

The rest of this Thesis is structured as follows:

Section 2 presents fundamental aspects relative to our work, such as Artificial Intelligence, Machine Learning, Neural Networks, Distributed Computing, Distributed Ledger Technology, Gossip Protocols, Distributed Hash Tables, Blockchain, and Internet of Things.

Section 3 presents our initial research, simulating a LEARNAE network using 10 Docker images on a single machine. This study offered insight about various parameters of the proposed environment.

Section 4 presents the experiments conducted on a real-life deployment. For this purpose, we used a typical local area network comprised by 15 commodity personal computers. This deployment was enhanced with the implementation of LEARNAE's proposal regarding IoT, where lightweight sensors were emulated by using Single Board Computers. Additionally, we studied and quantified the resilience of our architecture during network disruptions.

Section 5 presents our first incentivization mechanism, which aims to bring aboard peers who have no interest in the produced neural model. This is achieved by rewarding the helpful nodes with digital assets.

Section 6 consolidates and upgrades the incentivization algorithm, to mitigate the effect of malicious participants, by introducing the concept of Decentralized Autonomous Organizations.

Section 7 presents our conclusions on our study regarding LEARNAE, and poses future research directions.

2 Background

2.1 Artificial Intelligence

2.1.1 Steps towards Machine Learning

Conventional programming is a sequence of commands instructing a computer what to do in an explicit way. To deal with problems where the solution cannot be strictly formalized, we need heuristic methods; algorithms that manage to achieve good-enough results in acceptable timeframes, by following non-typical and non-optimal approaches. This area is divided into five main sectors:

(1) *Searching*

(2) *Pattern Recognition*

(3) *Learning*

(4) Planning

(5) *Induction*

The roughest way to solve a problem is to randomly scan for solutions in a -usually vast- solution space. *Pattern Recognition*, combined with *Learning*, can use the experience accumulated by *Search*, to construct generalizations that can further reduce searching time. *Planning* analyzes the problem and can replace *Search* with a significantly smaller, applied on an optimized solution space. During *Induction*, the system attempts to reconstruct the problem's environment. The created model is then used to solve similar problems in a broader area.

There is a minimum amount of information we need to have about a problem to be able to solve it. In cases where the system searches for a solution by consecutive trials, we must have some kind of comparator, a function that takes a pair of trial outcomes and returns the best one. Assuming that the relationship between trials is transitive, thus

$$IF\ (\boldsymbol{Trial_1}\ isBetterThan\ \boldsymbol{Trial_2})\ \&\ (\boldsymbol{Trial_2}\ isBetterThan\ \boldsymbol{Trial_3})$$
$$THEN\ (\boldsymbol{Trial_1}\ isBetterThan\ \boldsymbol{Trial_3})$$

we can define some kind of progress and get the best-found result in a given timeframe. The process can be significantly optimized if we have extra structure on the search space, like an *index of similarity* between two points. In this way we can send the exploration to more promising directions. Such a structure is called a *heuristic connection*. In most AI systems this structure is the *Loss Function*, which takes the problem's parameters as arguments and returns a value that indicates the quality of a candidate solution.

2.1.1.1 Searching

A popular way of searching unexplored solution spaces is *Hill Climbing*. Assume that there are N inputs, $x_1, x_2, \dots, x_N$, and one output $L(x_1, x_2, \dots, x_N)$. The goal is to maximize L by adjusting the N input values. The problem is that we are not aware of an analytical form for L function, thus we cannot use conventional differentiation methods. An approach is to randomly select a point and explore the area around it, always following the direction with the steepest ascent. The process is based on repeatedly estimating the gradient component $\partial L/\partial x_i$ separately for each x_i coordinate. This is the most fundamental form of *Hill Climbing*, while a large number of more sophisticated variations have been proposed.

An obvious problem of this method is that it can be trapped in a local peak which is not a satisfactory optimum. The system could overcome this dead-end by increasing the searching step. It is important to outline that there is no generic solution that applies to all problems. An efficient exploration pattern should probably include multiple techniques executed iteratively or even recursively.

2.1.1.2 Pattern Recognition

An intelligent system must be able to scan the input data and group them into categories, by extracting the *heuristically significant* features. The methods for such classification may vary, from simple matching against predefined prototypes, to complex extraction of important properties contained in a *property-list*. Perhaps the most challenging problem of this procedure is how to invent new heuristically-useful properties in order to create a recognition scheme.

There is an unlimited number of techniques, and their combinations, a system can use to solve a problem. To be successful it has to choose an optimal-enough subset of them, in order to finish the job using realistic resources. After the classification process, the objects grouped together should share a common characteristic that is of heuristic value [11]. For example, if the input objects are geometrical shapes, a pattern recognition system could extract a useful property by counting the number of corners, as shown in Figure 1.

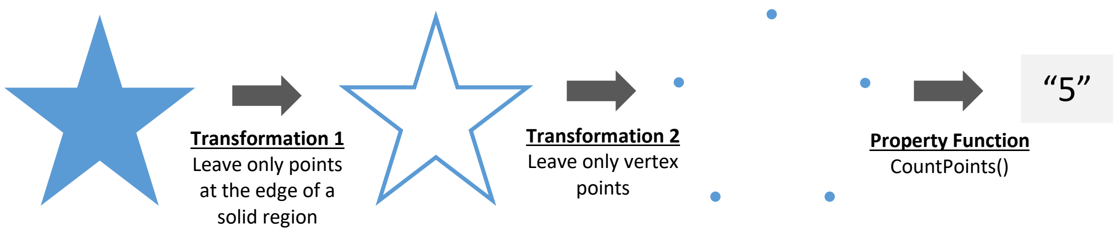


Figure 1: Image transformations for property extraction

2.1.1.3 Learning

A good way to solve a new problem is to follow a method called *Basic Learning Heuristic*. This method suggests that we should benefit from past related knowledge, by trying similar solutions that succeeded on similar problems. Obviously, a new problem will never be exactly the same as previous ones, so we have to create a generalization that can lead to useful inductive inference.

The learning capabilities of a system can be based on a reinforcement process. Such a system generates a variety of behaviors and uses a *Trainer* who applies a *Reinforcement Operator*, Z, on them. The role of this operator is to judge the quality of these behaviors, amplify the promising ones and diminish all others [12].

Suppose that, under predefined environment conditions, we want the system to make a specific choice. During the n^{th} trial we can reward this decision by amplifying its probability, p_n; this can be achieved if we apply the reinforcement operator Z_+:

$$p_{n+1} = Z_+(p_n) = \theta \cdot p_n + (1-\theta) \quad 0 < \theta < 1$$

In case the system makes a wrong decision and must be discouraged, we can apply a negative reinforcement [13]:

$$p_{n+1} = Z_-(p_n) = \theta \cdot p_n$$

This reinforcement scheme has a drawback: Its dependence on a rigid *Trainer*, which can limit the system's ability to solve more complex problems. This issue can be mitigated by a method called *Secondary Reinforcement*, which suggests that the machine can learn to generalize what the *Trainer* does. This can be seen as consecutive self-reinforcement corrections regarding relevant subproblems. The heuristic concept behind this is that any signal which in the past was linked to, for example, positive reinforcement is likely to indicate that something good has just happened.

2.1.1.4 Planning

Planning refers to the process of analyzing a problem, splitting it into a number of subproblems and selecting which of them to investigate, in order to reduce the resources needed for the final solution. The *Logic Theory Machine* [14][15] is considered to be a landmark in heuristic programming, and was a first attempt to prove theorems in logic.

Given that for a complex problem there is a vast number of inference methods that can be applied, perhaps the most challenging difficulty is to find an efficient way to decide where and when to use each one. This is not an easy task, because in a specific stage a method may not directly solve the problem but instead may help creating new subproblems that can assist to finding a solution.

The most straightforward concept of planning is using a *simplified model*. The first step is to find a similar problem but with less detail and complexity, that can be solved with a set of simpler methods. Then we can use this solution as a plan for the original problem. Probably the selected methods will need to be expanded in detail to keep up with the new requirements. But this enhancement will *add* to the total search time, not *multiply* it.

2.1.1.5 Induction

A machine with inductive ability can construct general statements about events beyond its recorded experience. Let's assume a system that groups statements into "true" and "false" ones. The system is initialized by a large number of labeled data which can easily recognize afterwards. But how can the system generalize the acquired knowledge to label new statements? An efficient method is to generate a formal language where the true statements are grammatical and the false ones are not. Using this language and obeying its rules we could create more statements, and presumably these will tend to be more like the true ones [16]–[19]. If the predictions fail to be consistent with new inputs, the system can periodically make small changes to the existing rules.

2.1.3 Unexplainability and Incomprehensibility

For decades AI systems were in fact a digital representation of the distilled knowledge of field experts. Frequently based on *decision trees*, were the perfect way of modeling human decision making. Thus, the function of those systems was naturally understandable by developers and even users. During the last years there was a cataclysmic paradigm shift in the leading AI methodology, towards Machine Learning systems based on Deep Neural Networks. Such systems manage to achieve exceptional performance by utilizing Big Data. Although these systems can produce optimal results with minimal human assistance, the sacrifice that has to be made is about explainability, comprehensibility, and transparency [20][21]. This lack of understanding is less in narrow AI systems applied on limited domains, but skyrockets when the application field broadens.

We are approaching an era when the size and complexity of Artificial Deep Neural Networks will exceed those of the human brain, generating results which rely on billions of contributing factors. Even in cases where the AI system can provide a reasoning on the solution it proposes, this explanation may be either too long to be surveyed [22][23], or too complex to be understood [24]. For example, a neural network could potentially be converted to a huge decision tree of *if* statements. That would not make it understandable, but just human-readable. As a result, developers face a dilemma: Either oversimplify their model to grasp understandability and lose in terms of accuracy, or allow the incomprehensible complexity and achieve optimal, often superhuman, results. Perhaps this aspect of DNN should not be surprising, since they are modeled after the very physical structure of the human brain, which is still considered a black box.

2.2 Artificial Neural Networks

2.2.1 General Structure

An *Artificial Neural Network* (ANN) is a model trying to simulate the structure and functionality of the biological brain. Its building block is the *Artificial Neuron*, an entity that transforms multiple inputs into one output. The inputs of a neuron are weighted, thus each one is multiplied by a different numeric value; the weighted inputs and a *bias value* are added together (Figure 2).

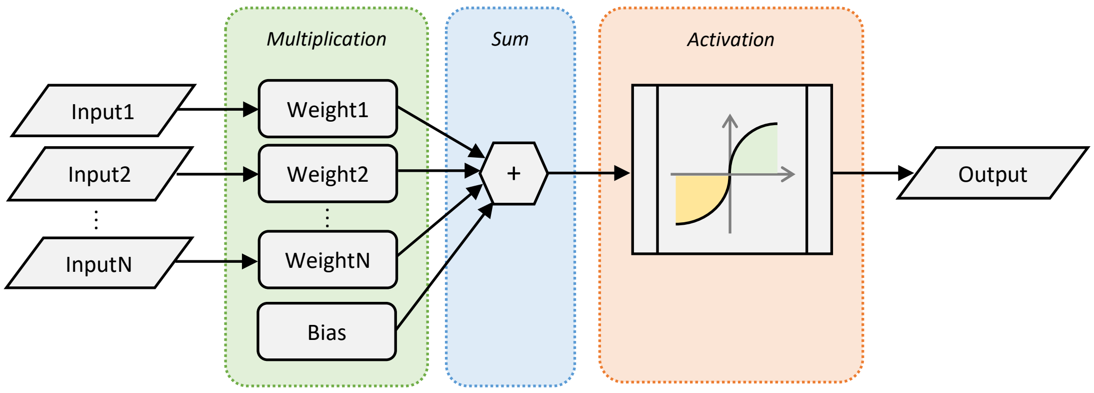


Figure 2: Workflow of an artificial neuron

The result serves as the input of an *Activation Function*, F, which determines the neuron's output:

$$y(k) = F\left(\sum_{i=0}^{m}\big(w_i(k)\cdot x_i(k)\big) + b\right)$$

The activation can be any mathematical function, and is selected according to the use case. The most useful functions are *Step*, *Linear* and *Sigmoid*.

Step function is binary, thus it has only two possible output values. Which one will be the result depends on whether the input exceeds a threshold value:

$$y = \begin{cases} A\,, input \geq threshold \\ B\,, input < threshold \end{cases}$$

Artificial neurons using this function are called *perceptrons* and are often used in the last layer for solving classification problems. In contrast, *Linear activation functions* can do simple transformations and are common in input layers. *Sigmoid*[1] *function* can be calculated with low processing cost, a feature important when dealing with a large number of weight updates.

[1] https://en.wikipedia.org/wiki/Sigmoid_function

The true potential is revealed when we connect multiple artificial neurons into layers, where the output of each layer serves as the input of the next one, forming a Neural Network (Figure 3). The connections between neurons are not random; researchers have come up with several standardized topologies, depending on the type of the problem that must be solved.

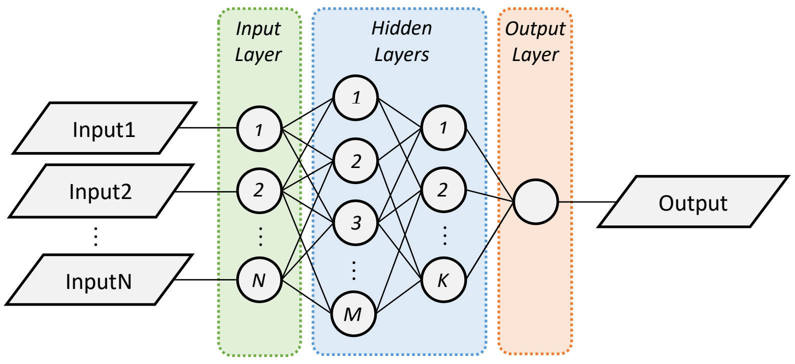
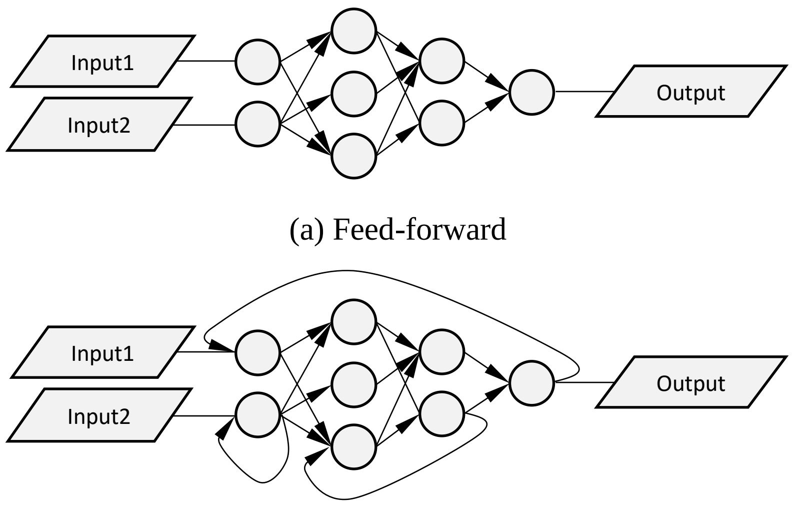


Figure 3: Structure of an Artificial Neural Network

Based on their interconnection topology, Artificial Neural Networks can be divided into two basic classes, *Feed-forward* and *Recurrent* (Figure 4). *Feed-forward* networks (FNN) form an acyclic graph, where the information flows in one direction from input to output. In *Recurrent* networks (RNN) neurons may supply data to other neurons in previous layers, forming a semi-cyclic graph. The ultimate goal of all methods is to set the values of weights and biases in a way that minimizes the model's cost function.



(b) Recurrent

Figure 4: Examples of neural network topologies

2.2.2 Learning Methods

The major paradigms of Machine Learning are (1) *Supervised*, (2) *Unsupervised* and (3) *Reinforcement*. The literature contains a large number of algorithms for each one of them.

2.2.2.1 Supervised Learning

This technique requires training data that contain pairs of input and anticipated output. During the learning phase, training data enter the system and the model's parameters are adjusted to agree with the desired outcome. Supervised Learning can also be referred as *Classification*. There is a wide range of classifiers that can be used, like *Multilayer Perceptron*, *Support Vector Machines*, *k-Nearest Neighbor*, *Naive Bayes*, *Decision Tree*, etc. A downside is that choosing the right classifier is not a strict deterministic process, but most of the times is a matter of experience and intuition. After training there is a validation stage where the neural network is checked against a limited testing dataset unknown to the system.

2.2.2.2 Unsupervised Learning

This method adjusts the model's parameters trying to minimize a given cost function. Unsupervised Learning usually targets estimation problems like statistical modelling, compression and clustering. The difference compared to other techniques is that the input data are unlabeled. The cost function is essentially a numeric representation of the solution's quality.

2.2.2.3 Reinforcement Learning

During Reinforcement Learning usually no data are given in advance. Input is generated by the interaction with the environment and the model is readjusted to maximize some notion of long-term reward. Reinforcement Learning may use several algorithms to find the policy that generates the maximum reward. A usual weakness of this method is the extremely large number of possible policies. Many methods have been proposed to overcome this weakness, including *Value Function Approaches* or *Direct Policy Estimation*. This type of machine learning generally excels in cases where there is a trade-off between a *short-term* versus a *long-term* reward. It can be successfully applied to tasks that require sequential decision making, such as telecommunications and games.

2.2.3 Evolutionary Artificial Neural Networks

Artificial Neural Networks adapt to input data by changing their parameters. It is worth noting that there is a special class of ANN that support an additional adaptation process, beyond training, called *Evolution*. Evolutionary Artificial Neural Networks (EANN) have the ability to alter various aspects of their functioning, using methods such as *Learning Rule Adaptation*, *Input Feature Selection*, *Connection Weight Initialization*, *Rule Extraction*, etc. In that way EANN have better ability to adapt to dynamic environments undergoing significant changes, with no need for human intervention.

Evolutionary Algorithms (EA) is a class of *population-based* stochastic search algorithms, based on concepts found in natural evolution. EA are useful in cases of complex problems with many local optima, since they are less likely to be trapped than gradient-based explore methods [25][26]. Evolution in ANN can be manifested in three major levels: *Connection weights*, *architecture* and *learning rules*. Thus, in practice, EANN have the ability to adapt the connection weights, alter their model's topology and discover novel learning rules.

2.2.4 Stochastic Gradient Descent

Machine Learning is an effort to analyze a data distribution in order to extract useful conclusions that will allow predictions on future data. A data distribution is a *probability distribution* D over a data domain Z. If X is a set of images and Y a set of words, a prediction task could work over the domain $Z = X \times Y$ to label each image according to what it contains. For the training process the system could use a subset of the data, $z_1, z_2, \ldots, z_n \in Z$. The function of the machine learning model is to generate a prediction for any given data point $z \in Z$. The prediction's quality is measured by differentiable non-negative scalar-valued *loss function*, $l(\theta, z)$, where θ are the parameters of z. Let's denote by L the *average training loss* [27] under a data set $S = (z_1, z_2, \ldots, z_n)$:

$$L(\theta, S) = \frac{1}{n}\sum_{i=1}^{n} l(\theta, z_i)$$

The aim of a Machine Learning process is to find parameters θ that minimize this loss. The dominant algorithm for training neural networks is the mini-batch *Stochastic Gradient Descent* (SGD) [28][29][30][31][32]. Given an initial point $\theta_0 \in \Theta$, SGD aims to iteratively minimize the stochastic gradient. The iterations are

$$\theta_t \leftarrow \theta_{t-1} - \eta_t \cdot g(\theta_{t-1}, B_t)$$

where B_t is a random subset of training examples and η_t the *learning rate*. For any $\theta \in \Theta$ and $B \subset S$, the estimate of the objective's gradient is

$$g(\theta, B) = \frac{1}{|B|} \sum_{z \in B} \nabla\, l(\theta, z) + \lambda \nabla R(\theta)$$

Some of the most commonly used variants are *SGD with momentum* [33][30][34], *Nesterov momentum* [35][34] and *Adam* [36]. Table 1 demonstrates the gradient iterations of the first two.

Table 1: Gradient iterations of popular SGD optimizers

SGD with momentum	Nesterov momentum
$\boldsymbol{v_{t+1} \leftarrow \gamma \cdot v_t + g(\theta_t, B_t)}$	$v_{t+1} \leftarrow \gamma \cdot v_t + g(\theta_t, B_t)$
$\boldsymbol{\theta_{t+1} \leftarrow \theta_t - \eta_t \cdot v_{t+1}}$	$\theta_{t+1} \leftarrow \theta_t - \eta_t \cdot g(\theta_t, B_t) - \eta_t \cdot \gamma \cdot v_{t+1}$

According to the above, these optimizers fallback to plain SGD for $\gamma = 0$. Considering a constant learning rate $\eta_t = \eta$, at a given iteration t:

$$\theta_{t+1} = \theta_t - \eta \cdot v_{t+1} = \theta_0 - \eta \cdot \sum_{u=0}^{t} v_{u+1} =$$

$$= \theta_0 - \eta \cdot \sum_{u=0}^{t} \sum_{s=0}^{u} \gamma^{u-s} \cdot g(\theta_s, B_s)$$

For any fixed $\tau \in \{0, \dots, t\}$, the coefficient of the gradient $g(\theta_t, B_\tau)$ in the update above is

$$\eta \cdot \sum\nolimits_{u=\tau}^{t} \gamma^{u-\tau}$$

To evaluate the contribution of a specific mini-batch gradient, we define the *effective learning rate*, η^{eff}, as the value of this coefficient at the end of the training:

$$\eta^{eff} = \lim_{T\to\infty} \sum_{u=\tau}^{T} \eta \cdot \gamma^{u-\tau} = \frac{\eta}{1-\gamma}$$

2.2.5 Neural Model Averaging

One way to parallelize neural network training, is to partition data and distribute them to different nodes. Local models can then be averaged every few mini-batches. The challenge in a such parallel process is to exploit the extra computing power and, in the same time, mitigate the communication cost. Neural network training is often a non-convex process. Roughly speaking, increasing the mini-batch size can give a better estimate of the gradient and, in result, can lead to a better convergence rate [37]. In a traditional distributed scheme, multiple nodes produce gradients of different minibatches; the gradients are then gathered and reduced by a node; finally, the averaged parameters are redistributed to all nodes in order to update their local models. It has been shown that increasing mini-batch size does not always result to a better model [37].

An alternative approach is to average parameters instead of gradients, reducing in this way the frequency of data exchange. If parameters are averaged after each weight update, the process is equivalent to gradient averaging. But if averaging is done every n minibatches, update can be described as

$$\theta_{t+n} = \theta_t + \sum_{i=1}^{n-1} \alpha \cdot g_{t+i} = \theta_t + \sum_{i=0}^{n-1} \alpha \cdot \frac{\partial}{\partial\theta} F(x, \theta_{t+i})$$

$$\overline{\theta_{t+n}} = \bar{\theta}_t + \sum_{i=0}^{n-1} \alpha \cdot \frac{\partial}{\partial\theta} \bar{F}(x, \theta_{t+i})$$

where θ is the model parameter and α is the learning rate. In case θ changes every n updates, this process can be viewed as an approximation to gradient averaging. Additionally, as proved in [38][39], model averaging for convex problems is guaranteed to converge, so an unsupervised pre-training phase could direct the process towards areas of minima that can offer better generalization.

2.3 Distributed Computing

2.3.1 Concerns

Large-scale cloud computing, despite many years of availability, remains out of reach for the majority of researchers. Although there are many successful platforms, like Hadoop[2] and Spark[3], backed by tech giants (Google, Microsoft, Amazon), average users trying to setup an infrastructure still have to deal with great difficulties. The reason is that almost all of these platforms were designed having in mind on-premise installations at large scale. A novice user has to decide on many, sometimes confusing, issues like *instance type*, *cluster size*, *pricing model*, *programming model*, *task granularity*, etc.

2.3.2 Communication Cost

Most of the modern distributing computing frameworks are inspired by the prevalent concepts *Map* and *Reduce*. During *Map* stage, nodes apply a computation on parts of the input data, producing an intermediate type of information. Next, they exchange this information with their peers. Finally, in the *Reduce* stage, they calculate the final results by applying a computation on the combined data.

Studies on many distributed systems, including machine learning algorithms, prove that a large portion of the needed time is consumed for exchanging data. Depending on the application, it is observed that 33% to 70% of the overall execution time is spent on communication [40][41].

Let's consider a distributed framework consisting of K nodes that takes N inputs and calculates Q arbitrary outputs. During the *Map* stage, each input is processed locally to generate Q intermediate values. Thus, the network calculates a total of $Q \cdot N$ values, which can be split into Q groups of N values. Then each subset can be used to calculate the correspondent output. During the data

[2] https://hadoop.apache.org
[3] https://spark.apache.org

exchanging phase, and for every output, a node must receive *N* intermediate values to proceed with *Reduce*. Of course, aiming to reduce communication cost, the system will not transfer values to a node that already have them in its local storage.

2.3.3 Distributed systems

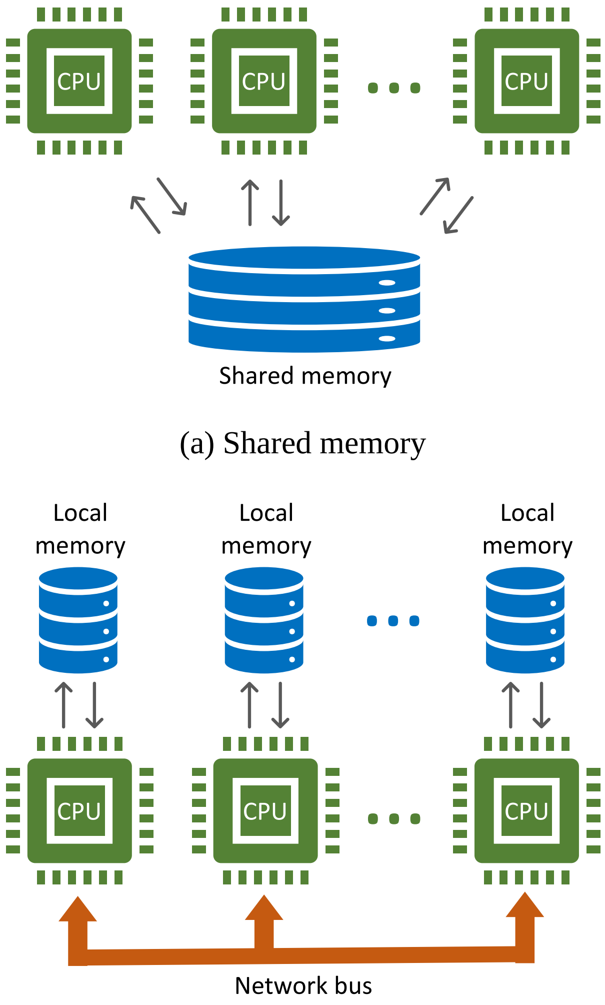
There are many different definitions of *Distributed Systems* in the literature. The one thing all of them have in common is the requirement of multiple processors. Of course, systems with multiple processors extend to a vast architectural space, from single machines to networks of independent workstations.

For example, *Vector Computers* [42] utilize many processors to execute the same operations to different sets of data. *Dataflow and Reduction Machines* [43] execute different operations to different data sets. *Multiprocessor Systems* [44] have several autonomous processors which have access to a shared memory. *Multicomputers* [45] are similar to multiprocessors but, instead of having a shared memory, they exchange information by messaging via a network. An approach that has no need for sophisticated hardware is a group of independent computers connected to a local or wide-area network, running a distributed operating system or application [46].

Experts disagree on which of the above architectures are to be considered *distributed*. Some claim that the term fits only the cases that have geographically dispersed entities, and that all others fall under the term *parallel systems*. A definition that, in our view, optimally addresses this issue is the following: *An architecture can be considered distributed if it consists of multiple autonomous processing units, which do not share memory but communicate using messages over a network of any type and any size.*

In such a system, each processor executes operations on its own set of data, using its own local memory and storage. The above definition of a *distributed system* is not concerned about the type of underlying network, as long as it serves its purpose as a medium every time a peer needs to exchange data.

The network speed has a significant impact to the performance and, thus, the use cases of the distributed system. Architectures with fast and reliable communication are termed *closely coupled*, while systems with slow and unreliable networking are referred to as *loosely coupled* (Figure 5).



(a) Shared memory

(b) Message passing

Figure 5: Types of distributed architectures

2.3.4 Types of distributed applications

When designing a new system, we should choose a distributed architecture for a number of important reasons. Leveraging execution on multiple workstations could result to one or more of the following benefits: (a) *Decreased execution time*, (b) *Increased reliability*, (c) *Increased availability*.

2.3.4.1 Improving performance

A usual reason for choosing a distributed system is the need for higher execution speed. This can be achieved by running different parts of an algorithm on different workstations. If the algorithm is properly optimized for parallel execution, this approach can often result to decreased execution time, despite the unavoidable communication overhead. The same effect can be achieved by using multiple-processor systems with shared memory. An important downside about the latter is that such systems cannot scale to large numbers, such as thousands of processors or more.

A design decision that can have major impact on the result is the *grain of parallelism* that will be used. This is the amount of computation between two subsequent communications. *Fine-grain* systems communicate frequently, while *large-grain* spend more of their time executing computations and communicate more rarely. Since communication is the performance bottleneck in most cases, *fine-grain* parallelism is best suited for *closely couped* systems [47][48]. On the other hand, *large-grain* can be used for both *closely* and *loosely coupled* distributed systems [49][50][51].

2.3.4.2 Improving fault-tolerance

Distributed systems have a fundamental property called *partial failure*. Since each processor is autonomous and independent, a failure in one of them does not affect the others. Reliability can therefore be enhanced by replicating critical functions on several processors. Although replication can be also applied to shared memory multiprocessors, such systems cannot survive intense physical disasters. So, in implementations that handle extremely critical data, the geographic dispersion of a *loosely coupled* distributed system is the only way to overcome this kind of danger.

2.4 Distributed Ledger Technology

2.4.1 The Elements of a Distributed Ledger

In a DLT network, the term *Ledger* refers to the set of data held by the majority of nodes. The most common key-concepts described in literature are:

- *Transaction*: A proposed event which attempts to change the current ledger state.
- *Log*: A list of proposed -but still unconfirmed- events, which are to be subject of network consensus rules.
- *Record*: An event which has already been subject to consensus rules. A record that has not yet been propagated to the network is often tagged as *candidate*.

- *Journal*: The set of records held by a node. Because most of DLT are *eventually consistent* systems, the data on different nodes may not be identical. Sometimes, while network has not reached consensus, they may even be contradicting.
- *Ledger*: The set of records held by the majority of the nodes, already approved by the network consensus. As time passes and new records are appended, it becomes increasingly difficult to revert an existing record.

The participants of a DLT network can have different roles, depending on the level of engagement they wish to have:

- *Auditors*: Check and validate proposed transactions and records. Also, may perform system audits.
- *Validators*: Generate candidate sets of records; they are often called miners or farmers.
- *Light nodes*: Query auditors regarding the current state of the ledger.
- *End-users*: Users who access the network via one of its gateways.

A DLT system can be divided to 3 different layers:

- *Protocol*: A fundamental set of rules which define how the system should operate under every condition. This layer also describes the initial state of the ledger, usually called the *genesis* record.
- *Network*: Defines the rules under which actors can participate to the network, how they communicate, what they can and cannot do. Ensures that critical processes, like proposing and validating transactions, are executed by the proper nodes.
- *Data*: This layer manages the information stored in the ledger. Sets the rules on how a record is structured and in what ways it can be modified. It can also contain rules on how code is executed, in DLT that support such a feature.

Putting it all together, a *protocol* is a theoretic construct which comes to life when utilized by a real *network*, thus a number of independent-yet-connected nodes. Protocol and network layers implement the *data* layer to enable the management of all information related to the DLT system (Figure 6).

- *Dependencies* set the boundaries of the DLT system and determine the need it may have for external resources in order to function properly.
- *Initialization* contains all the processes that are required for the successful launching of the DLT system. This includes the creation of the *genesis* component which describes in detail the starting state of the ledger.

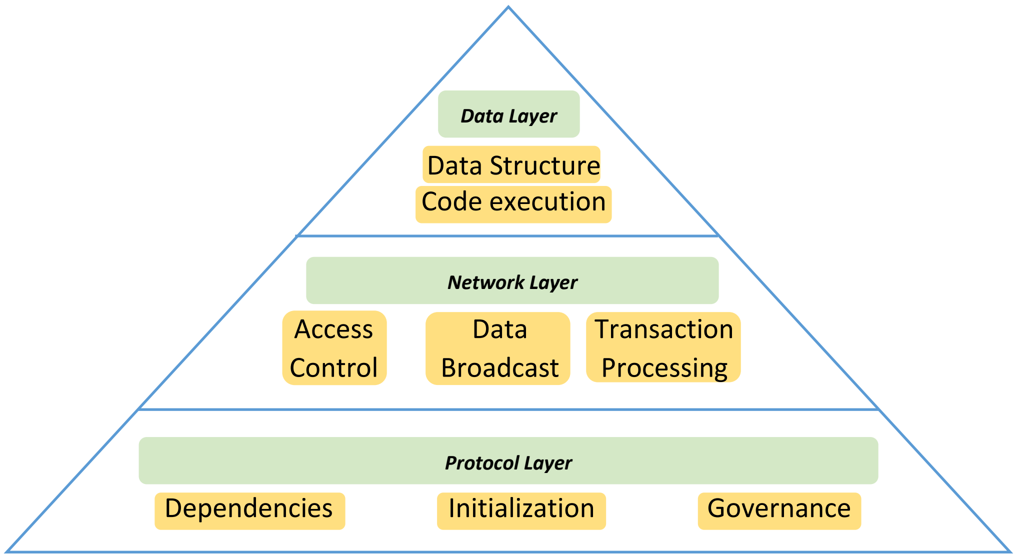


Figure 6: The three layers of a DLT system

- *Governance* refers to the predefined procedures which enable the modifications of the existing protocol. It specifies who and when can embed changes, aiming to fix potential security issues, add new features, or improve performance.
- *Access Control* contains the rules that describe the level of access each participant has to the ledger. This level can escalate from a simple transaction issuing, up to critical governance modifications.
- *Data Broadcast* is the process of transmitting information to connected nodes. In a typical use case, information is propagated to whole network. But in an attempt to reduce the synchronization latency, novel proposals introduce communication between a subset of peers. Such concepts may be found in literature under the term *channeling* (fast transactions between two specific parties) or *sharding* (network divisions sharing a common characteristic).
- *Transaction Processing* refers to the rules that define the way transactions are managed, including broadcasting and validation. Due to the probabilistic nature of most distributed consensus protocols, a record that is impossible (occurred after a checkpoint) or practically infeasible to revert, is called *permanently settled*. The rest -newer- transactions are called *provisionally settled*.
- *Data Structure* defines in strict terms the form of every bit of information stored or transmitted in the DLT network.

- *Code Execution* includes the structure of executable code, in cases where smart contracts are supported.

2.4.2 Challenges of Distributed Consensus

2.4.2.1 Fault-tolerant Distributed Consensus

The problem of reaching consensus in a fully distributed network has been studied by the scientific community for many decades. In contemporary DLT systems, where there is no shared memory, the participants have to agree on the valid global state by exchanging peer-to-peer messages. Each node N_i has its own state S_i, which can be updated as a result of the communication with its neighbors. In the case of open-access networks there is always the scenario of malicious actors trying to manipulate the system to their benefit. In centralized systems, this effect can be easily detected using straight-forward techniques; but in a decentralized environment the attackers can use the established rules in a seemingly legitimate way, trying to influence the consensus algorithm. The attempted manipulation could be either pushing the consensus to a state that yields profit to the attackers, or just sabotaging with contradicting messages, making it impossible for the network to reach *any* consensus. Such attack vectors are referred to as *Byzantine Failures* [52].

2.4.2.2 Byzantine Fault-Tolerant Consensus

A consensus protocol is called *Byzantine Fault Tolerant* (BFT) when it can withstand a number of attacks to the *peer reputation system* which supports its consensus mechanism. The problem can originate from one or more *Faulty-or-Malicious* (FoM) participants. Any distributed consensus protocol must meet the following four requirements [53][54][55] regarding its non-FoM nodes:

(a) *Termination*: Every node must come to a specific result as its output; (b) *Agreement*: Every node eventually ends up with the same output y; (c) *Validity*: If all nodes have the same input x, then all end up with the same output $y = x$; (d) *Integrity*: The final consensus value y must have been proposed by a number of nodes.

For a protocol to achieve these requirements, the underlying decentralized network must satisfy [56] the condition $N \geq 3 \cdot M + 1$, where N is the total number of nodes and M is the number of FoM ones.

2.4.3 Case study: The Nakamoto Consensus Algorithm

The *Nakamoto Consensus* is the emblematic algorithm behind the first widely adopted permissionless blockchain, *Bitcoin* [57], and served as the basis for many other consensus schemes. Its large-scale application allowed researchers to uncover its weaknesses and propose solutions in the form of new and improved consensus algorithms.

Nakamoto proposed the *Probabilistic Finality* specification, according to which a block can either be *accepted* or *rejected*. When accepted, there is still a chance it will be eventually rejected, but the probability of this scenario diminishes exponentially as new blocks are appended to the chain. This incrementally-achieved consensus is an unavoidable feature of any system containing asynchronous processes. Thus, for short periods of time the network may be found in inconsistent states, but in the end it will always reach consensus, a characteristic called *Eventual Consistency*[4].

Nakamoto consensus utilizes the *Proof of Work* mechanism to mitigate *Sybil Attacks*[5], making multiple identities practically impossible. Transactions are broadcasted immediately using *gossip protocols* and in case of conflicts the longest chain always wins. The production of new blocks is assigned to *miners*, participants who wish to dedicate processing power to the network. In return, they are rewarded with newly minted native tokens, and also fees paid by users who conduct transactions. To ensure that every block is sufficiently propagated before the next block is published, the difficulty of the mining cryptographic puzzle is adjusted every 2016 blocks, depending on the global hashing power. With this adjustment new blocks are generated approximately every 10 minutes (Algorithm 1).

Figure 7 shows how a new block is propagated in a peer-to-peer network that is structured as an *undirected graph*. The circles represent nodes and the numbers indicate the number of hops required for the new block to reach a node. Solid arrows are used when a peer informs its neighbors of the new block and sends it to them; dotted arrows show cases where the informed neighbor has already received the new block from another peer. In this example, the new block is generated by node A and propagated to nodes B, C and E in 1 hop. Node E propagates the block to node H (2 hops), but node F already received the block from C (2 hops), so this message from E will be ignored by F. In heterogeneous networks a node may receive a block from a path that may not be the shortest, due to the differences in peer processing power and connection speed.

[4] https://en.wikipedia.org/wiki/Eventual_consistency
[5] https://en.wikipedia.org/wiki/Sybil_attack

Algorithm 1: Overview of the Nakamoto consensus protocol

```
// Join the network
Bootstrap to hard-coded nodes;
start thread: Listen for messages;

// Work cycle
repeat {

      Select longest chain;
      call ComposeNewBlock();

      // Local block generation failed
      if received block from another peer {
            Append block to chain;
            interrupt ComposeNewBlock();
      }

      // Local block generation succeeded
      if ComposeNewBlock() returns block {
            Append block to chain;
            Broadcast block;
      }
}

// Block generation via Proof of Work
function ComposeNewBlock() {

      Create header:
            Embed digest of previous block’s header;
            Embed timestamp;

      repeat {
            Create new nonce;
            Calculate Hash(header|nonce);
      } until Hash has N leading zeros;
      // N sets the mining difficulty

      Embed proposed transactions;

      return block;
}
```

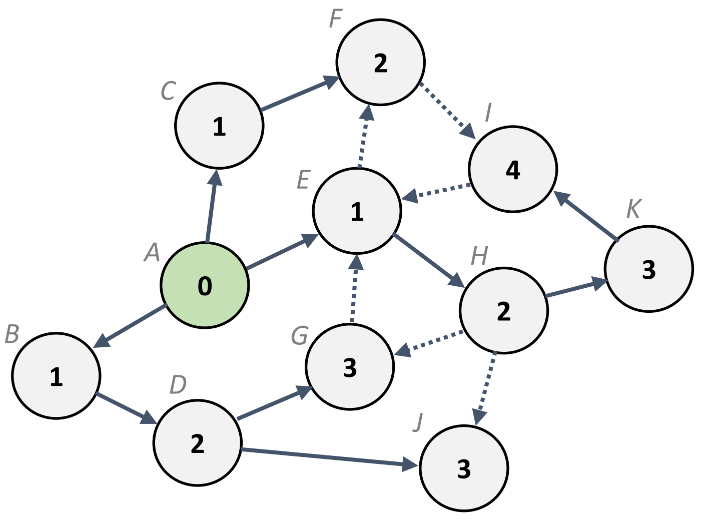


Figure 7: Data propagation using gossip protocol

The resilience of a classic distributed system is expressed simply as the number of FoM actors it can tolerate. For systems based on PoW consensus, tolerance is characterized by the percentage of adversarial hashing power the system is able to overcome. It is proved [58] that the critical condition a network must meet, is that the time PoW needs to generate a new block, to be longer than the time needed for the network to synchronize. When this condition is fulfilled, a majority (more than 50%) of benevolent hashing power will always manage to ensure proper convergence, even for an ever-growing size of blockchain.

2.4.4 The Security-Decentralization-Scalability Trilemma

A decentralized consensus protocol should seek a balance between three critical objectives:

- *Security;* refers to the system's ability to overcome both faulty and malicious peers.
- *Decentralization;* refers to the dispersed distribution of network governance power.
- *Scalability;* refers to the ability of efficiently dealing with rising event throughput and larger network size.

Most of the times there can be a tradeoff between these three features, the importance of which is dictated by each use case. Lower security can lead to higher scalability. For example, decreasing the time interval between blocks in a PoW network, can yield higher transaction rate but also leads to lower security, since it is easier to orchestrate a 51% attack [59]. Generally, more decentralization means less synchronization. So, higher geographic diversity makes censorship harder for suppressing regimes, but also introduces more heterogeneity: The discrepancy in connection speeds creates an unfair division, since peers with low latency have the advantage in winning the race of block publishing and validating. As a result, in a heterogeneous network well-connected nodes can commit consensus attacks with far less than 50% of the total hashing power. All these parameters must be evaluated and adjusted, so the distributed system can serve the purpose it was built for.

2.4.5 Identity management

A DLT user is uniquely identified by their *Public Key Certificate*. In most public systems the user first generates a key pair and then derives their identity as a hash of the public key. This hash can be used as their transaction address and can hold digital assets. To prove ownership and manage these assets, a user has to sign every transaction using the corresponding private key. Due to the absence of a centralized certificate authority, secure key management by the participants is essential [60]. Losing private keys has irrevocable impact, since the user permanently loses its ability to prove ownership.

2.4.6 How DLT can transform Artificial Intelligence

The integration of DLT and AI can be mutually beneficial. Both fields have weaknesses, many of which can be addressed by such a consolidation [61][62]. AI algorithms rely heavily on data to train models that can infer and make decisions. The quality of their results depends on the quality of the input data, which must be consistent and trustworthy. DLT networks have all the prerequisites to assure the integrity and credibility of the provided data. Every bit of information is cryptographically signed, validated, and agreed on by all participants. So, AI systems can be certain that the data have not been tampered with. Distributed ledgers ensure that training will be fed continuously, since the decentralized architecture will overcome any connectivity disruptions. The immutability of the provided data can help to mitigate unaccountability, by providing a trustful infrastructure which retains the training history and makes it easily available to all parties. In cases where the validity of results produced by code execution is a priority, hybrid systems can utilize on-chain execution of AI algorithms via smart contracts. The other way around, AI can offer important services to DLT systems, by analyzing their real-world performance. So, it can infer and propose ways to optimize their data exchanging, their consensus algorithm, etc.

2.5 Gossip Protocols

Generally, routing refers to how nodes are selected to relay data through the network. There are many proposals [62][63][64] aiming to classify routing methods, but *gossip protocols* used by modern decentralized systems have some fundamental differences compared to traditional networks.

2.5.1 Random Walk protocols

In *Random Walk* protocols, data are propagated using random paths. The peers use *hop-by-hop* routing, meaning that the initiator only selects the first relay node, which in turn picks the second and so on, until the message reaches its final destination. The selection is usually random. *Random Walk* protocols are often combined with peer-to-peer network topologies. In early systems designed for anonymous web browsing, like [66], a user randomly selects a peer and sends their message. The peer flips a biased coin to decide whether to send the message to another random peer, or directly to its final destination. The reply from the receiver follows the same path in reverse order.

Other studies [67] propose a dynamic peer-to-peer network, which establishes circuit-based connections using layered encryption. An anonymous route is iteratively set by the nodes on the route. Each node is aware only of a subset of peers and not the entire network. After an *initiator* selects the first node, it randomly selects a *witness* for each hop thereafter, and asks the next hop to extend the route with the assistance of the *witness*. Finally, each node proposes a set of candidates for the next hop, and the corresponding *witness* chooses one of them as the next hop.

There are also proposals [68] for fully decentralized network overlays that operate on IP level. At first, the *initiator* selects a set of nodes to form a route through the overlay network; then, a tunnel is established via these nodes. Using a gossip-based protocol based on *Name-Dropper* [69], node information is stored in a ring model and lookups are carried out using the *Chord Algorithm* [70].

2.5.2 DHT-based protocols

Tasks that are easy to complete in a traditional network, may be challenging in a fully distributed one. For example, the task of locating a node. One solution is to use *Distributed Hash Tables* (DHT) to overcome the lack of a centralized shared storage. The DHT is usually a database with key-value pairs. The keys are generated by hashing a unique piece of information that identifies the indexed data. The values may contain routing information, file contents, etc. To achieve efficient searching for DHT-based networks, several lookup strategies have been proposed. *Kademlia*[6] locates the nodes on their estimated distance using a XOR metric; *Chord*[7] uses a clockwise circle metric, where at each hop of the lookup the distance to the node is decreased at least by half; *Pastry*[8] carries out lookups based on numerical identifiers. DHT structures can work even in cases where nodes are not aware of all other peers. In fact, having a partial view of the network can result to improved scalability, higher resilience to attacks, and better load balancing.

2.6 Blockchain Technology

2.6.1 Definition

Blockchain technology introduces a fully decentralized and secure system based on a distributed ledger. It has no need for a central authority to bootstrap the trust among participants; in fact, it is designed to work among mutually distrustful parties. In contrast to traditional distributed computing with a clear client-server model, blockchains allow participants to be both clients (by issuing events) and servers (by validating events).

With the exception of a few privacy-oriented cases, blockchain events are publicly available, although the participants are identified by a pseudonym, usually an alphanumeric hash. To efficiently achieve consensus, researchers leveraged previous related work, like *Byzantine Fault Tolerant*[9] consensus and *Secure Multi-Party Computation*[10].

[6] https://en.wikipedia.org/wiki/Kademlia
[7] https://en.wikipedia.org/wiki/Chord_(peer-to-peer)
[8] https://en.wikipedia.org/wiki/Pastry_(DHT)
[9] https://en.wikipedia.org/wiki/Byzantine_fault
[10] https://en.wikipedia.org/wiki/Secure_multi-party_computation

In use cases where the real-world identity of the participants is a necessity, a *permissioned* blockchain can offer a solution. In order to join the network, everyone has to undergo an authentication procedure usually contacted by a centralized authority. Given the stronger identification framework, *permissioned* blockchains can implement multi-level access control. The network governance is assigned to a private entity, or a consortium of such [71].

2.6.2 Cryptography Fundamentals

2.6.2.1 Cryptographic Hash Functions

Hash Functions calculate a concise fixed-size output, called digest, from an input of arbitrary size ($Hash(data) = digest$). Hash functions have the following properties:

- Hashing the same input always produces the exact same output.
- Can provide a proof that data were not changed. Anyone can reapply the hash function and verify that it produces the same digest.
- Even a slight change to the input leads to a completely different result.
- They are *preimage resistant*. Given a specific output, it is computationally infeasible to find the corresponding input.
- They are *collision resistant*. It is computationally infeasible to find two different inputs that create the same output and it is unlikely that this will happen unintentionally.

As *computationally infeasible* is considered any attempt to explore a vast output space using brute-force methods. For example, a popular cryptographic hash function is the *Secure Hash Algorithm*[11] with an output length of 256 bits (SHA-256). The digest of this hashing function is usually displayed as a 64-character hexadecimal string (Table 2).

The output space contains 2^{256} (approximately 10^{77}) different hashes, making it resistant to exhaustive search. Modern distributed networks utilize hash functions to create unique identifiers and to ensure data immutability.

There are cases where a system must generate a hash string which is always different, even for the same input data. This can be achieved by the use of a *cryptographic nonce*, an arbitrary number that is used only one time.

[11] https://en.wikipedia.org/wiki/Secure_Hash_Algorithms

The unique digest is produced by hashing a combination of the input data and the nonce value:

$$hash(data + nonce) = digest$$

Table 2: Sample input texts and their digests

Input Data	SHA-256 Digest
`A`	`559aead08264d5795d3909718cdd05abd49572e84fe55590eef31a88a08fdffd`
`B`	`df7e70e5021544f4834bbee64a9e3789febc4be81470df629cad6ddb03320a5c`
`C`	`6b23c0d5f35d1b11f9b683f0b0a617355deb11277d91ae091d399c655b87940d`
`Cryptographic Hash Functions`	`e4526e7dcd04ce6816287b25da39ba08786ef85b74e5a5e05855773643077a61`

2.6.2.2 Asymmetric Cryptography

Using symmetric cryptography if two parties need a secure way to communicate, they need to exchange the key that will be used for both encryption and decryption. But this requires an alternative safe channel, which in many cases cannot be easily established.

Unlike symmetric method, asymmetric cryptography has no need for an already-secure channel. Every actor has a pair of keys, a *private* and a *public* one. The private key is known only to owner, while the public key can be freely broadcasted. The public key is generated from the private key, using well-established one-way mathematical functions. This process can be easily reproduced and its results can be verified by anyone. Due to the nature of these functions, one cannot follow the opposite direction, thus the private key cannot be derived when having the corresponding public key.

Asymmetric cryptography allows participants to verify both the authenticity and the integrity of messages in trustless communications. A useful feature of this method is that private key can be used for encryption and public key for decryption, or vice versa. A sender can encode a message with private key and send it to receiver; the receiver can use sender's public key to verify the transmission.

The other way around, a sender can encrypt a message using receiver's public key; the receiver can authenticate the message by decrypting it with its own private key.

A downside of asymmetric cryptography is that it is slow to execute. In order to enjoy the best of both worlds, in many cases systems use a hybrid approach: The slow asymmetric method is utilized to encrypt a -relatively small- symmetric key, which in turn is used for the rest of the communication. Such an approach speeds up the whole process by greatly reducing the needed encryption overhead. In many blockchain networks, private keys are used to digitally sign transactions, while public keys are used to derive addresses and verify signatures.

2.6.3 Addresses

Most permissionless decentralized systems advocate data ownership via pseudo-anonymity. Participants can freely create as many identifications they want, which are pairs of a private and a public key. These keys are not associated in any way with real-world identities, but all transactions are permanently recorded and publicly available. This last statement is false only in specific cases of privacy networks, where transactions are validated by the distributed consensus, but their details are not accessible.

Blockchain networks often apply hashing functions to public keys in order to generate addresses. An address is a public identification and can represent a *digital token* wallet, an issued transaction, or even a smart contract. Most blockchain networks offer services -called *explorers*- to browse the content and the previous activity of their addresses.

2.6.4 Transactions

Many of the contemporary decentralized cryptographic networks, such as blockchains, are constructed as immutable databases. Changes can only be implemented as cumulative additions to their state. In most of the cases, changes happen through *transactions*. These are interactions between participants and are used to express asset exchanges and -when the network supports it- code execution and data manipulation in smart contracts.

The consensus protocol will not allow a typical transaction to generate or destroy assets. So, in every transaction the total amount of input assets must be equal to the total amount of the output assets, as seen in Figure 8. In this example Bob owns 10 tokens which received at a previous time from $Alice$. He wants to send 7 tokens to $Charlie$, so he issues a new transaction. The algorithm will send 7 tokens to $Charlie$, but at the same time it will send the remaining 3 tokens back to Bob as change, in order to maintain the transaction balance. Depending on the implementation, the change can be sent either to the same or to a different address belonging to Bob.

Thus, what a transaction can alter is the participants owning the engaged assets. Particularly for blockchains, submitted transactions are enclosed to blocks. There can even be blocks with zero transactions, since empty blocks can also contribute to overall security by creating longer chain. In order for a bad actor to impose their malicious version of truth, they will need more resources to produce an even longer chain.

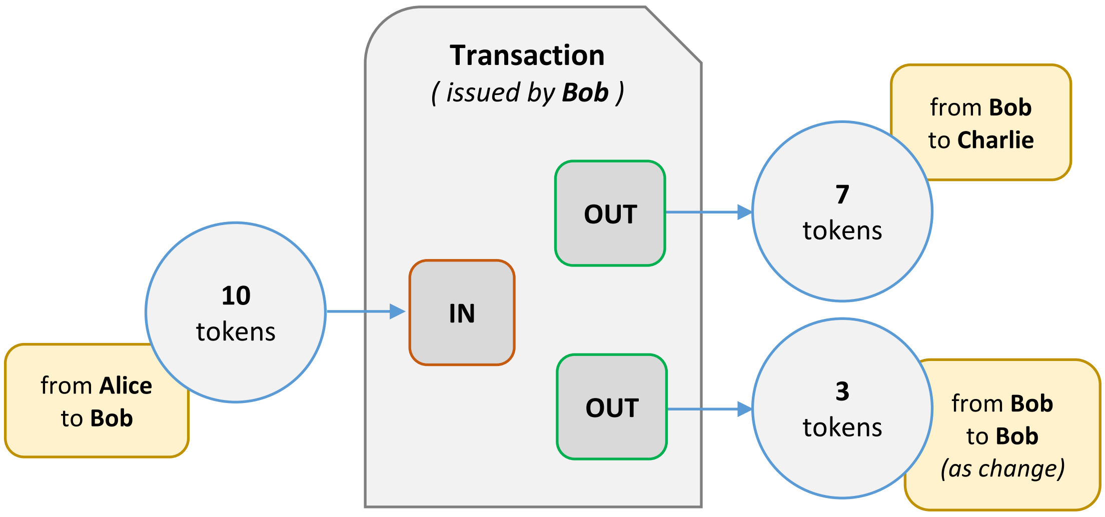

Transaction
(issued by **Bob**)
from **Alice** to **Bob**
10 tokens
IN
OUT
OUT
7 tokens
from **Bob** to **Charlie**
3 tokens
from **Bob** to **Bob** *(as change)*

Figure 8: Example of a Blockchain transaction

To ensure their authenticity, the broadcasted transactions are digitally signed by their issuers and contain their public key. The signing process proves access to the corresponding private key and can be easily verified by everyone.

2.6.5 Blocks

2.6.5.1 Block Structure

Transactions that are consistent to the ledger status and properly signed, are grouped into blocks and propagated to the network. The structure of a block may differ for each implementation, but most networks include the fields shown in Figure 9.

Networks that are based on a *Proof of Work* (PoW) consensus algorithm, utilize the nonce value to create a unique cryptographic puzzle for each block. Nodes will compete to solve it in order to win *mining rewards*. Since solving the puzzle requires processing power, the network is shielded against spamming from bad actors. The amount of protection is proportional to the total hashing power of all participating peers.

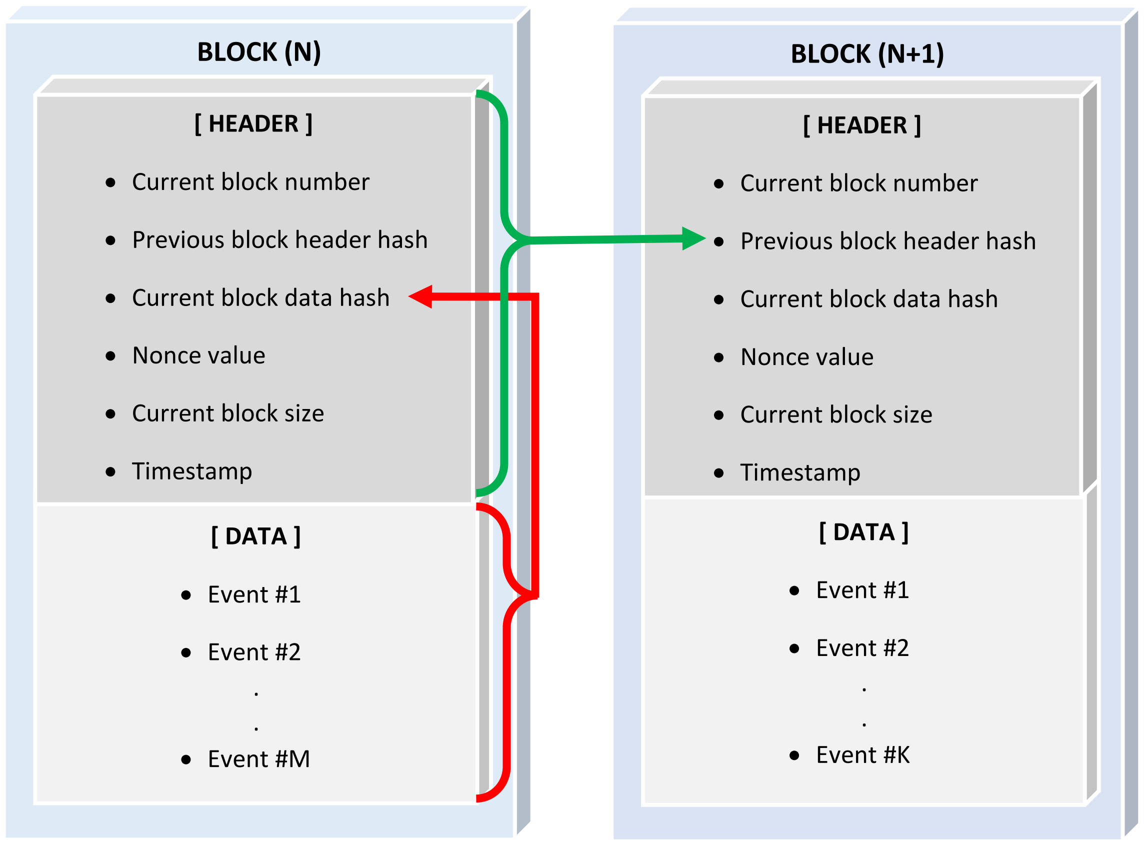


Figure 9: Typical structure and linking of blocks

2.6.5.2 *Block propagation and synchronization*

Block propagation mechanisms can be divided into the following categories:

- *Advertisement-based.* When node A receives the information of a block, it sends an inv message to its neighbors. If node B doesn't have the information of this block, will reply to A and ask for it. Neighbors who already have the information, ignore the message from A. When A receives the reply message from B, sends the requested block.

- *Sendheaders.* This is an improvement to the previous propagation mechanism. Node B sends a $sendheaders$ message to A. When A receives the information of a block, will immediately send the block header to B. Since this method has no need for inv messages, it can speed up the block propagation.
- *Unsolicited push.* When a node generates a new block, it broadcasts it to the network. There are no inv and $sendheaders$ messages, so the speed of block propagation process is further improved.
- *Relay network.* This mechanism uses a transaction pool shared by all peers. Each transaction is designated a global ID, hence mitigating the broadcasted block size and network load.
- *Hybrid.* This method utilizes a combination of push and advertisement propagation. If node A is connected to N peers, will push the block to $\sqrt{N}$ of them and will advertise block's hash to the rest $N - \sqrt{N}$.

The block synchronization mechanism can diversify for different blockchains. Let's assume node A requests synchronization from block B. A typical scenario could include the following steps:

1) Node A sends a $GetBlockHeaders$ message to node B, asking for the latest block.
2) Node B replies to this message by sending the requested block header.
3) Node A requests $MaxHeaderFetch$ blocks, in an attempt to find a common ancestor. The number of blocks returned by B may be equal to or less than the value of $MaxHeaderFetch$.
4) If node A cannot find a common ancestor, will continue to send $GetBlockHeaders$ messages, requesting one block header at a time.
5) When node A discovers a common ancestor, requests synchronization starting from that block. The blocks needed for the syncing will be grouped into packs that cannot contain more of $MaxHeaderFetch$.

2.6.6 Distributed Consensus

The most important function of a distributed system is to establish a *consensus*. There are many models a network can implement to achieve a global agreement regarding its state. There are issues that have to be dealt with, like who publishes the next block and how are conflicts resolved. Things get more complicated because of the lack of trust among the peers; the only information a node knows about its neighbors is their public address, which is usually just a hash string.

For any consensus model to work, blockchain networks need a solid starting point, an initial block called *genesis*. Every peer that joins the network must agree upon this block and, using this as a base, can verify all following blocks. If a valid second blockchain branch appears, most consensus algorithms will approve the longest one because it required the most effort.

2.6.6.1 Consensus Models

2.6.6.1.1 Proof of Work

One of the most widespread consensus algorithms is *Proof of Work* (PoW). In this model the next block is published by the peer that solves first a difficult mathematic problem. Although the problem is hard to solve, a proposed solution can be easily verified by everyone. Many distributed networks need an approximately constant rate of block production. This can be achieved by periodically modifying the difficulty of the PoW puzzle. The difficulty goes up when the total hashing power of the network is increased and vice versa. Such an adaption offers resistance to *Sybil Attacks* from actors who possess extensive processing power. A popular PoW puzzle among modern blockchain networks is using a hash function to produce digests with specific characteristics. For example, let's assume that the proposed block's header is the string "ProofOfWork" and nonce is an integer number. Participating peers could compete for it, by trying to find a nonce for which:

$$SHA256(header + nonce) = digest\ with\ N\ leading\ zeros$$

By changing N, the network can modify the PoW difficulty and with it the rate at which blocks are published. For $N = 1$, a node can solve the puzzle after 16 hashing attempts, as shown in Table 3.

Cryptographic hashing of lengthy data can be an extremely computational intense process. For the short string "ProofOfWork" of this example, a commodity computer would need exponentially increased amount of time for consequent values of N, as shown in Table 4.

On the other hand, when a peer broadcasts its solution, all other participants can instantly verify its validity and reach consensus.

Table 3: Attempts to solve PoW puzzle

Header + Nonce	SHA256 Digest
`ProofOfWork0`	`263a40e975437811bca5aab9ff9a82db6fcd725c0092a6a7a63d9cdf5a8a0997`
`ProofOfWork1`	`ea82db3755316339a31b1cd501c121b44721d6f494dea43f112efbd22631b6be`
`ProofOfWork2`	`5eecdd4dd548898572cd1dfdbbd0cfa0c43ec733832e246c3c042549040e79dd`
`ProofOfWork3`	`abd4ec4dd1703f7bc56b79733d473e1cea6a832fe82cd2c42fdf6c77fe8e7486`
`ProofOfWork4`	`c55da0001787f85554abdb72806bc20675aaf784a5d38e1c9a02ddd04b2a2476`
`ProofOfWork5`	`4f42c8dfd8266fbede41c3bfbc11540924b4c59dafcd523732192d1b60f68f3b`
`ProofOfWork6`	`1b32ee3f06829ce71087996d2f1ba47e3c861292aa7912c94d55e8f9d43f902a`
`ProofOfWork7`	`4f792c842a3c68d48627b066fa35cb012154724de8116fbbe33034a88ddde354`
`ProofOfWork8`	`4cfb10445e93145935864fe05f6b6793f1587635f0b866b8cefdf7572eaff8f8`
`ProofOfWork9`	`be37f6aeb9e8d1d30a62a0b6ea9688b9cd8dcf7c0edffd8f5908bd33527b9e98`
`ProofOfWork10`	`3e7110e326067c72f68c2357a73a384829f426fc8facc77873f48f19a5c45e52`
`ProofOfWork11`	`57635a7f1607d36cc24fe71620df274991c4174df31d0f6268365498c91b8c2c`
`ProofOfWork12`	`44d81a5c83e09ca63af5cbaa79694a69ec7b4f0acd2340e8b94dfe7c2dd7348f`
`ProofOfWork13`	`c01d9cc19907f549d5186554878855d4bd70faa347d55fad08d294efd9dca7f3`
`ProofOfWork14`	`4caed8b6251fbae092edf516eddf4b980a47845f001891fc3bea0bd02a002195`
`ProofOfWork15`	**`0`**`c3bb4677f061c875a48e3f48b3654bd740b7260a33dc2180bb0dfbf7e8f870b`

Table 4: Proof of Work simulation

Requirement *(number of leading zeros)*	Elapsed Time *(seconds)*	Iterations Tested *(number of hashes)*
1	0.016	16
2	0.018	101
3	0.055	3586
4	1.247	177859
5	6.690	878395
6	46.278	6297346

2.6.6.1.1.1 Mining Difficulty Analysis

Let's assume a network based on a PoW consensus algorithm. A random node n will need t_n time to solve the cryptographic puzzle associated to a specific block. As expected, this time depends on the instantaneous probability node n solves the puzzle, P_n. The value of this probability is related to (a) the hashing power, H_n, the node possesses, and (b) the current difficulty of the PoW puzzle, D, dictated by the network's total hashing power. Thus, the probability is:

$$P_n = \frac{H_n}{D}$$

The difficulty is periodically adjusted, so that the expected interval between two consequent blocks, T, is near-constant. Using the properties of exponential distributions and having N as the set of participating nodes:

$$T = \frac{1}{\sum_{i\in N} P_i} = \frac{1}{\sum_{i\in N} \frac{H_i}{D}} = \frac{D}{\sum_{i\in N} H_i}$$

The conclusion is that difficulty D is equal to the average block duration for block generation, multiplied by the network's total hashing power:

$$D = T\left(\sum_{i\in N} H_i\right)$$

Thus, as the network grows, PoW protocol increases difficulty and vice versa. Combining the above equations:

$$P_n = \frac{1}{T}\frac{H_n}{\sum_{i\in N} H_i}$$

The last equation shows that the probability for a node to win a block, increases by its hashing power but decreases by the combined power of all other participants.

2.6.6.1.2 Proof of Stake

Proof of Stake (PoS) consensus model is based on the principal that the more resources a user has invested (staked) into a system, the more likely they will work for its success. Most blockchain networks have native tokens the value of which is related to the amount of utility the system offers, as a store of value, transaction mechanism, decentralized application platform, etc. Peers who invest on these tokens are interested in the well-being of the ecosystem, so they can be trusted that they will never act maliciously. PoS also includes a cryptographic puzzle, but its difficulty decreases with the amount of staked assets. So, the energy required for block production can be significantly mitigated for high staking nodes.

In order to enhance the engagement needed for staking, most systems require the staked assets to be locked in a way that they cannot be spent for a period of time. The likelihood of a peer being selected to publish a block is tied to the ratio of its stake. For example, if a peer owns 1% of the total tokens, it will be selected 1% of the time. This directly proportional method is the simplest form of PoS consensus. Many other methods have been designed adding more complexity to the selection algorithm, such as voting systems (Delegated PoS - DPoS), staking age factors, etc.

An important advantage of PoS compared to PoW, is that it does not need intensive calculations and, thus, high energy consumption. For this reason, many existing ecosystems are redesigning their PoW consensus model in an attempt to be more friendly to the environment.

2.6.6.2 Conflict resolution

In all distributed systems, due to the lack of a central coordinating entity and the unavoidable network latency, for short periods of time there are different versions of truth propagating the network. The system has to deal with these conflicts as fast as possible in order to maintain its consistency.

For example, in a blockchain network two peers can broadcast blocks at almost the same time. It is possible these conflicting blocks are accepted by different subsets of participants, creating a temporary split to the blockchain, as shown in Figure 10.

Conflicts can occur even if there is no malicious intent. Usually, the reason behind them is the way information is propagated to the network via gossip protocols, allowing peers with slightly diverse views. In most implementations such a conflict will be solved after the network agrees on the next published block. That block will be attached to one of the conflicting blocks, making one of the branches longer than the other.

So, everyone will consider the longer chain as valid, and the other block will be flagged as orphaned. Since the conflicting blocks may contain different sets of events, the events of the orphaned block are queued to an event pool, in order to be included in a future block.

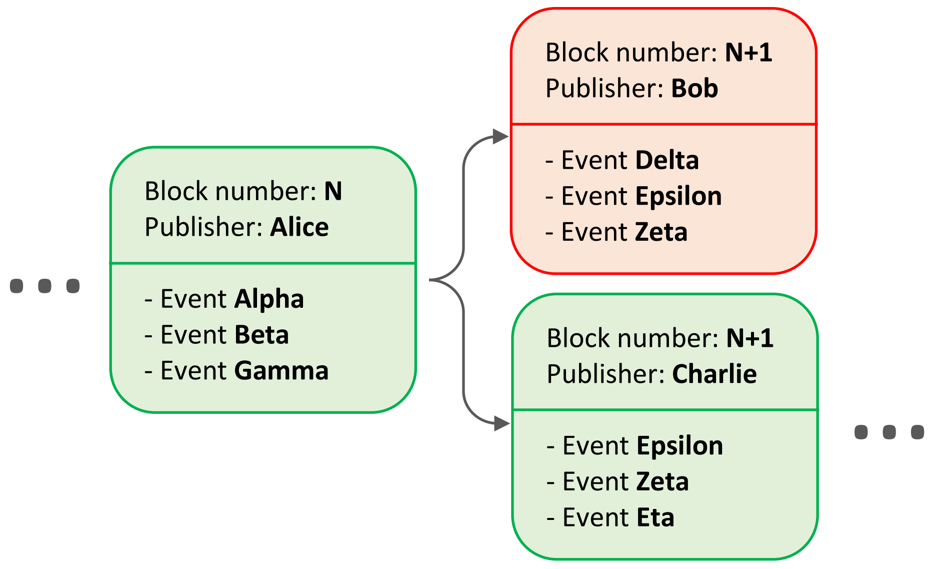


Figure 10: A split to the blockchain caused by conflicting blocks

In this example, if the next block is attached to the one published by $Charlie$, $Bob's$ block is marked as orphaned. Its events not present in $Charlie's$ block (that is $Delta$) will be stored into the event pool. Although event $Delta$ was valid at the time of its first submission, it is possible to be rejected as invalid in a later time. This could happen if the events of $Charlie's$ block (or a subsequent one) change the blockchain state in a way that turns $Delta$ event invalid. For example, if events represent token transactions, $Delta$ event will not be allowed to spend tokens that were already spent by $Epsilon$, $Zeta$ or Eta events.

As a conclusion, the validity of a blockchain event cannot be evaluated in strict terms. The event gains an increasing level of assurance over time, as other blocks are appended to its branch. Theoretically, an entity with access to enormous amounts of processing power could start building a new branch onto the genesis block and render invalid the entire blockchain. It is the size of the chain that can make such an attempt practically impossible. Furthermore, some implementations periodically generate checkpoints to consolidate the global state; all blocks up to the latest checkpoint are by default considered valid and cannot be discarded.

2.6.7 Smart Contracts

The term *Smart Contract* (SC) was defined [72] as "*a computerized transaction protocol that executes the terms of a contract. The general objectives of smart contract design are to satisfy common* *contractual conditions, minimize exceptions both malicious and accidental, and minimize the need for trusted intermediaries*".

Smart Contracts were introduced with 2nd generation blockchains, and allow decentralized code execution. There are specialized programming languages, created to allow users to easily develop such programs. The execution takes place in a protected virtual machine (VM) implemented in the node software.

There are numerous protection layers to prohibit abuse; for example, every SC is assigned a reasonable amount of time for its execution, based on the complexity of its code. The VM will terminate a SC that exceeds this limit to avoid infinite loops that could lead, intentionally or accidentally, to Denial of Service. The produced code is deterministic, thus, given the same state, the contract consistently produces the exact same result. Beyond the algorithm, a SC may have its own, private or public, resident data.

The code and the public data can be reviewed and executed by everyone. Inputs can be passed via parameters using a special type of on-chain transaction. The SC is executed by the participating peers and its results are validated by the network's consensus mechanism.

Since the execution of a SC requires processing power from several nodes, the user who requested its execution has to pay a fee, usually in the form of the blockchain's native token. The fee may vary significantly depending on the complexity of the contract's code.

2.7 Internet of Things

2.7.1 IoT Fundamentals

There are two major reasons behind the recent paradigm shift from conventional *computer-aided systems* to *smart systems*: The advances in IoT sector and the explosion of *Big Data*. The combination of these two factors allowed the analysis of enormous amounts of data and the extraction of hidden information, that can lead a system to intelligent decisions.

An IoT system, which is essentially a network of smart devices, consists of the following layers, as shown in
Figure 11:

- *Perception Layer:* A large variety of devices including sensors, controllers and tags. Some of them can affect their physical environment through actuators. The devices can communicate via wired or wireless connections.
- *Communication Layer:* Protocols that control information exchange in various environments. Depending on use case, connections may be wired or wireless, directional or omnidirectional, long or short ranged, with low or typical energy consumption.
- *Application Layer:* Almost any sector that utilizes data flows from geographically diverse devices, can benefit from the transition to IoT. The basic idea is to implement a strongly bound *digital twin* ecosystem, that can manage the data generated by events on the sector's physical environment.

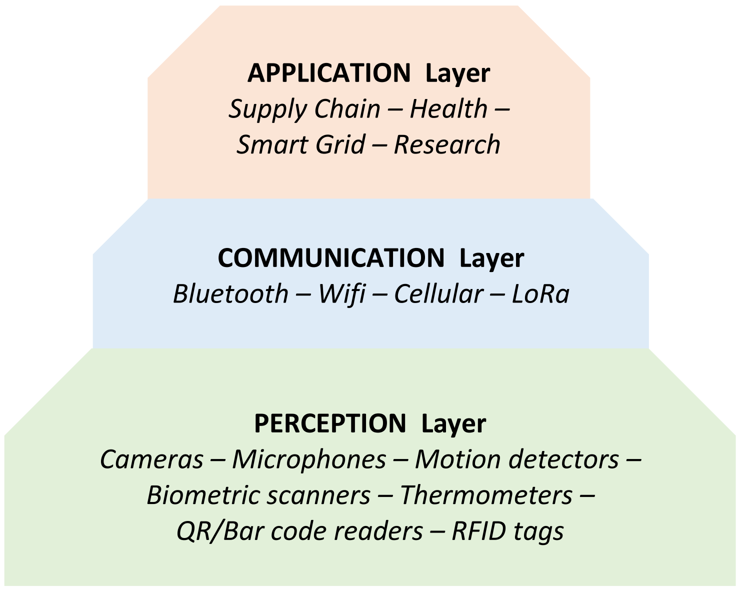

APPLICATION Layer
Supply Chain – Health – Smart Grid – Research
COMMUNICATION Layer
Bluetooth – Wifi – Cellular – LoRa
PERCEPTION Layer
Cameras – Microphones – Motion detectors – Biometric scanners – Thermometers – QR/Bar code readers – RFID tags

Figure 11: Layers of an IoT system with typical examples

2.7.2 DLT for Internet of Things

The recent advances in microelectronics and low-consumption communication technology pushed the evolution of smart systems, that can achieve data-driven decision making [73]. *Internet of Things* paradigm considers a physical environment and creates its digitized version, forming a cyber-physical system. In such a system the building blocks exchange information, based on which they can alter the system's behavior. In many use cases these blocks are heterogeneous, resource-constraint, and poorly connected, weaknesses that can be addressed by using DLT technology. In fact, DLT is the ideal complement to IoT, since it may bring a combination of critical features like *security*, *privacy*, *reliability*, *scalability* and *interoperability*. Architectures that extend to both worlds could be referred to as *DLT of Things* (DLToT).

2.7.3 DLT-IoT Interaction Level

When designing systems that leverage the power of both DLT and IoT, an aspect to take into account is the level of interaction between them. There are three alternatives, as described below:

- *Exclusive IoT-IoT communication:* This is the fastest implementation, since almost all traffic is conducted directly between the IoT devices. Only a minor amount of exchanged data is saved in DLT storage, for aspects that require decentralized authentication and immutability.
- *Exclusive DLT-IoT communication:* In this scenario all traffic goes through DLT nodes. This approach results to architectures with lower communication speeds due to increased overhead. On the other hand, it ensures full recording of every system interaction, guaranteed by the tamper-proof consensus mechanism of the DLT.
- *Hybrid communication:* Implementations that use both of the above types of communication in a balanced way [74][75]. One of the challenges in such cases is how to choose the optimal communication type for each operation. Usually, the system's processes are divided into two groups: those that need minimum latency and those that require immutability and/or permissionless trust.

2.8 IPFS

2.8.1 Design

Interplanetary File System (IPFS) combines features from Git [76] and BitSwap[12], which is an algorithm for incentivizing data replication (based on BitTorrent [77]). IPFS implements a decentralized filesystem for permissionless peer-to-peer topologies.

[12] https://github.com/ipfs/specs/tree/master/bitswap

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 hashes and are stored independently. In contrast to 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 [78][79][80]. The DHT contains information about where data chunks can be retrieved from.

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.

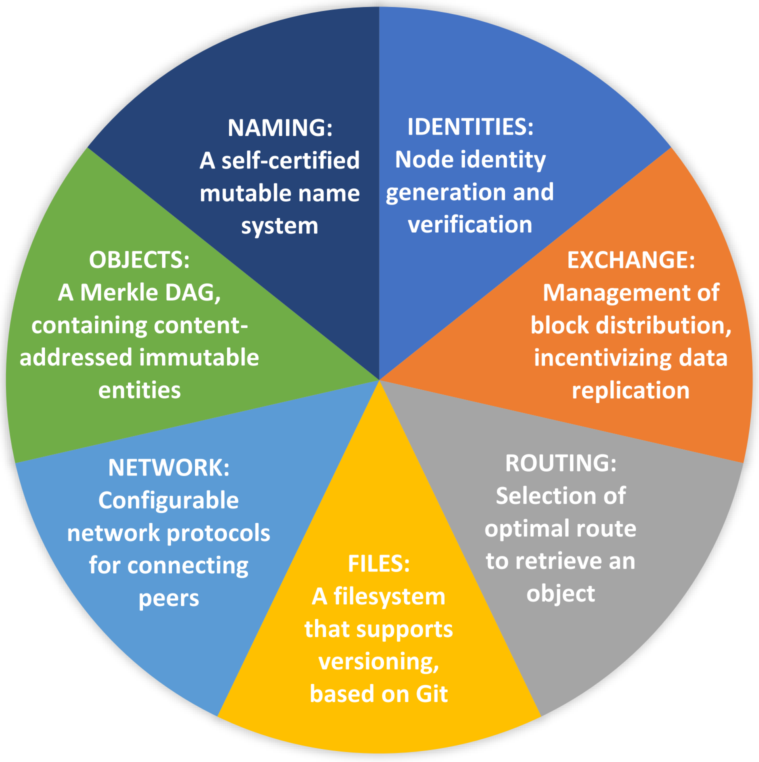

NAMING: A self-certified mutable name system
IDENTITIES: Node identity generation and verification
EXCHANGE: Management of block distribution, incentivizing data replication
ROUTING: Selection of optimal route to retrieve an object
FILES: A filesystem that supports versioning, based on Git
NETWORK: Configurable network protocols for connecting peers
OBJECTS: A Merkle DAG, containing content-addressed immutable entities

Figure 12: The protocol stack of IPFS

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.

The IPFS stack contains seven different sub-protocols, as presented in Figure 12.

2.8.2 Protocol Stack

2.8.2.1 Identities

Nodes participating to an IPFS swarm are identified by a cryptographic hash. This is called *NodeId* and is generated (Algorithm 2) using the static puzzle of S/Kademlia [78]. The NodeId consists of a public/private key pair, which are encrypted and stored locally. The NodeId is created by hashing the peer's public key. When a node detects a neighbor for the first time, checks whether

$$hash(neighbor.PublicKey) = neighbor.NodeId$$

If this check fails, the node terminates all communication with that neighbor. A node can use different identity each time it joins the network. But such a tactic is not recommended, since it would lose accrued benefits gained by its previous participation. IPFS can utilize multiple cryptographic hash functions, so in every use case the best-suiting one can be selected. Hash digests are stored in multihash format, which includes a header describing the function and the digest length in bytes.

2.8.2.2 Network

IPFS relies on heavy network communication; for that purpose, it can use any transport protocol like WebRTC[13] or uTP [81]. If necessary, reliability can also be ensured using SCTP [82]. For

[13] https://webrtc.org

connectivity and authenticity IPFS uses ICE NAT[14] traversal techniques and HMAC[15] with sender's public key.

Regarding peer addressing, IPFS does not rely on or assume access to IP, but uses a new way to express both an address and its corresponding protocol. This expression is called *multi-address* (multiaddr) and may contain encapsulation details. For example:

SCTP/IPv4 connection

`/ip4/10.20.30.40/sctp/1234/`

SCTP/IPv4 connection proxied over TCP/IPv4

`/ip4/5.6.7.8/tcp/5678/ip4/1.2.3.4/sctp/1234/`

Algorithm 2: IPFS algorithm for identity generation

```
type Multihash []byte
type PublicKey []byte
type NodeId Multihash

type PrivateKey []byte
type Node struct {
  NodeId NodeID
  PubKey PublicKey
  PriKey PrivateKey
}

difficulty = <integer parameter>
n = Node{}
do {
  n.PubKey, n.PrivKey = PKI.genKeyPair()
  n.NodeId = hash(hash(n.PubKey))
  p = count_preceding_zero_bits(n.NodeId)
} while (p < difficulty)
```

[14] https://en.wikipedia.org/wiki/Interactive_Connectivity_Establishment
[15] https://en.wikipedia.org/wiki/HMAC

2.8.2.3 Routing

IPFS nodes use a *Distributed Hash Table* to be able to find the address of other peers and objects. The routing protocol is based on S/Kademlia, Coral [82] and Mainline [80], so it makes a distinction for the values stored in the DHT. Small objects, less than 1KB, are stored directly on the DHT, while for larger objects the DHT stores the NodeIDs of peers that can serve these objects. Algorithm 3 shows the routing interface.

Algorithm 3: IPFS routing interface

```
type IPFSRouting interface {

  FindPeer(node NodeId)
  // gets a particular peer's network address

  SetValue(key []bytes, value []bytes)
  // stores a small metadata value in DHT

  GetValue(key []bytes)
  // retrieves small metadata value from DHT

  ProvideValue(key Multihash)
  // announces this node can serve a large value

  FindValuePeers(key Multihash, min int)
  // gets a number of peers serving a large value

}
```

2.8.2.4 Exchange

A block exchange strategy that works well in practice is a sigmoid scaled by a *debt ratio*. Let the debt ratio, r, between a node and another peer be

$$r = \frac{bytes\_sent}{bytes\_received\ +\ 1}$$

and the probability of sending a block to a peer be

$$P(send \mid r) = 1 - \frac{1}{1 + e^{6-3r}}$$

As shown in Figure 13, a node will provide more data to peers that have been helpful in the past, and will be sparing to unknown nodes. This exchange strategy (i) provides resistance to *sybil attacks* by actors who would attempt to create multiple identities, (ii) protects relationships that, although proven successful in the past, suffer from temporary issues, and (iii) downgrades unhelpful relationships until they start to provide value.

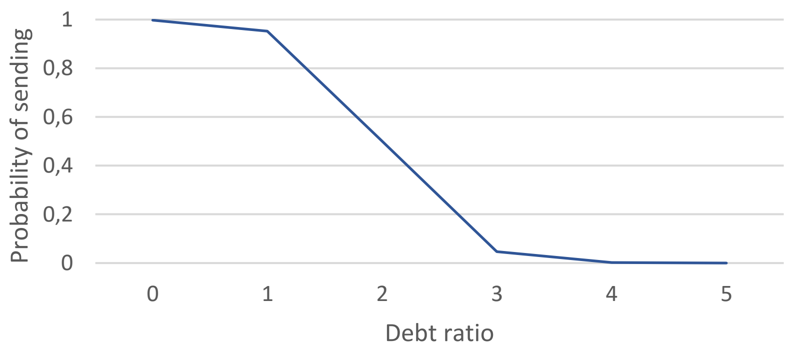


Figure 13: Probability of sending a block to a debtor

IPFS nodes maintain a local ledger containing the accrued credit or debt with all the known peers. Each time a connection is activated, the communicating nodes send each other the ledger of their previous exchanges (Algorithm 4). If discrepancies are found, the ledger is reinitialized from scratch.

Algorithm 4: IPFS exchange ledger & protocol

```
type Ledger struct {
  owner NodeId
  partner NodeId
  bytes_sent int
  bytes_received int
  timestamp Timestamp
}
```

```
type BitSwap struct {
  ledgers map[NodeId]Ledger
  // Ledgers known to this node, including inactive

  active map[NodeId]Peer
  // currently open connections to other nodes

  need_list []Multihash
  // checksums of blocks this node needs

  have_list []Multihash
  // checksums of blocks this node has
}

type Peer struct {
  nodeid NodeId
  ledger Ledger
  // Ledger between the node and this peer

  last_seen Timestamp
  // timestamp of last received message

  want_list []Multihash
  // checksums of all blocks wanted by peer
  // includes blocks wanted by peer's peers
}

interface Peer {
  open (nodeid :NodeId, ledger :Ledger);
  // peers send ledgers until they agree

  send_want_list (want_list :WantList);
  send_block (block :Block) -> (complete :Bool);
  // peers exchange want_lists and blocks

  close (final :Bool);
  // peers deactivate a connection
}
```

Algorithm 5: IPFS object format

```
type IPFSLink struct {
  Name string
  // name or alias of this link

  Hash Multihash
  // cryptographic hash of target

  Size int
  // total size of target
}

type IPFSObject struct {
  links []IPFSLink
  // array of links

  data []byte
  // opaque content data
}
```

2.8.2.5 Objects

On top of DHT and BitSwap, IPFS creates a *Merkle Directed Acyclic Graph* inspired by Git, where the links between objects are cryptographic hashes of the targets embedded in the sources. This layer provides properties such as *content addressing*, *tamper resistance* and *deduplication*. Algorithm 5 shows the general format of an *IPFS* object. IPFS allows applications to format the data field in whatever way they see fit. Due to the embedded IPFSLink list, a user can easily browse all referenced objects. For example, the (UNIX-inspired) *ls* command called with a multihash, will result:

```
> ipfs ls /QtLZ5T162Dmj5jng7SubMh5Dgyeayn5FR4vx

QtYk6gq61DYaQ8Nhk7cqyU7rLcnSa7dSHQ4  34556 ObjectMnemonicName1
QteHB3NmRQ5srJ7Jrd8gMPuu48hp9zeyTtR 457262 ObjectMnemonicName2
QtF54hhwVHsjVu5Z78FZ7Kk6fozf8Jj9lWE   1794 ObjectMnemonicName3

<object multihash> <object size> <link name>
```

Additionally, IPFS can recursively resolve all referenced objects:

```
> ipfs refs –recursive \ /QtLZ5T162Dmj5jng7SubMh5Dgyeayn5FR4vx

QtYk6gq61DYaQ8Nhk7cqyU7rLcnSa7dSHQ4
QtjH6JHhjH7jLiop8jYi9uHhkyY0T4GHopi
QteHB3NmRQ5srJ7Jrd8gMPuu48hp9zeyTtR
QtkGj7Fa52KlyV7VKEk73JsjVu5Z78FZ7Ke
QtF54hhwVHsjVu5Z78FZ7Kk6fozf8Jj9lWE
Qt89Jhecn84K48HWbser73JElu48hp9zeyt
```

Traversing IPFS objects can be achieved by using a simple string path API, similar to conventional filesystems or web URLs:

```
# format
/ipfs/<hash-of-object>/<name-path-to-object>

# example
/ipfs/QtgXLYk1gq61DYjaQ8Nhkcq4ykU7rLc5nSa/myfile
```

Since there is no global root, referencing an object must start from the hash of an object hierarchically above it. If an object resides deep inside a path, it can be retrieved by using any of the objects above it as the first reference hash. For example, the object $parent1/parent2/parent3/ myfile$, can be retrieved in any of the following ways:

```
/ipfs/<parent1hash>/<parent2>/<parent3>/myfile
/ipfs/<parent2hash>/<parent3>/myfile
/ipfs/<parent3hash>/myfile
/ipfs/<myfilehash>
```

IPFS can perform object-level cryptographic operations. An encrypted or signed object is wrapped in a special frame that allows encryption and verification of the raw data (Algorithm 6). Cryptographic operations alter the object’s hash, thus creating a new different object. IPFS can utilize user-specified keychains to verify signatures and decrypt data. The decryption key is needed for even traversing encrypted objects, since the links between them can be encrypted as well.

Algorithm 6: IPFS encrypted object format

```
type EncryptedObject struct {
  Object []bytes
  // raw object data encrypted

  Tag []bytes
  // optional tag for encryption groups
}

type SignedObject struct {
  Object []bytes
  // raw object data signed

  Signature []bytes
  // hmac signature

  PublicKey []multihash
  // multihash identifying key
}
```

2.8.2.6 Files

To implement a versioned filesystem on top of the *Merkle DAG*, IPFS defines the following set of objects: (i) *blob*, a variable-size block of data (ii) *list*, a collection of blobs or other lists (iii) *tree*, a collection of blobs, lists, or other trees (iv) *commit*, a snapshot in the version history of a tree.

Blobs are the most basic way to represent a file. It is an addressable amount and they cannot contain links to other objects:

```
{
  "user key": "user data"
  // blobs have no links
}
```

Lists contain an ordered sequence of blob or list objects. They are used to represent a large file, by concatenating the listed objects. More complex data structures, like linked lists and balanced trees, can be represented by embedding child lists:

```
{
  "data": ["blob", "list", "blob"],
  // lists have an array of object types as data

  "links": [
    { "hash": "Qte1Ykgq61DYa5Q8Nhrkcq6yU7rLcn7Sa7d",
      "size": 34680 },
    { "hash": "QtNm1RQ5sJJrdg4MPuu48pz5eygT6tRo39t",
      "size": 2341 },
    { "hash": "QtQDqxo9Kmd9zLyquoC9gAP8fCL1gWnHZ7z",
      "size": 12769 }
    // lists have no names in links
  ]
}
```

Trees can contain references to blobs, lists, other trees, or commits. They are used to represent a directory, leveraging the path naming mechanism provided by the Merkle DAG:

```
{
  "data": ["blob", "list", "blob"],
  // trees have an array of object types as data

  "links": [
    { "hash": "Qte1Ykgq61DYa5Q8Nhrkcq6yU7rLcn7Sa7d",
      "name": "MneumonicName1", "size": 65628 },
    { "hash": "QtNm1RQ5sJJrdg4MPuu48pz5eygT6tRo39t",
      "name": "MneumonicName2", "size": 7176 },
    { "hash": "QtQDqxo9Kmd9zLyquoC9gAP8fCL1gWnHZ7z",
      "name": "MneumonicName3", "size": 84913 }
    // trees do have names
  ]
}
```

Commits represent a snapshot in the version history of any IPFS object:

```
{
  "data": {
    "type": "tree",
    "date": "2021-06-05 09:26:08Z",
    "message": "This is a commit message."
  },
  "links": [
    { "hash": "Qte1Ykgq61DYa5Q8Nhrkcq6yU7rLcn7Sa7d",
      "name": "parent", "size": 54694 },
    { "hash": "QtNm1RQ5sJJrdg4MPuu48pz5eygT6tRo39t",
      "name": "object", "size": 4495 },
  ]
}
```

2.8.2.7 Naming

The *content-addressed* and *versioned* DAG of objects, which is the immutable backbone of the IPFS infrastructure, offers significant advantages. These include effortless caching and integrity checks, optimized bandwidth usage, and permanent links.
But a vast number of applications require a degree of mutability. IPFS can overcome this barrier by using self-certified naming, thus a mutable addressing system that allows the same name to always point to the latest version of an object. Using the naming scheme from SFS [83][84], IPFS constructs a mutable and cryptographically assigned global namespace.

Every node is assigned a mutable namespace at

```
/ipns/<NodeId>
```

where

```
NodeId = hash(node.PublicKey)
```

Thus, a node can publish objects under its namespace, as:

```
/ipns/QtQDqxo9Kmd9zLyquoC9gAP8fCL1gWnHZ7z/
/ipns/QtQDqxo9Kmd9zLyquoC9gAP8fCL1gWnHZ7z/folder1
/ipns/QtQDqxo9Kmd9zLyquoC9gAP8fCL1gWnHZ7z/folder1/file1
```

2.9 IOTA

2.9.1 The Tangle

The IOTA (meaning extremely small) project has as its goal to develop a completely decentralized network connecting all IoT devices. There are projections[16] 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.

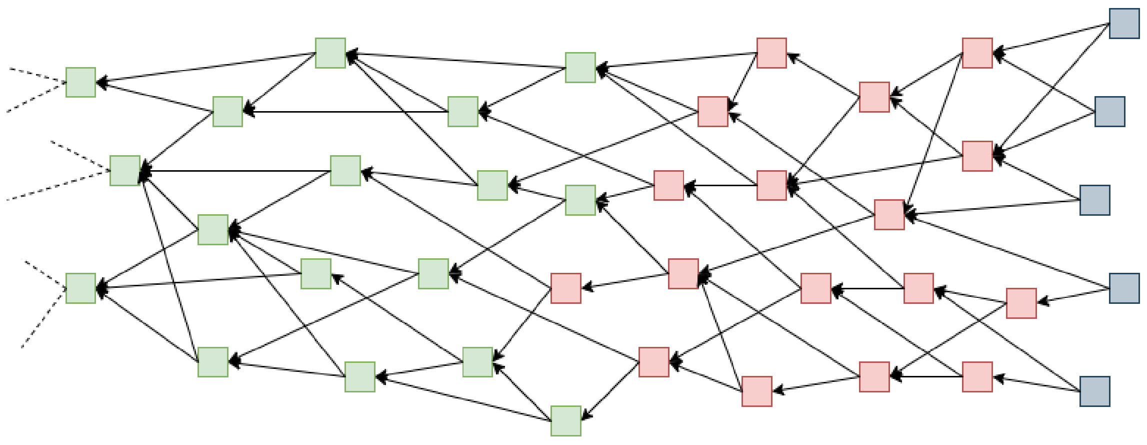


Figure 14: Representation of transactions in the Tangle (green: confirmed, red: unconfirmed, gray: tips)

To overcome the scalability issues encountered by blockchains, IOTA is based on a Directed Acyclic Graph (DAG), which was named *Tangle*[17]. The Tangle's building blocks are *transactions*. If

[16] https://www.statista.com/statistics/802690/worldwide-connected-devices-by-access-technology
[17] http://www.descryptions.com/Iota.pdf

a node needs to attach a new transaction, it must confirm two previous ones (Figure 14), so -unlike blockchains- increased activity is expected to result in better performance and security for the network. Thus, IOTA attempts to achieve consensus via a permissionless procedure, using the Tangle as DLT and propagating transactions throughout the network via a gossip protocol.

Each time a node needs to attach a transaction, chooses two previous unconfirmed transactions (called *tips*) to approve. The selection is made using *Markov Chain Monte Carlo*[18] (MCMC) algorithm. Tangle transactions are not synchronized and the nodes are not always aware of all of the broadcasted transactions. So, independent branches may appear, as shown in the example of Figure 15. Transactions 6, 7, 8, 11, 12 consist Branch A, and transactions 5, 9, 10, 13 consist Branch B. These two branches are temporarily independent, with no interconnection and no common transactions. The node which issues transaction 12 by validating 8 and 11, is not aware of the events in Branch B.

It is possible the two branches to have conflicting transactions. When transaction 14 is attached, it validates 10 and 11, and every transaction linked to them is verified. This simultaneous validation of the two branches will detect and eliminate any conflicts that might exist. Then the consensus protocol decides which branch will be chosen as valid, while the other one will be abandoned. In this example, transactions 12 and 13 are not referenced by 14; they will be verified with a subsequent new transaction.

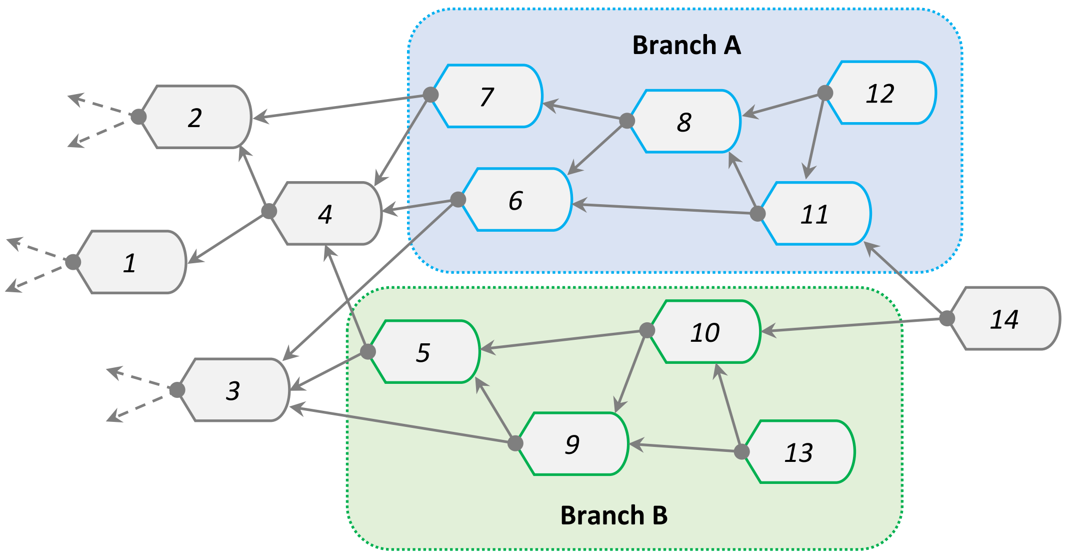


Figure 15: Conflict resolution in the IOTA Tangle

[18] https://en.wikipedia.org/wiki/Markov_chain_Monte_Carlo

The main rule used to decide between conflicting transactions is to execute the tip selection algorithm multiple times [85][86][87]. The MCMC algorithm has to select between the available tips, so it must use a metric to evaluate the importance of each branch. This metric is its *accumulated weight*, defined as the sum of the weights of all the transactions it contains. As a result, long branches tend to grow even further, while short ones tend to become isolated with low certainty probability. The level of adoption a transaction enjoys from the Tangle grows proportionally to the number of new tips, and can be calculated by using accumulated weight analysis. As proven in [85], at a specific time, t, the accumulated weight $W(t)$ is

$$W(t) \approx 2 \cdot e^{\frac{0.352t}{T}}$$

where T is the average time that a node needs to issue a new transaction, including PoW.

Figure 16 shows a plot of the behavior of the accumulated weight of a transaction. The graph contains two areas: (i) The *curved adaptation period*, where the weight is exponentially increased. During that time the transaction is referenced by an increasing subset of the new ones. (ii) The *linear certainty period*, where the transaction is linked to practically all new ones; thus, the transaction has a high probability that it will not be abandoned.

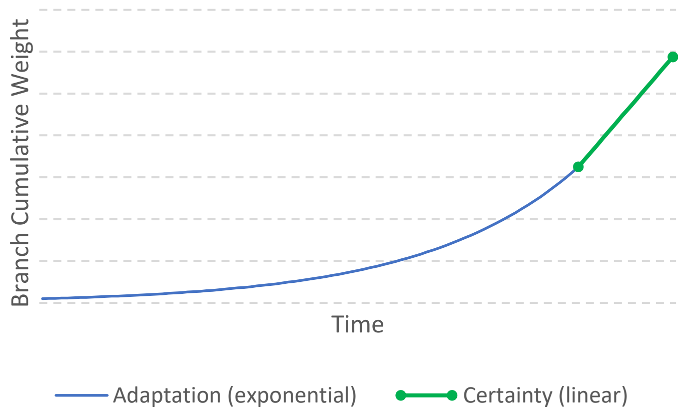


Figure 16: The Branch Cumulative Weight over time

2.9.2 System Stability

To provide an insight into the system's stability, we assume the following:

1) The total number of tips at time t, $L(t)$, fluctuates around a constant value and does not escape to infinity, since such a steep increase would result to unapproved transactions left behind. Let this value be L_0.
2) Transactions are issued by a large population of independent entities; thus, their number can be modeled after a Poisson point process [88].
3) The rate of this Poisson process, λ, remains constant in time.
4) All nodes have approximately the same processing power and in average they need time h to complete the computations needed for issuing a transaction.
5) At any given time, a node is not aware of the current state of the Tangle, but the one exactly h time ago. This means that a transaction attached at time t, becomes visible to the network at time $t + h$. The tips attached in the time interval $[t - h, t)$ are "hidden", meaning that they are not yet visible to the network.

If number of "revealed" tips (the ones that were attached before time $t - h$) is r, then

$$L_0 = r + \lambda \cdot h$$

At time t there are also around $\lambda \cdot h$ transactions that were tips at time $t - h$, but are not tips anymore. When a new transaction is about to be attached to the Tangle, it must select a transaction to approve. Since there are around r tips known to the node, and $\lambda \cdot h$ transactions which are not tips anymore, the selected transaction is a tip with probability

$$\frac{r}{r + \lambda \cdot h}$$

So, the mean number of selected tips is

$$\frac{2 \cdot r}{r + \lambda \cdot h}$$

According to our assumption that the number of tips remains roughly the same, the mean number of selected tips should be equal to 1:

$$\frac{2 \cdot r}{r + \lambda \cdot h} = 1 \rightarrow r = \lambda \cdot h$$

So:

$$L_0 = r + \lambda \cdot h = 2 \cdot \lambda \cdot h$$

If we generalize the Tangle's validation mechanism by assuming that new transactions reference not 2, but k existing tips, the calculation gives the following formula:

$$L_0^{(k)} = \frac{k \cdot \lambda \cdot h}{k - 1}$$

As noted, a new transaction cannot be approved during for a time interval of h after its attachment. When this period ends, the Poisson flow of approvals has a rate of

$$\frac{2 \cdot \lambda}{L_0}$$

So, the expected time for a transaction to be approved is approximately:

$$T_A = \frac{h + L_0}{2 \cdot \lambda} = 2 \cdot h$$

At any specific time, t, the set of transactions that were tips during the period

$$s \in [t, t + h(L_0, N)]$$

constitutes a *cutset*; any path from a transaction issued at time $t' > t$ to the *genesis*, must pass through this set.

Although useful to get an insight, the purely random tip selection is not the optimal algorithm for real-life applications. This is because it allows "lazy" nodes to always select a fixed pair of old transaction. It also permits bad actors to issue a huge number of new tips which all reference a fixed set of previous transactions. In that way these actors acquire for their transactions an inflated probability to be referenced by future tips.

The IOTA Tangle's tip selection algorithm is a work in progress. Improved implementations are proposed and tested, in order to achieve the security, scalability, and decentralization required for the wide-scale adoption of this novel technology.

2.9.3 Masked Authenticated Messages

The Tangle supports feeless zero-value transactions that serve the purpose of data streams, called Masked Authenticated Messages (MAM). We propose 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.

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* (Figure 17).

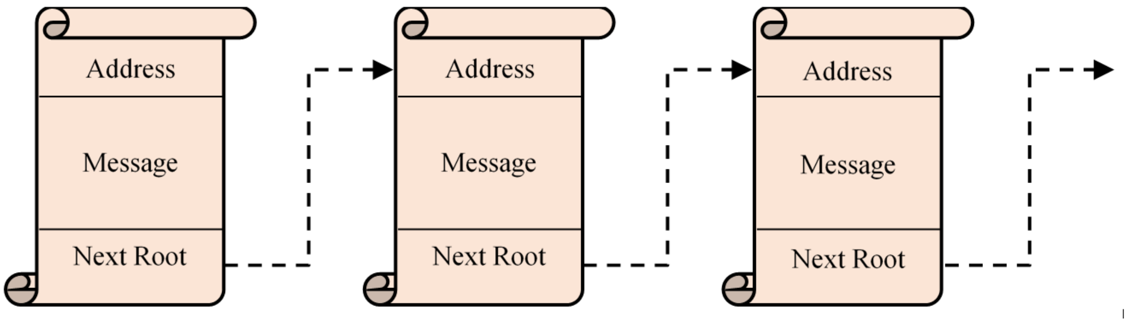


Figure 17: The singly-linked list of a MAM stream

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.

Table 5: Encryption types of MAM streams

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

MAM uses a signature scheme based on Merkle Trees [89] 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, 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 5).

2.9.4 Seeds & keys

In the IOTA Tangle network every new use gets a randomly generated *seed*, which is an alphanumeric string consisting of 81 *trytes*[19]. For every user an unlimited number of private keys can be created, by hashing the concatenation of the seed and the address index, a positive auto-increased integer [90]. By hashing a private key, the system can generate the corresponding public key, which is used as the user's address. The keys can be used to prove ownership of addresses. Assets from all the addresses that are linked to the same seed, are summed to create the account's total balance [91].

The IOTA protocol aims to be resistant to attacks from quantum computers and their vast processing abilities. For that purpose the network uses Winternitz One-Time Signatures (WOTS) [91][92] which -beyond their quantum resistance- are faster than elliptic curve cryptography [93]. The use of WOTS has a negative side effect; Each time a user sends assets from a specific address, a fraction of the corresponding private key can potentially be extracted by analyzing the transaction's signing data. So, repeatedly sending tokens from the same address can compromise the account's

[19] https://en.wikipedia.org/wiki/Ternary_numeral_system

security. There are many proposals to overcome these issues [91], some of which are to be implemented into the upcoming versions of the Tangle's protocol.

2.9.5 Coordinator

The security of the Tangle depends on the size of the network. As the number of participants grows, so does the resilience to attacks. To allow the network to reach that critical size, the first phase of the Tangle's operation is based on a centralized safeguard, called *Coordinator*. This is a special node which issues one transaction every 2 minutes. These transactions, called *milestones*, are by default instantly considered 100% confident [87]. Coordinator degrades the system's decentralization in order to increase security. Newer implementations of the Tangle with no need for coordinator have been proposed, and are already being evaluated in the IOTA's TestNet.

2.9.6 Snapshots & Permanodes

IOTA allows zero-value transactions with no fees. This fact can generate a large amount of exchanged data, skyrocketing the needed storage space. To mitigate this problem, there is a periodic process called *Snapshotting*. When a snapshot occurs, all data regarding the network history is purged, and the only that is kept is the current state, thus the amount contained in addresses that have a non-zero balance. In the latest implementation, snapshotting is a policy decided and applied locally, thus every node determines when and how to truncate its database via a snapshot. It is very likely that many applications will require access to the whole Tangle history. For these cases a node could -under proper incentivization- retain all transaction data; such nodes are called *Permanodes*.

2.10 Ethereum

Internet connectivity is steadily expanding to more sectors of human activity. Among many other similar projects, *Ethereum* is an attempt to build a generalized transaction-based state machine. Such a project could facilitate interactions between consenting parties who would otherwise have no means to trust each other. This goal could be achieved by a system which can autonomously enforce agreements that were described using an unambiguous language. A tamper-proof algorithmic decision system can offer attributes that are often difficult to find in the physical world, such as *clarity*, *incorruptibility*, *transparency* and *objectivity*.

Transactions are the mechanism for system transitions between valid states. Formally:

$$\sigma_{\tau+1} \equiv \Pi(\sigma_t, B)$$

$$B \equiv (\ldots, (T_0, T_1, \ldots))$$

$$\Pi(\sigma, B) \equiv \Omega(B, \Upsilon(\Upsilon(\sigma, T_0), T_1) \ldots)$$

Where σ_τ, $\sigma_{\tau+1}$ are the states before and after block B, Υ is the state transition function, Π is the block-level state transition function, and Ω is the block-finalization state transition function, which also includes the mining rewards.

The *World State* is a mapping between 160-bit addresses and account states, serialized as a Recursive Length Prefix (RLP). It is maintained in a Modified Merkle Patricia Tree (Figure 18), a persistent data structure for mapping between 256-bit binary fragments and arbitrary-length binary data. The trie provides a single value that identifies a given set of key-value pairs; traversing the trie from root to leaf, reconstructs a single key-value pair.

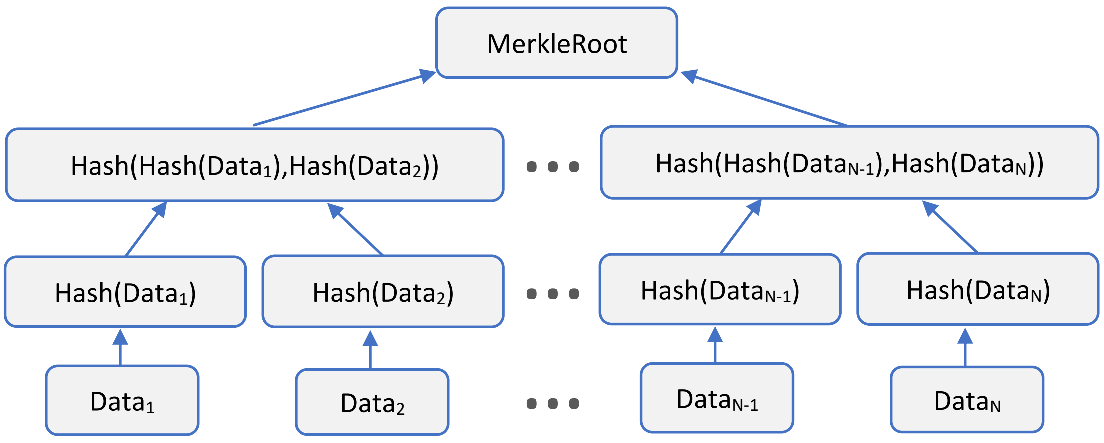


Figure 18: The structure of a Merkle Tree representing N data fragments

3 Privacy preserving distributed training of neural networks

3.1 Related work

The majority of AI/ANN applications include the use of a parameter server [94][10], which requires high-performance infrastructure. Although decentralized efforts, such as [95]–[97], use local optimization and asynchronous model merging to reduce communication requirements, the parameter server remains a bandwidth barrier, limiting scalability. Other distributed deep learning systems [98][99] are capable of circumventing this barrier, but they need low latency networking, which results in high setup cost and limited applicability.

While proposals such as [100] emphasize average-performance hardware, they rely on frameworks such as [101], which are optimized for synchronous environments, and therefore are not the best option for loosely linked peers.

The following Table 6 compares the aforementioned characteristics, taking into account alternative approaches of asynchronous data parallelization.

Table 6: Comparison of previous approaches

	Peer-to-peer	Resilience	Persistency	Privacy	Polymorphism	Heterogeneity
LEARNAE	✓	✓	✓	✓	✓	✓
Downpour SGD	✗	✓	✗	✗	✗	✓
Sandblaster L-BFGS	✗	✓	✗	✗	✗	✓
Elastic Averaging SGD	✗	✗	✗	✗	✗	✗
Hogwild	✗	✗	✗	✗	✗	✗

LEARNAE pushes the boundaries of decentralization and tolerance to the limit. Due to the fact that it is based entirely on distributed peer-to-peer technology, it does not need servers or any kind of synchronization. Its intended use cases are environments with commodity-hardware nodes and networking infrastructure that may have high latency and loose connectivity. Our method enables flexible data collection from a number of sources, including lightweight Internet of Things devices, through novel Distributed Ledger Technology.

A collaborative training scheme that has some similarities to our approach is Federated Learning (FL) [102]–[104]. In systems that are based on FL, model training takes place on the peers without explicitly exchanging data samples. The process can be either centralized or decentralized, and local models are periodically combined to generate a global model. Our proposal introduces both alterations and extensions to FL method. Primarily, during a LEARNAE session there is no global model; each peer attempts to improve its own local model by selective parameter averaging, thus the knowledge of its neighbors provides intuition for further solution space exploration. Additionally, our approach leverages novel DLT networks to propose: (a) a way for dealing with unreliable environments via data duplication, (b) a scheme for consolidating data aggregation from IoT devices, and (c) an incentivization mechanism to attract additional processing power. Considering the above, LEARNAE is a proposal for a complete ready-to-use ecosystem, which lowers the entry-level barrier for individuals who wish to experiment with Deep Neural Network training.

3.2 Implementation

3.2.1 Parallelism type

The initial choice was between model and data parallelism [105][8][9]. It's worth noting that there are proposals that implement systems with hybrid design. LEARNAE makes use of data parallelism, where each worker retains the full model locally and processes it using a portion of the training data.

3.2.2 Propagation

Following worker processing, the generated models must be merged. The two primary techniques for doing this are *weight* and *update* averaging, each of which has distinct benefits and disadvantages. LEARNAE employs *weight* averaging, which means that after the training phase, the actual values (not just the updates) of all model parameters are averaged -under specific conditions- with the actual values of the corresponding parameters of a model generated by a remote worker [106].

3.2.3 Coordination

Merging the models is also possible with varying degrees of decentralization, as shown in Table 7. A parameter server's job is to collect, combine, and re-distribute averaged data. While using a server speeds up training in most instances, it also introduces a single point of failure and – in large size networks – a bandwidth bottleneck. This disadvantage may be addressed by increasing the number of servers that collaborate with one another. When the presence of a server is not possible or desirable, some of the participating peers are given specific coordination responsibilities while simultaneously doing all other training duties. At the other end of this spectrum are systems in which no node assumes extra coordinating responsibilities, resulting in a completely distributed environment, which is the design choice taken in our proposal.

Table 7: 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

3.2.4 Synchronicity

The training procedure may be synchronous or asynchronous. In synchronous designs the coordinating entity guarantees that only results from the same training period are merged. There is no such need in asynchronous designs, and the outcomes of a worker may be integrated into the global model using more flexible criteria. Each method offers a number of advantages and disadvantages. Synchronous training may converge quicker since it avoids the merging of very dissimilar models, but it may encounter locks from slow peers, delaying the whole process. While asynchronous training maximizes worker utilization, it suffers from gradient staleness, which means that by the time a slow worker sends their results, they are already out of sync with the global model. Numerous mitigation methods [107] have been suggested to address these drawbacks, resulting in a large number of variations, particularly for Asynchronous Stochastic Gradient Decent. Although LEARNAE is designed to be asynchronous, it includes capabilities that, when utilized in future implementations, may introduce a configurable degree of synchronicity.

3.2.5 Data privacy

LEARNAE is capable of operating in environments where participants are reluctant to provide sensitive training data. In such situations, training-data related communications are disabled, and all transmitted data consist only of models generated by nodes after their training or averaging sessions. In that way the network indirectly leverages the useful information contained in all training data, by averaging the models these data produced.

3.3 System architecture

3.3.1 Overview

LEARNAE is built on a flexible scheme that adapts to a variety of environments. In terms of data, it supports use cases in which all training data are put into the network during the initial phase, before any training. Additionally, it supports scenarios in which data feeding is a continuous task and neural model improvement is an always-evolving procedure (online training).

In summary, there is no constraint on *when* training data can be fed to the network, which is important when streaming sensor data from IoT devices.

3.3.2 Node Roles

There are four distinct types of node roles that provide operational flexibility (Table 8). The role of a node is determined by the availability of training data and its computing capacity. The first three roles need sufficient computing power to support participation to IPFS swarm. The fourth role is reserved for the IoT sector, since data streaming through MAM messages may be done even by light-weight sensor devices. The dataflow between nodes with varying responsibilities is shown in Figure 19.

Table 8: 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 is the case with all publish-subscribe systems, the channel ID (IPFS or IOTA) serves as the connecting link between peers. By knowing this ID, a peer may join the network and participate in it by listening for and sending messages. Encryption and/or authentication may be implemented at any step of the data exchange process if necessary. The Listening thread of a node's process is shown in Figure 20.

3.3.3 Message Types

3.3.3.1 Slice hashlist

This message includes the hash of a file that was uploaded to IPFS by a node. The file includes a list of hashes for training dataslices that the same peer also made accessible through IPFS.

3.3.3.2 Remote model

This message includes the hash of a file that was uploaded to IPFS by a node. The file includes an HD5 model of that peer. Other model information is also provided, such as the model's attained accuracy, and maturity, defined as the number of training cycles completed prior to its construction.

3.3.3.3 Slice use stats

This message notifies all peers that a particular dataslice has been utilized for training by a peer. This information facilitates the *overuse threshold* feature, which allows limiting the number of times a dataslice may be utilized for training on various nodes.

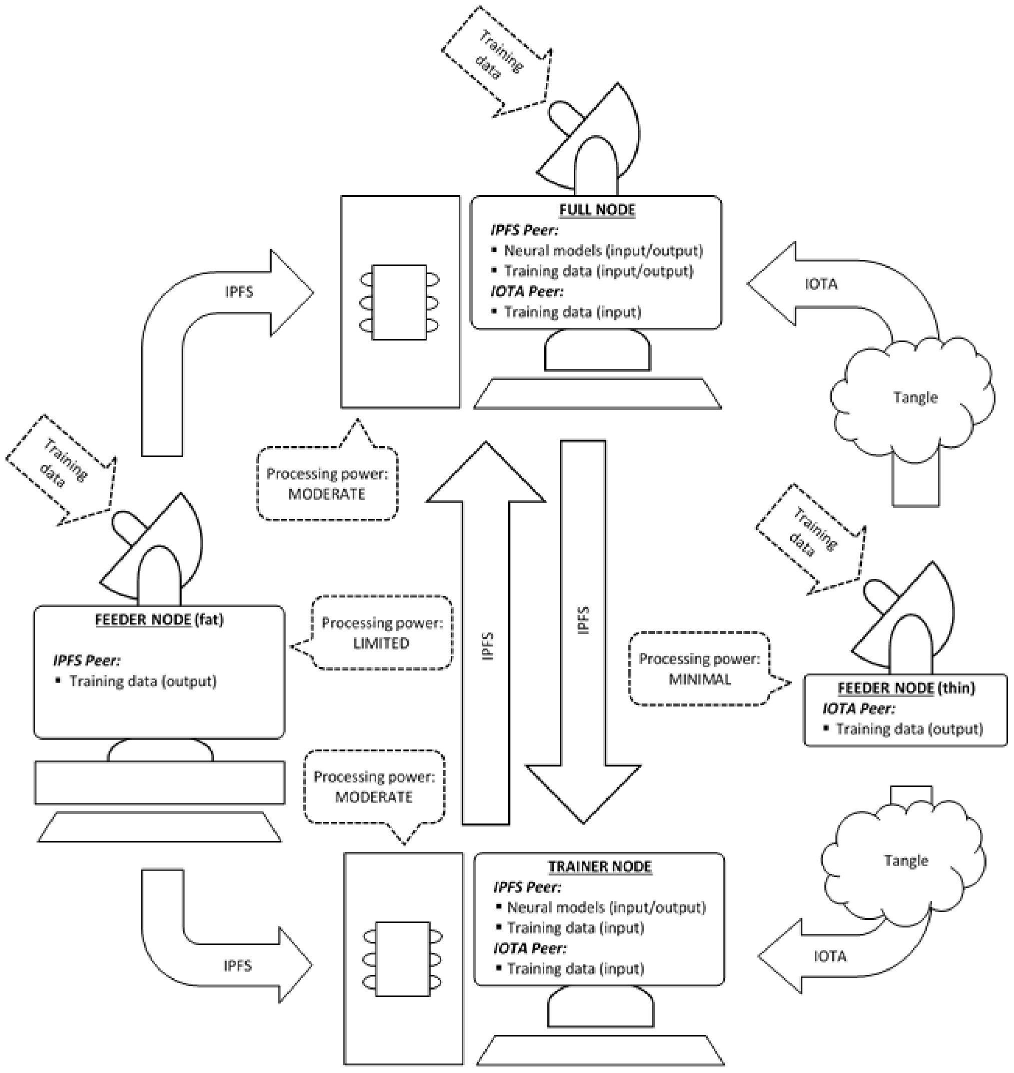


Figure 19: Dataflow between nodes of different roles

Table 9: 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

As stated in Table 9, a peer may perform up to four tasks. To optimize node usage, LEARNAE runs each task on a separate thread. All of them may be executed concurrently, except for the pair Training/Averaging, which requires read-write access to the local model. The workflow of a whole node is shown in Figure 21.

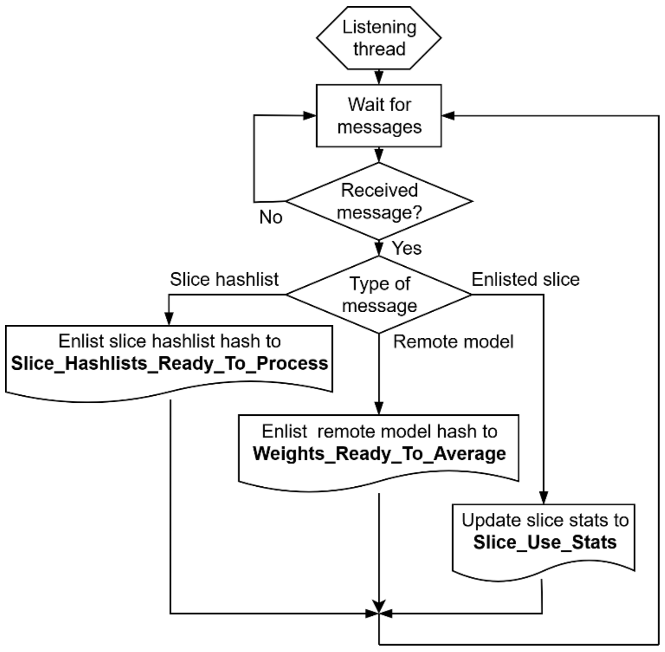

Listening thread
Wait for messages
Received message?
No
Yes
Type of message
Slice hashlist
Enlisted slice
Remote model
Enlist slice hashlist hash to **Slice_Hashlists_Ready_To_Process**
Enlist remote model hash to **Weights_Ready_To_Average**
Update slice stats to **Slice_Use_Stats**

Figure 20: Workflow of a node's "Listening thread"

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.

3.4 Simulation

This initial approach was simulated on a virtual network of ten workstations. The network was built on a single commodity computer using Docker containers. The coordinating application was developed in C#, including the node remote management and the logging system. As shown in the following figures, simulation was used to investigate the impact of many critical variables such as data slice size and overuse threshold, as well as other resilience-related factors such as duplication level and overhead. The simulation results indicated that increasing the slice size has a beneficial effect on the average accuracy of the generated models (Figure 22). Reduced overuse threshold leads in slightly improved accuracy, since it decreases repeated training with the same data (Figure 23).

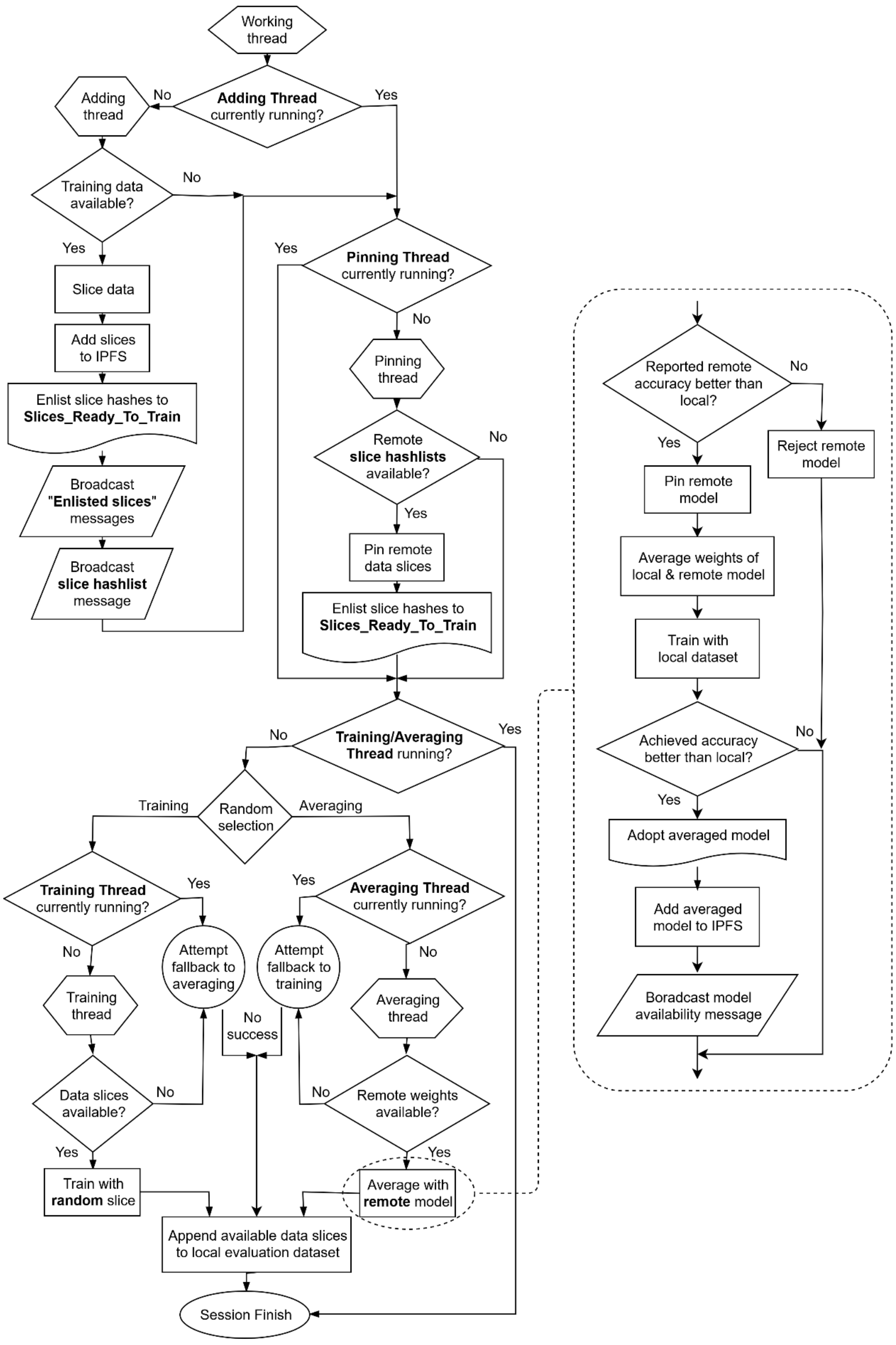

Working thread
Adding Thread currently running?
No
Yes
Adding thread
Training data available?
No
Yes
Slice data
Add slices to IPFS
Enlist slice hashes to Slices_Ready_To_Train
Broadcast "Enlisted slices" messages
Broadcast slice hashlist message
Pinning Thread currently running?
Yes
No
Pinning thread
Remote slice hashlists available?
No
Yes
Pin remote data slices
Enlist slice hashes to Slices_Ready_To_Train
Training/Averaging Thread running?
No
Yes
Random selection
Training
Averaging
Training Thread currently running?
Yes
No
Training thread
Attempt fallback to averaging
Data slices available?
No
Yes
Train with random slice
No success
Attempt fallback to training
Averaging Thread currently running?
Yes
No
Averaging thread
Remote weights available?
No
Yes
Average with remote model
Append available data slices to local evaluation dataset
Session Finish
Reported remote accuracy better than local?
No
Yes
Reject remote model
Pin remote model
Average weights of local & remote model
Train with local dataset
Achieved accuracy better than local?
No
Yes
Adopt averaged model
Add averaged model to IPFS
Boradcast model availability message

Figure 21: Workflow of a node's "Working thread"

As anticipated, the total number of bytes transmitted is proportional to the slice size chosen (Figure 24). Average resilience (Figure 25) is defined as the number of nodes that possess a requested slice/model. The percentage of duplicate data transmitted is shown in Figure 26, which is an inevitable overhead associated with gossip-based protocols.

Although initially very high, this percentage rapidly declines over time and is minimized for larger slice sizes.

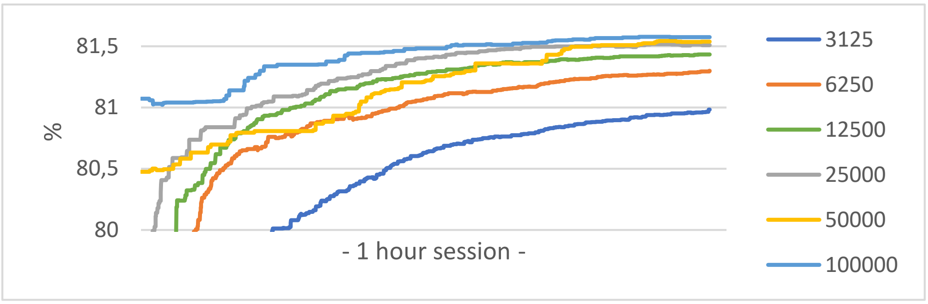


Figure 22: Average Accuracy per Slice Size (Overuse Threshold: 6)

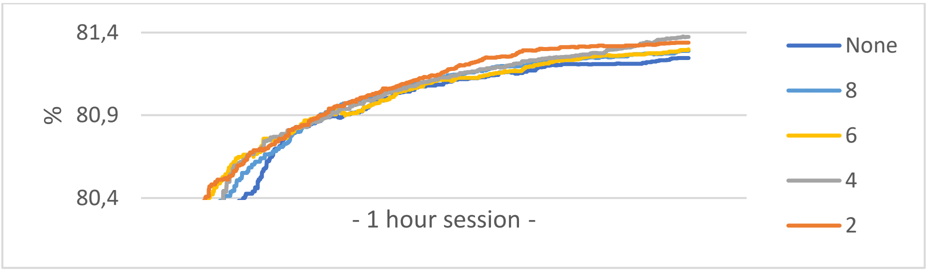


Figure 23: Average Accuracy per Overuse Threshold (Slice Size: 6250)

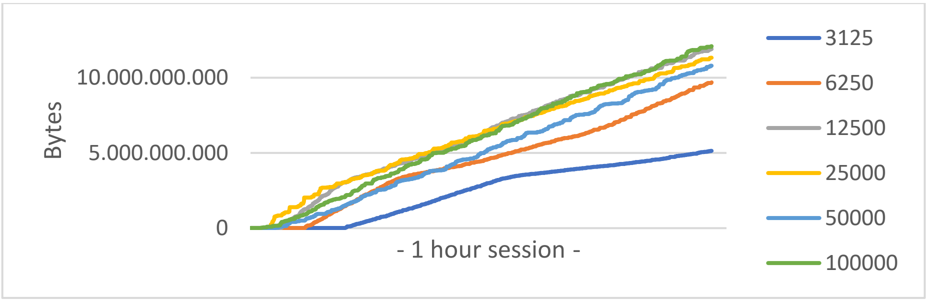


Figure 24: Total Bytes Sent per Slice Size (Overuse Threshold: None)

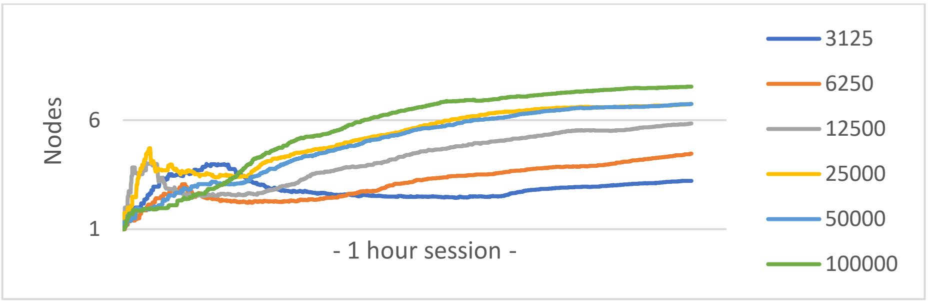


Figure 25: Average Resilience per Slice Size (Overuse Threshold: 2)

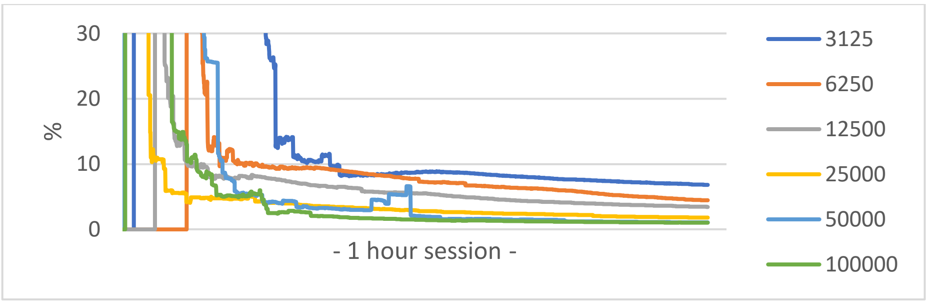


Figure 26: Duplicate Data Sent per Slice Size (Overuse Threshold: None)

3.5 Experiments

3.5.1 Scope

We demonstrate the first implementation of a real LEARNAE network, consisting of a typical LAN of 15 commodity PCs. Each machine has comparable hardware and software specs and lacks powerful GPUs, since we are comparing "stand-alone versus collaborative" training, and not actual values. The overall objective is to investigate the feasibility of utilizing contemporary DLTs as a mechanism for data diffusion, with the goal of gaining advantages from this cooperation that would not have been feasible with a stand-alone approach.

Additionally, the privacy benefits of this method will be discussed. The application scenarios are far from uncommon: A group of peers want to train a Neural Network cooperatively. They all agree to join as long as they are not required to disclose sensitive data. We propose a fault-tolerant approach for doing this via the use of distributed protocols to exchange just models, not training data.

LEARNAE's coordinating algorithm combines (averages) the transmitted models, resulting in improved models for all peers. The underlying assumption is that training data derived from diverse individuals exhibit some degree of consistency, which is true for the vast majority of the targeted use cases. Scenarios that allow for the exchange of training data will also be extensively investigated in the upcoming chapters.

3.5.2 Collaborative training sessions

The dataset used for this analysis (HEPMASS[20]) includes ten million instances of 28 attributes. The framework was evaluated with dataslice sizes of 500, 1000, 2000, 5000, 10000, and 50000 to show the effect of dataslice size on execution time, quantity of model data transferred, and obtained accuracy. All training sessions were conducted using Python and the Keras framework, with 32 instances as the default minibatch size. After training with a single slice, each node informs the rest of the network that the resultant model is ready for sharing and weight averaging.

3.5.3 Sample graphs

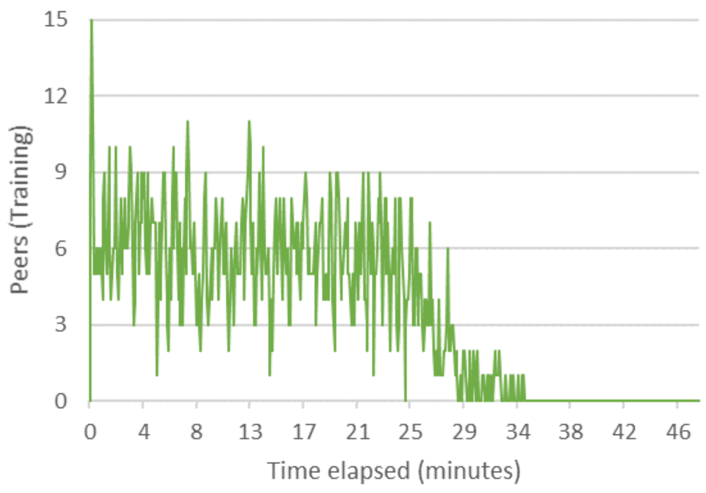
The remainder of this section highlights many important LEARNAE features via the use of sample graphs of sessions performed with 500 and 2000 instance dataslices. As anticipated, sessions based on 500-instance dataslices take longer to finish, a fact that is shown in all following figures. That is because, immediately after consuming a dataslice, a peer notifies others of the availability of a new model. Therefore, by increasing the number of dataslices results to a larger number of available models on the network, thus increased work in queue for all nodes.

3.5.3.1 Number of peers in training mode

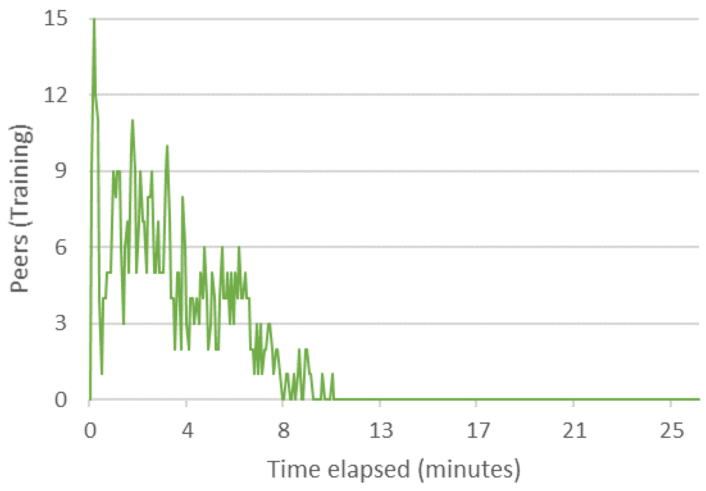
To begin averaging their local model, each peer must complete at least one training session (high values at graph start - Figure 27). Peer after peer consumes all local data, and the training process concludes. The time needed depends on the processing power of the node and the random rate at which it prefers training over averaging during each work cycle.

[20] https://archive.ics.uci.edu/ml/datasets/HEPMASS

As previously mentioned, 500-instance dataslices take longer to converge due to the additional processing cost associated with the increased number of models propagating the network.



(a) Dataslices of 500 instances



(b) Dataslices of 2000 instances

Figure 27: Number of peers in training mode

3.5.3.2 Number of peers in averaging mode

While averaging may begin concurrently with training, it may continue even after all available training data on all peers have been consumed. During the averaging phase, a peer inspects remote models via their broadcasted metadata and, if they meet specific criteria, fetches their chunks from several remote nodes and checks them locally to see whether they may contribute to the local model's improvement via weight averaging.

The technique a peer employs to determine if a distant model is a candidate for averaging is a point of high interest in our study, since an optimum strategy for selecting the most helpful models would result in increased overall efficiency. During these experiments, a remote model must pass two distinct tests. If its claimed accuracy (as specified in the metadata of its announcement message) is greater than the local one, an averaging operation will be attempted. If the averaged model improves the accuracy of the peer when using the local dataset, the peer adopts the model.

As shown in Figure 28, as peers attain a high level of accuracy, the likelihood of a successful averaging decreases. This fact leads in fewer new models being broadcasted to the network; as a consequence, the peers have less work to do and the whole process converges.

Because a node cannot conduct both training and averaging, these four graphs are complimentary when examined in detail.

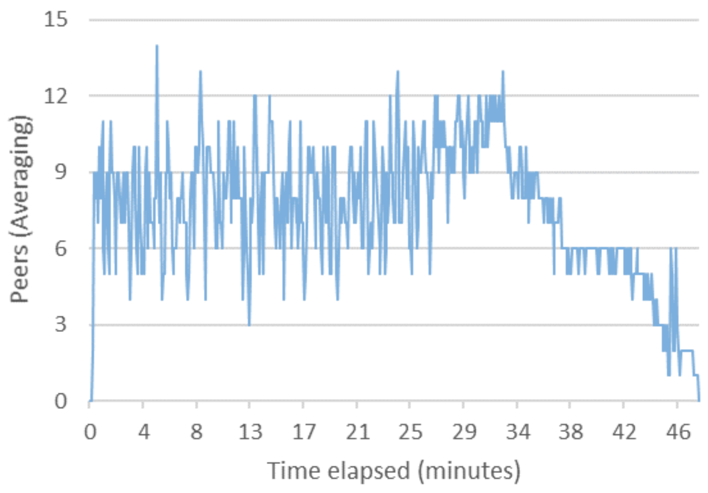
3.5.3.3 Data sent

Figure 29 shows 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.

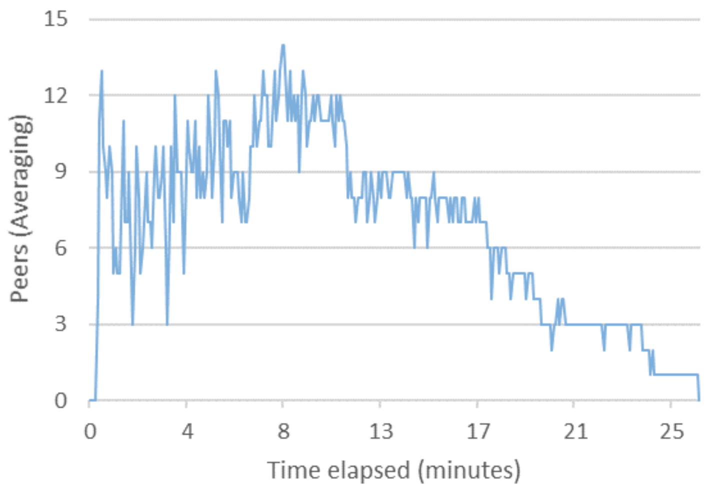
As expected, smaller size of dataslices results to higher network utilization, since there is a larger number of models announced to the participants, more averaging attempts, thus higher data transfer. Table 10 shows the exact values for the two sessions.

Table 10: Data sent – Peers & Total

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

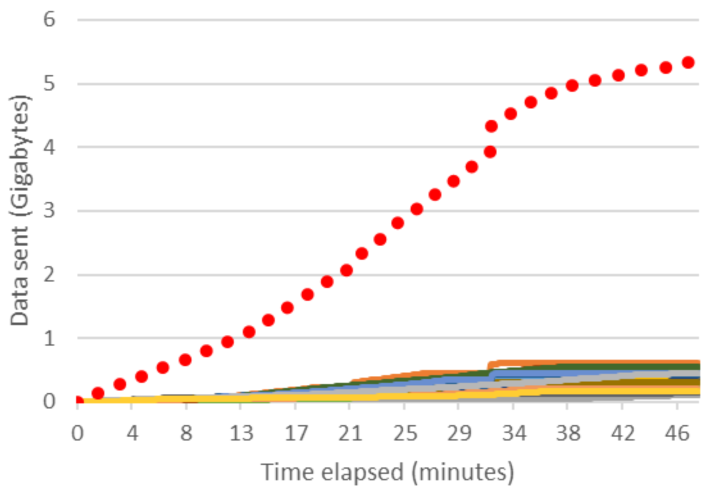


(a) Dataslices of 500 instances

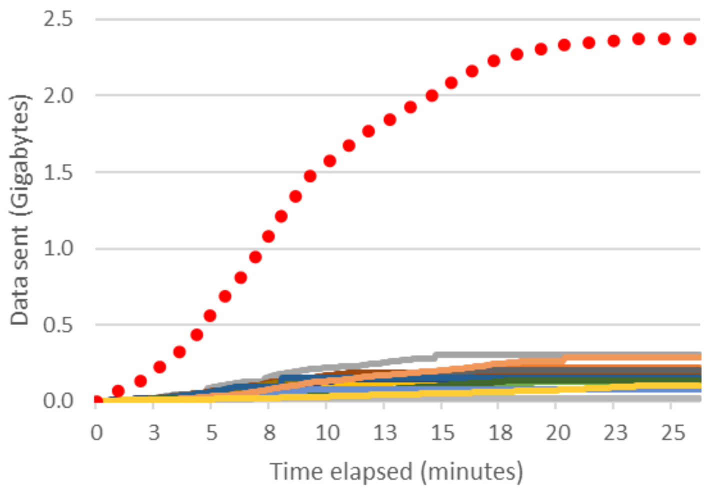


(b) Dataslices of 2000 instances

Figure 28: Number of peers in averaging mode

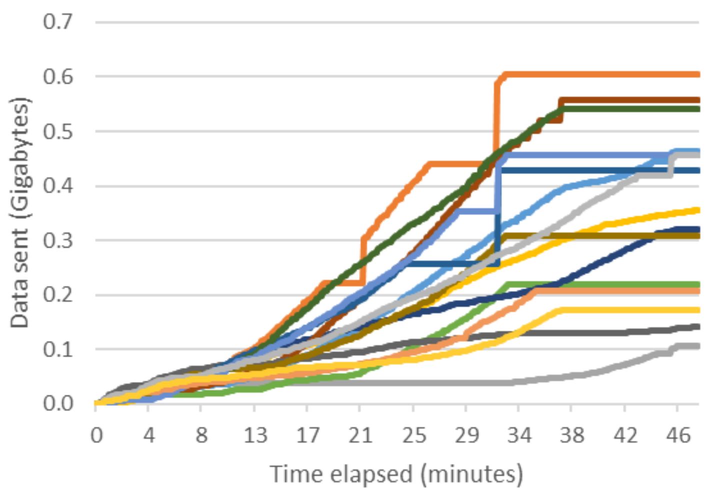


(a) Dataslices of 500 instances

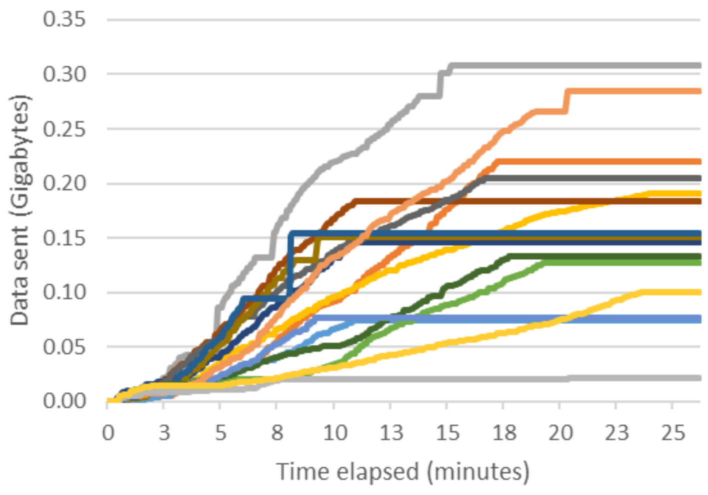


(b) Dataslices of 2000 instances

Figure 29: Data sent – Peers & Total



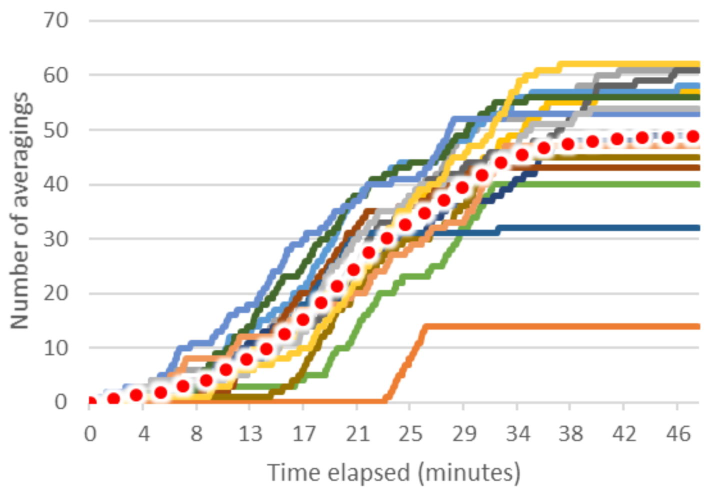
(a) Dataslices of 500 instances



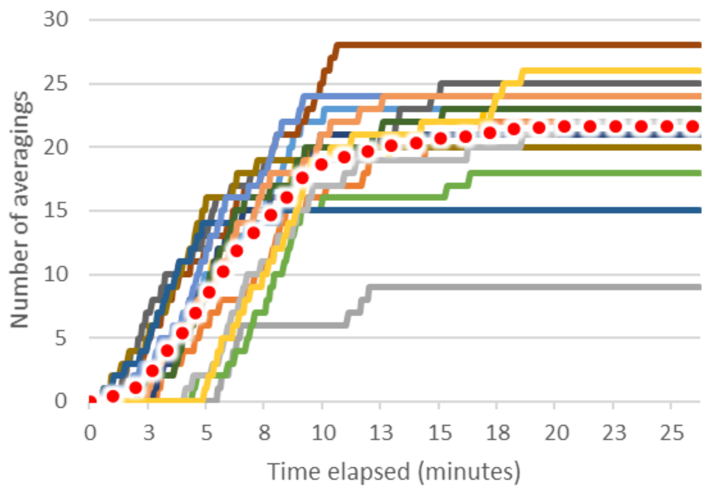
(b) Dataslices of 2000 instances

Figure 30: Data sent – Peers

The training data possessed by peers will never be of precisely equivalent quality, so each node will initially achieve different accuracy. As seen in Figure 30, peers that reach high accuracy during the early stages will be asked more frequently to send their local models to others.



(a) Dataslices of 500 instances)

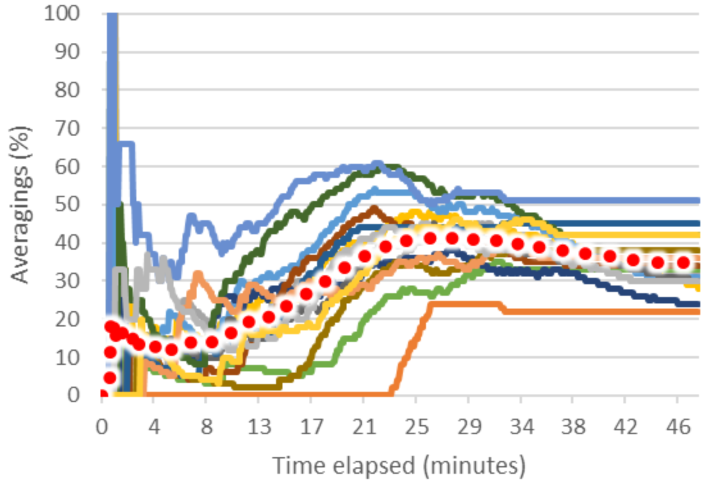


(b) Dataslices of 2000 instances

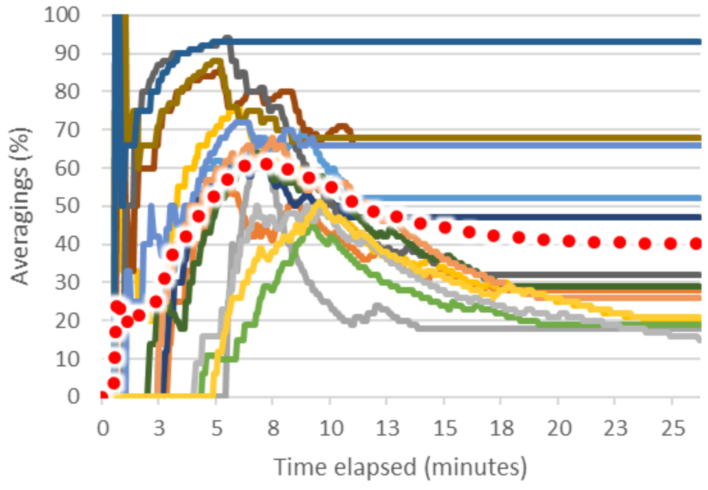
Figure 31: Successful average operations

3.5.3.4 Successful averagings

When a peer trains a new model, it makes it accessible to the rest of the network by broadcasting a packet containing the model's information. All participating nodes keep track of these messages and use them to determine if a remote model should be used to improve their local one. If a remote model satisfies specific criteria, the node retrieves it and attempts to average its own weights. This approach may or may not result in a better local model than the current one. If it does, the peer considers the averaging to be successful and accepts the produced model. As expected, the number of successful averagings is greater for smaller dataslices, as the total number of attempts is also greater (Figure 31).



(a) Dataslices of 500 instances



(b) Dataslices of 2000 instances

Figure 32: Successful average operations %

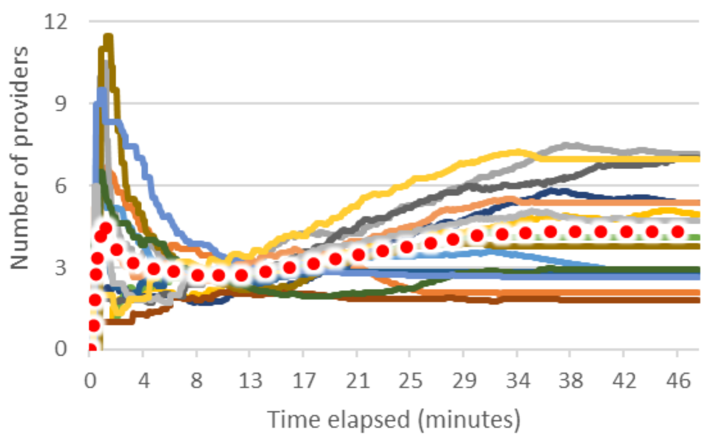
Figure 32 shows the typical curve of averaging success rate. At start, there is a noticeable accuracy disparity across peers, given the fact that the quality of training data used to train each of them may differ considerably. Initially, the graph shows a spike, as less accurate nodes benefit from the information given by their more accurate neighbors. When the accuracy discrepancies among the peers are reduced, the success rate of averagings marginally decreases for a brief period of time. The network then begins to profit from freshly consumed data and improved models published during the training phase. The success rate increases as the training period progresses. Following then, only averaging is performed and all members attempt to improve their models further by using the network's most recent combined models.

Finally, peers reach maximal convergence, all efforts are halted, and the successful averaging percent is stabilized at a terminal value.

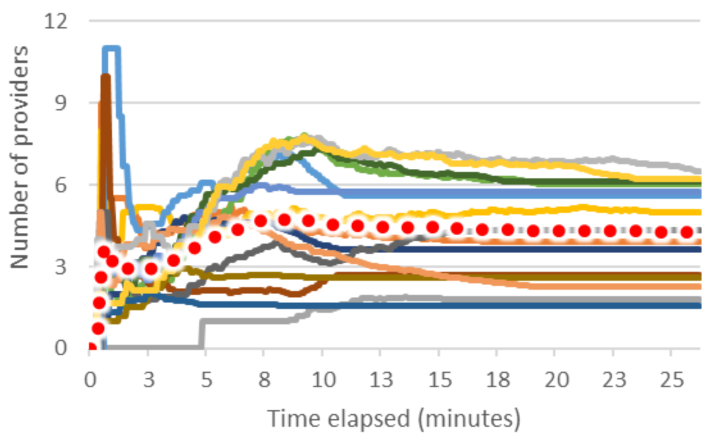
3.5.3.5 Resilience

The term "resilience" refers to the amount of different remote peers that can supply a requested chunk of either a model or training data at any given moment. Because this study focuses on privacy-preserving setups, all network data is related to stored neural models.

The initial surge (Figure 33) occurs because all nodes must first complete a training phase, during which their models are broadcasted and rapidly fetched by other peers. Due to the fact that averaging increases the number of broadcasted models, we notice a temporary decrease in data availability. Finally, as averaging declines, new model creation decreases, stabilizing the resilience value.



(a) Dataslices of 500 instances



(b) Dataslices of 2000 instances

Figure 33: Resilience

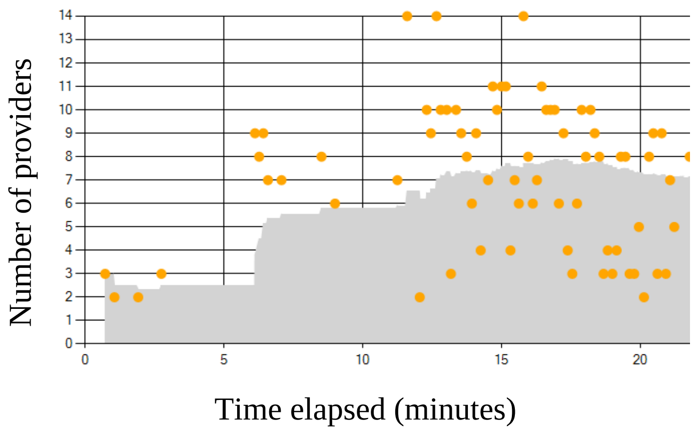
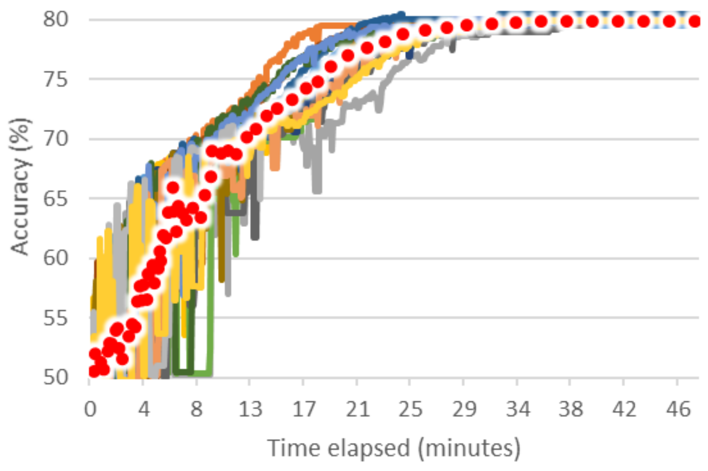


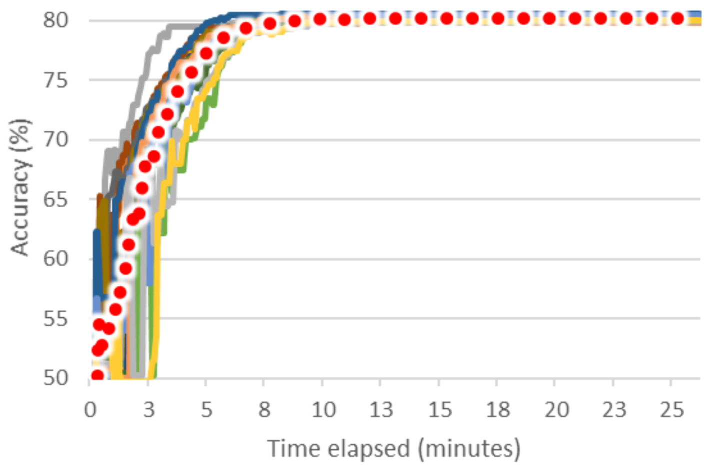
Figure 34: Sample snapshot of availability dispersion (random peer)

On both setups (50000-instance training data, 500 and 2000-instance dataslices), the terminal value of resilience was around 4.3, indicating that chunks requested by nodes were stored in an average of 4.3 other peers. Figure 34 illustrates the dispersion of chunk availability (orange dots) and resilience value (gray area) of a random peer participating in a 15-node training session utilizing 2000-instance dataslices.

Our proposal emphasizes resilience, since one of LEARNAE's primary objectives is fault tolerance and the ability to continue collaborative training even if a part of the network is disrupted. In practice, a resilience score of 4.3 indicates that the process could continue without obstruction even if a significant portion of the nodes were disconnected.



(a) Dataslices of 500 instances



(b) Dataslices of 2000 instances

Figure 35: Overall network accuracy

3.5.3.6 Overall network accuracy

The mean value of the final accuracy of all participating nodes was used as a measure for the network's achieved accuracy (Figure 35). The following graphs illustrate how the dataslice size selection impacts overall accuracy. From a theoretical standpoint, there are two critical opposing forces at work throughout a LEARNAE collaborative training session. By selecting bigger dataslices, the proportion of pure training in the whole process is enhanced. This leads in a more efficient training process and more accurate models. However, bigger dataslices have the disadvantage of decreasing the likelihood of effective averaging and the associated advantages to overall accuracy. As shown in Figure 36, there is a sweet spot for this tradeoff that results in optimum neural models. This was accomplished in the current setup, for dataslices containing 2000 instances.

The same image illustrates the advantages of LEARNAE's coordinating algorithm in comparison to a stand-alone scenario in which all peers train their models independently of one another. For the sweet spot, the accuracy gain is maximized (green diamond). The actual numerical statistics are included in Table 11.

At the graph's right edge, where all data is treated as a single dataslice, both techniques provide the same result since no effective averaging can occur, and the LEARNAE network degenerates into a collection of isolated workstations.

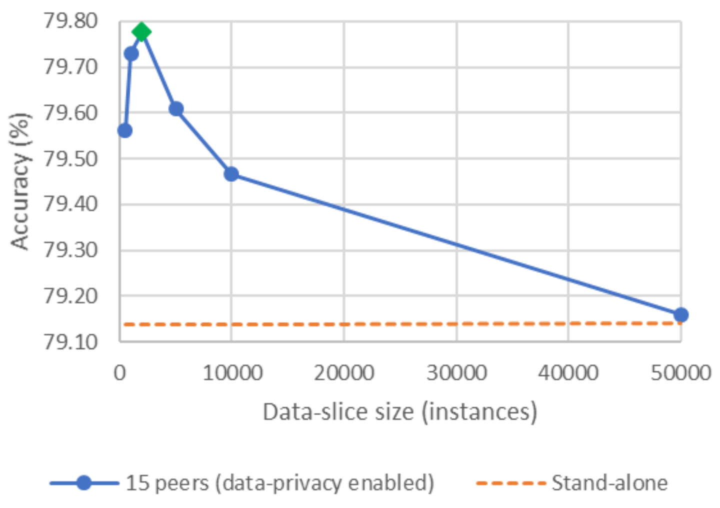


Figure 36: Collaborative overall network accuracy per dataslice size

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.

Table 11: 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	

3.6 Summary

Our work is a first approach on leveraging Distributed Ledger Technology for purely decentralized NN training. In first phase, being a proof-of-concept, it had a limited applicability and it was tested solely on a single machine using virtualization containers.

Then we expanded our work and applied it 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.

The experiments indicated that our proposal can offer tangible gain to collaborating peers, without compromising data privacy. The results showed a significant improvement compared to nodes working in isolation.

4 Using Distributed Ledger Technology to Democratize Neural Network Training

4.1 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 need the presence of a management entity, such as a parameter server or a node with enhanced permissions. This often results in a communication bottleneck and necessitates the use of high-performance networking; it also results in the creation of a single point of failure. On the other hand, synchronous approaches impose some kind of time-based coordination between the peers. There are solutions in which all nodes must operate in exact concurrent phases and others in which they must share a common clock. Often, these solutions are troubled by locks and stale updates caused by slow workers.

The following sections summarize previous proposals, which include the following:

1) centralized synchronous
2) centralized asynchronous
3) decentralized synchronous
4) decentralized asynchronous

4.1.1 Centralized synchronous

Sandblaster L-BFGS [10] and Parallel minibatch SGD [108] both need a parameter server and a high-performance infrastructure.

By estimating redundancy, Parameter Server [109] provides methods to minimize the impact of slow nodes and introduces fault management. FireCaffe [99] is built on the Caffe [110] framework, employs a proprietary MapReduce protocol, and needs high-speed networking. CaffeOnSpark is based on the DistBelief paradigm and is implemented on top of the Spark framework. Additionally, each peer acts as a parameter server for a subset of the model. BigDL [111] mostly follows the CaffeOnSpark concepts, with the exception of the parameter exchange method. It requires distinct training and data exchange cycles, which results in a strictly synchronous functioning scheme. MPCA

SGD [100] partitions the model into shards that are reduced and shared independently, requiring a driver node to coordinate the process.

4.1.2 Centralized asynchronous

HOGWILD [96] minimizes communication requirements by using local optimization and asynchronous model merging through a parameter server. DistBelief [10] is a well-known technique that makes use of a cluster of parameter servers and their peers. Each server-worker group is responsible for a subset of the model. Following each cycle, the peers must download the updated version of the joint model.

Adam [36] clusters the parameter servers and tries to reduce network traffic by offloading certain processing from the workers to the servers. Elastic Averaging SGD (EASGD) [95] runs the optimizers on the nodes, which interact with a parameter server separately every N work cycles. MXNet [98] defines a hierarchical parameter server topology in which intermediary nodes may function as proxy servers.

Petuum [112] introduces the concept of staleness norms in order to improve synchronicity and convergence speed. Selective SGD [97] minimizes communication requirements by using local optimization and asynchronous model merging via a parameter server. TensorFlow [94] may be regarded the successor of DistBelief, since it incorporates automated computation graph optimization, which significantly simplifies distributed model parallelism.

4.1.3 Decentralized synchronous

SparkNet [113] adopts FireCaffe's methodology while attempting to adapt to low-bandwidth networks. It implements decentralized synchronous training. Thus, for N cycles, each worker executes a separate optimizer in isolation. Averaging is then used to decrease the size of the resultant models.

Prior to the start of the next computing cycle, the averaged model is broadcast to all workers and takes the place of their local models. The nodes in Decentralized Parallel Stochastic Gradient Descent (D-PSGD) [114] are synchronized through a shared clock and exchange parameters after each training cycle.

4.1.4 Decentralized asynchronous

GoSGD [115] implements the EASGD algorithm using a mesh topology for peer organization. Every Nth cycle, a randomized algorithm selects the pairs of workers who will exchange data [116]. On top of the Spark framework, DeepSpark [117] tries to implement EASGD on commodity hardware. Asynchronous Decentralized Parallel Stochastic Gradient Descent (AD-PSGD) [118] is built on a ring-based network architecture, and after each iteration, each worker chooses a neighbor for averaging, at which point both workers replace their local models with the averaged one.
Recent work has focused on improving resource utilization, either via parallelization of computation and communication or through the use of intuitive scheduling. Dianne [119] is a Java-based distributed framework that makes use of the Torch backend. It decomposes the neural network into distinct building blocks and associates each block with a particular node. It contains components for training and assessing models with the help of a parameter server.

MXNet-MPI [120] is a modified version of MXNet that aims to integrate the best features of synchronous and asynchronous solutions. It proposes clustering the workers into groups that run SGD with AllReduce separately. Scalability and fault tolerance are enforced via this clustering. Horovod [121] utilizes MPI to build a new training layer for AllReduce and integrates it into Tensorflow. Because it is primarily concerned with GPUs, it makes extensive use of libraries that support many GPUs on a single worker. Due to the fact that it sets a priority on computing speed, it lacks scalability and fault tolerance.

ByteScheduler [122] is predicated on the premise that splitting and reordering tensor transmissions may enhance performance in distributed environments. It proposes that by altering the transmission order of the neural layers, training speed may be improved and the effect of communication overhead can be minimized. BytePS [123] is an open source project that focuses on heterogeneous architectures using GPU clusters. It makes an effort to optimize distributed training tasks by using their spare CPU and bandwidth resources. [124] proposes a compressed SGD using Nesterov's momentum that is two-way (both to and from workers). To minimize communication costs, it divides the gradient into blocks that are compressed and sent in a 1-bit format via a scaling factor. Independent Subnet Training [125] divides the neural network into a set of equal-depth subnetworks. Each work cycle entails training the subnet on a local level and sharing results with other peers. The method does not need parameter aggregation, since there are no shared parameters across the subnets. This feature increases subnet independence and lowers the necessity for communication.

The Computation and Communication Decoupled SGD (CoCoDSGD) algorithm [126] proposes a method for parallelizing computation and communication. The workers do not share data on a per-iteration basis, but rather on a periodic basis. This leads in more efficient resource use and cost savings associated with communication. Geryon [127] is a proposal for speeding CNN training by including a network-level scheduling algorithm. It categorizes model parameters according to their degree of urgency. The parameters that have the greatest effect on the model's quality are prioritized and transmitted first. Geryon accelerates the training process but has no impact on the model accuracy achieved.

Another communication scheduler is Preemptive AllReduce Scheduling for Expediting Distributed DNN Training (PACE) [128]. It is based on scheduling AllReduce tensors in advance using the DAG from DNN training. It aims to determine the optimum level of granularity for tensor communication in order to maximize the overlap between communication and computation functions. As a consequence of this scheduler, overhead is reduced and bandwidth utilization is maximized. Priority-based Parameter Propagation for Distributed DNN Training (P3) [129] is a two-part technique that includes parameter slicing and priority-based updating. The first method subdivides the model layers into smaller sublayers and independently synchronizes them. The second determines the priority of each slice depending on its recurrence in the following iteration. P3 guarantees that the most critical slices always have the necessary network resources.

The concepts around which SwitchML [130] is founded are as follows: To begin, parameter changes may be split into pieces that can be handled separately. Second, SGD aggregation may be performed independently on various subsets of the input data, regardless of their order. Third, machine learning training is robust to approximations of its compute operations. The authors suggest a technique in which the switch pipeline processes the generated chunks and packet loss may be tolerated via the use of a lightweight switch scoreboard mechanism and a retransmission mechanism controlled exclusively by end hosts.

TicTac [131] is a TensorFlow-based framework that aims to anticipate the sequence in which parameters will be used by the underlying computational model. As a result, it optimizes network transfers to minimize communication time. TonY [132] is a free and open-source orchestrator that includes a client and a scheduler. Users may submit machine learning tasks via the client, and the scheduler takes care of assigning resources, configuring the environment, and executing the machine learning work in a distributed manner. [133] addresses the scalability issue that exists on public cloud clusters due to their low interconnection bandwidth. It proposes a top-k sparsification library for computation and communication that will be optimized. Additionally, it incorporates a multi-level data caching technique to improve I/O functions and introduces a new parallel tensor operator to speed up update operations. The aforementioned characteristics are used by a hierarchical communication algorithm, which tries to combine sparsified gradients in order to maximize processing power utilization.

4.2 LEARNAE system

4.2.1 Coordinating algorithm

The data flow between various kinds of nodes is shown in Figure 37. The workflow of a Full Node is shown in Figure 38. When a collaborative session begins, peers who possess training data divide it into predefined-size "dataslices." These files are added to IPFS individually and their hashes are aggregated into a hashlist. This file is then added to the network's shared DHT, and its hash is broadcasted (purple area). Each time a peer consumes a dataslice, the peer broadcasts a message informing the other peers that the dataslice has been used. This facilitates LEARNAE's *overuse threshold* feature, which allows users to set the maximum number of times a particular dataslice may be utilized for training throughout the network.

When a peer gets a message indicating the availability of a dataslice, it retrieves its chunks from several neighbors and adds it to the queue containing locally accessible dataslices; unless the slice has already exceeded the preset overuse threshold, in which case the peer ignores it (red area).

Parallel to the aforementioned functions, peers randomly choose between training and averaging in every work cycle, increasing the overall stochasticity. Following each training phase, peers add the created local model to IPFS and announce its availability (green area). When a peer chooses averaging, it looks through its local list of remote models. If a model with a claimed accuracy higher than the local one is discovered, the peer fetches the remote model and averages it with the local one. If this procedure results in an increase in local accuracy, the peer accepts the averaged model, adds it to the shared DHT, and announces its availability to the network. The message regarding available models includes information about the broadcaster's accuracy and the model's maturity, e.g., how many work cycles preceded its development (blue area). After all averaging attempts have been performed and the network has attained maximum convergence, each peer retrieves the one remote model with the highest accuracy. This model is compared against the local dataset and, if it performs better, it is accepted as is by the peer.

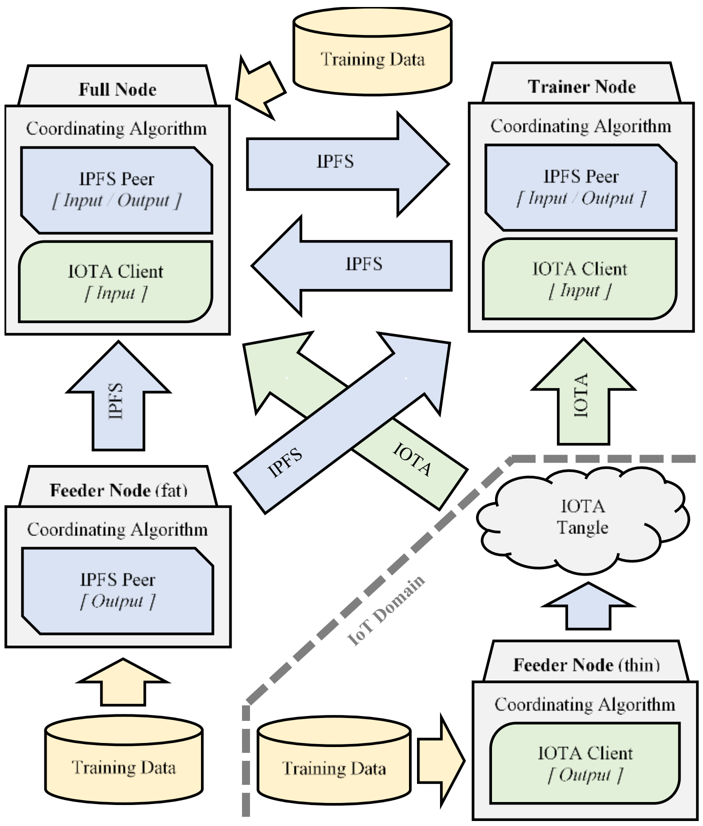


Figure 37: Data flow between different node roles

4.2.2 IoT Implementation

Our work introduces a way to incorporate data from lightweight IoT devices using the IOTA Tangle (Figure 39). The participating peers that are capable of training models scan the Tangle for new messages on a periodic basis. If they find new data, they save them locally.

Due to the Tangle's IoT-centric design, it works best when transmitted packages are not too big. The next chapter contains details on the experimental metrics. The saved sensor data are kept in a buffer and are added to the list of dataslices that are ready to be utilized for training when their size reaches the preselected dataslice size; the buffer is then purged.

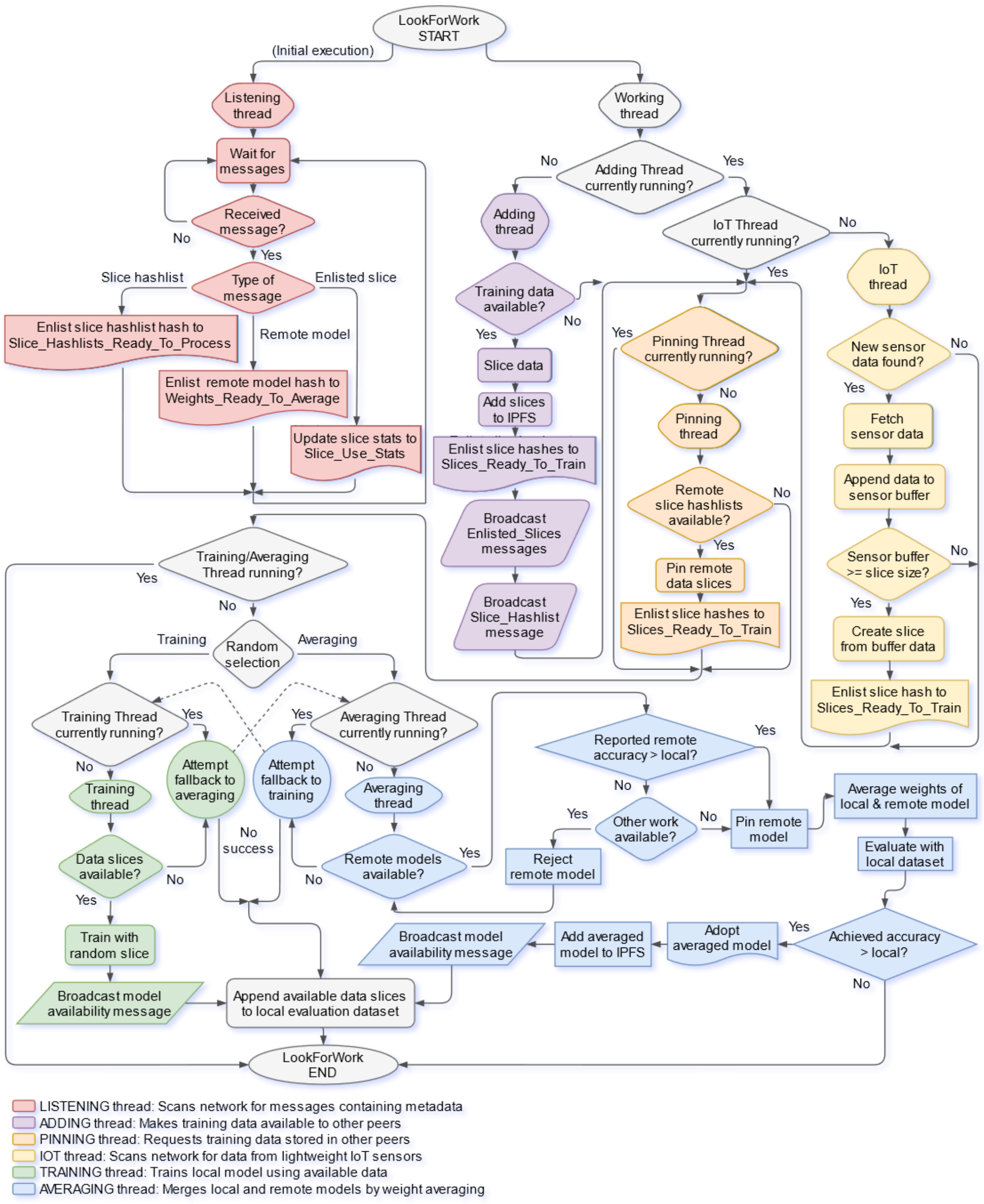

LookForWork START
(Initial execution)
Listening thread
Wait for messages
Received message?
No
Yes
Type of message
Slice hashlist
Enlisted slice
Remote model
Enlist slice hashlist hash to Slice_Hashlists_Ready_To_Process
Enlist remote model hash to Weights_Ready_To_Average
Update slice stats to Slice_Use_Stats
Working thread
Adding Thread currently running?
No
Yes
Adding thread
Training data available?
Yes
No
Slice data
Add slices to IPFS
Enlist slice hashes to Slices_Ready_To_Train
Broadcast Enlisted_Slices messages
Broadcast Slice_Hashlist message
IoT Thread currently running?
No
Yes
IoT thread
New sensor data found?
Yes
No
Fetch sensor data
Append data to sensor buffer
Sensor buffer >= slice size?
Yes
No
Create slice from buffer data
Enlist slice hash to Slices_Ready_To_Train
Pinning Thread currently running?
Yes
No
Pinning thread
Remote slice hashlists available?
Yes
No
Pin remote data slices
Enlist slice hashes to Slices_Ready_To_Train
Training/Averaging Thread running?
Yes
No
Random selection
Training
Averaging
Training Thread currently running?
Yes
No
Training thread
Data slices available?
Yes
No
Train with random slice
Broadcast model availability message
Attempt fallback to averaging
Attempt fallback to training
No success
Averaging Thread currently running?
Yes
No
Averaging thread
Remote models available?
Yes
No
Reported remote accuracy > local?
Yes
No
Other work available?
Yes
No
Reject remote model
Pin remote model
Average weights of local & remote model
Evaluate with local dataset
Achieved accuracy > local?
Yes
No
Adopt averaged model
Add averaged model to IPFS
Broadcast model availability message
Append available data slices to local evaluation dataset
LookForWork END
🟥 LISTENING thread: Scans network for messages containing metadata
🟪 ADDING thread: Makes training data available to other peers
🟧 PINNING thread: Requests training data stored in other peers
🟨 IOT thread: Scans network for data from lightweight IoT sensors
🟩 TRAINING thread: Trains local model using available data
🟦 AVERAGING thread: Merges local and remote models by weight averaging

Figure 38: The principal parts of a Full Node's workflow

4.2.3 Configuration of the MAM Stream.

A MAM message may have a tag field in addition to the data part. Because the MAM API allows searching by tag, LEARNAE uses it to connect all sensors associated with a particular training session. Each sensor is linked with its own MAM stream. The first message is tagged with a preagreed codename (which may correspond to the IPFS PubSub topic name) and includes the sensor ID and its creation date. This head message indicates the location of the first data message, and so on. All data is in JSON format.

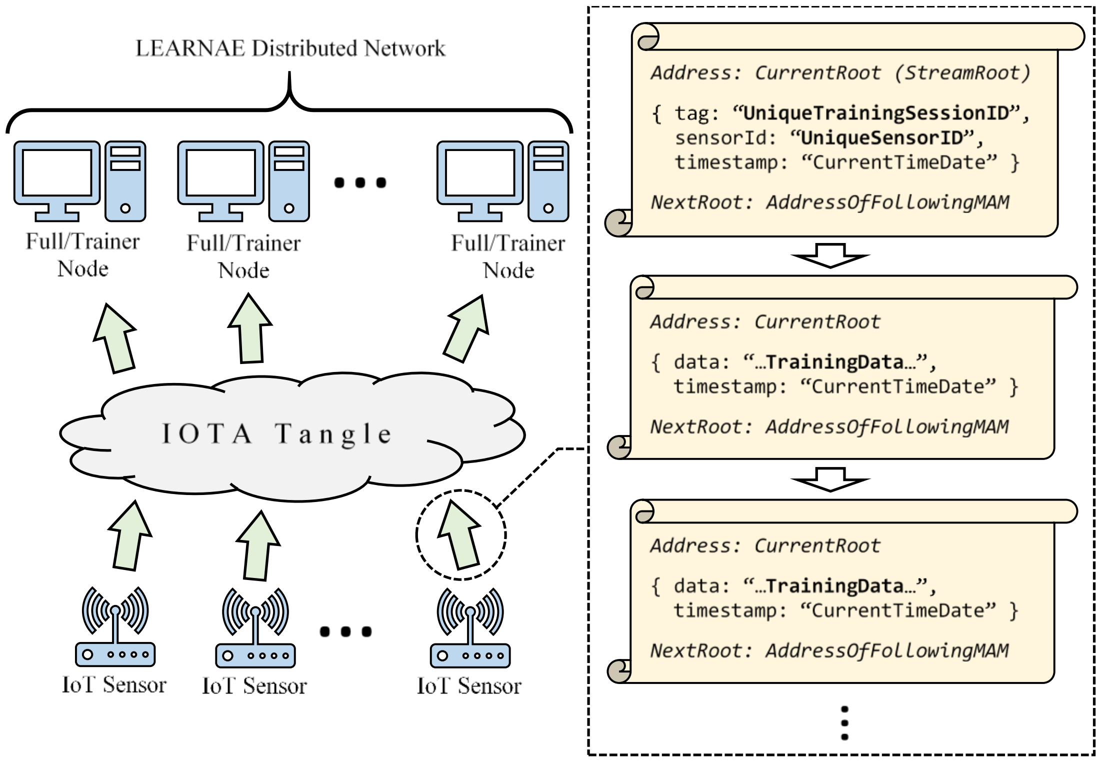

LEARNAE Distributed Network
Full/Trainer Node
Full/Trainer Node
Full/Trainer Node
IOTA Tangle
IoT Sensor
IoT Sensor
IoT Sensor
Address: CurrentRoot (StreamRoot)
{ tag: "UniqueTrainingSessionID", sensorId: "UniqueSensorID", timestamp: "CurrentTimeDate" }
NextRoot: AddressOfFollowingMAM
Address: CurrentRoot
{ data: "…TrainingData…", timestamp: "CurrentTimeDate" }
NextRoot: AddressOfFollowingMAM
Address: CurrentRoot
{ data: "…TrainingData…", timestamp: "CurrentTimeDate" }
NextRoot: AddressOfFollowingMAM

Figure 39: The structure of MAM messages created by LEARNAE

The method of embedding an IoT sensor into a LEARNAE network consists of these steps:

(a) To begin, the enquiring peer seeks MAM messages with the specified tag from the Tangle. If it finds a new message (e.g., with a first-seen address), attaches it to the list of known sensors.

(b) Then, each time the peer scans for new IoT data, it starts checking the linked list's tip of each known sensor.

(c) 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 12: Configuration 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

The enhancements to LEARNAE described here were tested via the use of a large number of distributed training sessions. The network comprised of twenty workstations that were set as Virtual Private Servers in the cloud (VPS). Their configuration was chosen to match that of a typical personal computer, as shown in Table 12. A particularly compelling use of distributed training is in environments where each peer has a very limited number of instances. In this case, cooperation may help alleviate the data scarcity restriction and provide more accurate models.

Table 13: 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
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 prior research, the experiments in this phase concentrate only on such use cases, limiting each peer's available training data to 5,000 instances. Because all tests are designed to ensure anonymity, participating peers share just models, not training data. The following sections quantify the proposed architecture's advantages.

4.2.4 IoT evaluation

4.2.4.1 Sensor Setup

To mimic an IoT sensor, we used a low-power SBC: the Raspberry Pi 3 Model B. The hardware specs and major software/development decisions used to implement these tests are listed in Table 13.

4.2.4.2 Dataset Characteristics

The dataset employed was HEPMASS, which was created for the purpose of training systems on exotic particle identification via the utilization of a high number of collisions. It contains 7 million training instances (5 GB) and 3 million testing instances (2.5 GB). Each instance is approximately 750 bytes in size and is composed of 30 floating point integers (including metadata).

For these experiments, the data are sent uncompressed and unencrypted. Each dataset instance simulates the reception of a new signal by a sensor. A transmission packet is composed of several instances. This number is related to the instance size in order to generate packets with optimum propagation across the Tangle.

4.2.4.3 Data Publishing

The training dataset was divided into 50-instance packets. As a result, each MAM message broadcast to the Tangle was about 35 KB in size. This was determined to be the optimal value, since lower values could not achieve maximum network utilization and higher values were sometimes refused by IOTA nodes due to congestion or restrictions preventing misuse. Table 14 presents the metrics of the sensor data publishing procedure.

4.3 Fault tolerance evaluation

To assess the negative impact of network disruptions, a new subsystem was implemented in LEARNAE to simulate peers shutting down or experiencing connection issues. Three new testing parameters control the severity of the simulated issues:

- *Offline Cycle*, which specifies the frequency with which the disconnection status is updated;
- *Offline Duration*, which specifies the duration of the disconnection;
- *Offline Probability*, which specifies the likelihood of the disconnection to occur.

Table 14: 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

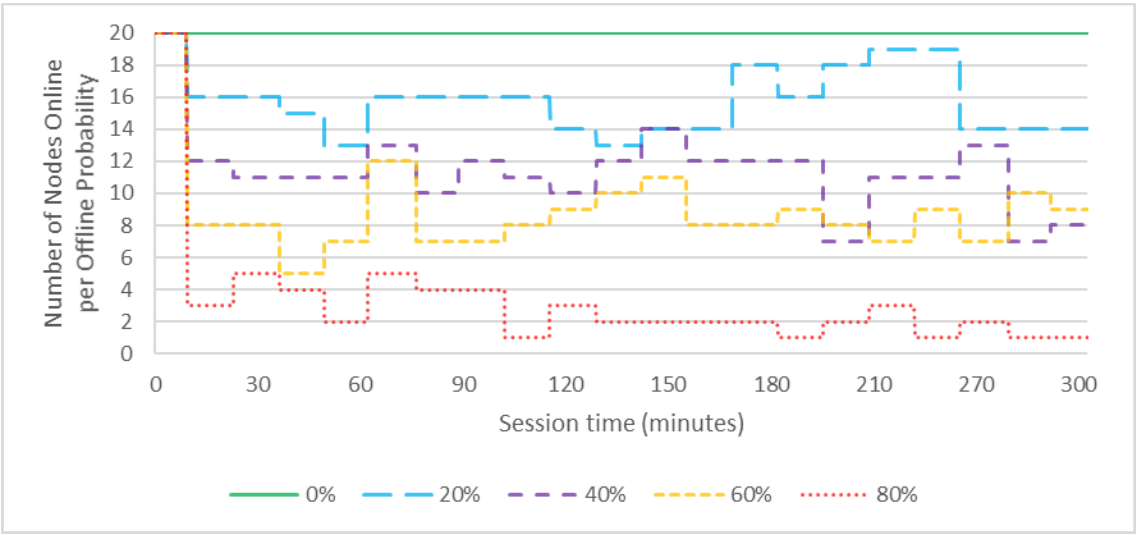


Figure 40: The number of online peers for different Offline Probability

For these experiments, the first two values were set to 15, indicating that peer connectivity was updated every 15 minutes and that the new state persisted until the next update. To study the impact of these issues, all experiments were conducted with 0%, 20%, 40%, 60%, and 80% Offline Probability. The fluctuation in the number of online peers throughout the trials is shown in Figure 40.

4.3.1 Low Epoch Conditions

A model may be enhanced by training and/or by averaging. When the number of epochs in a use case is low, the effect of training is decreased and averaging plays a greater role. To illustrate the full impact of network interruptions, early experiments were performed using a single training epoch. The achieved mean accuracy and spread for each offline probability are shown in Figure 41, while they are compared in Figure 42. For reference, these figures also depict the outcomes for a stand-alone configuration, in which each peer trains its model independently using just its own data. As shown in these figures, the proposed algorithm outperformed the stand-alone setup except for the 80% Offline Probability. The extended final spread (orange region) in the latter example (f) shows that this was the only case in which the network failed to reach proper convergence.

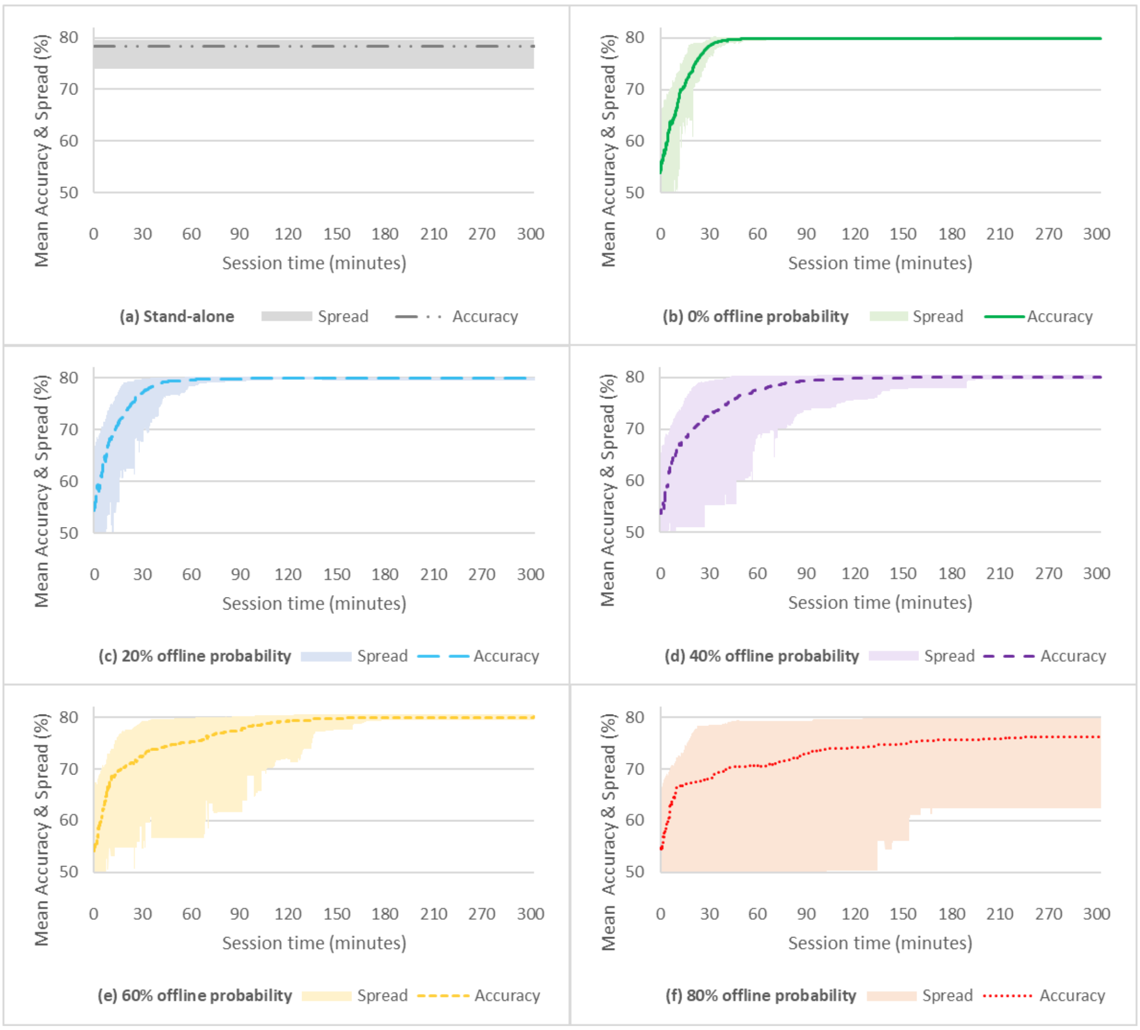


Figure 41: Mean Accuracy and Spread per Offline Probability (Low Epochs)

The ability to overcome difficulties caused by unavailable peers is described in LEARNAE's terminology as resilience, an index defined as the total number of neighbors from whom a data chunk may be retrieved at a given moment. To illustrate the obtained data replication, Figure 43 depicts the evolution of mean resilience and its spread over the period of a training session with no simulated disconnections. The final result of 6 indicates that on average, a requested chunk may be obtained from 6 distinct remote peers out of a total of 20.

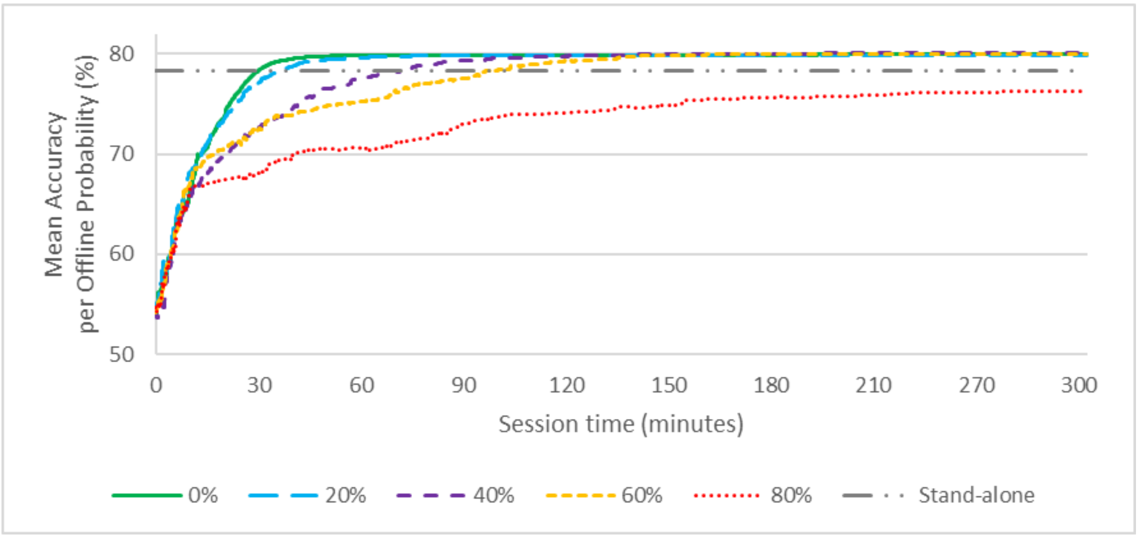


Figure 42: Comparison of Mean Accuracy for different Offline Probability (Low Epochs)

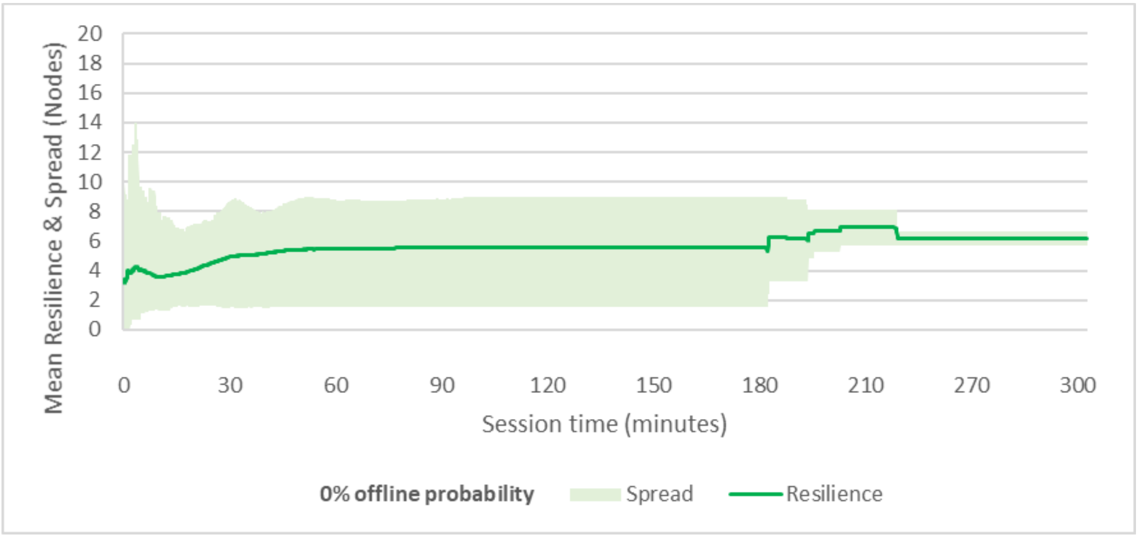


Figure 43: Mean Resilience and Spread

4.3.2 Optimal Epoch Conditions

The optimum number of epochs for the configuration of these experiments was determined to be 10. The following tests were conducted appropriately to maximize the beneficial impact of the training phases. As shown in Figure 44, disruptions with an Offline Probability of up to 80% have no impact on overall performance. The comparison of the achieved mean accuracy for various disconnection rates is shown in Figure 45. Even with an 80% disconnection rate, the proposed averaging algorithm was able to reduce the accuracy spread across peers by up to 3.24% (Table 15).

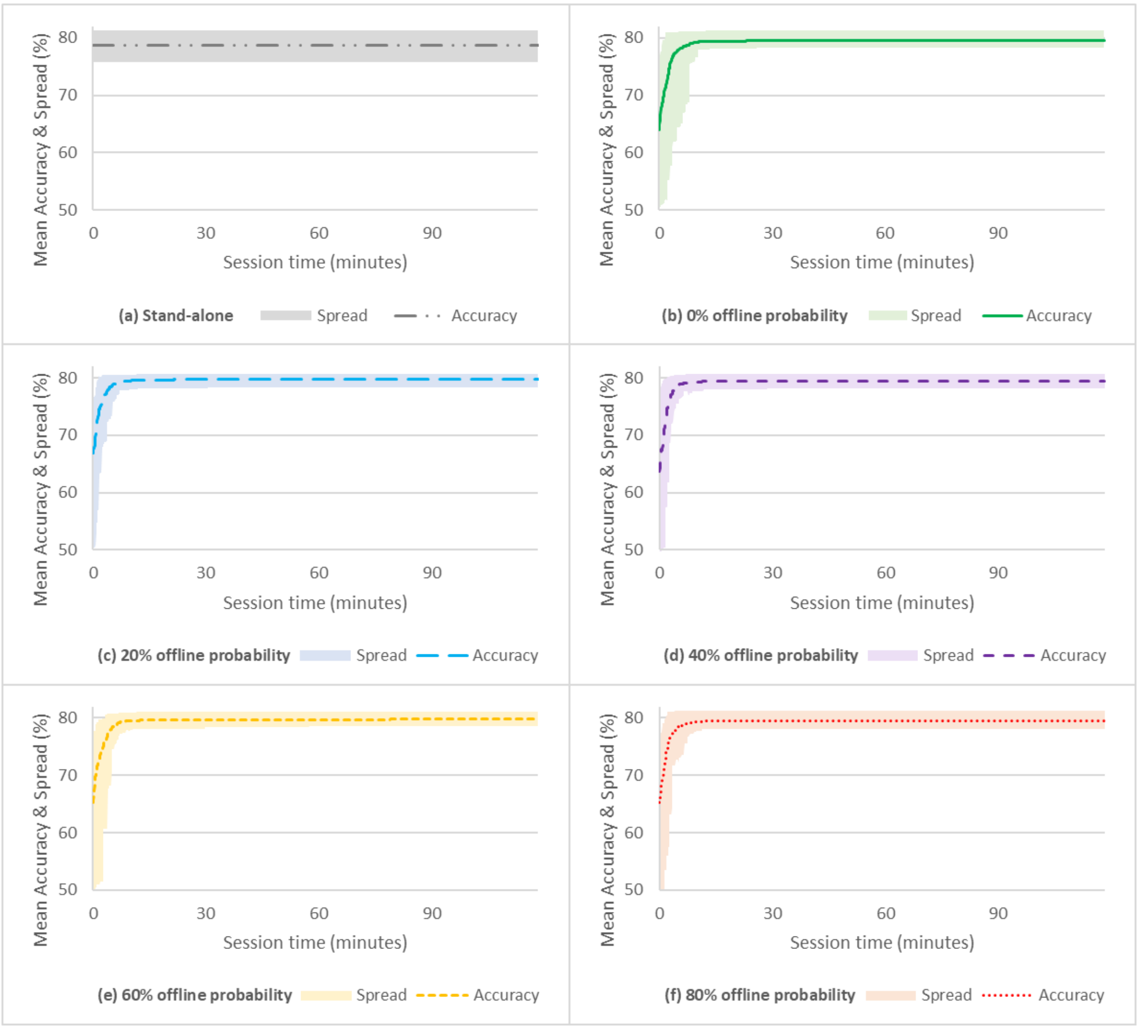


Figure 44: Mean Accuracy and Spread per Offline Probability (Optimal Epochs)

4.4 Data balancing

Due to the fact that the quality of the training data differs across peers, the produced models will have varying degrees of accuracy. This discrepancy is mode intense at the start of the collaborative session and diminishes as the session progresses and averaging is finished. The proposed architecture's distributed nature guarantees load balancing and the lack of congestion points. The quantity of data provided by each participant is shown in Figure 46. Peers who develop more accurate models earlier in the process must provide more data, since their model is heavily fetched by others. However, this spike is just transitory, lasting only as long as their data's replication factor in the network remains low.

Table 15: The improvement in convergence (distributed vs stand-alone)

Offline Probability *(%)*	Accuracy Spread *(%)* *[Stand-alone]*	Accuracy Spread *(%)* *[Distributed]*	Change in Spread *(%)*
0		3.04	-2.56
20		2.39	-3.21
40	5.60	2.52	-3.08
60		2.36	-3.24
80		3.15	-2.45

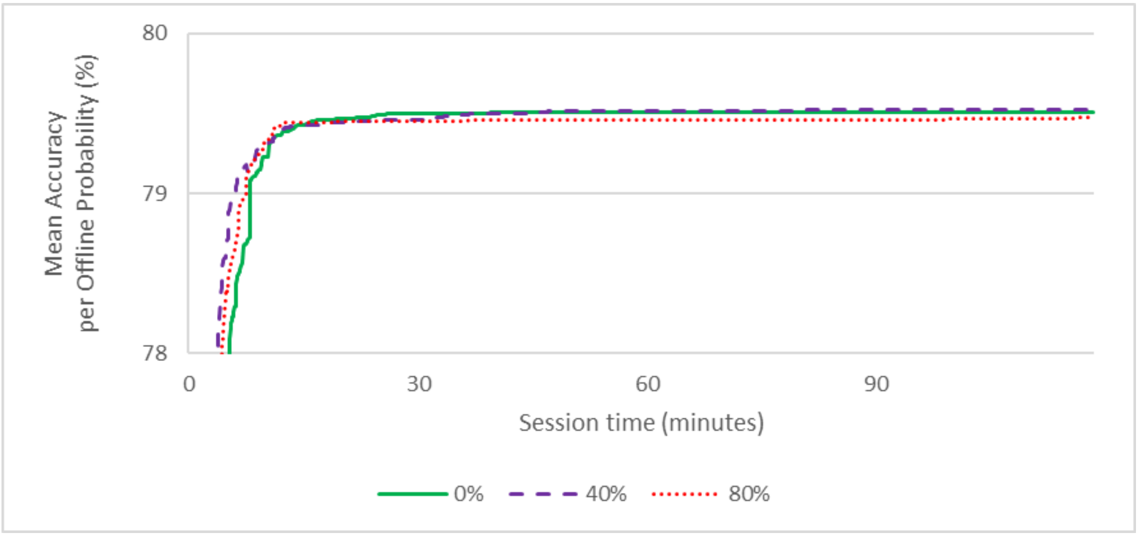


Figure 45: Comparison of Mean Accuracy for different Offline Probability (Optimal Epochs)

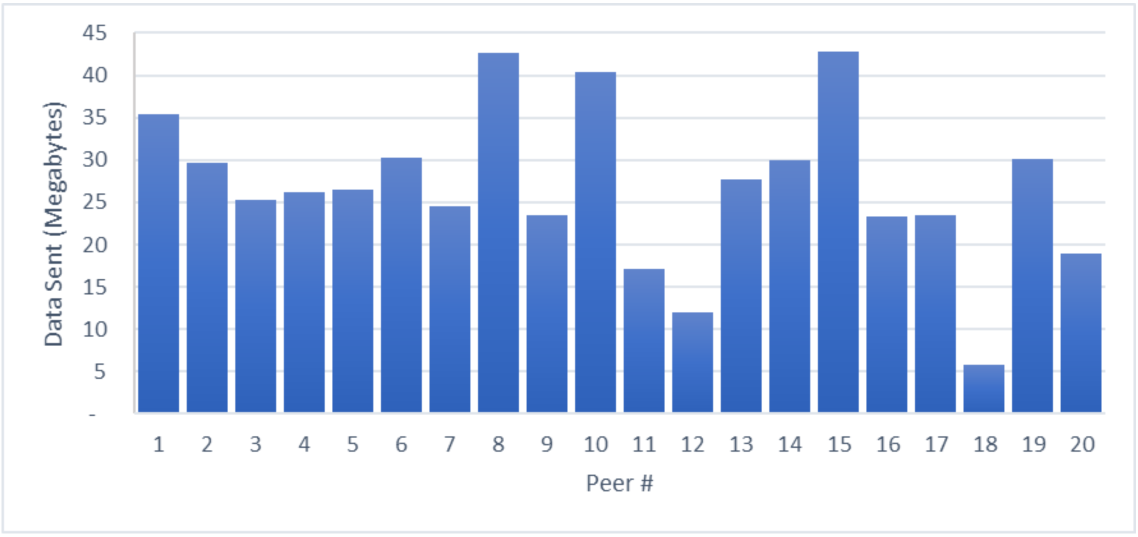


Figure 46: Total amount of data sent by each peer

4.5 Benefits of proposed architecture

As shown in Figure 47, at the first stage when data are poured into the network, peers are evenly split into two groups: training (green) and averaging (blue). After consuming all available data, no training processes are invoked, and weight averaging takes over. This procedure concludes with global convergence, at which point no peer can profit from model merging and thus the averaging phase ends. The rightmost spikes in the blue region represent peers who already have high-quality models. To maximize utilization when peers have no other queued work, they try averaging with newly published remote models, even if their reported accuracy is lower than the local one. This results in further parameter space exploration and, potentially, model improvement, without sacrificing valuable processing time. The percentage of successful averaging attempts reached a maximum of 32% throughout the trials, while the mean value for all participants was 10%. These are the final values obtained at the end of the session after the network had converged. During the first phase, peers improved their accuracy at a rate of up to 70% via model merging (Figure 48).

To evaluate the proposed architecture's accuracy improvement, two sessions were conducted using the optimum number of epochs. In the first instance, each peer trained their models independently. The second was a LEARNAE session that had privacy features enabled. As shown in Figure 49 and Table 16, there is a substantial increase in the accuracy of the generated models of 1.12% despite the absence of training data exchange.

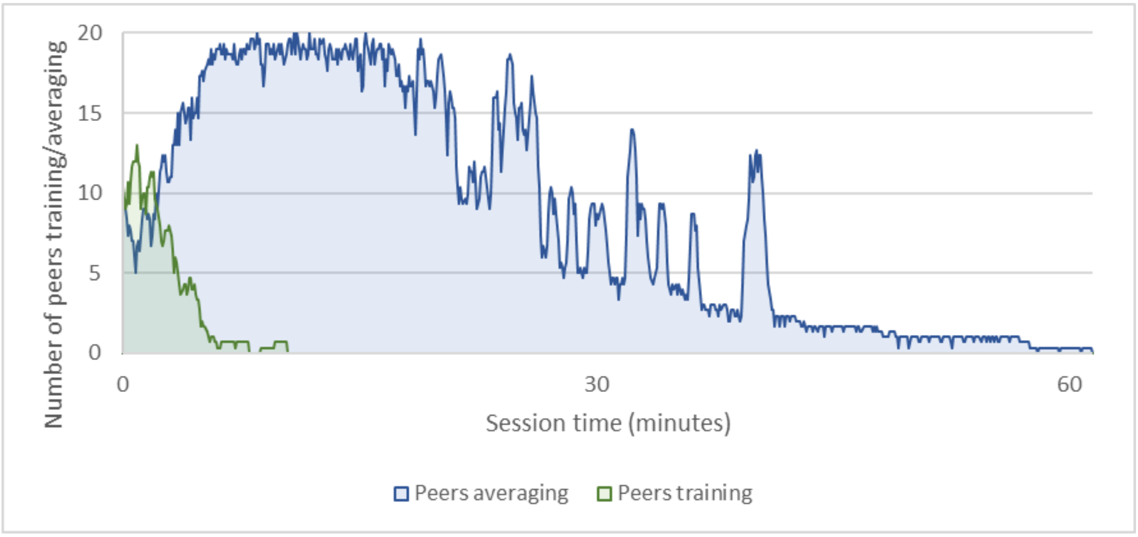


Figure 47: Number of peers training/averaging

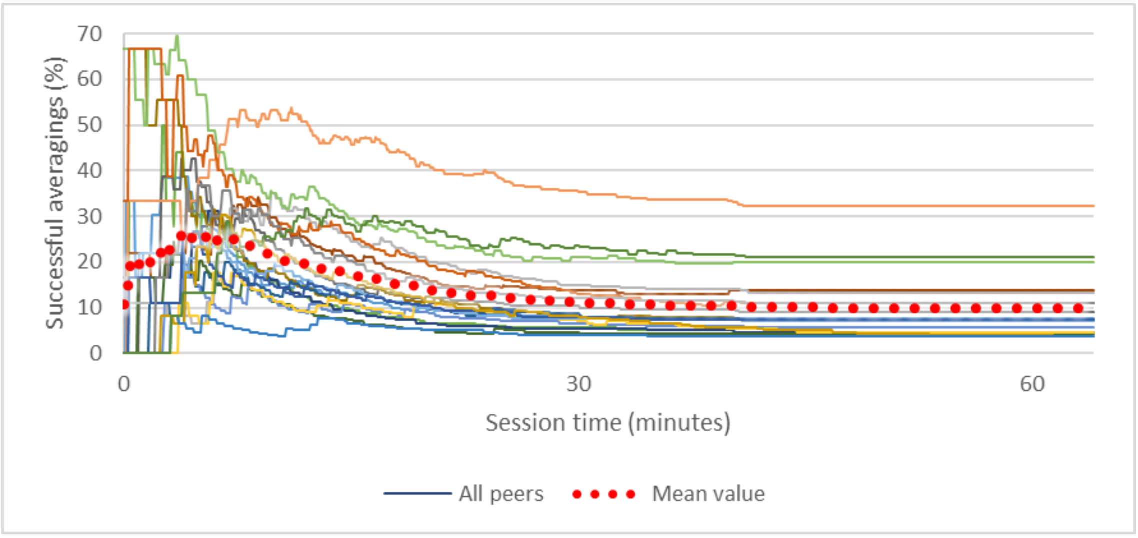


Figure 48: Successful averaging attempts

Table 16: Mean model accuracy values (stand-alone vs distributed)

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

4.6 Summary

This study proposes and evaluates a novel method for embedding IoT sensors without compromising the architecture's distributed nature. Additionally, it examines a critical feature known as resilience, as well as the effect of network disruptions and untrustworthy peers.

As shown by the experiments, LEARNAE collaboratively achieves models with an increased accuracy of up to 1.12% for the present configuration , without the need to share training data, but rather by leveraging neighbor information through selective parameter averaging.

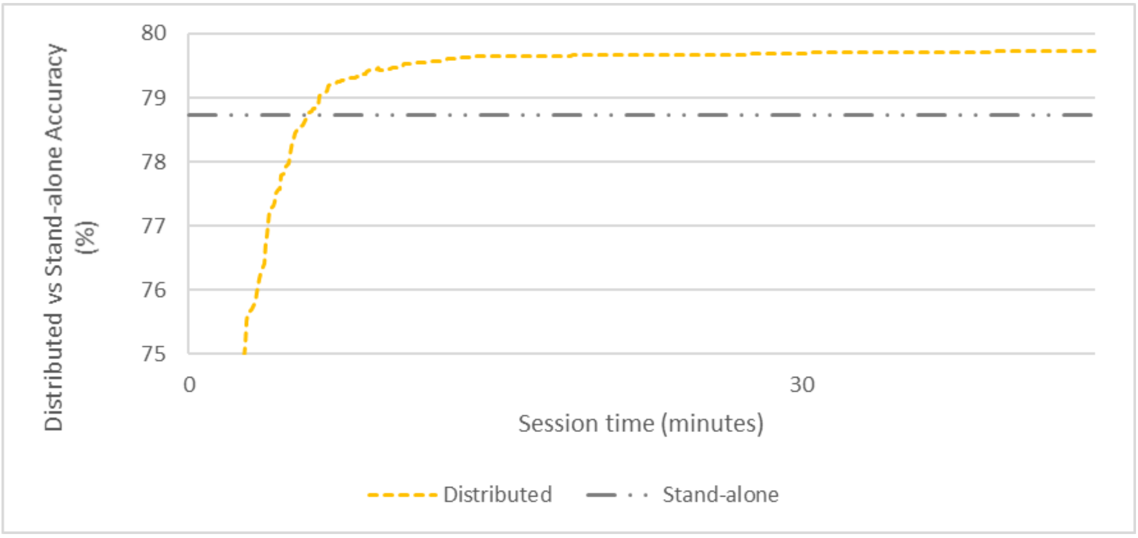


Figure 49: Mean model accuracy (stand-alone vs distributed)

5 Incentivizing Participation to Distributed Neural Network Training

We suggest a method for incentivizing peers to participate in LEARNAE's collaborative training sessions. So far, a LEARNAE swarm was composed of nodes that shared an interest in the outcome and partnered to improve model accuracy. A novel incentivization system is now included; peers may join a session and profit from their constructive averaging.

5.1 Proposed architecture

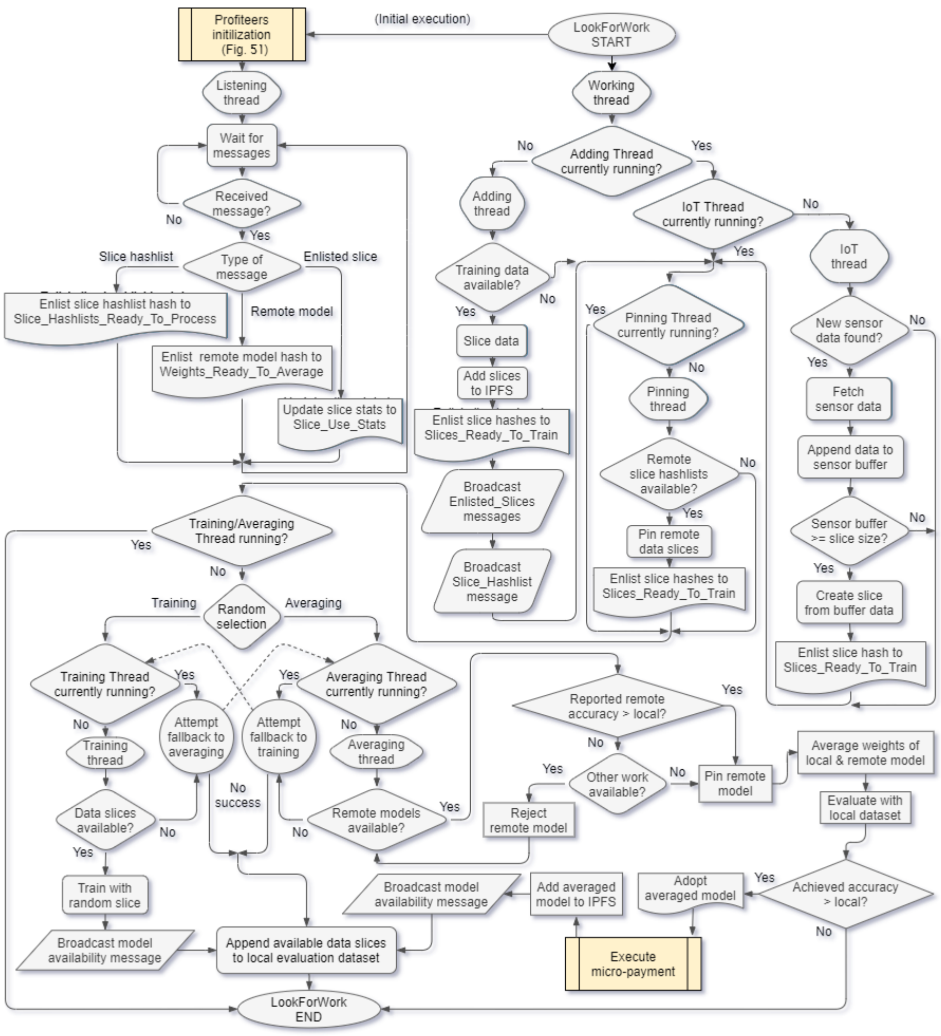

Profiteers initilization (Fig. 51)
(Initial execution)
LookForWork START
Listening thread
Working thread
Wait for messages
Adding Thread currently running?
Received message?
Adding thread
IoT Thread currently running?
IoT thread
Type of message
Slice hashlist
Enlisted slice
Remote model
Enlist slice hashlist hash to Slice_Hashlists_Ready_To_Process
Enlist remote model hash to Weights_Ready_To_Average
Update slice stats to Slice_Use_Stats
Training data available?
Slice data
Add slices to IPFS
Enlist slice hashes to Slices_Ready_To_Train
Broadcast Enlisted_Slices messages
Broadcast Slice_Hashlist message
Pinning Thread currently running?
Pinning thread
Remote slice hashlists available?
Pin remote data slices
Enlist slice hashes to Slices_Ready_To_Train
New sensor data found?
Fetch sensor data
Append data to sensor buffer
Sensor buffer >= slice size?
Create slice from buffer data
Enlist slice hash to Slices_Ready_To_Train
Training/Averaging Thread running?
Random selection
Training
Averaging
Training Thread currently running?
Averaging Thread currently running?
Attempt fallback to averaging
Attempt fallback to training
Training thread
Averaging thread
Data slices available?
No success
Remote models available?
Train with random slice
Broadcast model availability message
Append available data slices to local evaluation dataset
Broadcast model availability message
Add averaged model to IPFS
Adopt averaged model
Execute micro-payment
Reported remote accuracy > local?
Other work available?
Reject remote model
Pin remote model
Average weights of local & remote model
Evaluate with local dataset
Achieved accuracy > local?
LookForWork END
Yes
No

Figure 50: The major parts of a node's workflow (incentivization in yellow color)

The aforementioned incentivization mechanism is accomplished via the incorporation of a gateway, capable of communicating with blockchains in order to both publish and collect data. The architecture is platform agnostic and is compatible with any blockchain capable of executing code. With a credit system in place, peers may reward helpful neighbors by sending them digital micropayments for their contribution to successful averagings.

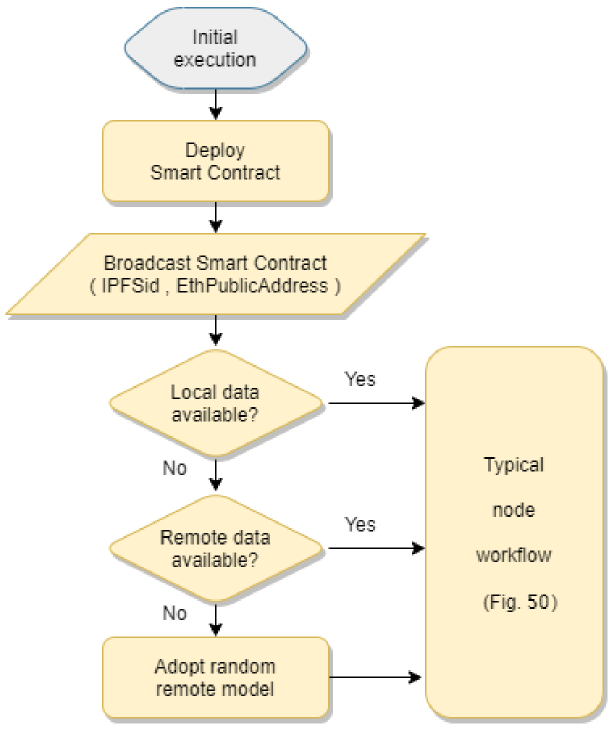

Initial execution
Deploy Smart Contract
Broadcast Smart Contract (IPFSid , EthPublicAddress)
Local data available?
Yes
No
Remote data available?
Yes
No
Adopt random remote model
Typical node workflow (Fig. 50)

Figure 51: Additional workflow section

Our prior work includes a comprehensive description of the workflow of a node. Figure 50 illustrates how the new components are integrated into the current workflow, while Figure 51 shows the modifications required to create the incentivization subsystem. At the time of the first execution, a peer wishing to receive payments must deploy LEARNAE's Smart Contract to the Ethereum Network. Following that, it must notify its neighbors by broadcasting a new type of metadata message containing its IPFS ID and the Ethereum public address of its Smart Contract.

When LEARNAE is in private mode, no training data is sent, which means that peers who joined only for the reward will almost certainly have no data of their own. In these cases, peers will randomly adopt a remote model that was made available to the network in order to properly initialize their local model. After that, they will fall back to the typical node workflow.

5.2 Incentivization algorithm

Each time a peer improves its local model by averaging with a remote one, the peer's incentivization algorithm sends a micropayment in Eth to the creator of the remote model. The payment amount is determined by a Reward Function that takes into account the amount of improvement in local model accuracy.

5.2.1 Reward Function

For the conducted experiments, in order to evaluate the whole procedure, we used a simple proportional formula:

$$Payment = RewardFactor \cdot (AchievedAccuracy - CurrentAccuracy)$$

In general, any Reward Function, whether linear or not, may be utilized. Eventually, the law of *Supply and Demand* will guide participants to the optimal amount of reward. This first implementation assumes that peers who engage in and benefit from the generated model are benevolent and will reward their helpful neighbors in order to maintain a relationship of trust.

We acknowledge that this method may be improved by anticipating bad actors who would benefit from others' contributions without rewarding them. So, study in this area, within the scope of Game Theory, will be a part of our future work.

5.3 Distributed proof of identification

Peers must be able to prove their identity in order for the incentivization to even exist. This would be trivial if a certificate authority could be used, but in the case of LEARNAE every aspect must be completely decentralized. As a result, we use Ethereum's Smart Contract architecture to provide a mechanism for Distributed Proof of Identity (DPoID).

5.3.1 Smart Contract Deployment

When a peer joins a LEARNAE session, it must deploy a specific Smart Contract on the Ethereum blockchain. Two data fields are included in this contract: "PoID" (Proof of Identification) and "Timestamp." When a contract is created, its constructor method is automatically executed. The creator-peer provides it with a single parameter (PoID), which is composed of its IPFSid and the public Ethereum address associated with its digital wallet. Internally, the constructor assigns the current date and time to the Timestamp field (Figure 52).

5.3.2 PoID Propagation and Rewarding

Following the deployment of the Smart Contract, the peer broadcasts the address of the contract to its neighbors. To obtain the required data, all other nodes execute this Smart Contract's getPoid() and getTimestamp() methods on the blockchain. Each node keeps a local directory containing the IPFSid, the public Ethereum wallet address, and the timestamp for each Smart Contract broadcasted to the network. For example: Peer A improves its local model by averaging it with a remote model sent by peer B. Peer A scans its local directory for the Ethereum public wallet address associated with peer B's IPFSid and sends a micropayment to that address.

5.3.3 Shielding against fraudulence

A malicious actor might try to hijack payments by broadcasting Smart Contracts with its own Ethereum public wallet address but using the IPFSid of a more active peer. This attempt would be recognized and ignored by its neighbors. This is due to the fact that the same IPFSid would be linked with two different Ethereum public addresses. In such situations, the Smart Contract with the later timestamp would be immediately rejected as fraudulent.

5.4 Conducted experiments

Our previous work demonstrated that collaborative training may provide advantages – in terms of model accuracy – even with a small number of participants. The point of this study is to discuss some early metrics for the incentivization system.

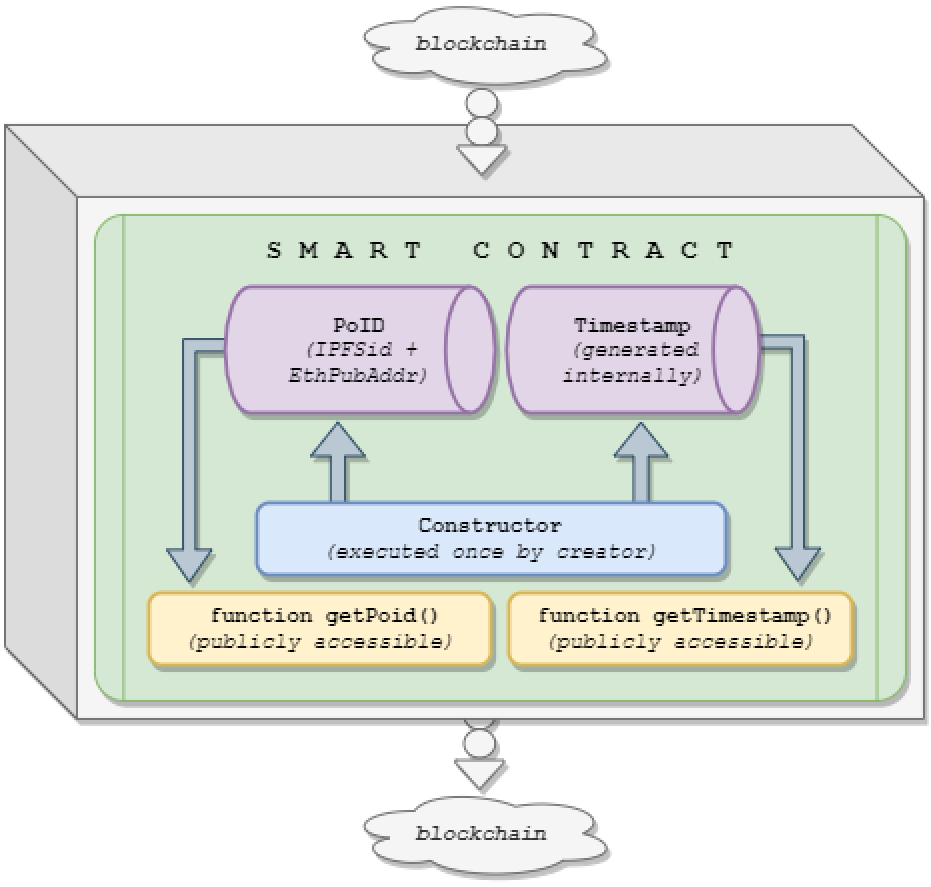


Figure 52: Structure of DPoID Smart Contract

The following graphs depict a LEARNAE session of 20 peers. Each of them possessed a local dataset of 10000 examples, which was split into 10 groups of 1000 instances to facilitate averaging. Data privacy was enabled, which means that nodes did not exchange training data.

As shown in Figure 53, the first phase is distributed evenly between training and averaging. Once all available data has been consumed, peers devote their whole attention to averaging efforts. Finally, as the swarm approaches convergence, averaging is unable to improve the models any further, and the session concludes.

As shown in Figure 54, at the beginning the percentage of successful averagings is high, and peers rapidly benefit from their neighbors' models. Even at the session's end, the mean percentage of successful averagings is above 20%.

Figure 55 depicts the evolution of the mean model's accuracy.

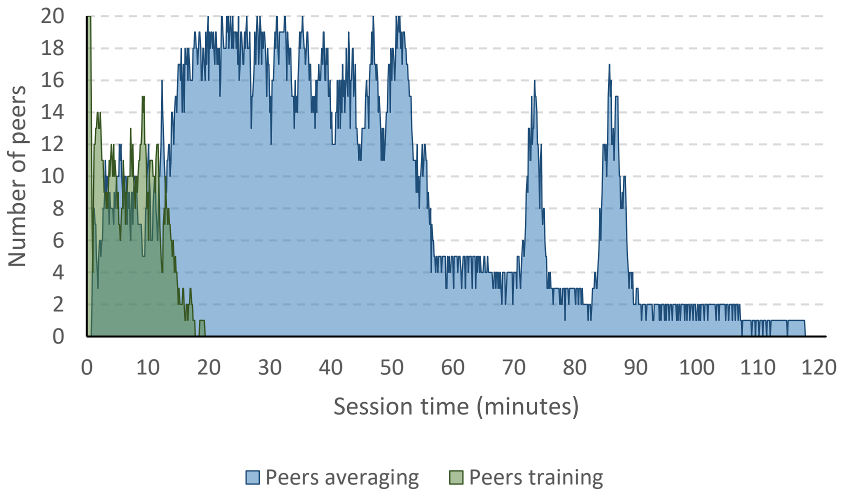


Figure 53: Work type distribution

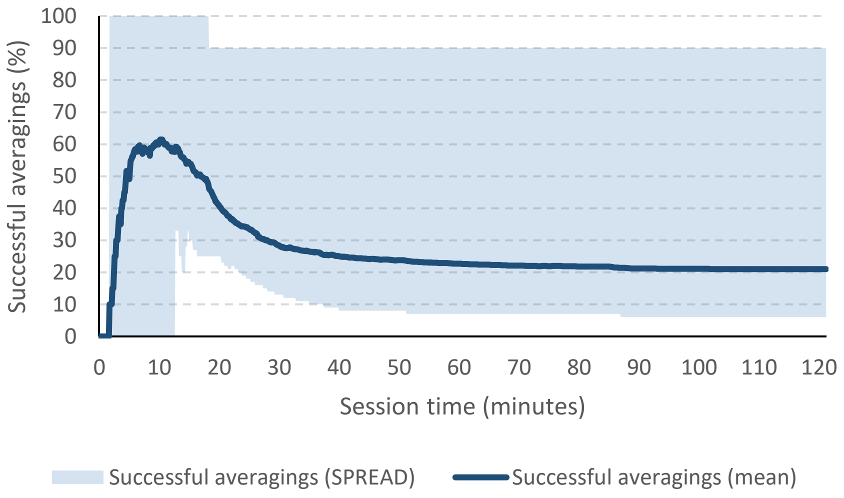


Figure 54: Cumulative success rate of averaging process

Figure 56 shows the number of Eth payments throughout the session. For the conducted experiments, peers sent micro-payments 3-16 times (with a mean value of 9.4).

As shown in Figure 57, during the collaborative training, peers sent 0.0005-0.0068 Eth to their neighbors (with a mean value of 0.003).

It is important to outline that the conducted experiments serve as a proof of concept; the actual values of a real-life session would be auto-regulated by supply-and-demand of processing power.

5.5 Summary

We extended our previous research on Distributed Neural Network Training by including a subsystem that incentivizes peers to participate in a LEARNAE session. In this scenario, peers who have no interest in the generated model may join the swarm to profit from their constructive averaging. We developed a technique for implementing a completely distributed proof of identification using Ethereum's Smart Contracts and conducted proof-of-concept experiments to evaluate the basic metrics.

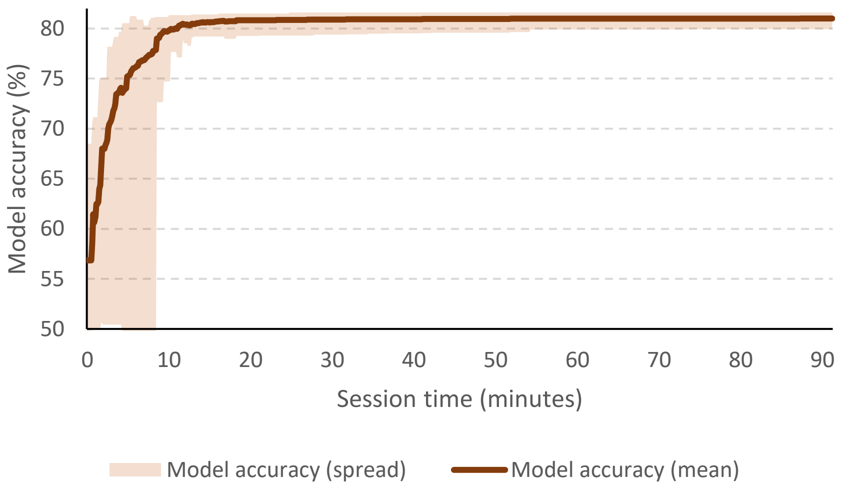


Figure 55: Progress of model accuracy

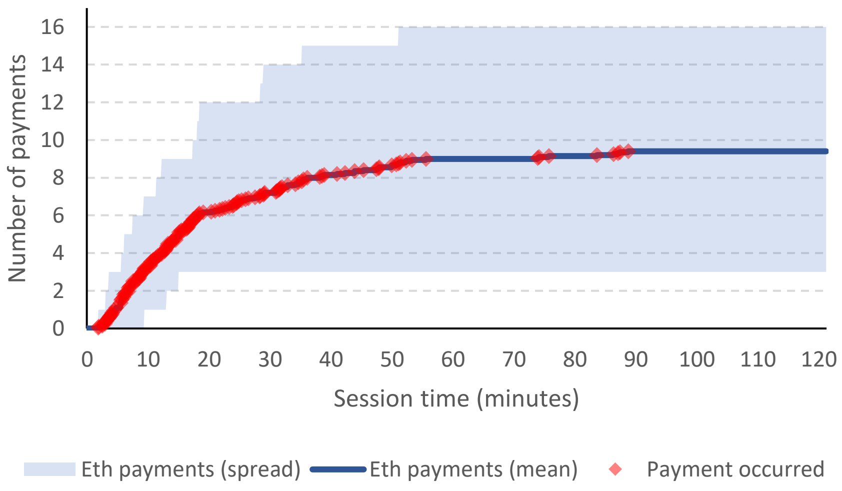


Figure 56: Cumulative number of Eth payments

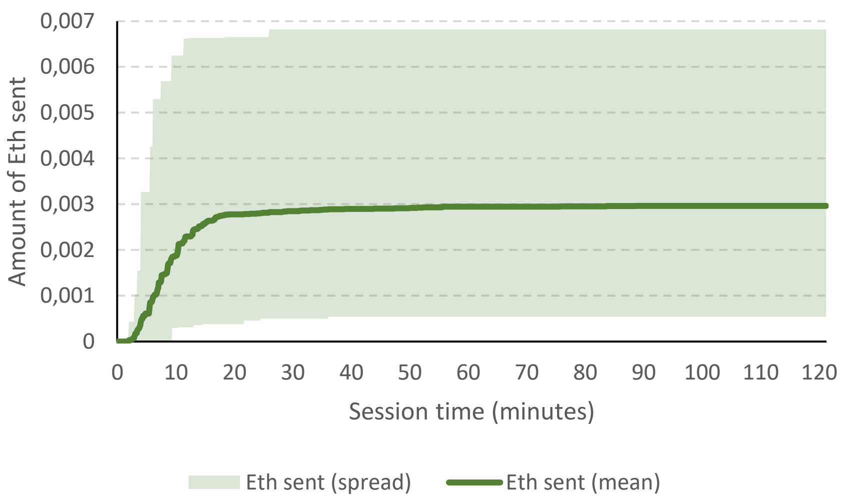


Figure 57: Cumulative amount of Eth sent

6 Consolidating Incentivization in Distributed Neural Network Training via Decentralized Autonomous Organization

Our previous incentivization method is unable to deal with bad actors who refuse to pay the proper rewards. To deal with this issue we introduce a novel incentivization scheme, based on the concept of *Decentralized Autonomous Organizations*.

6.1 Proposed architecture

In order for any incentivization to work, peers must be able to verify their identities. A certificate authority would make this straightforward, but in the case of LEARNAE everything must be fully decentralized. We use Ethereum's Smart Contracts to build a Distributed Proof of Identity (DPoID).

To participate in a LEARNAE session, each participant must first deploy a Smart Contract to the Ethereum network. This contract has two data fields: PoID (Proof of Identification) and Timestamp. The creator-peer passes one parameter (PoID) to the network, which is comprised by (a) its IPFSid, (b) its public Ethereum address of its digital wallet, and (c) its PublicKey. Internally, the constructor assigns the current date and time to the Timestamp field.

After deploying the Smart Contract, the peer broadcasts the contract's address to its neighbors, and all other nodes in the network execute the $getPoid()$ and $getTimestamp()$ methods to get the required data, which are kept locally in every node's storage.

A malicious actor could attempt to highjack payments by broadcasting a Smart Contract that contains its own Ethereum public wallet address but with the IPFSid from another –more active– peer. This effort would be detected and dismissed by its neighbors, due to the fact that the same IPFSid would be associated with two distinct Ethereum public addresses. In such cases, The Smart Contract with the later timestamp is immediately rejected as fraudulent. Detailed analysis of this process can be found in our previous work.

6.2 The concept of Decentralized Governance

6.2.1 Background Theory

There is a novel technological development that has entered the domain of public organizations, thanks to algorithmic systems [134]. The thing which defines these systems is Machine Learning, which provides new ways to discover knowledge [135]. Artificial Intelligence applications use huge datasets and statistical methods to infer connections that are sometimes hidden or not obvious. These algorithms have the potential to offer significant new insights, therefore they are regarded as a very strong tool for governance purposes [136].

Decision-making based on algorithmic analysis is grounded on a completely different logic than traditional bureaucracy, since the latter usually leads to a decision taking into consideration single-case information. Even though uncertainty is inherent in many decision-making processes, algorithmic systems allow governance structures to use data analysis to quantify the uncertainty and state the information as probabilities, to better rationalize the process. This method could offer, for example, a powerful optimization tool to classify cases based on whether they should be the subject of further investigation. Thus, algorithmic systems are a major shift in how an organization is structured and they have the potential to fundamentally affect how governance is achieved. The term *Algocracy* introduces the concept of exerting governance using algorithms. It is the evolution step after *Machine Bureaucracy* and *Infocracy*.

Machine Bureaucracy is an administrative body that is defined by a distinct set of guidelines for getting things done [137]. Work procedures in this context are quite organized and rule-based. A single, standardized way of doing things dominates the Machine Bureaucracy, since every task must be done in a very rigid manner. Hierarchization and formalization are both high, as is centralization, with the buildup of decision-making capacity at the top of the organization. The underlying technological infrastructure is the pivotal element of the organization. Because it is in charge of standardization, it can thus be said that it is accountable for that implementation.

Infocracy is a form of organization, much like Machine Bureaucracy. The standardization of work is programmed into the technology utilized by the organization, eliminating the need for participants to learn rules that would improve uniformity [138]. Work standardization is extremely prevalent, but all processes are set in motion by information systems following preset laws and regulations [139][140]. This solidifies the technological structure's important role inside the organization. Work is supported by information systems, which have created a fine-grained division of roles with plenty of vertical control. Decisions at a lower organizational level may be taken at some extend, sometimes with moderate decentralization. In addition, hierarchical and formalized organizational structures are decreased, as compared to Machine Bureaucracy.

In Algocratic systems, non-routine labor may be done by using sophisticated technology. Algorithms for data mining, pattern recognition, and prediction are all built using Machine Learning [141][142]. Inherent uncertainty is measured and minimized by the algorithmic system's data analysis [143].

Algocracy proposes that sophisticated technologies can push technology into decision-making domains, by converting human judgement into standardized processes [144]. The algorithm is to be used in both a unidirectional and bidirectional fashion, contrary to Machine Bureaucracy and Infocracy, where work control is unidirectionally administered via the organization and information infrastructure, respectively.

It is noteworthy that many of those algorithms depend on developer inputs, like their judgments, perceptions and opinions, etc, that may have an impact on the learning process. This is significant because the programmers may actually predetermine and guide the Machine Learning algorithm, and in that way influence the decision-making results given by the system.

6.2.2 Decentralized Autonomous Organizations

A Decentralized Autonomous Organization (DAO) [145][146] is defined as a structure controlled by laws written in code. Its members exert control through the transparency of these computerized governance rules, and do not rely on central control. Transactions and regulations for a DAO are stored on a blockchain [147][148][149]. The consensus mechanism of the blockchain ensures the proper function of DAO, a critical feature since so far there is no concrete legal framework that covers this novel kind of organization [150]. The benefits gained by the use of blockchain technology include fault-tolerant distributed database, cryptography-based identification and permissionless timestamping. When someone uses this method, they no longer have to rely on a trustworthy third party in their transactions, making things easier and more straightforward.

The costs of a blockchain-based transaction, and the associated data-reporting, may be significantly lower than traditional methods, due to the elimination of the need for multiple and independent bureaucratic records and third-party fees that are typically charged in conventional procedures. Thus, blockchain data might, at least in principle, replace public papers like deeds, titles and contracts, so long as regulations allow it. Once a DAO launches, it could be structured to operate autonomously, with Smart Contracts managed by a Turing-complete platform that would maintain full-scale administrative support [151][152]. Decentralized self-government organizations aspire to be open platforms where people control their relationships, their identity and their personal data [153].

In the Ethereum blockchain the *Solidity* programming language is what is used to create DAO code. This code can be executed by creating on-chain Ether transactions. Ether, the digital asset of the Ethereum network, is the fuel for all applications that leverage its blockchain. In order to start functioning, a DAO needs Ether in its account, and thus, its priority is to obtain it. During the creation phase the code is released, and the system allows Ether to be transferred to the DAO's Smart Contract address.

In order to compensate senders of ether, a DAO generates tokens and assigns them to the senders of the ether. The tokens provide to the participants the ability to vote and be part-owners. The amount of created tokens depends on the amount of transferred Ether. There are no transfer fees for moving tokens around after the Genesis phase has concluded.

When deployed, the settings for the *Minimum DAO Creation Objective* and *Creation Phase Time-period* are given as arguments to the code. In case the total of DAO Creation Objective does not meet the minimum before the end of the creation phase, all of the ether will be refunded. After the Creation Phase has ended, the total Ether raised is denoted by Ξ_{raised} and the total amount of tokens created is denoted by T_{total}. Essentially, DAO is a structure that holds Ether and other Ethereum-based tokens, and transfers them according to the organization's code.

An individual who owns a DAO token may ask for DAO funds (denoted $\Xi_{transfer}$) by proposing a contract. If the proposal is accepted by the majority of the voting power, the DAO sends the requested Ether to a smart contract representing the proposed project. The process of selecting a contract can be enhanced with advanced features, such as collaboration with other DAOs and fetching data from external sources called "Oracles".

The number of votes someone has, is proportional to the number of tokens they own. Each proposal has an allotted amount of time for discussion and a vote. Once a proposal has been approved, token holders will be able to execute a DAO contract function which verifies that (i) the majority of votes were in favor of the proposal, (ii) the minimum quorum percentage was met, and (iii) the proposal has been approved. The DAO will fund the proposal if it has been approved, otherwise the proposal will be closed.

A token holder has a say if they have at least one DAO token. The minimum number of tokens a person must have to be able to influence a decision is denoted by q_{min}. An example of how some of the most popular DAO calculate q_{min} is as follows

$$q_{min} = \frac{T_{total}}{d} + \frac{\Xi_{transfer} \cdot T_{total}}{3 \cdot (\Xi_{DAO} + R_{DAO})}$$

where d is the $minQuorumDivisor$, Ξ_{DAO} is the amount of ether owned by a DAO and R_{DAO} is the amount of reward tokens owned by this DAO. The sum $(\Xi_{DAO} + R_{DAO})$ is equal to the amount of ether used to create DAO tokens plus the rewards received, or said another way, the total amount of ether a DAO has ever received.

The above formula means that, initially and for $d = 5$, a quorum of 20% of all tokens is required for any proposal to pass. In the event $\Xi_{transfer}$ equals the amount of ether a DAO has ever received, then a quorum of 53.33% is required.

To prevent proposal spam, a deposit is needed in order to have a proposal reviewed; the deposit will be returned if the majority approves the plan. The DAO will retain the deposit if a quorum is not met. The required deposit amount may be modified by the DAO in a later proposal.

The DAO as a decentralized entity cannot be manipulated by any outside influences. Because it is open-source, the organization and all of its code are visible and therefore impossible to corrupt, since all program functions are managed on the blockchain.

Stakeholders must have complete consensus on every choice they need to make, such if one member wishes to pull out their money. Bugs and other problems requiring democracy in the decision-making process are also a concern, and all can be addressed by the DAO’s rules.

From a technological standpoint, a DAO is made up of one or more Smart Contracts which are executed on the Ethereum blockchain using its distributed consensus mechanism. Ethereum offers a blockchain with a built-in Turing-complete programming language, where users may design and implement applications on their own terms. Smart Contract transaction costs are paid using Ethereum's currency Ether. Figure 58 shows the theoretical layers of a Decentralized Autonomous Organization.

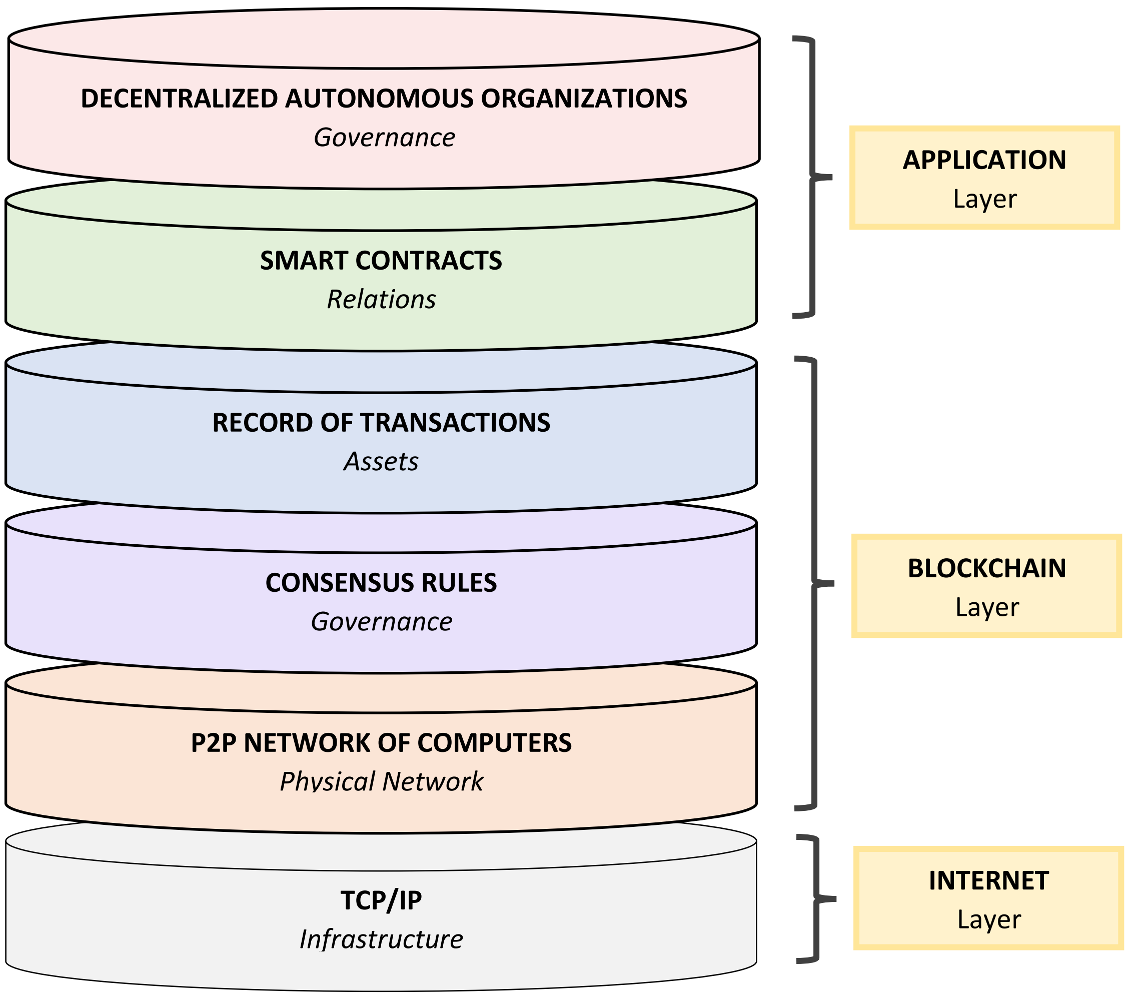


Figure 58: Architectural layers of DAO

The data structure of a single proposal depends on the characteristics of each use case. For a typical DAO, a proposal could have the parameters shown in Table 17.

6.3 LEARNAE's Decentralized Autonomous Organization

The first stage of a LEARNAE training session is the establishment of its Decentralized Autonomous Organization. During this period all participants have to deposit a predefined amount of digital assets. These assets are immediately locked and constitute the DAO's Contribution Fund. Each deposit contains the participant's Ethereum public address. This information, in conjunction with the data contained in the DPoID Smart Contracts, are used to link a deposit to an IPFS ID. Since voting is not weighted, the DAO sends in return a single LEARNAE token to each member. Owning this token grants the right to vote and submit proposals.

Table 17: Typical parameters of a DAO proposal

Parameter	Description
$recipient$	The address where the amount of assets will go to if the proposal is accepted.
$amount$	The amount of assets to transfer to recipient if the proposal is accepted.
$description$	A plain text description of the proposal.
$votingDeadline$	A Unix timestamp, denoting the end of the voting period.
$open$	A Boolean which is false if the votes have already been counted, true otherwise.
$proposalPassed$	A Boolean which is true if a quorum has been achieved with the majority approving the proposal.
$proposalHash$	A hash to check validity of a proposal. Equal to $sha3(recipient, amount, transactionData)$.
$proposalDeposit$	The deposit the creator of a proposal has send to submit a proposal. It is taken from the $msg.value$ of a $newProposal$ call;
yea	Number of tokens in favor of the proposal.
nay	Number of tokens opposed to the proposal.
$votedYes$	Simple mapping to check if a token holder has voted for it.
$votedNo$	Simple mapping to check if a token holder has voted against it.
$creator$	The address of the token holder that created the proposal.

When the period for fund raising expires, the network starts the collaborative training. Every time a peer improves its local model by averaging with a remote one, the peer updates, signs with its private key, and broadcasts its own *Shared Contribution Vector* (SCV). SCV is a record that contains information on how much help a peer received by its neighbors, in its effort to improve its local neural model (Figure 59).

	Node_00	Node_01	Node_02	⋯	Node_97	Node_98	Node_99
Node_i	*[value]*		*[value]*	⋯		*[value]*	

Figure 59: Shared Contribution Vector of Node #i (assuming 100 participants)

Peers gather broadcasted SCV messages and use them to construct their local copy of *Shared Contribution Ledger* (SCL). SCL is an array comprised by all known SCV records and is used to determine a general view of the contribution level throughout the network (Figure 60). Thus, the value in row X and column Y of the SCL represents the help offered to peer X by peer Y.

When training phase concludes, participants submit their proposals regarding the amount of digital assets they claim as reward for their contribution. Peers review the submitted proposals by comparing them to their copy of SCL. They vote in favor of every consistent proposal and against of all others. When voting period ends, the DAO automatically releases the whole amount of available funds, distributed proportionally to all approved proposals (Figure 61, Figure 62). The procedure is taking place on the blockchain and requires no central coordination.

	Node_00	Node_01	Node_02	⋯	Node_97	Node_98	Node_99
Node_00	▨	*[value]*		⋯	*[value]*	*[value]*	
Node_01	*[value]*	▨	*[value]*	⋯			
Node_02			▨	⋯		*[value]*	
⋮	⋮	⋮	⋮	⋱	⋮	⋮	⋮
Node_97		*[value]*	*[value]*	⋯	▨		*[value]*
Node_98	*[value]*			⋯		▨	*[value]*
Node_99		*[value]*		⋯	*[value]*		▨

Figure 60: Shared Contribution Ledger (assuming 100 participants)

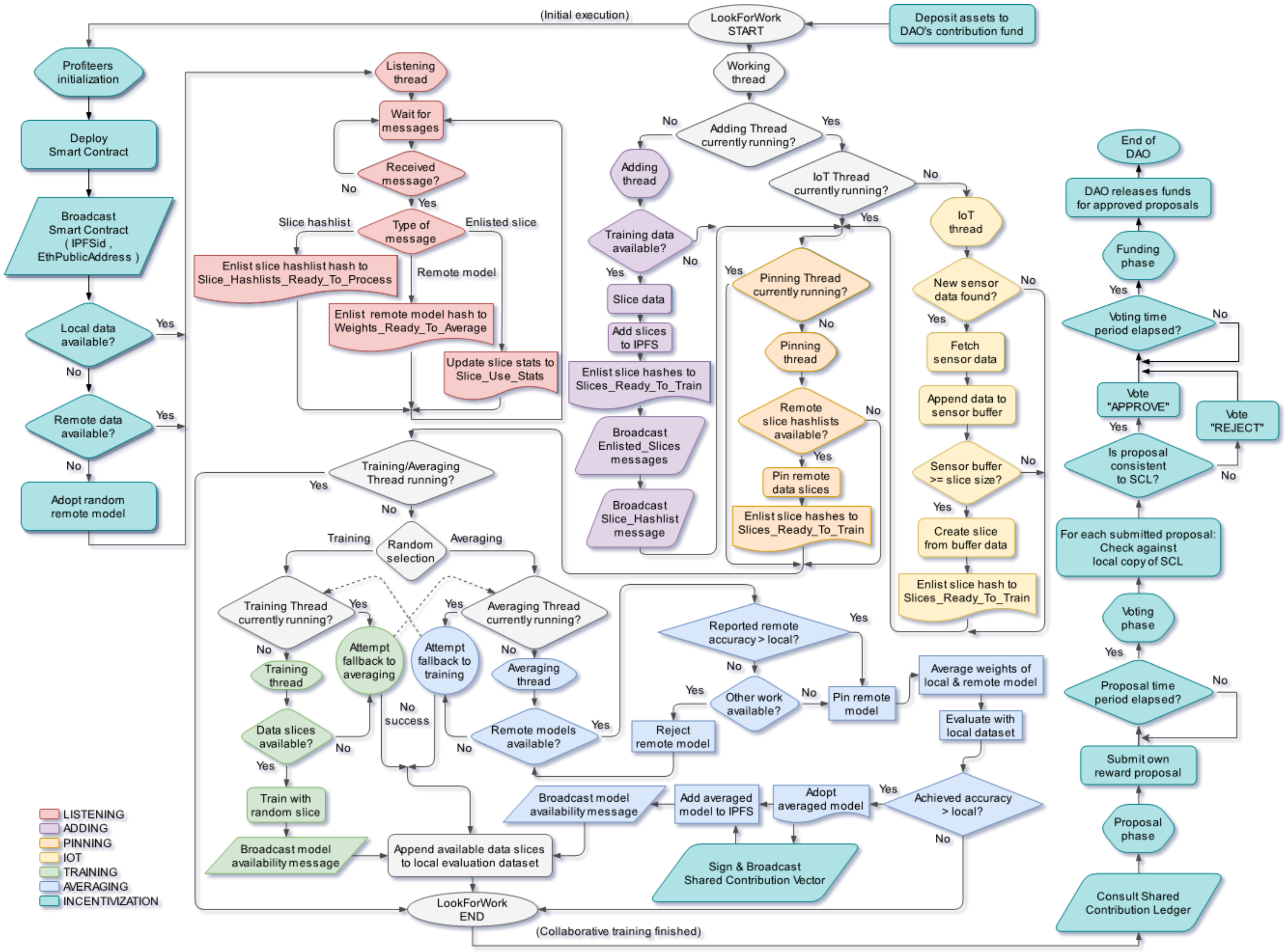

(Initial execution)
LookForWork START
Deposit assets to DAO's contribution fund
Working thread
Adding Thread currently running?
No
Yes
Adding thread
IoT Thread currently running?
No
Yes
IoT thread
Training data available?
No
Yes
Slice data
Add slices to IPFS
Enlist slice hashes to Slices_Ready_To_Train
Broadcast Enlisted_Slices messages
Broadcast Slice_Hashlist message
Pinning Thread currently running?
Yes
No
Pinning thread
Remote slice hashlists available?
No
Yes
Pin remote data slices
Enlist slice hashes to Slices_Ready_To_Train
New sensor data found?
No
Yes
Fetch sensor data
Append data to sensor buffer
Sensor buffer >= slice size?
No
Yes
Create slice from buffer data
Enlist slice hash to Slices_Ready_To_Train
Profiteers initialization
Deploy Smart Contract
Broadcast Smart Contract (IPFSid , EthPublicAddress)
Local data available?
Yes
No
Remote data available?
Yes
No
Adopt random remote model
Listening thread
Wait for messages
Received message?
No
Yes
Type of message
Slice hashlist
Enlisted slice
Remote model
Enlist slice hashlist hash to Slice_Hashlists_Ready_To_Process
Enlist remote model hash to Weights_Ready_To_Average
Update slice stats to Slice_Use_Stats
Training/Averaging Thread running?
Yes
No
Random selection
Training
Averaging
Training Thread currently running?
Yes
No
Averaging Thread currently running?
Yes
No
Attempt fallback to averaging
Attempt fallback to training
No success
Training thread
Averaging thread
Data slices available?
No
Yes
Remote models available?
Yes
No
Train with random slice
Broadcast model availability message
Append available data slices to local evaluation dataset
LookForWork END
(Collaborative training finished)
Reported remote accuracy > local?
Yes
No
Other work available?
Yes
No
Reject remote model
Pin remote model
Average weights of local & remote model
Evaluate with local dataset
Achieved accuracy > local?
Yes
No
Adopt averaged model
Add averaged model to IPFS
Broadcast model availability message
Sign & Broadcast Shared Contribution Vector
End of DAO
DAO releases funds for approved proposals
Funding phase
Yes
Voting time period elapsed?
No
Vote "APPROVE"
Vote "REJECT"
Yes
Is proposal consistent to SCL?
No
For each submitted proposal: Check against local copy of SCL
Voting phase
Yes
Proposal time period elapsed?
No
Submit own reward proposal
Proposal phase
Consult Shared Contribution Ledger
🟥 LISTENING
🟪 ADDING
🟧 PINNING
🟨 IOT
🟩 TRAINING
🟦 AVERAGING
🟦 INCENTIVIZATION

Figure 61: The updated workflow of a LEARNAE full node

The new DAO-based incentivization mechanism is a major paradigm shift compared to the tipping method of our previous implementation. Although tipping was a more direct way for a peer to reward the neighbors that actually offered help to that peer, it could not ensure the compliance of the participants. Thus, there could be cases where malicious actors might tamper the procedure to avoid paying. The new implementation is more community-oriented: Peers reward neighbors according to their overall assistance to the swarm.

All participants have to declare their engagement to the process, by depositing digital assets to DAO’s Contribution Fund. Depending on their assistance during the training phase, peers can end up with less or more assets in their balance, compared to their initial deposit amount. Which one it will be, it depends on their “give help” / “get help” equilibrium

6.4 Experimental results

To provide a proof-of-concept for the proposed incentivization architecture, we construct and evaluate the results of an algorithm simulating the collaborative neural network training. In these experiments the LEARNAE swarm consists of 100 participants. The session is comprised of 50 cycles; during each cycle peers attempt to average their model with a remote one.

The result of this attempt may be successful or not. The level of change to the local model’s accuracy is expressed as a random number (range: *Uniform*[-10..10]), where negative values indicate unsuccessful attempts. To simulate the discrepancy in node’s hardware performance, every peer is uniformly assigned a random *Contribution Factor* (range: *Uniform*[1..9]). The final level of change occurred by an averaging attempt is calculated as:

```
$ModelChange = Uniform[-10, 10]$
$if\ (ModelChange > 0)\ then$
    $ModelChange = NeighborContributionFactor \times ModelChange$
```

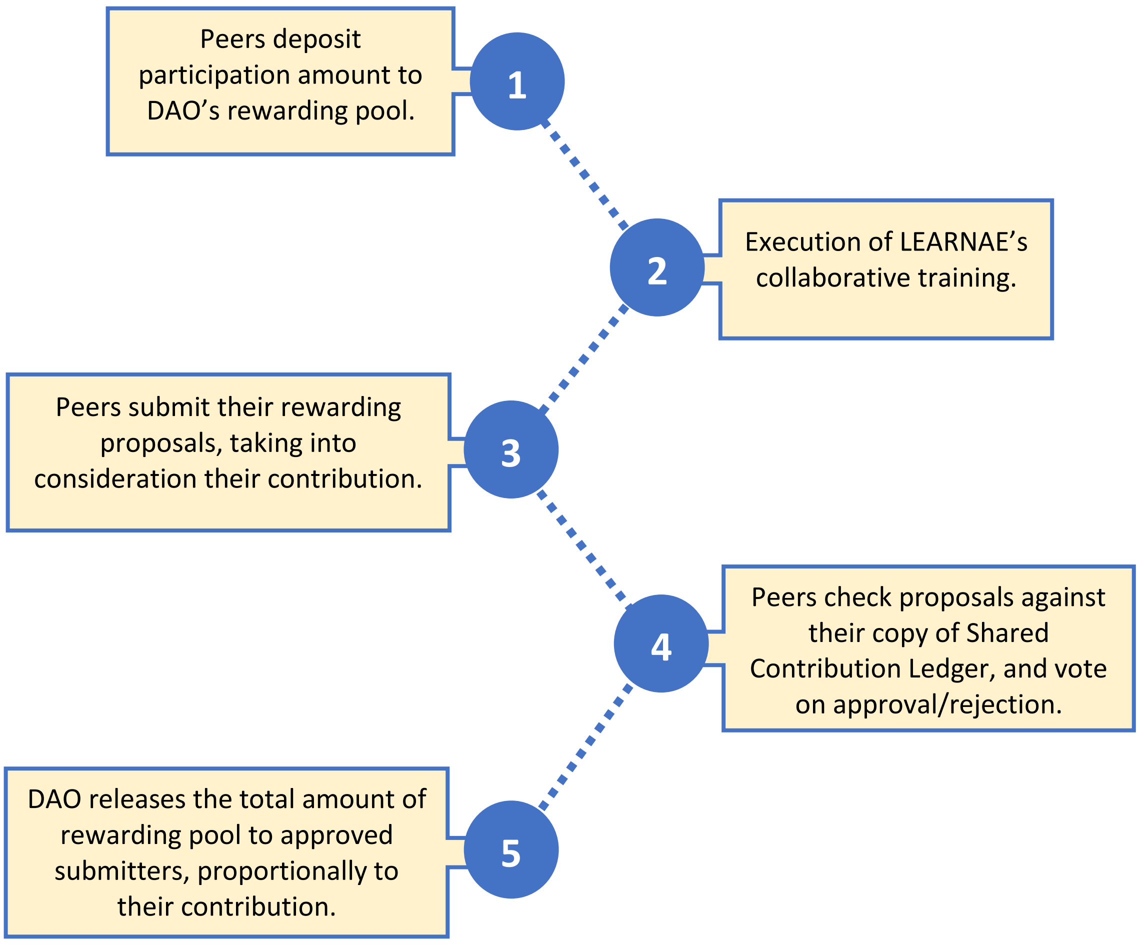

Peers deposit participation amount to DAO's rewarding pool.
1
2
Execution of LEARNAE's collaborative training.
Peers submit their rewarding proposals, taking into consideration their contribution.
3
4
Peers check proposals against their copy of Shared Contribution Ledger, and vote on approval/rejection.
DAO releases the total amount of rewarding pool to approved submitters, proportionally to their contribution.
5

Figure 62: Stages of LEARNAE's DAO

The value of $ModelChange$ is also used as the amount of the reward given for this successful averaging. During the initial phase, all participants have to deposit 1000 credit units to DAO's Contribution Fund.

To assess the efficiency of our incentivization proposal on mitigating the impact of malicious nodes, we conducted experiments for both cases, (a) no malicious actors, and (b) 10% of participants being malicious. In the following experiments, the period during which DAO adapts the peer rewards in order to comply with the new incentivization scheme, is denoted as *Consolidation Phase*.

6.4.1 Network with no malicious nodes

Figure 63 demonstrates the balance of each peer during a session with no malicious actors. This means that when a peer achieves a successful averaging, willingly broadcasts the reporting message to the swarm, to let everyone know that a neighbor's help resulted to a specific level of model improvement. The message that is broadcasted is essentially the updated version of its own SCV, signed with the node's private key. The lines indicate how a peer's balance is progressing from this peer's point of view.

During Consolidation Phase, the DAO applies the corrections introduced with our new incentivization algorithm. Thus, every peer rewards its neighbors for their overall contribution to the LEARNAE swarm. As expected, this paradigm shift results to different final balances compared to simple tipping method. Figure 64 depicts these differences. Out of 100 nodes, 47 ended up with increased balance at an average value of $+241.84$ credit units; 53 nodes ended up with decreased balance at an average value of -214.46 credit units.

6.4.2 Network with 10% malicious nodes

Figure 65 demonstrates the progress of every peer's balance during a collaborative session with 10 malicious nodes. For these experiments we define a "malicious" actor as one who refuses to broadcast its updated SCV, in an attempt to avoid paying rewards to others. The malicious peers are denoted with thick dashed lines. As shown in the chart, if we just use our previous tipping method, the balance of the malicious nodes is only increasing, since, although they avoid paying others, they still get rewards for their contribution. The DAO's Consolidation Phase is eliminating this malicious effort; dishonest peers ultimately get rewarded not based on their tampered version of truth, but rather according to the evaluation they got from the entire network. The correction is visually expressed by the uniform distribution of malicious balances after the Consolidation Phase.

Figure 66 depicts the balance differences between the two rewarding schemes. Out of 100 nodes, 51 ended up with increased balance at an average value of $+272.59$ credit units; 49 nodes ended up with decreased balance at an average value of -283.71 credit units. Regarding malicious actors, the achieved correction leads to a rough decline in their final balance. The first 10 nodes being malicious, experience a substantial decrease in their final balance. The average decrease for the malicious nodes is -608.56 credit units, while the average value for all other reductions is -200.42 credit units.

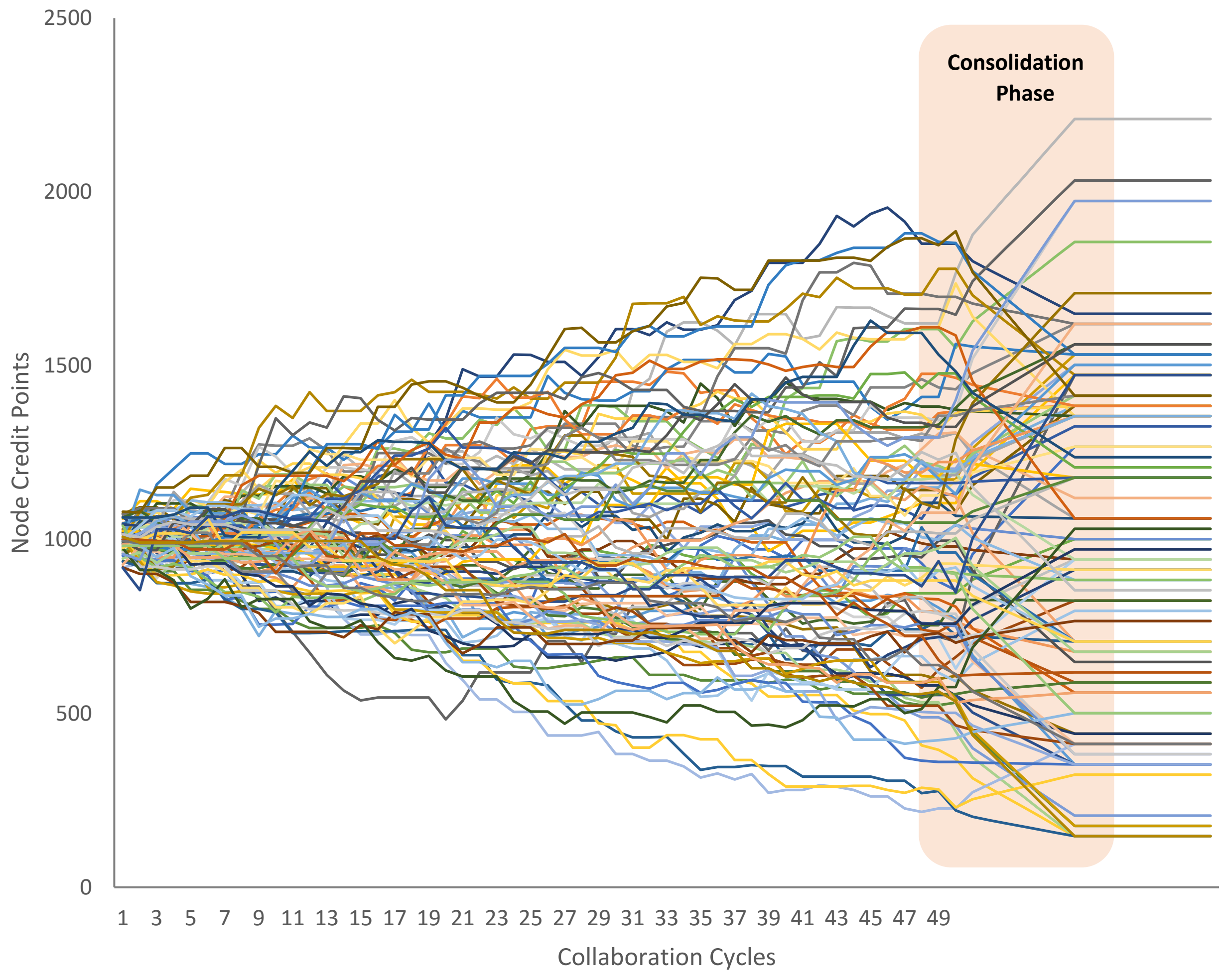


Figure 63: Progress of credit balances in case of no malicious peers

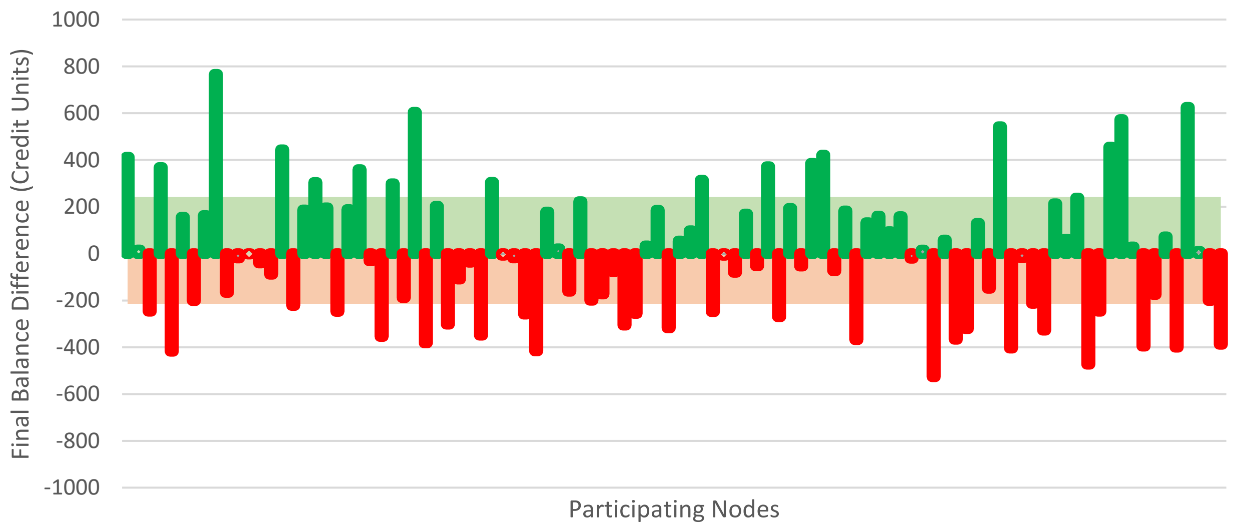


Figure 64: Correction in credit balances in case of no malicious peers

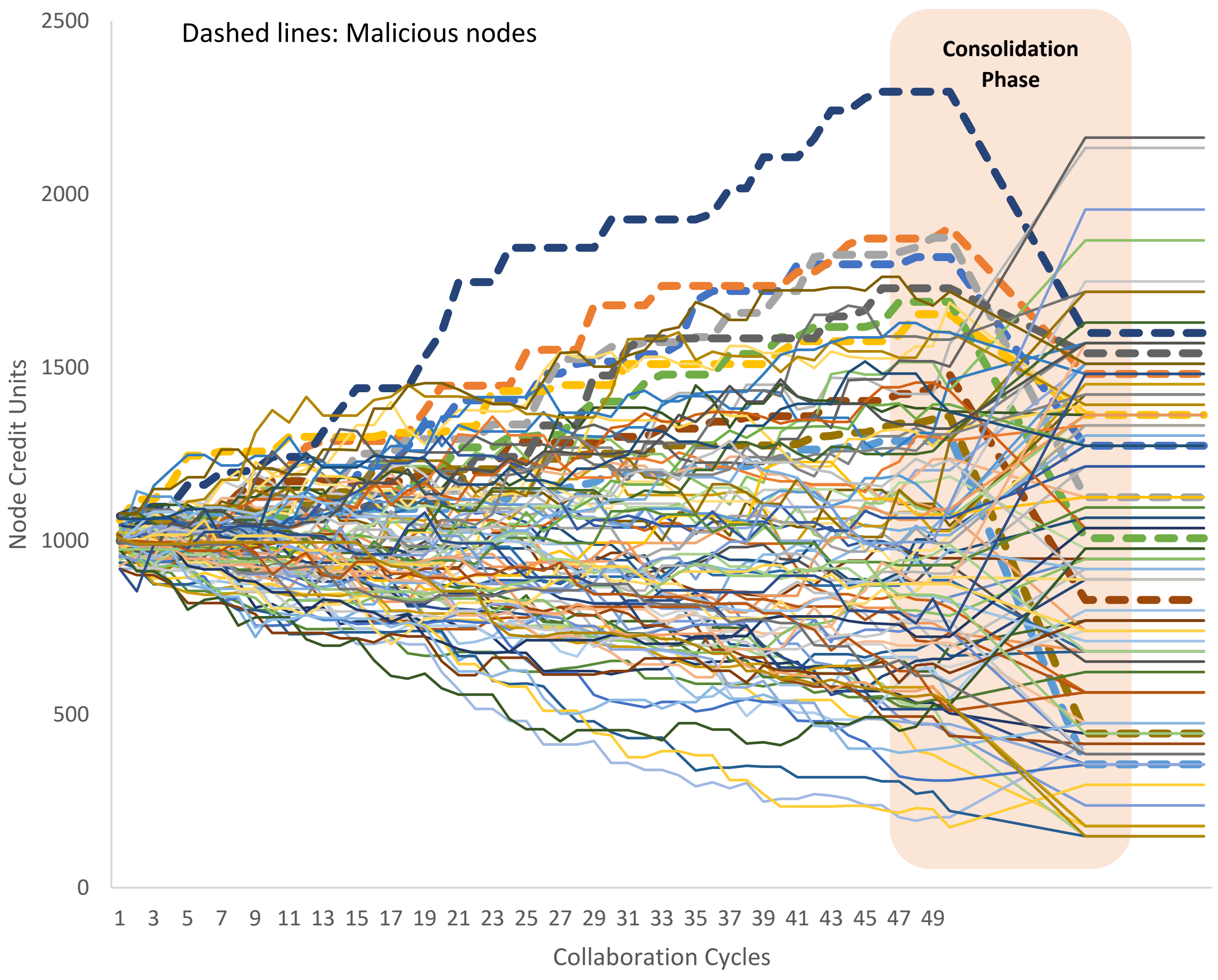


Figure 65: Progress of credit balances in case of 10% malicious peers

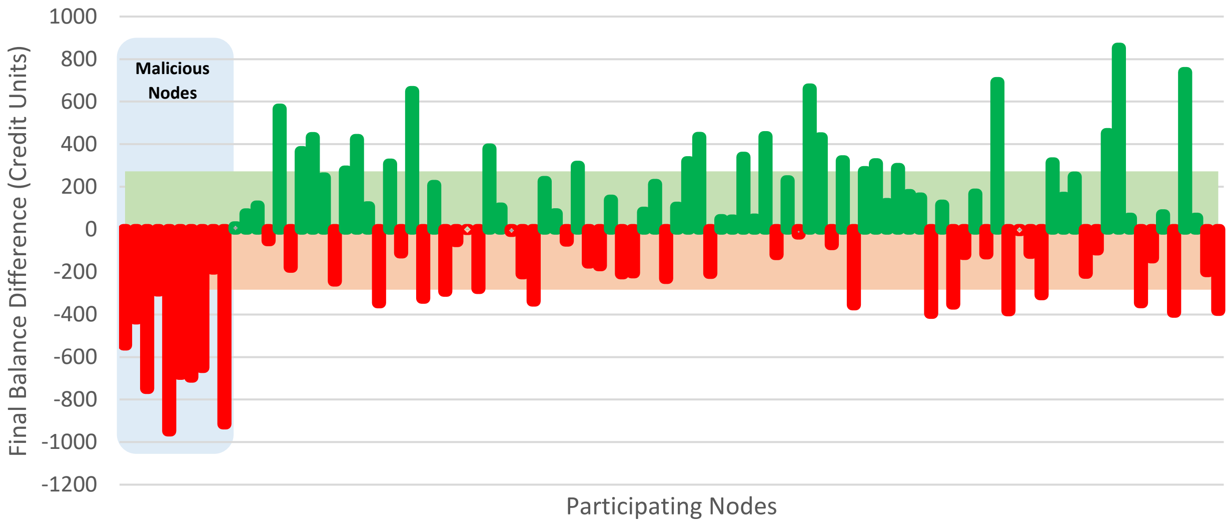


Figure 66: Correction in credit balances in case of 10% malicious peers

6.4.3 Discussion

To evaluate our proposal's efficiency on mitigating the effects of malicious actors who attempt to abuse the incentivization mechanism, we conducted two distinct experiments: (i) a session with no malicious actors and (ii) a session where 10% of the participants denied to report the help they enjoyed from their neighbors, in order to evade giving the appropriate rewards. Although most node metrics were randomly generated for the simulations, in both cases we used the same randomization seed; thus, the contribution of all peers was identical in both experiments. This allows us to quantify the correction achieved by the Consolidation Phase. As shown in Figure 67, the mean discrepancy of final balances between the two experiments was 3.2%. These results suggest that the effect of a significant portion of malicious peers was successfully eliminated at the expense of a minor decline in the final reward amount.

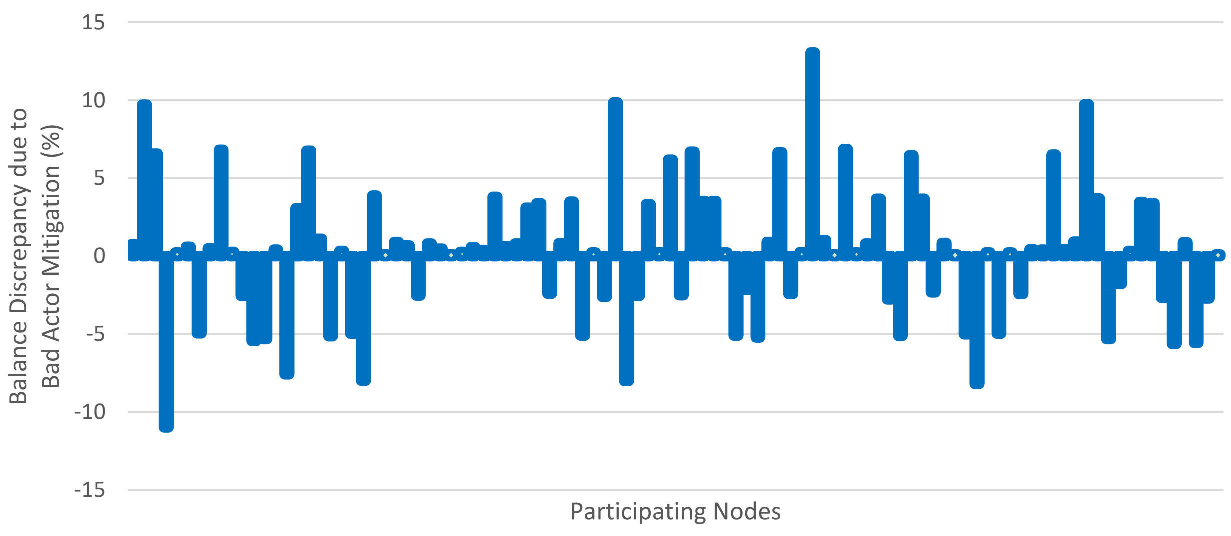


Figure 67: Credit balance discrepancy due to malicious peers

7 Conclusions and Future Work

7.1 Conclusions

In Chapter 3, we simulated a first approach for decentralized DNN training utilizing Distributed Ledger Technology as the data diffusion mechanism. The simulation that was contacted served as a proof-of-concept for the particular formula. Although the dataset employed contains about 10 million instances, only a limited subset (100,000) of those were used in this phase. The results indicated that such an architecture can offer useful features in collaborative DNN training.

We then extended our research to include more realistic conditions. The coordination algorithm was tested on a local network of PCs with moderate hardware specifications. Our methodology was solely comparative in nature, since we were interested in the potential advantages of decentralized collaboration over single node training, rather than actual performance.

Our framework was tested in a variety of configurations to investigate the effects of each parameter on the final outcome. Each of these setups was performed several times to verify that all measurements were adequately consistent. Experiments demonstrated that our approach may provide real benefits to participating peers without jeopardizing data privacy. Finally, this research determined the optimum set of parameters to achieve the greatest benefit in terms of model accuracy.

Although our experiments used just 15 workstations, they revealed a substantial improvement over nodes operating in isolation. According to the information gained from these trials, it is firmly concluded that increasing the total number of participating peers should have a positive effect on this improvement.

In Chapter 4, we developed and tested a novel method for embedding IoT sensors without compromising the architecture's distributed nature. Additionally, we examined a critical feature called resilience, and the effect of network disruptions and unreliable peers. Our experiments demonstrated that disruptions with an Offline Probability of up to 80% had no effect on overall efficiency.

In Chapter 5 we extended our research on Distributed Neural Network Training, by adding a subsystem that incentivizes peers to participate to a LEARNAE session. In this scenario, peers who have no interest in the generated model can join the swarm to profit from their constructive averaging. We developed a completely distributed Proof of Identification method and performed proof-of-concept tests to assess the key metrics.

In Chapter 6 we addressed the limitations of the tipping incentivization method, since it was not able to deal with bad actors who refuse to pay the proper rewards. We proposed a novel mechanism that relies on a collaboratively generated contribution profile for each participant. The whole swarm exchanges knowledge in order to build a Shared Contribution Ledger, which is then used to allocate the rewards for the peers.

With these contributions, LEARNAE fulfills the following properties:

(a) Because it is built on a peer-to-peer architecture, it does not need a central coordinating entity or nodes with elevated privileges.

(b) It accommodates participants with heterogeneous hardware, while avoiding locks caused by slow workers.

(c) It offers a variety of roles, depending on processing power and the availability of training data.

(d) It supports privacy mode, allowing peers to maintain ownership of their sensitive data.

(e) By using data replication, it is capable of recovering from severe network failures.

(f) It can retain all (meta)data indefinitely, allowing for the addition of new peers at any moment.

(g) It democratizes the process, since participants have complete access to all information and generated models and do not need costly or sophisticated hardware.

(h) It offers a robust incentivization mechanism, to attract participants who have no interest in the produced neural model.

Apart from typical training scenarios, this method can provide a solution for use cases in which training data is constantly accumulated, such as via IoT sensors. Thus, peers with same interests might establish a community to ensure that the best model is available at any given moment. As shown by the experiments, LEARNAE collaboratively produces models with an increased 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 selective parameter averaging. According to our tests, increasing the total number of participants will enhance the benefits of the proposed algorithm.

7.2 Future Work

Numerous intriguing issues remain to be addressed. How effectively can an ecosystem of this kind scale? How much can a large-scale deployment ultimately improve the obtained results? How does the performance/resilience trade-off work in practice? How effectively can the implemented algorithm work in conjunction with techniques such as model sharding to achieve faster convergence? What extra information may be utilized to improve the selection of candidate remote models for averaging? Are there any experimental DLT platforms that can be a better match outperforming the ones reviewed? Is this approach applicable to other training techniques, such as Random Forests, in which each worker trains a single tree or a set of trees using their own sample of data?

The incentive algorithm must be evaluated on the following scenarios: (a) Sessions without data privacy; in this case, peers who join only for the purpose of profiting would donate their processing power to both training and averaging. (b) Sessions with two separate groups: data suppliers and profiteers. These trials would enable an in-depth examination of the digital asset exchange between these two groups.

All of the above will constitute the backbone of our future work.

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*, Cham, 2019, pp. 286–298. doi: 10.1007/978-3-030-20257-6_24.

[2] S. Nikolaidis and I. Refanidis, “Privacy preserving distributed training of neural networks,” *Neural Comput & Applic*, vol. 32, no. 23, pp. 17333–17350, Dec. 2020, doi: 10.1007/s00521-020-04880-0.

[3] S. Nikolaidis and I. Refanidis, “Using distributed ledger technology to democratize neural network training,” *Appl Intell*, Mar. 2021, doi: 10.1007/s10489-021-02340-3.

[4] S. Nikolaidis and I. Refanidis, “Incentivizing Participation to Distributed Neural Network Training,” in *Proceedings of the 22nd Engineering Applications of Neural Networks Conference*, Cham, 2021, pp. 364–374. doi: 10.1007/978-3-030-80568-5_30.

[5] S. Nikolaidis and I. Refanidis, “Consolidating Incentivization in Distributed Neural Network Training via Decentralized Autonomous Organization,” *(submitted)*.

[6] J. Benet, “IPFS - Content Addressed, Versioned, P2P File System,” *arXiv:1407.3561 [cs]*, Jul. 2014, Accessed: Apr. 01, 2021. [Online]. Available: http://arxiv.org/abs/1407.3561

[7] S. Popov, O. Saa, and P. Finardi, “Equilibria in the Tangle,” *Computers & Industrial Engineering*, vol. 136, pp. 160–172, Oct. 2019, doi: 10.1016/j.cie.2019.07.025.

[8] X. Zhang, J. Trmal, D. Povey, and S. Khudanpur, “Improving deep neural network acoustic models using generalized maxout networks,” in *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, May 2014, pp. 215–219. doi: 10.1109/ICASSP.2014.6853589.

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

[10] J. Dean *et al.*, “Large Scale Distributed Deep Networks,” in *Advances in Neural Information Processing Systems*, 2012, vol. 25. Accessed: Aug. 11, 2021. [Online]. Available: https://papers.nips.cc/paper/2012/hash/6aca97005c68f1206823815f66102863-Abstract.html

[11] O. G. Selfridge, “Pattern recognition and modern computers,” in *Proceedings of the March 1-3, 1955, western joint computer conference*, New York, NY, USA, Mar. 1955, pp. 91–93. doi: 10.1145/1455292.1455310.

[12] B. F. Skinner, *Science and human behavior.* New York: Macmillan, 1953.

[13] R. R. Bush and F. Mosteller, *Stochastic models for learning*. Oxford, England: John Wiley & Sons, Inc., 1955, pp. xvi, 365.

[14] A. Ehrenfeucht, “Allen Newell and Herbert A. Simon. The logic theory machine. A complex information processing system. Institute of Radio Engineers, Transactions on

information theory, vol. IT-2 no. 3 (1956), pp. 61–79.," *The Journal of Symbolic Logic*, vol. 22, no. 3, pp. 331–332, Sep. 1957, doi: 10.2307/2963663.

[15] A. Newell, J. C. Shaw, and H. A. Simon, "Empirical explorations of the logic theory machine: a case study in heuristic," in *Papers presented at the February 26-28, 1957, western joint computer conference: Techniques for reliability*, New York, NY, USA, Feb. 1957, pp. 218–230. doi: 10.1145/1455567.1455605.

[16] R. J. Solomonoff, "An Inductive Inference Machine," in *IRE Convention Record, Section on Information Theory, Part 2*, 1957, pp. 56–62.

[17] R. J. Solomonoff, "The Mechanization of Linguistic Learning," in *Proceedings of the Second International Congress on Cybernetics*, Namur, Belgium, May 1958, pp. 180–193.

[18] R. J. Solomonoff, "A new method for discovering the grammars of phrase structure languages," Unesco, Paris, France, 1959.

[19] R. J. Solomonoff, "A Preliminary Report on a General Theory of Inductive Inference. (Revision of Report V-131)," Zator Co. and Air Force Office of Scientific Research, Cambridge, Mass., ZTB-138, Nov. 1960.

[20] D. Silver *et al.*, "Mastering the game of Go without human knowledge," *Nature*, vol. 550, no. 7676, pp. 354–359, Oct. 2017, doi: 10.1038/nature24270.

[21] N. Bostrom, *Superintelligence: Paths, Dangers, Strategies*. Oxford, New York: Oxford University Press, 2014.

[22] O. B. Bassler, "The Surveyability of Mathematical Proof: A Historical Perspective," *Synthese*, vol. 148, no. 1, pp. 99–133, 2006.

[23] E. Coleman, "The Surveyability of Long Proofs," *Foundations of Science*, vol. 14, pp. 27–43, Mar. 2009, doi: 10.1007/s10699-008-9145-8.

[24] R. V. Yampolskiy, "Efficiency Theory : a Unifying Theory for Information, Computation and Intelligence," *Journal of Discrete Mathematical Sciences and Cryptography*, vol. 16, no. 4–5, pp. 259–277, Oct. 2013, doi: 10.1080/09720529.2013.821361.

[25] D. B. Fogel, "An introduction to simulated evolutionary optimization," *IEEE Transactions on Neural Networks*, vol. 5, no. 1, pp. 3–14, Jan. 1994, doi: 10.1109/72.265956.

[26] T. Back, U. Hammel, and H.-P. Schwefel, "Evolutionary computation: comments on the history and current state," *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 3–17, Apr. 1997, doi: 10.1109/4235.585888.

[27] S. Shalev-Shwartz and S. Ben-David, *Understanding Machine Learning: From Foundations to Algorithms*. Cambridge University Press, 2014.

[28] H. Robbins and S. Monro, "A Stochastic Approximation Method," *The Annals of Mathematical Statistics*, vol. 22, no. 3, pp. 400–407, Sep. 1951, doi: 10.1214/aoms/1177729586.

[29] J. Kiefer and J. Wolfowitz, “Stochastic Estimation of the Maximum of a Regression Function,” *The Annals of Mathematical Statistics*, vol. 23, no. 3, pp. 462–466, Sep. 1952, doi: 10.1214/aoms/1177729392.

[30] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning representations by back-propagating errors,” *Nature*, vol. 323, pp. 533–536, Oct. 1986, doi: 10.1038/323533a0.

[31] L. Bottou and O. Bousquet, “The Tradeoffs of Large Scale Learning,” in *Advances in Neural Information Processing Systems*, 2008, pp. 161–168. Accessed: Aug. 06, 2021. [Online]. Available: http://leon.bottou.org/publications/pdf/nips-2007.pdf

[32] O. Bousquet, S. Gelly, K. Kurach, O. Teytaud, and D. Vincent, “Critical Hyper-Parameters: No Random, No Cry,” *arXiv:1706.03200 [cs]*, Jun. 2017, Accessed: Aug. 06, 2021. [Online]. Available: http://arxiv.org/abs/1706.03200

[33] B. Polyak, “Some methods of speeding up the convergence of iteration methods,” *Ussr Computational Mathematics and Mathematical Physics*, vol. 4, pp. 1–17, Dec. 1964, doi: 10.1016/0041-5553(64)90137-5.

[34] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, “On the importance of initialization and momentum in deep learning,” in *International Conference on Machine Learning*, May 2013, pp. 1139–1147. Accessed: Aug. 06, 2021. [Online]. Available: http://proceedings.mlr.press/v28/sutskever13.html

[35] Y. Nesterov, “A method for unconstrained convex minimization problem with the rate of convergence o(1/k^2),” *undefined*, 1983, Accessed: Aug. 06, 2021. [Online]. Available: https://www.semanticscholar.org/paper/A-method-for-unconstrained-convex-minimization-with-Nesterov/ed910d96802212c9e45d956adaa27d915f5d7469

[36] D. P. Kingma and J. Ba, “Adam: A Method for Stochastic Optimization,” *arXiv:1412.6980 [cs]*, Jan. 2017, Accessed: Aug. 06, 2021. [Online]. Available: http://arxiv.org/abs/1412.6980

[37] F. Seide, G. Li, and D. Yu, “Conversational speech transcription using context-dependent deep neural networks,” in *in Proc. Interspeech ’11*, pp. 437–440.

[38] R. McDonald, K. Hall, and G. Mann, “Distributed Training Strategies for the Structured Perceptron,” in *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, Los Angeles, California, Jun. 2010, pp. 456–464. Accessed: Mar. 29, 2021. [Online]. Available: https://www.aclweb.org/anthology/N10-1069

[39] R. Mcdonald, M. Mohri, N. Silberman, D. Walker, and G. Mann, “Efficient Large-Scale Distributed Training of Conditional Maximum Entropy Models,” *Advances in Neural Information Processing Systems*, vol. 22, 2009, Accessed: Aug. 06, 2021. [Online]. Available: https://proceedings.neurips.cc/paper/2009/hash/d81f9c1be2e08964bf9f24b15f0e4900-Abstract.html

[40] M. Chowdhury, M. Zaharia, J. Ma, M. I. Jordan, and I. Stoica, “Managing data transfers in computer clusters with orchestra,” *SIGCOMM Comput. Commun. Rev.*, vol. 41, no. 4, pp. 98–109, Aug. 2011, doi: 10.1145/2043164.2018448.

[41] Z. Zhang, L. Cherkasova, and B. T. Loo, “Performance Modeling of MapReduce Jobs in Heterogeneous Cloud Environments,” in *2013 IEEE Sixth International Conference on Cloud Computing*, Jun. 2013, pp. 839–846. doi: 10.1109/CLOUD.2013.107.

[42] R. M. Russell, “The CRAY-1 computer system,” *Commun. ACM*, vol. 21, no. 1, pp. 63–72, Jan. 1978, doi: 10.1145/359327.359336.

[43] P. C. Treleaven, D. R. Brownbridge, and R. P. Hopkins, “Data-Driven and Demand-Driven Computer Architecture,” *ACM Comput. Surv.*, vol. 14, no. 1, pp. 93–143, Mar. 1982, doi: 10.1145/356869.356873.

[44] A. K. Jones and P. Schwarz, “Experience Using Multiprocessor Systems&#x2014;A Status Report,” *ACM Comput. Surv.*, vol. 12, no. 2, pp. 121–165, Jun. 1980, doi: 10.1145/356810.356813.

[45] W. C. Athas and C. L. Seitz, “Multicomputers: message-passing concurrent computers,” *Computer*, vol. 21, no. 8, pp. 9–24, Aug. 1988, doi: 10.1109/2.73.

[46] A. S. Tanenbaum and R. van Renesse, “Distributed Operating Systems,” *ACM Comput Surv*, vol. 17, no. 4, pp. 419–470, Dec. 1985, doi: 10.1145/6041.6074.

[47] C. L. Seitz, “The cosmic cube,” *Commun. ACM*, vol. 28, no. 1, pp. 22–33, Jan. 1985, doi: 10.1145/2465.2467.

[48] S. Ranka, Y. Won, and S. Sahni, “Programming a hypercube multicomputer,” *IEEE Software*, vol. 5, no. 5, pp. 69–77, Sep. 1988, doi: 10.1109/52.7944.

[49] E. H. Baalbergen, “Design and implementation of parallel make,” *Computing Systems*, vol. 1, pp. 135–158, 1988.

[50] H. E. Bal and A. S. Tanenbaum, “Distributed Programming with Shared Data,” *1988 Internation Conference on Computer Languages*, pp. 82–91, 1988, doi: 10.1109/ICCL.1988.13046.

[51] T. A. Marsland, M. Olafsson, and J. Schaeffer, “Multiprocessor tree-search experiments,” in *Advances in computer chess*, USA: Pergamon Press, Inc., 1986, pp. 37–51.

[52] L. Lamport, R. Shostak, and M. Pease, “The Byzantine Generals Problem,” *ACM Trans. Program. Lang. Syst.*, vol. 4, no. 3, pp. 382–401, Jul. 1982, doi: 10.1145/357172.357176.

[53] Y. Xiao, N. Zhang, J. Li, W. Lou, and Y. Hou, “Distributed Consensus Protocols and Algorithms,” 2019, pp. 25–50. doi: 10.1002/9781119519621.ch2.

[54] H. Attiya and J. Welch, *Distributed Computing: Fundamentals, Simulations, and Advanced Topics*. John Wiley & Sons, 2004.

[55] C. Dwork, N. Lynch, and L. Stockmeyer, “Consensus in the presence of partial synchrony,” *J. ACM*, vol. 35, no. 2, pp. 288–323, Apr. 1988, doi: 10.1145/42282.42283.

[56] M. Pease, R. Shostak, and L. Lamport, “Reaching Agreement in the Presence of Faults,” *J. ACM*, vol. 27, no. 2, pp. 228–234, Apr. 1980, doi: 10.1145/322186.322188.

[57] S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System," *Cryptography Mailing list at https://metzdowd.com*, Mar. 2009.

[58] J. Garay, A. Kiayias, and N. Leonardos, "The Bitcoin Backbone Protocol: Analysis and Applications," in *Advances in Cryptology - EUROCRYPT 2015*, Berlin, Heidelberg, 2015, pp. 281–310. doi: 10.1007/978-3-662-46803-6_10.

[59] C. Decker and R. Wattenhofer, "Information propagation in the Bitcoin network," *IEEE P2P 2013 Proceedings*, 2013, doi: 10.1109/P2P.2013.6688704.

[60] S. Eskandari, D. Barrera, E. Stobert, and J. Clark, *A First Look at the Usability of Bitcoin Key Management*. 2015. doi: 10.14722/usec.2015.23015.

[61] A. Panarello, N. Tapas, G. Merlino, F. Longo, and A. Puliafito, "Blockchain and IoT Integration: A Systematic Survey," *Sensors*, vol. 18, no. 8, Art. no. 8, Aug. 2018, doi: 10.3390/s18082575.

[62] T. Marwala and B. Xing, "Blockchain and Artificial Intelligence," *arXiv:1802.04451 [cs]*, Oct. 2018, Accessed: Aug. 05, 2021. [Online]. Available: http://arxiv.org/abs/1802.04451

[63] P. Bell and K. Jabbour, "Review of point-to-point network routing algorithms," *IEEE Communications Magazine*, vol. 24, no. 1, pp. 34–38, Jan. 1986, doi: 10.1109/MCOM.1986.1092937.

[64] L. M. Feeney, *A Taxonomy for Routing Protocols in Mobile Ad Hoc Networks*. Swedish Institute of Computer Science, 1999. Accessed: Aug. 06, 2021. [Online]. Available: http://urn.kb.se/resolve?urn=urn:nbn:se:ri:diva-21975

[65] X. Zou, B. Ramamurthy, and S. Magliveras, "Routing techniques in wireless ad hoc networks — classification and comparison," 2002.

[66] M. K. Reiter and A. D. Rubin, "Crowds: anonymity for Web transactions," *ACM Trans. Inf. Syst. Secur.*, vol. 1, no. 1, pp. 66–92, Nov. 1998, doi: 10.1145/290163.290168.

[67] M. Rennhard and B. Plattner, "Introducing MorphMix: peer-to-peer based anonymous Internet usage with collusion detection," in *Proceedings of the 2002 ACM workshop on Privacy in the Electronic Society*, New York, NY, USA, Nov. 2002, pp. 91–102. doi: 10.1145/644527.644537.

[68] M. J. Freedman and R. Morris, "Tarzan: a peer-to-peer anonymizing network layer," in *Proceedings of the 9th ACM conference on Computer and communications security*, New York, NY, USA, Nov. 2002, pp. 193–206. doi: 10.1145/586110.586137.

[69] M. Harchol-Balter, T. Leighton, and D. Lewin, "Resource discovery in distributed networks," in *Proceedings of the eighteenth annual ACM symposium on Principles of distributed computing*, New York, NY, USA, May 1999, pp. 229–237. doi: 10.1145/301308.301362.

[70] I. Stoica, R. Morris, D. Karger, M. Kaashoek, and H. Balakrishnan, "Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications," *ACM SIGCOMM Computer Communication Review, vol. 31*, vol. 31, Dec. 2001, doi: 10.1145/964723.383071.

[71] C. Cachin and M. Vukolic, “Blockchain Consensus Protocols in the Wild (Keynote Talk),” in *31st International Symposium on Distributed Computing (DISC 2017)*, Dagstuhl, Germany, 2017, vol. 91, p. 1:1-1:16. doi: 10.4230/LIPIcs.DISC.2017.1.

[72] N. Szabo, “Smart Contracts.” 1994. Accessed: Aug. 05, 2021. [Online]. Available: https://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/smart.contracts.html

[73] P. Lade, R. Ghosh, and S. Srinivasan, “Manufacturing Analytics and Industrial Internet of Things,” *IEEE Intelligent Systems*, vol. 32, no. 3, pp. 74–79, May 2017, doi: 10.1109/MIS.2017.49.

[74] “Eth(Embedded) – Ethereum Clients on Embedded Devices,” 2017. http://ethembedded.com/ (accessed Aug. 06, 2021).

[75] “raspnode,” 2017. https://raspnode.com/ (accessed Aug. 06, 2021).

[76] A. J. Mashtizadeh, A. Bittau, Y. F. Huang, and D. Mazières, “Replication, history, and grafting in the Ori file system,” in *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, New York, NY, USA, Nov. 2013, pp. 151–166. doi: 10.1145/2517349.2522721.

[77] B. Cohen, “Incentives build robustness in BitTorrent,” *Workshop on Economics of PeertoPeer systems*, vol. 6, Jun. 2003.

[78] I. Baumgart and S. Mies, “S/Kademlia: A practicable approach towards secure key-based routing,” in *2007 International Conference on Parallel and Distributed Systems*, Dec. 2007, pp. 1–8. doi: 10.1109/ICPADS.2007.4447808.

[79] M. J. Freedman, E. Freudenthal, and D. Mazières, “Democratizing content publication with coral,” in *Proceedings of the 1st conference on Symposium on Networked Systems Design and Implementation - Volume 1*, USA, Mar. 2004, p. 18.

[80] L. Wang and J. Kangasharju, “Measuring Large-Scale Distributed Systems: Case of BitTorrent Mainline DHT,” presented at the IEEE International Conference on Peer-to-Peer Computing, Sep. 2013. doi: 10.1109/P2P.2013.6688697.

[81] S. Shalunov, G. Hazel, B. Inc, J. Iyengar, and M. Kuehlewind, “Low Extra Delay Background Transport (LEDBAT,” 2012.

[82] M. J. Freedman, E. Freudenthal, and D. Mazières, “Democratizing content publication with coral,” in *Proceedings of the 1st conference on Symposium on Networked Systems Design and Implementation - Volume 1*, USA, Mar. 2004, p. 18.

[83] D. Mazieres, “Self-certifying file system,” phd, Massachusetts Institute of Technology, USA, 2000.

[84] D. Mazières, D. M. Eres, and M. F. Kaashoek, “Escaping the Evils of Centralized Control with self-certifying pathnames,” in *In the Proceedings of the 8th ACM SIGOPS European*, 1998, pp. 118–125.

[85] S. Popov, “The Tangle,” 2015. https://www.semanticscholar.org/paper/The-Tangle-Popov/43586b34b054b48891d478407d4e7435702653e0 (accessed Aug. 06, 2021).

[86] M. Divya and N. B. Biradar, “IOTA-Next Generation Block chain,” 2018, doi: 10.18535/IJECS/V7I4.05.

[87] A. Gal, “The Tangle: an Illustrated Introduction,” *IOTA Foundation Blog*, Jan. 31, 2018. http://blog.iota.org/the-tangle-an-illustrated-introduction-4d5eae6fe8d4 (accessed Aug. 06, 2021).

[88] S. M. Ross, *Introduction to Probability Models*. Academic Press, 2014.

[89] R. C. Merkle, “A Digital Signature Based on a Conventional Encryption Function,” in *Advances in Cryptology — CRYPTO ’87*, Berlin, Heidelberg, 1988, pp. 369–378. doi: 10.1007/3-540-48184-2_32.

[90] “One seed to sow your key(s),” *IOTA Foundation Blog*, Oct. 01, 2018. http://blog.iota.org/one-seed-to-sow-your-key-s-f074f1bb6714 (accessed Aug. 06, 2021).

[91] S. Shafeeq, S. Zeadally, M. Alam, and A. Khan, “Curbing Address Reuse in the IOTA Distributed Ledger: A Cuckoo-Filter-Based Approach,” *IEEE Transactions on Engineering Management*, vol. 67, no. 4, pp. 1244–1255, Nov. 2020, doi: 10.1109/TEM.2019.2922710.

[92] J. Buchmann and J. Ding, *Post-Quantum Cryptography: Second International Workshop, PQCrypto 2008 Cincinnati, OH, USA October 17-19, 2008 Proceedings*. Springer Science & Business Media, 2008.

[93] S. Rohde, T. Eisenbarth, E. Dahmen, J. Buchmann, and C. Paar, “Fast Hash-Based Signatures on Constrained Devices,” in *Smart Card Research and Advanced Applications*, Berlin, Heidelberg, 2008, pp. 104–117. doi: 10.1007/978-3-540-85893-5_8.

[94] M. Abadi *et al.*, “TensorFlow: A System for Large-Scale Machine Learning,” 2016, pp. 265–283. Accessed: Mar. 29, 2021. [Online]. Available: https://www.usenix.org/conference/osdi16/technical-sessions/presentation/abadi

[95] S. Zhang, A. Choromanska, and Y. LeCun, “Deep learning with Elastic Averaging SGD,” *arXiv:1412.6651 [cs, stat]*, Oct. 2015, Accessed: Mar. 29, 2021. [Online]. Available: http://arxiv.org/abs/1412.6651

[96] F. Niu, B. Recht, C. Re, and S. J. Wright, “HOGWILD!: A Lock-Free Approach to Parallelizing Stochastic Gradient Descent,” *arXiv:1106.5730 [cs, math]*, Nov. 2011, Accessed: Mar. 29, 2021. [Online]. Available: http://arxiv.org/abs/1106.5730

[97] R. Shokri and V. Shmatikov, “Privacy-Preserving Deep Learning,” in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, New York, NY, USA, Oct. 2015, pp. 1310–1321. doi: 10.1145/2810103.2813687.

[98] T. Chen *et al.*, “MXNet: A Flexible and Efficient Machine Learning Library for Heterogeneous Distributed Systems,” *arXiv:1512.01274 [cs]*, Dec. 2015, Accessed: Mar. 29, 2021. [Online]. Available: http://arxiv.org/abs/1512.01274

[99] F. N. Iandola, K. Ashraf, M. W. Moskewicz, and K. Keutzer, “FireCaffe: near-linear acceleration of deep neural network training on compute clusters,” *arXiv:1511.00175 [cs]*, Jan. 2016, Accessed: Mar. 29, 2021. [Online]. Available: http://arxiv.org/abs/1511.00175

[100] M. Langer, A. Hall, Z. He, and W. Rahayu, “MPCA SGD—A Method for Distributed Training of Deep Learning Models on Spark,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 29, no. 11, pp. 2540–2556, Nov. 2018, doi: 10.1109/TPDS.2018.2833074.

[101] M. Zaharia *et al.*, “Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing,” 2012, pp. 15–28. Accessed: Mar. 29, 2021. [Online]. Available: https://www.usenix.org/conference/nsdi12/technical-sessions/presentation/zaharia

[102] K. Bonawitz *et al.*, “Towards Federated Learning at Scale: System Design,” in *Proceedings of Machine Learning and Systems*, 2019, vol. 1, pp. 374–388. [Online]. Available: https://proceedings.mlsys.org/paper/2019/file/bd686fd640be98efaae0091fa301e613-Paper.pdf

[103] T. Li, A. K. Sahu, A. Talwalkar, and V. Smith, “Federated Learning: Challenges, Methods, and Future Directions,” *IEEE Signal Processing Magazine*, vol. 37, no. 3, pp. 50–60, Feb. 2020, doi: 10.1109/MSP.2020.2975749.

[104] P. Kairouz *et al.*, “Advances and Open Problems in Federated Learning,” *MAL*, vol. 14, no. 1–2, pp. 1–210, Jun. 2021, doi: 10.1561/2200000083.

[105] A. Coates, B. Huval, T. Wang, D. Wu, B. Catanzaro, and N. Andrew, “Deep learning with COTS HPC systems,” in *International Conference on Machine Learning*, May 2013, pp. 1337–1345. Accessed: Mar. 29, 2021. [Online]. Available: http://proceedings.mlr.press/v28/coates13.html

[106] D. Povey, X. Zhang, and S. Khudanpur, “Parallel training of DNNs with Natural Gradient and Parameter Averaging,” *arXiv:1410.7455 [cs, stat]*, Jun. 2015, Accessed: Mar. 29, 2021. [Online]. Available: http://arxiv.org/abs/1410.7455

[107] 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,” pp. 1058–1062, Jan. 2014.

[108] O. Dekel, R. Gilad-Bachrach, O. Shamir, and L. Xiao, “Optimal Distributed Online Prediction using Mini-Batches,” *arXiv:1012.1367 [cs, math]*, Jan. 2012, Accessed: Mar. 29, 2021. [Online]. Available: http://arxiv.org/abs/1012.1367

[109] M. Li, “Scaling Distributed Machine Learning with the Parameter Server,” in *Proceedings of the 2014 International Conference on Big Data Science and Computing - BigDataScience ’14*, Beijing, China, 2014, pp. 1–1. doi: 10.1145/2640087.2644155.

[110] Y. Jia *et al.*, “Caffe: Convolutional Architecture for Fast Feature Embedding,” *arXiv:1408.5093 [cs]*, Jun. 2014, Accessed: Mar. 30, 2021. [Online]. Available: http://arxiv.org/abs/1408.5093

[111] J. Dai *et al.*, “BigDL: A Distributed Deep Learning Framework for Big Data,” *arXiv:1804.05839 [cs]*, Nov. 2019, doi: 10.1145/1122445.1122456.

[112] E. P. Xing *et al.*, “Petuum: A New Platform for Distributed Machine Learning on Big Data,” *IEEE TRANSACTIONS ON BIG DATA*, p. 17, 2015.

[113] P. Moritz, R. Nishihara, I. Stoica, and M. I. Jordan, “SparkNet: Training Deep Networks in Spark,” *arXiv:1511.06051 [cs, math, stat]*, Feb. 2016, Accessed: Mar. 30, 2021. [Online]. Available: http://arxiv.org/abs/1511.06051

[114] 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,” *Advances in Neural Information Processing Systems*, vol. 30, 2017, Accessed: Mar. 29, 2021. [Online]. Available: https://proceedings.neurips.cc/paper/2017/hash/f75526659f31040afeb61cb7133e4e6d-Abstract.html

[115] M. Blot, D. Picard, M. Cord, and N. Thome, “Gossip training for deep learning,” *arXiv:1611.09726 [cs, stat]*, Nov. 2016, Accessed: Mar. 30, 2021. [Online]. Available: http://arxiv.org/abs/1611.09726

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

[117] H. Kim, J. Park, J. Jang, and S. Yoon, “DeepSpark: A Spark-Based Distributed Deep Learning Framework for Commodity Clusters,” *arXiv:1602.08191 [cs]*, Sep. 2016, Accessed: Mar. 30, 2021. [Online]. Available: http://arxiv.org/abs/1602.08191

[118] X. Lian, W. Zhang, C. Zhang, and J. Liu, “Asynchronous Decentralized Parallel Stochastic Gradient Descent,” p. 10.

[119] E. De Coninck *et al.*, “DIANNE: a modular framework for designing, training and deploying deep neural networks on heterogeneous distributed infrastructure,” *JOURNAL OF SYSTEMS AND SOFTWARE*, vol. 141, pp. 52–65, 2018, doi: 10.1016/j.jss.2018.03.032.

[120] A. R. Mamidala, G. Kollias, C. Ward, and F. Artico, “MXNET-MPI: Embedding MPI parallelism in Parameter Server Task Model for scaling Deep Learning,” *arXiv:1801.03855 [cs]*, Jan. 2018, Accessed: Mar. 30, 2021. [Online]. Available: http://arxiv.org/abs/1801.03855

[121] A. Sergeev and M. Del Balso, “Horovod: fast and easy distributed deep learning in TensorFlow,” *arXiv:1802.05799 [cs, stat]*, Feb. 2018, Accessed: Mar. 30, 2021. [Online]. Available: http://arxiv.org/abs/1802.05799

[122] Y. Peng *et al.*, “A generic communication scheduler for distributed DNN training acceleration,” in *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, New York, NY, USA, Oct. 2019, pp. 16–29. doi: 10.1145/3341301.3359642.

[123] Y. Jiang, Y. Zhu, C. Lan, B. Yi, Y. Cui, and C. Guo, “A Unified Architecture for Accelerating Distributed {DNN} Training in Heterogeneous GPU/CPU Clusters,” 2020, pp. 463–479. Accessed: Aug. 11, 2021. [Online]. Available: https://www.usenix.org/conference/osdi20/presentation/jiang

[124] S. Zheng, Z. Huang, and J. Kwok, “Communication-Efficient Distributed Blockwise Momentum SGD with Error-Feedback,” *Advances in Neural Information Processing Systems*, vol. 32, 2019, Accessed: Mar. 30, 2021. [Online]. Available: https://papers.nips.cc/paper/2019/hash/80c0e8c4457441901351e4abbcf8c75c-Abstract.html

[125] B. Yuan, C. R. Wolfe, C. Dun, Y. Tang, A. Kyrillidis, and C. M. Jermaine, “Distributed Learning of Deep Neural Networks using Independent Subnet Training,” *arXiv:1910.02120 [cs, stat]*, Nov. 2020, Accessed: Mar. 30, 2021. [Online]. Available: http://arxiv.org/abs/1910.02120

[126] S. Shen, L. Xu, J. Liu, X. Liang, and Y. Cheng, “Faster Distributed Deep Net Training: Computation and Communication Decoupled Stochastic Gradient Descent,” *arXiv:1906.12043 [cs, math, stat]*, Sep. 2019, Accessed: Mar. 30, 2021. [Online]. Available: http://arxiv.org/abs/1906.12043

[127] S. Wang, D. Li, and J. Geng, “Geryon: Accelerating Distributed CNN Training by Network-Level Flow Scheduling,” in *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications*, Toronto, ON, Canada, Jul. 2020, pp. 1678–1687. doi: 10.1109/INFOCOM41043.2020.9155282.

[128] Y. Bao, Y. Peng, Y. Chen, and C. Wu, “Preemptive All-reduce Scheduling for Expediting Distributed DNN Training,” in *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications*, Toronto, ON, Canada, Jul. 2020, pp. 626–635. doi: 10.1109/INFOCOM41043.2020.9155446.

[129] A. Jayarajan, J. Wei, G. Gibson, A. Fedorova, and G. Pekhimenko, “Priority-based Parameter Propagation for Distributed DNN Training,” *arXiv:1905.03960 [cs]*, May 2019, Accessed: Mar. 30, 2021. [Online]. Available: http://arxiv.org/abs/1905.03960

[130] A. Sapio *et al.*, “Scaling Distributed Machine Learning with In-Network Aggregation,” *arXiv:1903.06701 [cs, stat]*, Sep. 2020, Accessed: Mar. 30, 2021. [Online]. Available: http://arxiv.org/abs/1903.06701

[131] S. H. Hashemi, S. A. Jyothi, and R. H. Campbell, “TicTac: Accelerating Distributed Deep Learning with Communication Scheduling,” *arXiv:1803.03288 [cs]*, Oct. 2018, Accessed: Aug. 11, 2021. [Online]. Available: http://arxiv.org/abs/1803.03288

[132] A. Hsu, K. Hu, J. Hung, A. Suresh, and Z. Zhang, “TonY: An Orchestrator for Distributed Machine Learning Jobs,” p. 4.

[133] S. Shi *et al.*, “Towards Scalable Distributed Training of Deep Learning on Public Cloud Clusters,” *arXiv:2010.10458 [cs]*, Oct. 2020, Accessed: Mar. 30, 2021. [Online]. Available: http://arxiv.org/abs/2010.10458

[134] R. Peeters and M. Schuilenburg, “Machine justice: Governing security through the bureaucracy of algorithms,” *Inf. Polity*, 2018, doi: 10.3233/IP-180074.

[135] R. Kitchin, “Big Data, new epistemologies and paradigm shifts,” *Big Data & Society*, vol. 1, no. 1, p. 2053951714528481, Apr. 2014, doi: 10.1177/2053951714528481.

[136] M. Veale and I. Brass, “Administration by Algorithm? Public Management Meets Public Sector Machine Learning,” Social Science Research Network, Rochester, NY, SSRN Scholarly

Paper ID 3375391, 2019. Accessed: Dec. 05, 2021. [Online]. Available: https://papers.ssrn.com/abstract=3375391

[137] H. Mintzberg, “Structure in 5’s: A Synthesis of the Research on Organization Design,” *Management Science*, vol. 26, no. 3, pp. 322–341, 1980.

[138] A. Zuurmond, “De infocratie: een theoretische en empirische herori??ntatie op Weber’s ideaaltype in het informatietijdperk,” Phaedrus, Den Haag, 1994.

[139] L. Lessig, *Code: And Other Laws of Cyberspace, Version 2.0*. 2006.

[140] A. Aneesh, “Global Labor: Algocratic Modes of Organization,” *Sociological Theory*, vol. 27, no. 4, pp. 347–370, Dec. 2009, doi: 10.1111/j.1467-9558.2009.01352.x.

[141] T. H. Cormen, *Algorithms Unlocked*. Cambridge, MA, USA: MIT Press, 2013.

[142] J. Wirtz *et al.*, “Brave new world: service robots in the frontline,” *Journal of Service Management*, vol. 29, no. 5, pp. 907–931, Jan. 2018, doi: 10.1108/JOSM-04-2018-0119.

[143] R. Mohabbat Kar, *(Un)berechenbar?. Algorithmen und Automatisierung in Staat und Gesellschaft*. Berlin: Fraunhofer FOKUS, 2018.

[144] J. Danaher *et al.*, “Algorithmic governance: Developing a research agenda through the power of collective intelligence,” *Big Data & Society*, vol. 4, no. 2, p. 2053951717726554, Dec. 2017, doi: 10.1177/2053951717726554.

[145] N. Prusty, *Building Blockchain Projects*. Birmingham Mumbai: Packt Publishing Limited, 2017.

[146] U. W. Chohan, “The Decentralized Autonomous Organization and Governance Issues,” Social Science Research Network, Rochester, NY, SSRN Scholarly Paper ID 3082055, Dec. 2017. doi: 10.2139/ssrn.3082055.

[147] P. Vigna and M. J. Casey, *The Age of Cryptocurrency: How Bitcoin and the Blockchain Are Challenging the Global Economic Order*, Reprint edition. New York, N. Y: Picador, 2016.

[148] “Bitcoin moves beyond mere money | New Scientist.” https://www.newscientist.com/article/dn24620-bitcoin-moves-beyond-mere-money/?ignored=irrelevant (accessed Dec. 05, 2021).

[149] “The DAO of accrue,” *The Economist*, May 19, 2016. Accessed: Dec. 05, 2021. [Online]. Available: https://www.economist.com/finance-and-economics/2016/05/19/the-dao-of-accrue

[150] N. Popper, “A Venture Fund With Plenty of Virtual Capital, but No Capitalist,” *The New York Times*, May 22, 2016. Accessed: Dec. 05, 2021. [Online]. Available: https://www.nytimes.com/2016/05/22/business/dealbook/crypto-ether-bitcoin-currency.html

[151] D. J. Pangburn, “The Humans Who Dream Of Companies That Won’t Need Us,” *Fast Company*, Jun. 19, 2015. https://www.fastcompany.com/3047462/the-humans-who-dream-of-companies-that-wont-need-them (accessed Dec. 05, 2021).

[152] “Vapor No More: Ethereum Has Launched,” *TechCrunch*. https://social.techcrunch.com/2015/08/01/vapor-no-more-ethereum-has-launched/ (accessed Dec. 05, 2021).

[153] John Clippinger and David Bollier, *From Bitcoin To Burning Man & Beyond*. 2014. Accessed: Dec. 05, 2021. [Online]. Available: http://archive.org/details/FromBitcoinToBurningManBeyond