future internet

MDPI

Article

A Joint Survey in Decentralized Federated Learning and TinyML: A Brief Introduction to Swarm Learning

Evangelia Fragkou * and Dimitrios Katsaros *

Department of Electrical and Computer Engineering, University of Thessaly, 38334 Volos, Greece
* Correspondence: efragkou@uth.gr (E.F.); dkatsar@inf.uth.gr (D.K.)

Abstract: TinyML/DL is a new subfield of ML that allows for the deployment of ML algorithms on low-power devices to process their own data. The lack of resources restricts the aforementioned devices to running only inference tasks (static TinyML), while training is handled by a more computationally efficient system, such as the cloud. In recent literature, the focus has been on conducting real-time on-device training tasks (Reformable TinyML) while being wirelessly connected. With data processing being shift to edge devices, the development of decentralized federated learning (DFL) schemes becomes justified. Within these setups, nodes work together to train a neural network model, eliminating the necessity of a central coordinator. Ensuring secure communication among nodes is of utmost importance for protecting data privacy during edge device training. Swarm Learning (SL) emerges as a DFL paradigm that promotes collaborative learning through peer-to-peer interaction, utilizing edge computing and blockchain technology. While SL provides a robust defense against adversarial attacks, it comes at a high computational expense. In this survey, we emphasize the current literature regarding both DFL and TinyML/DL fields. We explore the obstacles encountered by resource-starved devices in this collaboration and provide a brief overview of the potential of transitioning to Swarm Learning.

Keywords: TinyML; decentralized federated learning; swarm learning

Citation: Fragkou, E.; Katsaros, D. A Joint Survey in Decentralized Federated Learning and TinyML: A Brief Introduction to Swarm Learning. *Future Internet* **2024**, *16*, 413. https://doi.org/10.3390/fi16110413

Academic Editor: Paolo Bellavista

Received: 31 August 2024
Revised: 11 October 2024
Accepted: 21 October 2024
Published: 8 November 2024

1. Introduction

Nowadays, the burgeoning paradigm of using machine/deep learning algorithms to enhance not only sectors of every day life, such as healthcare and smart devices for home automation but also security and surveillance, industrial monitoring, smart agriculture, etc. [1], necessitates shifting the training process and the deployment of machine/deep learning algorithms toward the edge, where the data are gathered. The ability of an edge device to process its own raw data locally is of a great importance since it can make decisions in real time whilst data privacy and independence are retained. However, the hardware resources are limited, making the deployment of conventional deep learning algorithms impossible. In taking into consideration Moore's controversial law [2], which first appeared in 1965 (stating that the density of transistors per circuit system is doubled every two years, meaning that the hardware becomes not only smaller and faster but also its cost is ever decreasing) and has existed over the last 50 years, the aforementioned edge devices will probably become powerful enough to support computationally expensive on-device training. Although Cortex-M-based devices have achieved unprecedented success in performance [3] as both on-chip SRAM and embedded Flash are available on MCUs, this kind of device has yet to meet the requirements to support the running of traditional machine/deep learning algorithms hitherto. For instance, ultra-low-power devices/boards (see Section 3), such as microcontrollers (MCUs) with limited RAM (some MB or even KB of RAM) and computational power cannot handle the GB of data necessary for the successful training of ML/DL models or support energy-intensive and complex algorithms. In the literature, there is a lot of work conducted toward the optimization of the *inference* phase

in these devices. The training phase of a model that is going to be deployed on an edge device takes place on a server, where both the required energy and a plethora of necessary data are abundant. Then, one uses suitable tools (e.g., TensorFlow lite) so that the model's size is then increased to some KB of RAM. The converted compact model can be applied to resource-limited target devices, and this is what *TinyML* is called [1,4]. There are a lot of works (see the section on pruning) in which a DL model undergoes processing with the aim of minimizing its size, like pruning, quantization, or both of them, in order to meet the memory capacity requirements of edge devices.

Adversely, the inference phase is a static procedure since its main purpose is to make predictions, based on the learned task, on new unseen data that the model is exposed to. The neural network model remains unchanged at its core during the inference phase, which limits its ability to learn new patterns and improve generalizations, or else it lacks *interpretability*.

In order to alleviate the problem of model adaptability to new tasks, *reformable* [5] and *on-device* TinyML were developed. Of course, reformable or otherwise *dynamic* TinyML has to be conducted under conditions that will not make the model "forget" about its previous acquired tasks; otherwise, it will cause the "catastrophic forgetting" effect [6] while trying to expand its possibilities. Although it has been proven that there is a trade-off between the precision of saved information and non-harmful forgetting, known as the *stability/plasticity dilemma* [7], it is of great importance to minimize this phenomenon as much as possible. A lot of nascent techniques have been introduced, such as Continual Learning (CL) (see Section 3.2.2); however, the real question is how could we enable real-time energy-and-storage-efficient training (without inference) on resource-starving devices? This is how *federated learning* (FL) [8] came to light.

In FL, many node devices cooperatively train a model either by sharing their parameters (not data themselves) with a server—centralized federated learning (CFL)—which makes the global aggregations and forwards the new updated weights back to the nodes in order to continue training, or by exchanging their parameters with one another—decentralized [9] or distributed federated learning (DFL)—in order to perform training without a central coordinator. There is also a hybrid form of both of them (see the section on Hybrid FL). In CFL, nodes usually form a star-shaped network, with the server being in the center of the network, so all nodes have access to it. In DFL, although nodes usually connect only with their one-hop neighbors, so that they form a ring-shaped network (ad hoc or peer-to-peer networks), there are also works that have implemented d-ring or mesh [10] topology under DFL settings. FL has also been proven to mitigate the lack of data availability of these devices during training, since they do not have to own all of the data required for training in order to reach convergence. Although there are a lot of approaches [11] regarding CFL, they do not actually meet the needs of TinyML environments, like IoTs, due to their requirement of a coordinator server. The energy required by the edge nodes to send their parameters to the server when embarking on the learning process may cause the depletion of their total energy, and/or *communication* overhead/flooding, since every node is obliged to communicate with the server. Moreover, the fact that the training process still takes place outside of the end devices hinders the ability of the devices to be independent. Despite the fact that FL was first introduced in order to handle the problem of privacy concerns, since nodes do not have to send their collected data over the network, only some of their parameters, as shown in [12], there is a possibility of reconstructing the initial data sent to the server by aggregating their total gradients. Thus, the literature has focused on DFL, which is a scheme a) that operates without a central orchestration; b) in which every node shares information with its neighbor nodes, which are wirelessly connected; c) that has unsolved problems like the broadcast storm problem [13] in ad hoc networks; and d) that lacks trustworthiness, since in ad hoc networks, there are often attacks that resource-starving devices cannot recognize/handle (due to their lack of energy/memory needed to always authenticate their collaborators).

Moreover, *Swarm Learning* (SL) [14] has emerged, which successfully combines DFL methodology in wireless networks with blockchain-based protocols for retaining trustwor-

thiness in the network. SL practices are not amenable to being deployed to end devices, since they are also computationally expensive.

The emergence of the aforementioned distributed deep learning technologies and the progress in the development of products running over distributed environments create the pressing need to describe the most significant advances in this research landscape. In light of this need, the main question that our survey aspires to answer is the following: What are the most recent and important solutions for addressing deep learning challenges in resource-starving environments that can be encountered in (a) highly distributed architectures running either over wireless ad hoc networks or over conventional peer-to-peer networks and (b) miniaturized devices, e.g., microcontrollers and sensors? Since security is always an important issue in distributed architectures, a second question that this survey will adequately cover is about What has been proposed for guaranteeing privacy and security in distributed deep learning?

The anatomy of this survey is the following: we aimed to provide answers to the aforementioned questions by drawing on academic papers, published in well-respected academic databases and digital libraries of publishers, e.g., Google Scholar, IEEEXplore, ACM DL, Scopus, and MDPI (see Figure 1), and whose publication age is no larger than five years. We ensured that the surveyed articles appeared in ACM/IEEE Transactions and Journals, and/or in journals with a significantly large impact factor, or in premium quality conferences such as NeurIPS, IJCNN, ICDM, and so on. We also included some yet unpublished articles (e.g., from arxiv.org), but only very recent articles for which (some of) the authors are prominent scientists or when we felt that the article is going to be a significant one. We refrained from setting hard "performance" thresholds such as number of citations or h-index as paper selection criteria, because we are well aware of how citation inflation efforts distort the scientific value of an article.

Our survey overlaps to a small extent with some earlier surveys; the kind/extent of overlap and our discrepancy is described in the rest of this paragraph. Table 1 lists published surveys most relevant to our work: Hu et al. [15] comprehensively presented the possibilities of deploying distributed deep learning and deep reinforcement learning in wireless networks, emphasizing distributed computing, e.g., Hadoop. In this survey, the model was mainly split, and every device trains on a part of the whole model, instead of collaboratively training the same model by sharing model parameters (the case we focus on). In parallel, Ajani et al. [1] presented a detailed review about machine learning optimization in order to fit in low-power devices (e.g., embedded systems, MCUs, etc.); nevertheless, they mainly focused on machine learning instead of deep learning approaches. Bellavista et al. [16] pointed out the necessity of shifting toward decentralized learning techniques and the need to construct suitable learning models for edge devices; however,they solemnified the network infrastructural perspective without referring to model optimization. Despite the benefits of working on edge devices, there are pivotal concerns about both communication safety and data adequacy among the participating entities, as analyzed by Han et al. in [14]. In this work, they address *trust* issues, regarding node cooperation in fully distributed environments, using blockchain-based methods; however, swarm nodes need more than enough computational power in order to meet both the high-energy and memory expectations of blockchain methodology.

Table 1. Reporting the most relative existing surveys.

Title	Year	Purpose
Distributed Machine Learning for Wireless Communication Networks: Techniques, Architectures, and Applications [15]	2021	it focuses on wireless networks
An Overview of Machine Learning within Embedded and Mobile Devices—Optimizations and Applications [1]	2021	it emphasizes machine learning for embedded devices
Demystifying Swarm Learning: A New Paradigm of Blockchain-based Decentralized Federated Learning [14]	2022	the main goal of this work was to tackle the imbalanced training dataset problem by leveraging a blockchain-based infrastructure
Decentralized learning in federated deployment environments: A system-level survey [16]	2021	it focuses on distributed FL

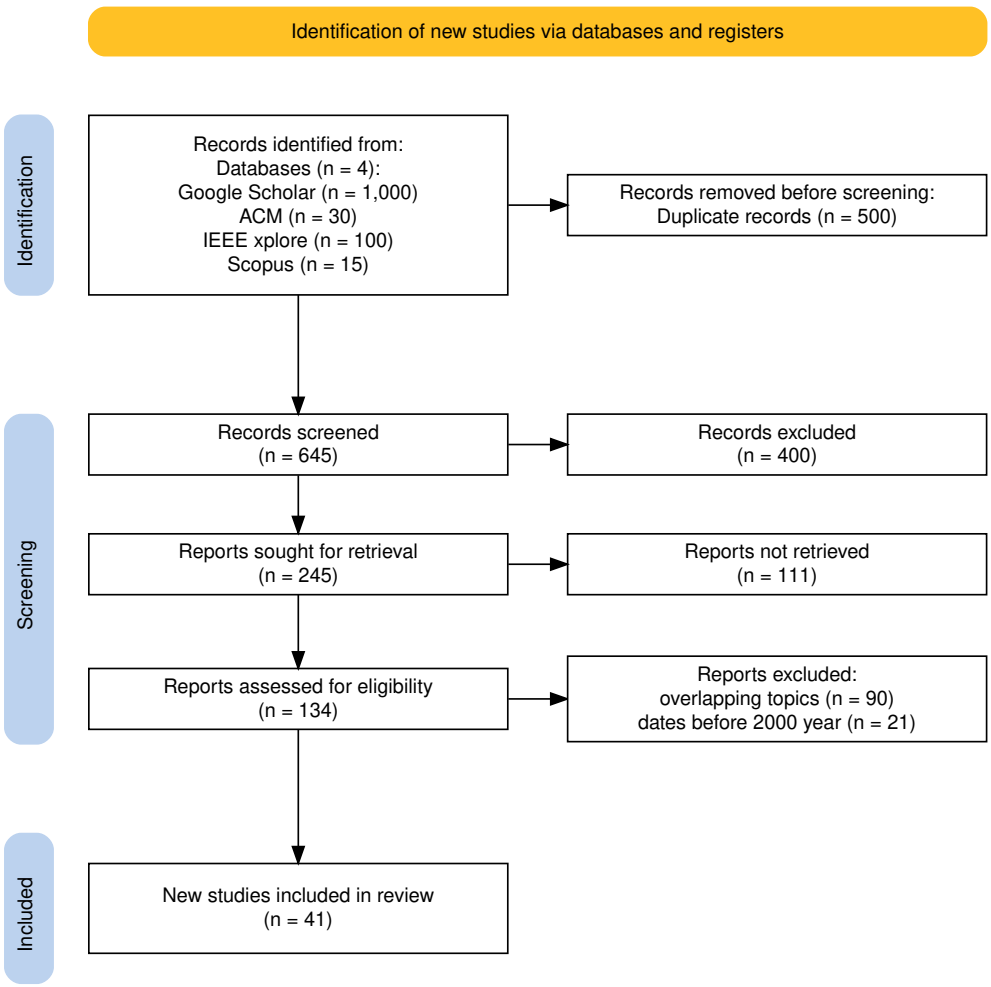

Figure 1. Screening process [17] of relevant academic works.

The present survey aimed to contribute the following:

- The current advancements of the amalgamation of three wide areas, TinyML, decentralized federated learning, and Swarm Learning, for which their amalgamation revolutionize both the viability and autonomy of resource-starving decentralized environments;
- The incorporation of new up-to-date entries, published in the recent literature, focusing on cutting-edge cross-device decentralized federated learning methodologies, e.g., on-device reformable TinyML, peer-to-peer communication among tiny devices, blockchain contributions to communication safety, etc.;
- The exploration of current challenges that have yet to be fully addressed in the deployment of deep neural networks on resource-starving devices;
- The presentation of challenges concerning the *ethical concerns* that this rapidly increasing technological era (e.g., Generative AI models) faces.

The organization of this paper is as follows: The introduction of decentralized federated learning and TinyML/DL is covered in Sections 2 and 3, respectively. The current progress of tiny federated learning is discussed in Section 4, followed by a brief introduction to Swarm Learning in Section 5. Lastly, the potential challenges in the realm of Tiny DFL are analyzed in Section 6.

2. The Rise of the Federated Learning Approach

The *federated learning* technique was first introduced by McMahan, from Google, in 2017 [8] and was the alternative of a privacy-preserving machine/deep learning paradigm. In the original form of FL, there are three basic steps to performing FL: *selection*, *configuration*, and *reporting*. First of all, the *selection* process takes place when we select the participating node devices. After this, the aggregation technique adopted by the server is declared and the server sends the model to all participants (*configuration* process). After that, the nodes collaboratively train the defined neural network model by sharing their gradients/weights but not their exact data with the server. Finally, in the *reporting* phase, the server is informed about the nodes' updates (about sending gradients/weights), and through a specific algorithm, namely FedAvg, it sends the renewed model to the participating nodes.

Every node runs its own *local* training procedure for some epochs, before the reporting stage, and then it sends its produced parameters, either periodically or after a specific number of local training rounds, to the server in order for the server to make the global aggregations. After that, the newly updated weights are sent back to the devices, which incorporate the newly updated weights and continue their local training. The global aggregations continue until the model reaches its optimal solution, or until it reaches *convergence* [11].

2.1. Problem Formulation

To explain further, let us assume that every node i produces a dataset D_i with $|D_i|$ data points. Each data point $(\mathbf{x}, y) \in D_i$ consists of a multi-dimensional feature vector $\mathbf{x} \in R^m$ and a label $y \in R$ space. We also assume that the $f(\mathbf{x}, y, \mathbf{w})$ declares the loss function, regarding the data point $(\mathbf{x}, y)$, based on the learning model parameter vector $\mathbf{w} \in R^m$. Then, the local loss function at node i is defined:

$$F_i(\mathbf{w}) = \frac{1}{|D_i|} \sum_{(\mathbf{x},y)\in D_i} f(\mathbf{x}, y, \mathbf{w}). \quad (1)$$

And the total loss function is then defined as the average loss of the aforementioned local losses as follows:

$$F(\mathbf{w}^*) = \frac{1}{|D|} \sum_{(\mathbf{x},y)\in D} f(\mathbf{x}, y, \mathbf{w}). \quad (2)$$

where the goal is to find the optimal learning parameters $\mathbf{w}^{\star}$ for F so that the total loss function is minimized.

$$\mathbf{w}^{\star} = argmin_{\mathbf{w} \in R^m} F(\mathbf{w}). \quad (3)$$

2.2. Taxonomy of FL Models

In federated learning, we have the following categories: scale of federation, data partitioning, and federated learning architecture.

2.2.1. Data Partitioning

In this case, we have three categories: *horizontal*, *vertical* and *transfer federated learning* [18]. Horizontal federated learning refers to the process of distributing the same resources to many node devices (parallelism). On the perspective of data, in horizontal FL (or sample-based FL), the feature space (a vector space that includes all the possible feature vectors from a population of a specific domain) of the training data is the same for all used samples but may differ in the sample space. In contrast, in vertical federated learning, the resources of a powerful device, e.g., a server, are bisected in order for more nodes to leverage its capability (concurrency). In vertical or feature-based FL, the datasets share the same sample ID space but are different in respect to the feature space. Furthermore, we also have the terminology *homogeneous or heterogeneous* federated learning, which stipulates whether there is harmonization or not, respectively, regarding device specifications and model architecture. For example, if all node devices have the same architecture and learn the same task, we have homogeneity. If either the task or the model differs, it is called heterogeneity. Horizontal FL tends to coincide with homogeneous FL, while vertical FL is commonly associated with heterogeneous FL. Nevertheless, in [19], Mori et al. addressed the challenge of heterogeneous data in a horizontal federated learning scenario. In FL, the workload accomplishment can realize, either from multiple cores of a single device (vertical setting) or from multiple devices, communication with each other (horizontal setting) [20]. The third category, namely *transfer* federated learning, combines data either with common feature, or a common sample space, with the aims of alleviating the problem of limited resources among the nodes and, hence, building an efficient neural model.

2.2.2. Scale of Federation

The federation scale can be classified into two types: *cross-device* or *cross-silo*. In the case of cross-device federation, nodes are typically mobile devices with imperceptible computational power, and their number can reach up to a scale of millions (there is a wide range of connectivity). Furthermore, in this case, node devices are more susceptible to possible attacks due to their lack of resources (less reliable). On the contrary, in cross-silo setting, nodes are organizations or companies, e.g., banks, hospitals, etc. (meaning that they do not lack computational power), but in this case, we have a small range of connected nodes (e.g., within a hundred). In this situation, the main goal is to retain privacy among the cooperative nodes, since they exchange sensitive personal data. Either in cross-device or cross-silo scaling, the training procedure may or may not involve a central server as a neutral party, and this depends on the type of the training algorithm and the type of the data it uses, the network topology architecture, and how trusted the communication is for the nodes to exchange model parameters among one another in order to assist the global model's gradient calculations.

2.2.3. Federated Learning Architectures

Centralized Federated Learning

Centralized federated learning[8] with the FedAvg algorithm was first introduced in order to address the problem of data privacy. The FedAvg algorithm is the reference point in the evolution of centralized federated learning algorithms, since it has been already proven that it converges and achieves a sub-optimal convergence range of O(1/T) when training the model using the *stochastic gradient decent* algorithm under settings where the convexity

of the loss function holds. After FedAvg, a lot of variants were introduced with remarkable convergence rates and high statistical accuracy levels, e.g., DFedSGD, FedProx, SCAFFOLD, VanillaSGD, etc. [21]. However, CFL was also a great solution motivating resource-scarce devices to run DL models on their own. The devices run their own local training for a few epochs and then send their parameters to the server for further calculations, which in turn forwards back the updated weights to the participating entities. However, even if the server mitigates the computational cost of running the collaborative learning, the communication overhead remains at high levels. In taking into consideration that in FL the bandwidth of uploading/downloading data is very small (about 1 MB/s) and we often deal with "asymmetric" networks [22], which probably means that not all devices have the same capabilities in the network and specifically the velocity of uploading or downloading differs from device to device, the continual communication among node devices and the server conducted in order for the model to be trained may cause cross-device *overflow* in communication links. Also, when the energy of these devices is also constrained, this data exchange may deplete all their energy. There are a lot works in the literature that justify the advancements of this technique to small and tiny devices, for instance, the *edge stochastic gradient descent (eSGD)* [11] algorithm for updating weights, based on the importance of parameter values. The eSGD algorithm takes into account a few significant gradients, which are further forwarded to the server in order to update the global model. The eSGD algorithm compares the loss values of two consecutive iterations of the algorithm, and if the present value is smaller than the previous one, it means that these gradients are important for the model training, and thus, these weights are kept. There are a lot works in the literature that justify the advancements of this technique to small and tiny devices, for instance, the *edge stochastic gradient descent (eSGD)* [11] algorithm for updating weights, based on the importance of parameter values. The eSGD algorithm takes into account a few significant gradients, which are further forwarded to the server in order to update the global model. The eSGD algorithm compares the loss values of two consecutive iterations of the algorithm, and if the present value is smaller than the previous one, it means that these gradients are important for the model training, and thus, these weights are kept.

Another *selective communication protocol* is introduced in [23]. Pan et al. present a centralized federated learning paradigm in which not all nodes participate in the learning process but only the nodes that achieve high rewards. The authors propose a Neural *Contextual Combinatorial Bandit (NCCB)* algorithm for rewarding nodes not only for their contribution, extracted from system *experience*, but also for their data cross-device similarity, while using the K-means clustering algorithm to fit nodes into groups. There is a lot of remarkable work in centralized federated learning [11]; however, the existence of a central server that takes over the orchestration of the training procedure is still considered a disadvantage in recent environments. Liu et al. [24], for instance, proposed a method of encrypted message transmission between the server and the participating nodes in order to enhance data security, with imperceptible client-side overhead for message encryption. However, the existence of a central coordinator remains a "bottleneck" in contemporary environments, due to retaining data sovereignty over the devices. In bearing in mind the work in [12], a malicious node may intrude the network and manage to reconstruct the produced-by-devices data that are sent to the server by knowing only their gradients. Also, in cases with cross-silo settings, the organization and storage of all the sensitive parameters, e.g., medical tests, may cause severe problems in the case of an attack to the server.

Decentralized Federated Learning (Purely Distributed)

Decentralized federated learning [18] is a form of federated learning in which there is no server to coordinate the communication among the cooperative nodes and thus the training procedure. Every participating device communicates with its one-hop neighbor node (peer-to-peer or device-to-device networks), and all nodes are usually connected, forming a symmetric topology (see Figure 2), e.g., a ring, d-ring, or grid network topology and more rarely, a fully connected mesh [10] network. It is of a great importance to

have symmetric topologies in DFL, meaning that the node resources are balanced, and communication among the nodes is uniform, since these factors impact both the scalability and fault-tolerance of the network directly. However, in [25], Li et al. investigated the case when non-symmetric topologies (e.g., star topologies in which the central node has a more crucial role than the rest of the connected nodes) are used and proposed the *push-sum* protocol in order to address this imbalance. The training procedure continues to be executed until the devices reach a *consensus* [26], meaning that the global model has converged, e.g., as described in over-the-air approaches [27–29]. This mechanism bridges the gap between data safety and communication overhead, since every device exchanges information and communicates only with its one-hop neighbor. However, due to a lack of memory storage, devices also lack complex algorithms that determine if a node is trusted or provide defense from a contingent attack of a malicious node, e.g., byzantine faults (adversarial learning [30]). The objectives in DFL are summarized in Figure 3.

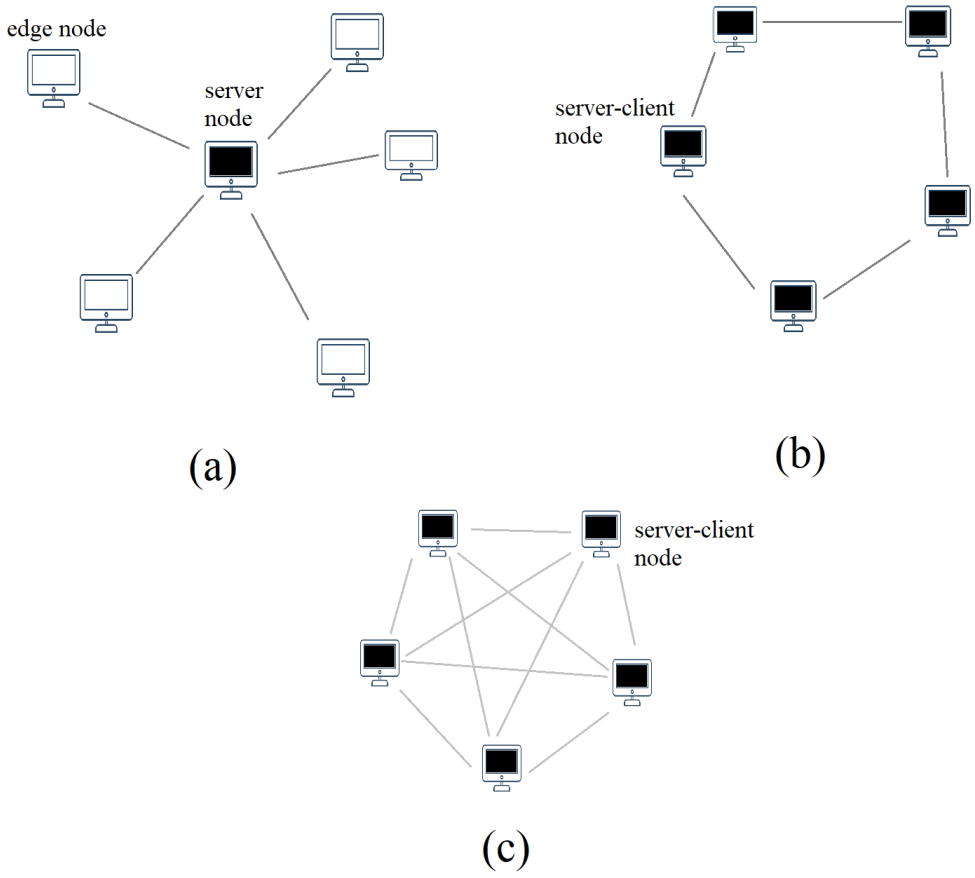

Figure 2. Different topologies in FL: (**a**) star-shaped topology, (**b**) ring-shaped topology, and (**c**) the topology of a mesh network.

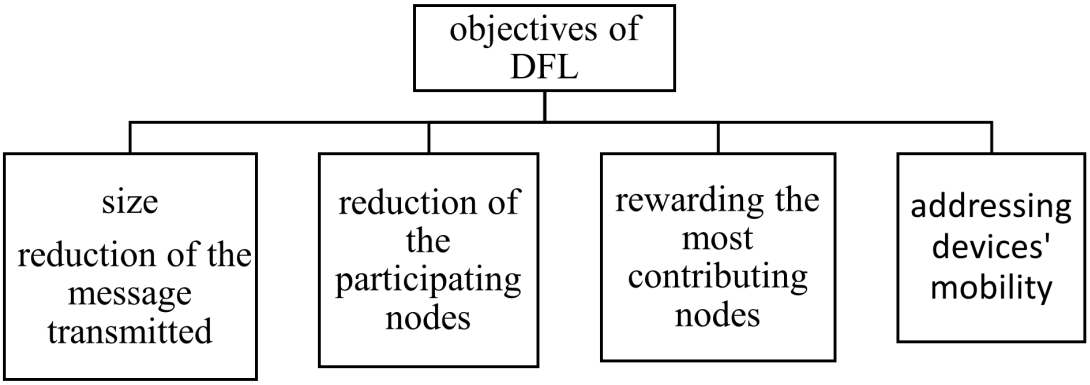

Figure 3. Objectives in DFL.

Hybrid FL

Although a server-less mechanism is ideal, especially in IoT networks, since device independence is preserved, it is not fully scalable for large-scale networks due to the unreliable connection of D2D communication [31]. Some recent works [32–35] tried to deploy a hybrid model, or else a *semi-decentralized* federated learning setting, in order to exploit advantages from both the aforementioned categories. In Semi-DFL, the connection of the entities with the coordinator server is configured with clustering algorithms, e.g., max-min d clustering, k-means, etc., which go beyond the idea of flat networks, and as a result, the central server communicates only with the nodes that represent the respective cluster (the clusterheads). For instance, Hosseinalipour et al. introduced a tree-shaped hierarchy of layers between devices on the edge and the server, namely, multi-stage hybrid federated learning *(MH-FL)* [36], and later, Lin et al. [31] advanced this idea by introducing two-timescale hybrid federated learning *(TT-HF)*, in which the model reaches consensus irregularly. To facilitate training, the network of devices is organized into smaller groups (clusters), with devices sending their model to their sub-clusterhead aperiodically, and the sub-clusterhead also intermittently broadcasting the updated local model to the server. It is worth highlighting that TT-HF converges at a rate of $O(1/t)$, which is similar to the centralized stochastic gradient decent (SGD) algorithm.

2.3. Delving into Decentralized Federated Learning (DFL)

In this section, we focus on purely distributed or otherwise decentralized federated learning (DFL) [26,37,38]. For all the works that follow, the major competitor is the efficient FedAVG [8] and its proven convergence rate. Unfortunately, in DFL, it is more difficult to reach convergence, either due to the heterogeneity of data distribution or the model deployed on the participating devices, which delays the learning process, or due to the difficulties that wireless network topologies cause in communication integrity. The main objectives in DFL, in order to become a viable learning scheme, as depicted in Figure 3, are the following:

- The direct communication among devices lessens the network latency; however, the throughput of the messages must be lessened too, in order to avoid network *flooding*.
- To address the resource limitations of edge devices, such as energy, one strategy is to minimize communication between nodes in the network, focusing only on those that contribute actively to the federation process.
- In real-time scenarios, we cannot guarantee that all cooperating devices follow the same specifications (e.g., flash memory, energy capacity, etc.) or equally contribute to the learning process. Thus, there is a need for stipulating methods that reinforce the lower-contribution nodes in order to ensure accuracy *uniformity*.
- Last but not least, wirelessly interconnected devices, used for, e.g., surveillance and wearability, are usually mobile devices that are not stably positioned in a specific place, impeding the configuration of a strong topology.

2.3.1. Communication Reduction

In order to achieve communication reduction, Zhou et al., in [39], proposed a heuristic (CEDFed) in which a one-bit signal is transmitted among selected neighbors in order to reduce the communication requirements of the network. Parameters (which are about to be sent) are firstly encoded to the node i in which they were trained, and this node i takes advantage of an encoding matrix Φ, is accessed by specific nodes belonging to a neighborhood M_i. Accordingly, node i, after receiving the one-bit message, decodes it using the *1BCS* algorithm. Authors also claim that if the Φ matrix follows a *Gaussian* distribution, the dimension d_i of the transmitted dense vector of node i is upper bounded ($d_i < n$), and the message is sparsified, then the decoded message has fewer chances to be corrupted. The algorithm was tested, using a linear regression task in a randomly generated decentralized topology, while the competitor was the same algorithm under ideal communication conditions, and the results verified that there are negligible losses in message conversion under the aforementioned assumptions. Furthermore, Decentralized Federated Averaging (DFedAvgM) [40] is based on the DSGD algorithm. In particular, DFedAvgM proposes a scheme in which participating neighbor nodes communicate with each other after multiple local updates, while DSGD requires communication among the neighbors after each local iteration. In taking into account that the communication cost is the main bottleneck in peer-to-peer communication schemes, DFedAvgM is introduced as more communication-efficient than DSGD, since fewer communication rounds are taking place. Moreover, a quantized version of the proposed algorithm has been introduced, alleviating communication flow and thus time. It is worth highlighting that authors have demonstrated theoretical proofs regarding both convex (for which the worst-case convergence rate coincides with the DSGD one) and non-convex cases, in which the Polyak–Lojasiewicz (PL) condition is used, and under this assumption, quantized DFedAvgM converges faster than the simple DFedAvgM, which mainly depends on both the network topology and the computational resources of the participating nodes. Regarding the experimental evaluation, a CNN deep neural network was trained in both image classification and language modeling tasks, executing the quantized DFedAvgM algorithm, and the results proved the sufficiency of this method, independently of the number of bits transmitted and the data distribution used (either in cases where IID or non-IID data). Finally, the algorithm was also tested for its privacy-preserving capabilities as it is exposed to a membership inference attack (MIA) either in ring-based or d-regular-based network infrastructures. The results verified the efficiency of the algorithm in facing membership attacks, while the lower the AUC, the safer the deployed model (the ideal value of the AUC was proven to be 0.5). Consequently, DFedAvgM demonstrates higher performance than FedAvg in terms of communication costs, paving the way for secure D2D large-scale implementations.

In distributed systems, the *gossip* protocol is an appealing choice for dealing with large-scale networks. In the machine/deep learning field, the *gossip learning* protocol technique was first introduced in [41], in which a portion of the participating nodes share the sum of the weights of the global model (GoSGD) in both a fully decentralized and asynchronous manner. In [42], a gossip-based protocol scheme, namely Combo, was proposed in conjunction with a *split* learning approach. To elaborate, in every participating node i, the model i is divided into a specific amount of pieces, in which the trainable parameters are distributed equally and they do not coincide. During training, the node i undertakes to aggregate only one segment of its own with the respective segment of k out of the total participant nodes. In order for the algorithm to reach convergence, it is suggested that the k value be less than the value of the total participants (workers). The gossip-based heuristic (Combo) was tested using a CNN model and the CIFAR-10 dataset [43] against both the conventional FedAvg algorithm and classic gossip approach (without model segmentation). The results showed that Combo outperforms the aforementioned baselines in regard to convergence speed (about 3× faster) with a negligible drop (from 1% to 2%) in validation accuracy.

Moreover, in [42], a decentralized federated learning scheme is proposed, in which the model, whose updates are propagated toward the nodes participating in training, is split and every worker transmits a part of the update that it is responsible for in a peer-to-peer way. Based on the gossip protocol, this approach decreases transmission bandwidth and acquired training time with a little trade-off regarding accuracy by splitting the model and using the gossip strategy in worker communication, while it also replicates the data from different non-overlapping nodes so that it can ensure that enough information is gathered for better convergence during aggregation.

Another algorithm in [44], namely *Soft-DSGD*, was proposed in order to alleviate communication unreliability regarding UDP protocol end-device use in order to communicate in peer-to-peer networks. Soft-DSGD updates model parameters during training using a part of the weights successfully received, while it also strengthens the weights originating from devices with reliable links (e.g., a coordinator's link), taking into consideration a *reliability matrix* to achieve brisker convergence (reaching *consensus*). In order to deal with the package transmission failures (which the UDP protocol is susceptible to due to the fact that packets are forwarded to all the possible receivers and there is no validation if the message reached its destination) and thus normalize the training procedure, every device replaces lost packet information with a part of its local parameters. Not only numerical results but also convergence analysis proofs (treating it as a convex optimization problem) have shown both similar behaviours and asymptotic convergence rates between *banilla decentralized SGD* (in its ideal communication conditions) [21] and Soft-DSGD.

Prototype learning is also an interesting way of alleviating overflow in communication links. A *prototype* is a data value that represents the values of a class. So, prototype learning in decentralized networks is a technique that exploits this feature in order to further reduce the communication load by sending only prototype representations of the gradients and not the exact ones in order for the nodes to perform training. A prototype, namely *DeProFL*, was introduced by Li et al. [45] in order to deal with both the heterogeneity, caused by the time-dependable shifting (dynamic) communication infrastructure (a feature that characterizes wireless networks), and communication overhead. In every round, a different network topology is randomly defined, while every node performs some local updates and then propagates only the prototypes to its neighbors. Practically, according to the authors, nodes send the vector generated after the first layer, which is responsible for understanding the given data features and not the vector, produced as a prediction of the supervised model (as usual) across their neighbors. Regarding data heterogeneity, authors have set the degree of heterogeneity to 0.1 while they conducted experiments with the *Dirichlet Non-IID* dataset. As far as model heterogeneity is concerned, all nodes have different deep learning model structures, making centralized (SOTA) FL approaches like Ditto or FedAVG fail to keep up with these requirements. Plenty of experiments have been conducted, using well-known datasets (Mnist [46], FMnist [47], Cifar-10 and Cifar-100 [43], Tiny-Imagenet [48]) and both DeProFL and centralized SOTA algorithms, and the proposed method achieves comparable results regarding accuracy (accuracy degradation approximately from 0.4% to 5%), while it was proved to converge faster under non-convex loss function assumptions.

After McMahan et al.'s work [8], Lalitha et al. in [49] introduced a federated learning approach based on collaborative learning among one-hop connected devices (fully decentralized federated learning—DFL), taking advantage of their own localized personal data. The proposed scheme is completely server-aware, since no centralized control is required in order for the training procedure to be performed. While the collaboratively learned model functions as a Bayesian-like model, the authors explored the potential for extending it to deep neural networks (DNNs). The efficiency of the aforementioned algorithm was tested in [38], in which the results of a linear regression task, being performed on both Bayesian-like and DNNs models, showed that DFL provides better results compared to the cases in which nodes learn only from their own data.

In [50], the authors provide up-to-date information about the evolution of the decentralized federated learning algorithm (DSGD), taking stochastic gradient descent (SGD)

for granted in the proposed implementations. They provide theoretical analysis and convergence rates in both convex and non-convex [51], fashion [52], taking network topology adaptivity and data heterogeneity into consideration. For example, they offer an improved version of theoretical proofs regarding the cases of parameters-abundant models, Local SGD [52] and cooperative SGD.

It is worth highlighting that DSGD paves the way for large-scale machine learning implementations in a D2D topology fashion. Plenty of variations, presented in this section, have been introduced in order to cope with the challenges that a peer-to-peer topology brings, e.g., communication cost, convergence speed (reaching consensus), etc. However, DSGD has been proved to converge to the optimal level when assumptions regarding convexity are applied.

Roy et al. [53] proposed a server-less dynamic peer-to-peer environment, namely *BrainTorrent* (see Table 2 for more frameworks) with the aim of alleviating data privacy concerns while decentralized topology approaches are used on training deep neural networks with medical images (which are very sensitive personal data). It is one of the first frameworks to implement server-less approaches in collaborative learning. The results of the experimental evaluation showed that this server-less peer-to-peer framework with a range of 5 to 7 participating clients performs about 1–2% better than centralized federated learning approaches, while it reaches almost the same accuracy (86.3%) as the *pooled* model (86.6%).

In [37], the authors were the first to implement the DSGD algorithm to edge networks. They used a D2D network topology in order for the nodes to collaboratively train a shared ML model. In order to meet the transmission standards of the network, each device prunes the trainable model to align with the device's lowest data rate (limited bandwidth). Experiments were conducted, using both digital (D-DSGD) and analog (A-DSGD) schemes. In the latter one, Over-The-Air (OTA) SGD-based approaches are used like the one mentioned in [54], which seemed to have a better performance than DSGD only in cases where a conventional star-like topology is used, given the Fashion-Mnist dataset as a training set in a classification task.

In their work [55], Liu et al. present a DFL framework that involves both local updates and inter-node communication. They also propose a compressed algorithm, C-DFL, to address communication overhead and facilitate cross-device convergence. Nodes, after completing local training, send information to all the nodes they are connected to and hence compute the local average of the received data. This process is based on the C-SGD algorithm, which adopts *compute-then-communicate* (or else computes the local average and then broadcasts it to the whole network) rather than *communicate-and-compute* that the D-SGD algorithm employs. Transmitted information in cross-device communication via the C-DFL algorithm is compressed either by taking advantage of sparsification (using random k values out of the total ones or k values with the higher-magnitude values) methods or randomized gossip protocols (like the one reported by Onoszko et al. [56] for addressing non-IID data heterogeneity in a random gossip communication protocol), achieving a linear convergence rate, indicating strong theoretical proofs without taking convexity for granted.

As described in the Hybrid FL section, the hierarchical network topology is a step forward in the transition of the decentralization of the learning process in FL. However, in the aforementioned section (see the Hybrid FL section), there is a central coordinator. Here, there are also works that take advantage of non-flat networks in a fully decentralized spectrum.

Fragkou et al. [57] introduced a two-tier hierarchical fully distributed learning scheme, with the aim of reducing communication rounds among the entities, using similarity-based criteria. The max-min d clustering algorithm is used in order to create clusters and carry out betweenness centrality measures so that they place the clusterhead of every cluster at the center, and hence, it is easily accessible by all cluster members. In calculating small summaries (using Fourier transformation), the process identifies similarities in data from nodes in the same cluster. Extracting the Fourier coefficients requires a computational complexity of $O(mlog_2m)$, where m denotes the size of the data. Since the Fourier coefficients

sent to clusterheads are much smaller than the exchanged weights and can be packed in *beacon* messages, we do not consider it as communication overhead. Based on the presence of "diverse data", the clusterhead chooses which cluster members to incorporate into the federation. In comparison to the conventional all-to-all method, the proposed heuristic reduces communication overhead by up to 42%, with a preserved accuracy of up to 96%. The proposed method significantly reduces the computational complexity of the algorithm, making it feasible for implementation in decentralized environments.

2.3.2. Privacy-Preserving Methodologies

Although FL is a method introduced to securely enable devices to take advantage of the collaborative learning effect without divulging their data, attackers (adversarial networks) can still find ways to take control of the device's data. The methods used in order to ensure the privacy in the federated learning process are categorized as follows: *Differential Privacy (DP) methods, secure multi-party computation, and Homomorphic Encryption* [15].

In DP methods, noise is added to the training data whilst the algorithm remains capable of learning and hence makes precise predictions. The purpose is to impede the malicious nodes from distinguishing whether a user's data point or dataset was incorporated into the training phase. For instance, Kalra et al. [58] introduced a method for secure cross-silo decentralized federated learning. Each client (which represents an institution, like finance or healthcare) trains two models concurrently, the local one and the exchangeable one (ProxyFL model). The ProxyFL model is trained using *Differential Privacy* guarantees (DP-SGD, based on Gaussian mechanisms) and deployed to decentralized environments, while these two conditions justify the reason that ProxyFL is diversified by conventional *Deep Mutual Learning (DML)* [59]. While training, each client communicates solely with its peers, exchanging one proxy model per round. The method was tested on the *Camelyon-17* dataset, containing medical images (images of lymph nodes) from different medical centers (the distribution of which was skewed in some cases for the approach to be tested on non-IID data) and using CNN networks (Res-Net), while some of the competitors were centralized FedAvg, AvgPush, CWT, and FML. ProxyFL outperformed every other aforementioned approach and continued to upgrade its performance until the end of the training.

Although there is a lot of successful work regarding securing privacy in DFL, this case will be further analyzed in Section 5.

2.3.3. Fairness

It is worth highlighting that not all nodes participating in the global training have the same resources (asymmetry), and hence, the produced outcome cannot be of the same importance regarding all nodes. In their paper [60], Li et al. promoted fairness in decentralized federated machine learning by maintaining uniformity in accuracy distribution across cooperating devices. The algorithm called q-FFL is an alternative algorithm to FedAvg that incorporates a parameter q in the loss function to achieve balanced weight updates. For example, the higher the loss value, the larger the weights, with the aim of rewarding the most contributing nodes and thus reach a consensus. Compared to FedAvg, q-FFL reduces approximately 45% of the average accuracy fluctuations among devices, while q-FedAvg performs better regarding accuracy metrics on different types of data (synthetic non-IID, synthetic IID, hybrid–synthetic). The authors also ran experiments with parameterized FedSGD (q-FedSGD) and concluded that its performance behaviour was proven to be superior to q-FedAvg when dealing with heterogeneous data (non-IID) and is even better, unlike the FedSGD variant that tunes the step size of the algorithm.

Table 2. Frameworks used in federated learning.

Framework/Platform	Architecture	Purpose	Capabilities
TensorFlow Federated (by Google)	CFL	smart-grid applications	open source
FATE	CFL	finance	open source
BrainTorrent [53]	DFL	healthcare application	not open source
Scatterbrained [11]	DFL	academic usage	open source, developer-friendly API
FedML [61]	CFL	academic usage, suitable for IoT networks, benchmarking	open source, scalability, reliability
FedStellar [62]	CFL, Semi-Decentralized FL, DFL	addresses heterogeneous FL-based problems	easy customization, deployment of complex network topologies, secure communication, enough storage for FL models
FL-SEC [11]	DFL, Blockchain	orientation of Artificial Internet of Things (AIoT)	not open source, defense in poisoning attacks, communication reduction

3. On the Prospect of TinyML/DL

3.1. Problem Formulation

TinyML/DL is a subset of machine/deep learning that is specified for deployment in resource-limited devices (lack of memory capacity, energy, etc.) such as MCUs, IoT end-devices, etc. (see Table 3). In the literature (see Section 3.2.1), the only way of implementing deep learning algorithms on the aforementioned devices is to pre-train the ML/DL model on a server, which requires both resources and a plethora of data, and then the ML/DL model has to be compiled through suitable software, e.g., micro TVM (see Table 4), to run on MCUs. After that, the model runs on devices without updating its weights; it only runs inference tasks. However, this creates static models, which are unable to adapt to new scenarios, e.g., different distributions of streaming data—Independent and Identically Distributed (non-IID) data [63]—triggering severe problems, e.g., concept drift [64–66], which decreases neural network accuracy. Nonetheless, this inference is efficient. It forms static models that are unable to adapt to new tasks. The main challenge regarding TinyML is to manage to run training with resource-starving devices, without sacrificing performance, e.g., prediction accuracy.

3.2. Taxonomy of TinyML/DL Models

3.2.1. Static Model Inference

Low-powered devices, such as sensors and MCUs, have been utilized in the literature to perform efficient machine learning tasks. They have wide-ranging applications in areas like everyday life, security, surveillance, industrial monitoring, smart agriculture, etc. [1]. The field of on-edge data processing using ML algorithms (TinyML) is a new and emerging area. With its fast and real-time performance, it also enhances devices' capabilities and privacy and reduces communication costs between them and a server. Of course, conventional ML techniques cannot be deployed on the aforementioned devices because these devices lack the computational and memory resources required for an ML/DL algorithm to run efficiently. Consequently, multiple methods are recommended to optimize the inference phase of an ML model by developing compact models, as outlined in the following sections.

Pruning

In this section, pruning aims to reduce the memory requirements of the model by reducing its trainable parameters—hyperparameters. Thus, pruning is bisected into two main categories: *static*, in which the model completes training and is then pruned,

and *dynamic—dynamic sparsity during training*—in which the model is both pruned and regrown iteratively (the latter one will be further analyzed in Section 3.2.2.

Despite the phase in which the pruning is conducted (static or dynamic), we can define other categorizations of pruning as follows [67]: structured pruning (like channel pruning), which can easily be used in conjunction with any other optimization method, and unstructured pruning, which provides the network with the largest size reduction possible but makes it more vulnerable to performance decrease [68]. According to the work in [69], there is a smaller sub-network whose randomly initialized parameters make it capable of replacing the initial fully connected network and therefore undergoes effective training while the redundant information of the network can be ignored. This technique is called *Lottery Ticket Hypothesis*, and the chosen sub-networks are called *winning tickets*. This pruning technique effectively reduces the trainable parameters of feedforward neural networks (such as MLPs and CNNs) by up to 90%. As a result, memory requirements are reduced, enabling deployment on resource-limited devices for inference tasks without any loss in accuracy. In fact, in some cases, it can even improve the initial results. Similarly, the technique of *Depth Pruning*, described in [70], involves discarding layers after training to create a sub-network. An auxiliary network is then used as the head of the pruned network, resulting in a marginal decrease in accuracy for an already pre-trained model.

Of course, there are plenty of pruning techniques described in the literature, like the one presented in [71], which is based on the L1 regularization technique and a light-weighted network.

Quantization

The quantization technique refers to the process of reducing the precision of the model parameters, e.g., weights or gradients, from floating-point (e.g., 32-bit or 64-bit) to lower-bit-width representations (e.g., 8-bit integers). For instance, Li et al. [72] proposed a novel compression method; they combined both pruning and quantization methods and achieved a conversion from the 64-bit floating-point representation to almost 8 bits, while in [73], a 2-bit precision representation was proposed, when running a person detection task, losing approximately only 3% of its accuracy. A new software accelerator library, running on Arm Cortex – M processors, in which kernels are implemented in a way that supports both 8-bit and 16-bit data precision, is presented in [3] and used in [74] in order to smooth out the differences among data representations, which is a hard task for a CPU to run. An interesting quantization pipeline is proposed in [75], in which the weights are saved to an external L3 memory, and in every epoch, only the weights being used are downloaded to an L2 cache with a capacity of 512 KB.

Neural Architecture Search—NAS

NAS summarizes a category of techniques that aim to automatically find better deep learning models for each case [76] regarding the available resources. It is possible to find not only the type of model needed, e.g., a convolutional neural network in the case of image processing, but also the number of filters, parameters, activation functions, etc.

Knowledge Distillation—KD

The approach known as knowledge distillation was first introduced by Buciluǎ et al. [77] and further generalized by Ba et al. [78,79], and it is a method that aims to compress neural network models. Knowledge distillation involves training a smaller neural network, known as the “student”, to imitate a larger network, the “teacher”, by minimizing a loss function shared by both models. Despite the large model being trained already, the smaller one can utilize the larger one’s knowledge to learn raw data online, correcting itself with a loss function when it makes misleading predictions. This approach allows for the real-time training of small and energy-efficient models [80], but it necessitates the student model to depend on a pre-trained model, referred to as the teacher.

Conventional Transfer Learning

The biggest challenge in static TinyML is adapting the model to a new task. For example, if a model is well trained on recognizing images of cats and suddenly captures an image of a dog, it will probably misclassify the dog as a cat, since it does not have much knowledge to understand patterns out of the learned category (concept drift [66]). In order to address the problem, techniques such as *transfer learning (TL)* are introduced. The TL technique uses a pre-trained model on a large dataset, e.g., Imagenet [81], and fine-tunes some of its last layers while freezing the earlier ones, when deployed to a tiny device, with the aim of learning new tasks. However, fine-tuning the last layers (mainly the last fully connected layers since the cost of retraining more layers is prohibitive in tiny devices) may cause imperceptible improvement or no improvement at all, since in this way, we affect only the weights of the classifier and we do not enter the main core of the network. For instance, inspired by the way a convolution neural network (CNN) works, in which every layer has a specific task to learn, retraining only the last fully connected layer will probably be ineffective, since we modify the results of the training procedure and not the procedure itself in order to embody the weights of the new task before the classification outcome. There are works, like [82], that try to retrain more than the last layer but it has a trade-off regarding energy consumption. We further describe transfer learning techniques in Section 3.2.2.

Inference-Based Applications

Inference-based TinyML/DL has paved the way for ML/DL applications in healthcare, industry, everyday life, etc. For instance, Khaled et al. designed a prototype system to deploy ML inference tasks on microcontroller-powered edge devices for predicting blood pressure-related metrics [83], while T'Jonck et al. developed a real-time application on low-powered devices to support nurses' work with elderly individuals [84]. Regarding industry, an efficient TinyML model [85] is also used in industries, in order to detect anomalies through IoT devices related to production process. Furthermore, the use of TinyML models in detecting pavement anomalies through intelligent vehicles [86], gas leakage [87], coughs [88], falls [89] and smart agriculture [90] are explored. There are also hardware-based implementations, like the one in [91], in which an ultra-low-power IoT monitoring device is presented, showcasing the integration of Bluetooth and NFC connectivity for monitoring asset activity and so on.

3.2.2. On-Device Real-Time Learning (Reformable TinyML)

The majority of the existing literature focuses on TinyML solutions that are based on offline settings and only support neural network inference on low-powered and ultra-low-powered devices like MCUs (see Table 3). On the contrary, the training phase of a network is very costly in terms of both energy and storage. In this section, we introduce methods/techniques that pave the way for online on-device training.

Continual (or lifelong) learning (CL), unlike conventional transfer learning, allows for real-time online training using raw data. In this way, the model can learn new tasks without forgetting the knowledge gained from the original task. In the literature, CL methodologies are divided into three main categories [92]: *architecture-*, *regularization-*, and *rehearsal-*based methodologies [19]. Architecture-based approaches adapt layers and activation functions to prevent forgetting, depending on the deployed device, while regularization-based approaches reinforce existing knowledge through penalties in the loss function or (*knowledge distillation*) [93]. With the term *rehearsal or native rehearsal* [94–96], we refer to a learning technique in which for every training batch, a randomly chosen subset of training data is saved to an external storage and re-forwarded for training, replacing another random subset of the incoming batch patterns, mainly aiming at handling *catastrophic forgetting* (a phenomenon in which acquired knowledge tends to vanish, due to repeatedly learning new incoming data). Despite being the most suitable solution for combating catastrophic forgetting, this method raises data privacy concerns as sensitive raw data must be stored externally from the edge device. The key is to identify the optimal solution

by considering the memory requirements and the specific data that need to be stored for accurate outcomes. We can say that there is a trade-off between the precision of saved information and non-harmful forgetting, known as the *stability/plasticity dilemma* [7]. Thus, the sub-category of the rehearsal-based approaches which alleviates this trade-off is the one called *latent replay* [97]. *Latent replay*-based approaches store activation outputs of past data at some random intermediate layer, instead of the data themselves, reducing the storage requirements compared to those of conventional rehearsal-based approaches. One of the first latent replay- and gradient-based approaches, designed for MCUs in the literature, is the one described in [98]. Combining fully 32-bit precision continual learning, the multi-core parallelization capability of a 22nn Risc-v-based microprocessor, namely *VEGA*, and a quantization strategy, Ravaglia et al. provided a platform (QLR-CL) that outperformed a low-power STM32 L4 microcontroller, and specifically, it is 65x faster and 37x more energy-efficient than the aforementioned microcontroller. Additionally, experiments were carried out using both 8-bit and 7-bit quantized memory, resulting in a small decrease of at least 0.26% and 5% in accuracy, respectively, compared to the 32-bit implementations. Generally, tested on Core50 dataset, this method achieves up to 77% accuracy, while it reduces the memory footprint (about 64 MB), paving the way for more online adaptive strategies for TinyML to emerge.

Hinton et al., in [99], questioned the efficacy of the backpropagation algorithm on online on-device training in MCUs due to the fact that there is a need of gradient computation, which further means that both enough computation power and memory storage are required. Instead, they proposed the Forward-Forward (FF) algorithm, in which the forward pass and the backward pass procedures are replaced by two forward ones that contain dissenting type of data (positive and negative ones, respectively). The goal is to either increase (FF on positive data) or decrease (FF on negative data) the sum of the squares of the activities in every layer, namely *goodness*, and based on this metric, to reconfigure the weights of the aforementioned layers. In [100], Vita et al. present an improved variation of FF, called μ-FF, with the aim of further reducing the memory and computation requirements of the above heuristic by separating the method into two *blocks*: the first one functions as a feature extractor, for the better understanding of non-linear data, while in the second phase of the algorithm, the training procedure is executed, using Mean Square Error (MSE) as the loss function along with a ridge regularization term. Experiments were conducting in an STM32 MCU, using the X-cube-AI tool for quantization and the Fashion-Mnist dataset, and it was shown that the MCU performance when running μ-FF online on-device is comparable with the corresponding one when the backpropagation algorithm is deployed in the same MCU in an offline setting. Although this heuristic is an attractive means for resource-starving devices, it has a low-speed convergence rate, and in general, its performance is inferior to the conventional back-propagation algorithm.

However, dynamic pruning ([67]), as firstly introduced in Section 3.2.1, can enhance online training as it is based on the "iterate and regrow" technique. Elements (weights or neurons) are dynamically removed and re-added in order for the model to keep its composition (otherwise, it is going to progressively lose all of its connections after a few epochs). The main idea behind this method is that in a fully connected network, the connections of neurons that are going to be removed in a layer are close to zero in weight, and hence, they are of less importance. For example, Mocanu et al. [101], inspired by network science, proposed a scheme of random close-to-zero weight connection removal, and in following a preferential attachment, the retrieval of the connection completes after every epoch, following a power law distribution. The basic idea using the power law distribution is to create scale-free networks, which are constructed to reinforce the high-influence connections/nodes in the network and hence retain the high performance of the network. Furthermore, the works outlined in [102,103] take this idea a step further by sparsifying a fully connected neural network, e.g., an MLP, dynamically either by creating an original scale-free network or modifying the number of removable connections following

dynamic functions during training, focusing on lessening the training time of the network while achieving better accuracy than the initial network in some cases.

In this section, the papers that follow are based on the fact that the knowledge being transmitted, or the weights of a pre-trained network, is information that contributes to the model generalization [104], but the models have to be able to adapt to new circumstances in order to retain high performance. For instance, in the work presented in [105], a transfer learning (TL) kind of scheme is proposed. In contrast with conventional TL, TinyOL does not retrain the last fully connected layerm but it functions as an extra layer connected to the last layer of the already pre-trained model. The weights in TinyOL are adapted to new data through online stochastic gradient descent (SGD), while it runs in RAM, with pre-trained weights frozen once TinyOL starts being fine-tuned. Furthermore, [106] presents a scheme that combines transfer learning with K-nearest neighbors, enabling training on embedded systems and IoT units using minimal data. In this way, networks can evolve over time, enhancing their performance by implementing independent local training.

Table 3. Devices/boards used for the experiments.

Devices/Boards	Power	Instruction Set	SRAM	Flash Memory	CPU Clock
Raspberry Pi family ([106–108]) Raspberry Pi 3B+	Low	ARM (Cortex-A53)	1 GB SDRAM	–	1.4 GHz
Raspberry Pi 4B	Low	ARM (Cortex-A72)	256 KB	–	1.5 GHz
Raspberry Pi Pico	Ultra low	ARM (Dual core Cortex-M0+)	264 KB	2 MB	133 MHz
Arduino family [74,108,109] Arduino Nano 33 BLE Sense (nrF52840 SoC)	Ultra low	ARM (Cortex-M4)	256 KB	1 MB	64 MHz
Arduino Portenta	Low	ARM (Cortex-M7−M4)	8 MB SDRAM	16 MB	240–480 MHz
STM Microcontrollers (MCU) [70,106,110] STM32F7 board	Ultra low *(high-performance MCU)*	ARM (Cortex-M7)	512 KB	2 MB	216 MHz
STM32H743	Ultra low *(high-performance MCU)*	ARM (Cortex-M7)	1 MB	1–2 MB	480 MHz
STM32F401	Ultra low *(general-purpose MCU)*	ARM (Cortex-M4)	96 KB	128 KB– 512 KB	84 MHz
STM NUCLEO **L4**96ZG	Ultra low *(general-purpose MCU)*	ARM (Cortex-M4)	320 KB	1 MB	80 MHz
STM NUCLEO **F7**67ZI	Ultra low *(high-performance MCU)*	ARM (Cortex-M7)	512 KB	2 MB	216 MHz
Adafruit Feather Family [108,111] Bluefruit Sense board (nrF52840 SoC)	Ultra low	ARM (Cortex-M4F)	256 KB	1 MB	64 MHz
M4 express	Ultra low	ARM (Cortex -M7)	192 KB	2 MB	120 MHz
Microprocessors GAP8 [98,112]	Parallel ultra-low-power processing platform (PULP)	RISC-V (FC)	80 KB+ 8 MB SDRAM	512 KB	250 MHz
VEGA [98] (22 nm technology)	Parallel ultra-low-power processing platform (PULP)	RISC-V (FC)	L2 (interleaved) 1.5 MB + 64 KB	64 MB	250 MHz
Mr.Wolf [113] (40 nm LP CMOS technology)	Parallel ultra-low-power processing platform (PULP)	RISC-V (RVC32IMF)	latch-based memory instead of SRAM (25 Gbit/s at 100 MHz)	–	32 KHz– 450 MHz

Table 4. Most-used tools for implementing TinyML.

Tool	Capabilities	Where to Apply
TensorFlow Lite macro [114]	The most widespread method (inference library) for deploying ML models to resource-limited machines. The TensorFlow pipeline consists of a classic TensorFlow model that is converted into a compressed flat buffer using the TensorFlow Lite Converter. After that, a file with a .tflite extension is created, which can efficiently carry out an inference task on the aforementioned devices.	32-bit platforms (e.g., Arduino nano 33 BLE, STM32F746 Discovery etc.)
uTensor [115]	A free embedded learning environment, supporting neural network training using Keras. uTensor produces c++ code from the trained model in order to fit edge devices.	Mbed, K64, ST boards
Edge Impulse [115]	A cloud service for deploying ML models on edge devices.	edge devices (e.g., smartphones)
NanoEdge AI Studio [115]	A software that tests a library's performance so that it can be cognizant of this library being the most suitable one, according to the learning process' needs.	STM32 Nucleo−32, Arduino Nano 33 board
Pytorch Mobile [115]	It is subjected to Pytorch software, and it advocates for both model training and deployment on edge devices.	smartphones (e.g., Android, iOS)
Embedded Learning Library [115]	A library suitable for embedded learning, developed by Microsoft. There is no need for cloud access.	Raspberry Pi, Arduino
STM32Cube.AI [115]	An optimization software that reinforces ML/DL tasks to be deployed in microcontrollers (MCUs).	STM32 ARM Cortex−M boards
μTVM (MicroTVM) [20,115]	Is an evolution of *tensor virtual machines* (TVMs) for deploying models on MCUs. This framework takes an already trained model and converts it so as it can be applicable to different hardware settings.	MCUs
CMSIS−NN [3]	Optimization library that is compatible with ARM processors.	ARM Cortex−M processors
CMix-NN [116]	A mixed low-precision CNN library for memory-constrained edge devices.	ARM Cortex−M processors
Runes [20]	An ML package that provides containers to encapsulate and deploy edge ML pipelines and applications.	edge devices
TinyCNN [117]	A framework introduced for accelerating CNNs in FPGAs.	FPGAs
edX MOOC [118]	Educational platform created by the collaboration of academia (Harvard University) and industry (Google) with the aim of encouraging researchers to remotely develop complete applications using application-oriented instructions through TinyML and to find solutions regarding the fields of data gathering for application deployment.	all new ML/DL enthusiasts

4. Current Advancements of FL and TinyML for Edge Devices

The first work implementing TinyML training in an exact resource-constrained MCU in a federated learning setting was by Kopparapu et al. [119]. The authors took advantage of the benefits that the transfer learning (see the Conventional Transfer Learning section) technique offers, by pre-training a model for a computationally efficient system, e.g., a server, and then deploying the model to the specific device. Then, fine-tuning only the last fully connected layer, with raw data, completes the training process, enabling a kind of on-device training in devices with less than 1 MB of memory. In this work, the global averaging of the model parameters took place in the server (centralized federated learning).

As it is elaborately described in Section 5, blockchain techniques are preferred in order for data privacy and secure communication to be ensured. However, the cost of implementing these kinds of techniques is high and when implemented in real-time scenarios, they tend to be prohibitive due to the fact that convergence is delayed. In order to address this drawback, Zheng et al. [120] proposed a semi-decentralized blockchain-based FL scheme that accelerates model convergence, using resource allocation for balancing computation/communication time, energy consumption, etc.

Trying to alleviating the problem of memory constrains in edge devices, Huang et al. [121] introduced a distributed framework for efficient on-device pruning, namely TinyFed. This framework generates unbiased models that are efficient, capable of local

training, and adaptable to various task scenarios. The experimental evaluation showed that FedTiny outperforms conventional small sparse models regarding not only accuracy levels but also memory footprint, aiming to take the decentralization of TinyML devices a step further in future research.

5. A Brief Introduction to Swarm Learning (SL)

The *Swarm Learning* [14] methodology is a decentralized federated learning paradigm that advocates peer-to-peer collaborative learning by combining edge computing methodologies and blockchain technology. It offers strong guarantees regarding data privacy and secure communication among the nodes, using Blockchain-empowered techniques [122], and hence, it is an appealing means in decentralized networks.

For instance, Ma et al. [123] presented a consensus lightweight blockchain protocol, namely TORR, which lessens the latency that conventional blockchain-empowered FL may incur due to its functionality. TORR provides latency reduction since it "categorizes" the users as reliable (if they are active and interact with other nodes) or not which means that the former devices are assigned a larger probability, and hence, they are more likely to be randomly chosen to participate in the collaborative learning. Therefore, "straggling" nodes affect the training procedure less. In order to protect the system from a *sybil* attack (in which a node creates non-existing nodes in order to persuade the network that it is interacting with many nodes, hence being reliable, or when a model is asked to be retrieved from a node via hashing decoding and then the "malicious" node returns an incorrect model), each node verifies the rest of the nodes in the blockchain. Whenever a node is delayed in answering, the node is excluded of the process as a suspected malicious node.

A typical attack in decentralized networks is the Byzantine fault, where malicious nodes aim to infiltrate the network to manipulate or corrupt exchangeable information. Ghanem et al. [124] presented a decentralized framework that consists of both trainer and validator nodes to deal with the Byzantine fault at the level of *trust*. A validator node receives the updates from its trainers, amalgamates the gradients, and then computes the validation score. The better the score, the more trustworthy the validator node. Another proposed consensus protocol, emphasizing fault tolerance against Byzantine attacks, was described by Wang et al. [125]. In the proposed algorithm, the nodes are considered to be *trusted* if their accuracy is the same or higher than a specific accuracy threshold, based on both the number of participant nodes and a specific ratio.

In Ref. [126], Li et al. presented a mechanism called Blade-FL based on the PoW consensus protocol in order to succeed secure decentralized federated learning without the need of a central aggregator. Furthermore, they proved that the loss function bound of the global learned model is characterized as convex and depends on the number of the total training rounds. Finally, they analyzed the effect of the *lazy* nodes (clients) on the general learning process.

In Ref. [127], Wang et al. dealt with the privacy concerns of conventional FL by providing a blockchain-based mechanism with differential privacy guarantees (adding Laplace noise). Either the Proof-of-Stake mechanism was used or a block was received from another client (miner) in order to reach a consensus. Finally, the miners with similar gradient values create clusters and build trust among them while they cooperate to learn their cluster's global model. It is also worth highlighting that both quantization and pruning techniques are used in order to alleviate communication among edge devices.

There are also published works regarding blockchain architecture implemented in industry 4.0 [128] and in IoT networks [129–131]. However, SL is a computationally expensive methodology, making its real-time implementation an unfeasible task. Blockchain-based practices in DFL incur high latency and thus more time to reach consensus, compared to centralized federated learning. In real-time applications, there is not much time for checking for malicious nodes/data in the very secure way blockchain-based techniques entail. Moreover, ML models demand enough storage capacity, something that blockchain nodes are not capable of. This is why most common works store the model in a decentralized

manneer (eg. IPFS) and save only hashed information to the blockchain, but this demands extra effort in order to guarantee the reliability of the saved model.

6. Exploring Challenges in Tiny Decentralized Federated Learning Environments

6.1. Scalability

The scalability of communication reliability in ad hoc large-scale networks entails difficulty to a great extent. The direct communication among nodes in decentralized networks is beneficial regarding reduced latency, bandwidth rates, etc.; however, it requires strong network infrastructure, when nodes are geographically expanded. The major question is if we need all the devices to participate in the federation process? In Section 2.3, there are many works that paved the way for implementing communication reduction using clustering algorithms. For instance, in [57], it was demonstrated that devices in close proximity generate similar data, and thus, redundant ones can be excluded from the federation process, leading to a decrease in information circulation within the network. However, taking into consideration that the size of the messages transmitted is also significant is of great importance to find ways not only to lessen the size but also to secure that the message will be received. In this case, edge devices have to ensure that the connection between them will be stable. In addition, due to close proximity, nodes in distributed networks are encouraged to communicate with their one-hop neighbors. However, this does not ensure that the neighbor nodes are also the most trustworthy ones.

6.2. Imbalanced Dataset Classification Problems

Class imbalance is a classification predictive modeling problem, in which every class of the dataset, during the training phase, has an unequal amount of samples or not every class is equally significant. Consequently, this class distribution is biased or skewed. In taking into consideration that we want to achieve on-device training in TinyML environments, the use of online raw data is likely to face this problem and hence cause our mdoel to *overfit* the data. Han et al. solemnified Swarm Learning [14] as a possible alternative to this case; however, swarm nodes demand ample computational power in order to operate in cross-device decentralized settings. Another approach to mitigate the occurrence of this phenomenon was described by Liu et al. [132]. They proposed a clustering algorithm based on bloom filter methodology in order for distributed connected devices to be grouped by their contribution in balancing data classes in their cluster, paving the way for even more efficient solutions to this problem.

6.3. Catastrophic Forgetting

This is a phenomenon caused due to model adaptations made to prevent concept drift (which occurs when we have data belonging to different feature spaces) [5,6]. Inspired by biological definitions, the replacement of data in order for our model to fit more cases may destroy some already important features for the main problem this model was constructed for.

6.4. Heterogeneity

Dealing with data or model heterogeneity, or the problem of "worker drift" or "data distribution shift", is a very common phenomenon in training with real-time (raw) data. Real incoming data are often shifted regarding their statistical properties or they are amalgamated with noisy data, which impedes the learning efficiency of the model, leading to its accuracy degradation [98].

6.5. Benchmarking

Having tools and datasets that can ensure the testing of tinyML algorithms in order to evaluate its efficacy is of great importance. There has been some progress in this field, e.g., *TinyMLPerf* [133], which extends the MLPerf benchmark in order to fit the needs of tinyML, and Tiny-Imagenet [48], which originates from the well-known Imagenet dataset and consists of 100,000 images, shrinked and downsized to 64x64 pixels in order to fit

TinyML devices. However, there is a lot of research and experiments that have to be performed in order to come up with specific comparative details adjusted to tiny federated learning settings.

6.6. Attacks

Due to a lack of memory resources, there is not enough capacity for taking security measures for end-devices (or frugal devices) in order to cope with attacks, e.g., Jamming–Backhaul–white-box attacks (adversarial ML). For example, in [134], a new non-continuous activation function (a different variation of ReLU) was proposed in order for a neural network to be less susceptible to gradient-associated adversarial attacks (specifically in white-box attacks) during training. In the case of a federated learning setting, data privacy is almost ensured, since every node participating in the learning procedure shares only gradients not the data themselves, in order to train the model. However, in this case, the corruption of the network is the main goal of attacker/malicious nodes, e.g., free-riding attacks, Byzantine faults [11], sybil attacks, or Competitive Advantage Attacks [30] in blockchain-based settings. It is worth highlighting that there is much research regarding detecting and/or counteracting against the possible attacks; however, the increase in the percentage of attacks will always be proportional to the advancements in ML/DL and significance of these advancements.

6.7. Fairness in FL

The implementation of federated learning does not entail the assurance of a symmetric end-node topology. In other words, it is not taken for granted that all devices contribute equally to the performance of the network and have both the same capabilities (e.g., machine specifications) and responsibilities, regarding the training procedure. Finding non-computationally expensive ways for end-nodes to identify the supportive nodes and rewarding them is of great importance. For instance, Li et al. [60] developed an algorithm in which devices with higher values in their local loss functions are given an also higher contributing factor in order to guarantee the generalization of the global model or else to ensure *a more uniform accuracy distribution.* However, it is of great importance to find even more efficient methodologies in which tiny devices can run.

6.8. Ever-Changing Topology

In real-time DFL settings, cooperative nodes may not be geographical stable—a node can depart or join the network any time (mobility of devices). Hence, the network topology is ever-changing through time [135]. This necessitates the implementation of the used clustering algorithms multiple times alongside the training process, meaning higher energy consumption.

6.9. Ethical Concerns

The widely known Generative AI models, e.g., GPT models, are designed to enhance many aspects including healthcare, industry, education, etc. [136]. GPT models can undoubtedly analyze large medical datasets, and given access to patient clinical records, they can detect types of disease and/or make estimations about patient treatments faster, enhancing the work of scientists. GPT models are also included in industries in the field of human resources, enabling faster screening and hence faster recruiting of new potential employees. A Suitable AI tool conducts the first round of the candidate selection process: only candidates that meet the job requirements are promoted to the interview stage. Contrarily, the ascent of Generative AI bears risk, and we have to place restrictions. For instance, these models [137] can also produce tampered models to some extent, such that we cannot discern whether the result is illusory or not. For example, these tools can easily generate images that depict fake content, leading to misleading information that is difficult to verify [137]. Regarding fairness, it is of great importance that the data given to this type of AI model for training be unbiased. For example, if the training dataset includes

more men instead of women in chief positions, the model will automatically interpret it as a crucial detail for deciding to whom the job is assigned. It will behave similarly in cases where the applicants have disabilities or racial differences, leading to the discrimination and hence marginalization of a large portion of people. There is need for society to put stringent legal limitations in order to ensure *auditability* and *accountability* for AI systems not only during the training phase of the model but also during deployment, stipulating the impact it can have on the progress of society, environment [138], etc. Ultimately, the rates of growth of both ML/DL algorithms and people's critical ability have to coincide in order for these tools to be advantageous to humanity [139].

7. Conclusions

The necessity of performing real-time data processing on edge devices in IoT networks has brought deep learning to the forefront of research. In this survey, we analyzed the possibilities of deploying reformable TinyML/DL algorithms in low-powered wirelessly connected devices in purely distributed (Decentralized) environments, using federated learning (FL) techniques. As secure parameter exchange is of major importance in DFL, the introduced blockchain-based paradigm, namely Swarm Learning, guarantees the effective and secure distribution of a significant amount of information, which is exchanged among edge devices, with a trade-off regarding the energy cost. The combination and the further advancement of the aforementioned fields pave the way for effective, secure, and energy-efficient algorithms.

Despite the advancement of the aforementioned technological areas, there are some issues that hinder the full exploration of their potential. For example, in cooperative learning, lessening communication overhead among participating devices is of great importance. A proposed way of doing so is to exclude several nodes from the training procedure; however, we have to define sophisticated ways of implementing this in order not to sacrifice the model's performance. The problem of data class imbalance is also a great challenge. The raw data of the participating entities may be skewed or biased, harming the training models' generalization by causing the overfitting effect. In taking into consideration the lack of resources, it is of great importance to introduce ways that autonomous devices (without a server coordination) can deal with this phenomenon in real-time scenarios. Last but not least, the introduction of nascent deep large language models spur the need of establishing stringent rules in order to ensure that this technological advancement is aligned with the core tenets of human life and nature (resource sustainability) in the long run.

Author Contributions: All authors contributed to the study conception and design. The first draft of the manuscript was written by E.F.; D.K. revised it substantially. All authors have read and agreed to the published version of the manuscript.

Funding: The research work is supported by the Hellenic Foundation for Research and Innovation (HFRI) under the 3rd Call for HFRI PhD Fellowships (Fellowship Number: 5631).

Data Availability Statement: No data were generated or processed during this study.

Conflicts of Interest: The authors declare that they have no known conflicting and/or competing financial or non-financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

1. Ajani, T.S.; Imoize, A.L.; Atayero, A.A. An Overview of Machine Learning within Embedded and Mobile Devices–Optimizations and Applications. *Sensors* **2021**, *21*, 4412. [CrossRef] [PubMed]
2. Waldrop, M.M. The chips are down for Moore's law. *Nature* **2016**, *530*, 144–147. [CrossRef] [PubMed]
3. Lai, L.; Suda, N.; Chandra, V. CMSIS-NN: Efficient Neural Network Kernels for Arm Cortex-M CPUs. *arXiv* **2018**, arXiv:1801.06601.
4. Abadade, Y.; Temouden, A.; Bamoumen, H.; Benamar, N.; Chtouki, Y.; Hafid, A.S. A comprehensive survey on TinyML. *IEEE Access* **2023**, *11*, 96892–96922. [CrossRef]
5. Rajapakse, V.; Karunanayake, I.; Ahmed, N. Intelligence at the Extreme Edge: A Survey on Reformable TinyML. *ACM Comput. Surv.* **2023**, *55*, 1–30. [CrossRef]

6. Kirkpatrick, J.; Pascanu, R.; Rabinowitz, N.C.; Veness, J.; Desjardins, G.; Rusu, A.A.; Milan, K.; Quan, J.; Ramalho, T.; Grabska-Barwinska, A.; et al. Overcoming catastrophic forgetting in neural networks. *Proc. Natl. Acad. Sci. USA* **2016**, *114*, 3521–3526. [CrossRef]
7. Mermillod, M.; Bugaiska, A.; Bonin, P. The stability-plasticity dilemma: Investigating the continuum from catastrophic forgetting to age-limited learning effects. *Front. Psychol.* **2013**, *4*, 504. [CrossRef]
8. McMahan, B.; Moore, E.; Ramage, D.; Hampson, S.; Blaise Aguera y Arcas, B. Communication-efficient learning of deep networks from decentralized data. In Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS), Fort Lauderdale, FL, USA, 20–22 April 2017; pp. 1273–1282.
9. Ram, S.S.; Nedić, A.; Veeravalli, V.V. Distributed Stochastic Subgradient Projection Algorithms for Convex Optimization. *J. Optim. Theory Appl.* **2008**, *147*, 516–545.
10. Giménez, N.L.; Solé, J.M.; Freitag, F. Embedded federated learning over a LoRa mesh network. *Pervasive Mob. Comput.* **2023**, *93*, 101819. [CrossRef]
11. Lim, W.Y.B.; Luong, N.C.; Hoang, D.T.; Jiao, Y.; Liang, Y.C.; Yang, Q.; Niyato, D.; Miao, C. Federated Learning in Mobile Edge Networks: A Comprehensive Survey. *IEEE Commun. Surv. Tutorials* **2020**, *22*, 2031–2063. [CrossRef]
12. Taya, A.; Nishio, T.; Morikura, M.; Yamamoto, K. Decentralized and model-free federated learning: Consensus-based distillation in function space. *IEEE Trans. Signal Inf. Process. Over Netw.* **2022**, *8*, 799–814. [CrossRef]
13. Wang, J.; Sahu, A.K.; Yang, Z.; Joshi, G.; Kar, S. MATCHA: Speeding Up Decentralized SGD via Matching Decomposition Sampling. In Proceedings of the Indian Control Conference (ICC), New Delhi, India, 9–11 January 2019; pp. 299–300.
14. Han, J.; Ma, Y.F.; Han, Y.; Zhang, Y.; Huang, G. Demystifying Swarm Learning: A New Paradigm of Blockchain-based Decentralized Federated Learning. *arXiv* **2022**, arXiv:2201.05286.
15. Hu, S.; Chen, X.; Ni, W.; Hossain, E.; Wang, X. Distributed Machine Learning for Wireless Communication Networks: Techniques, Architectures, and Applications. *IEEE Commun. Surv. Tutorials* **2021**, *23*, 1458–1493. [CrossRef]
16. Bellavista, P.; Foschini, L.; Mora, A. Decentralised learning in federated deployment environments: A system-level survey. *ACM Comput. Surv.* **2021**, *54*, 1–38. [CrossRef]
17. Haddaway, N.R.; Page, M.J.; Pritchard, C.C.; McGuinness, L.A. An R package and Shiny app for producing PRISMA 2020-compliant flow diagrams, with interactivity for optimised digital transparency and Open Synthesis. *Campbell Syst. Rev.* **2022**, *18*, e1230. [CrossRef]
18. Beltr'an, E.T.M.; Pérez, M.Q.; S'anchez, P.M.S.; Bernal, S.L.; Bovet, G.; Pérez, M.G.; P'erez, G.M.; Celdr'an, A.H. Decentralized Federated Learning: Fundamentals, state-of-the-art, frameworks, trends, and challenges. *IEEE Commun. Surv. Tutorials* **2023**, *25*, 2983–3013. [CrossRef]
19. Mori, J.; Teranishi, I.; Furukawa, R. Continual Horizontal Federated Learning for Heterogeneous Data. In Proceedings of the International Joint Conference on Neural Networks (IJCNN), Padua, Italy, 18–23 July 2022; pp. 1–8.
20. Lootus, M.; Thakore, K.; Leroux, S.; Trooskens, G.; Sharma, A.; Ly, H. A VM/Containerized Approach for Scaling TinyML Applications. *arXiv* **2022**, arXiv:2202.05057
21. Sun, Y.; Shen, L.; Tao, D. Which mode is better for federated learning? Centralized or decentralized. *arXiv* **2023**, arXiv:2310.03461.
22. Sánchez-García, R.J. Exploiting symmetry in network analysis. *Commun. Phys.* **2020**, *3*, 87. [CrossRef]
23. Pan, Q.; Cao, H.; Zhu, Y.; Liu, J.; Li, B. Contextual Client Selection for Efficient Federated Learning over Edge Devices. *IEEE Trans. Mob. Comput.* **2024**, *23*, 6538–6548. [CrossRef]
24. Liu, K.; Uplavikar, N.; Jiang, W.; Fu, Y. Privacy-Preserving Multi-task Learning. In Proceedings of the IEEE International Conference on Data Mining (ICDM), Singapore, 17–20 November 2018; pp. 1128–1133.
25. Li, Q.; Zhang, M.; Yin, N.; Yin, Q.; Shen, L. Asymmetrically Decentralized Federated Learning. *arXiv* **2023**, arXiv:2310.05093.
26. Savazzi, S.; Nicoli, M.; Rampa, V. Federated Learning with Cooperating Devices: A Consensus Approach for Massive IoT Networks. *IEEE Internet Things J.* **2020**, *7*, 4641–4654. [CrossRef]
27. Michelusi, N. Decentralized Federated Learning via Non-Coherent Over-the-Air Consensus. In Proceedings of the IEEE International Conference on Communications (ICC), Rome, Italy, 28 May–1 June 2023; pp. 3102–3107.
28. Yang, P.; Jiang, Y.; Wen, D.; Wang, T.; Jones, C.N.; Shi, Y. Decentralized Over-the-Air Federated Learning by Second-Order Optimization Method. *IEEE Trans. Wirel. Commun.* **2024**, *23*, 5632–5647. [CrossRef]
29. Shi, Y.; Zhou, Y.; Shi, Y. Over-the-Air Decentralized Federated Learning. In Proceedings of the IEEE International Symposium on Information Theory (ISIT), Melbourne, Australia, 12–20 July 2021; pp. 455–460.
30. Jia, Y.; Fang, M.; Gong, N.Z. Competitive Advantage Attacks to Decentralized Federated Learning. *arXiv* **2023**, arXiv:2310.13862.
31. Lin, F.P.C.; Hosseinalipour, S.; Azam, S.S.; Brinton, C.G.; Michelusi, N. Semi-Decentralized Federated Learning With Cooperative D2D Local Model Aggregations. *IEEE J. Sel. Areas Commun.* **2021**, *39*, 3851–3869. [CrossRef]
32. Tao, Y.; Zhou, J.; Yu, S. Efficient Parameter Aggregation in Federated Learning with Hybrid Convergecast. In Proceedings of the IEEE Consumer Communications & Networking Conference (CCNC), Las Vegas, NV, USA, 9–12 January 2021; pp. 1–6.
33. Chou, L.; Liu, Z.; Wang, Z.; Shrivastava, A. Efficient and Less Centralized Federated Learning. In Proceedings of the Joint European Conference on Machine Learning and Knowledge Discovery in Databases (ECML/PKDD), Bilbao, Spain, 13–17 September 2021; pp. 772–787.
34. Hosseinalipour, S.; Brinton, C.G.; Aggarwal, V.; Dai, H.; Chiang, M. From Federated to Fog Learning: Distributed Machine Learning over Heterogeneous Wireless Networks. *IEEE Commun. Mag.* **2020**, *58*, 41–47. [CrossRef]

35. Lee, J.W.; Oh, J.; Lim, S.; Yun, S.Y.; Lee, J.G. TornadoAggregate: Accurate and Scalable Federated Learning via the Ring-Based Architecture. *arXiv* **2020**, arXiv:2012.03214
36. Hosseinalipour, S.; Azam, S.S.; Brinton, C.G.; Michelusi, N.; Aggarwal, V.; Love, D.J.; Dai, H. Multi-Stage Hybrid Federated Learning Over Large-Scale D2D-Enabled Fog Networks. *IEEE/ACM Trans. Netw.* **2020**, *30*, 1569–1584. [CrossRef]
37. Xing, H.; Simeone, O.; Bi, S. Decentralized Federated Learning via SGD over Wireless D2D Networks. In Proceedings of the IEEE International Workshop on Signal Processing Advances in Wireless Communications (SPAWC), Atlanta, GA, USA, 26–29 May 2020; pp. 1–5.
38. Lalitha, A.; Kilinc, O.C.; Javidi, T.; Koushanfar, F. Peer-to-peer Federated Learning on Graphs. *arXiv* **2019**, arXiv:1901.11173
39. Zhou, S.; Xu, K.; Li, G.Y. Communication-Efficient Decentralized Federated Learning via One-Bit Compressive Sensing. *arXiv* **2023**, arXiv:2308.16671.
40. Sun, T.; Li, D.; Wang, B. Decentralized Federated Averaging. *IEEE Trans. Pattern Anal. Mach. Intell.* **2021**, *45*, 4289–4301. [CrossRef] [PubMed]
41. Blot, M.; Picard, D.; Cord, M.; Thome, N. Gossip training for deep learning. *arXiv* **2016**, arXiv:1611.09726
42. Hu, C.; Jiang, J.; Wang, Z. Decentralized Federated Learning: A Segmented Gossip Approach. *arXiv* **2019**, arXiv:1908.07782
43. Krizhevsky, A.; Nair, V.; Hinton, G. *Learning Multiple Layers of Features from Tiny Images*; Technical Report; Computer Science Department, University of Toronto: Toronto, ON, USA, 2009.
44. Ye, H.; Liang, L.; Li, G.Y. Decentralized Federated Learning with Unreliable Communications. *IEEE J. Sel. Top. Signal Process.* **2021**, *16*, 487–500. [CrossRef]
45. Li, B.; Gao, W.; Xie, J.; Gong, M.; Wang, L.; Li, H. Prototype-based Decentralized Federated Learning for the Heterogeneous Time-varying IoT Systems. *IEEE Internet Things J.* **2024**, *11*, 6916–6927. [CrossRef]
46. LeCun, Y.; Bottou, L.; Bengio, Y.; Haffner, P. Gradient-based learning applied to document recognition *Proc. IEEE* **1998**, *86*, 2278–2324. [CrossRef]
47. Xiao, H.; Rasul, K.; Vollgraf, R. Fashion-MNIST: A novel image dataset for benchmarking machine learning algorithms. *arXiv* **2017**, arXiv:1708.07747.
48. Le, Y.; Yang, X. Tiny ImageNet visual recognition challenge. *Comput. Sci.* **2015**, *7*, *3*.
49. Lalitha, A.; Shekhar, S.; Javidi, T.; Koushanfar, F. Fully decentralized federated learning. In Proceedings of the Third Workshop on Bayesian Deep Learning (NeurIPS), Montréal, QC, Canada, 7 December 2018.
50. Koloskova, A.; Loizou, N.; Boreiri, S.; Jaggi, M.; Stich, S.U. A Unified Theory of Decentralized SGD with Changing Topology and Local Updates. *arXiv* **2020**, arXiv:2003.10422.
51. Li, X.; Yang, W.; Wang, S.; Zhang, Z. Communication Efficient Decentralized Training with Multiple Local Updates. *arXiv* **2019**, arXiv:1910.09126.
52. Patel, K.K.; Dieuleveut, A. Communication Trade-Offs for Synchronized Distributed SGD with Large Step Size. *arXiv* **2019**, arXiv:1904.11325.
53. Roy, A.G.; Siddiqui, S.; Pölsterl, S.; Navab, N.; Wachinger, C. BrainTorrent: A Peer-to-Peer Environment for Decentralized Federated Learning. *arXiv* **2019**, arXiv:1905.06731.
54. Amiri, M.M.; Gündüz, D. Machine Learning at the Wireless Edge: Distributed Stochastic Gradient Descent Over-the-Air. In Proceedings of the IEEE International Symposium on Information Theory (ISIT), Paris, France, 7–12 July 2019; pp. 1432–1436.
55. Liu, W.; Chen, L.; Zhang, W. Decentralized Federated Learning: Balancing Communication and Computing Costs. *IEEE Trans. Signal Inf. Process. Over Netw.* **2021**, *8*, 131–143. [CrossRef]
56. Onoszko, N.; Karlsson, G.; Mogren, O.; Zec, E.L. Decentralized Federated Learning of Deep Neural Networks on Non-IID Data. *arXiv* **2021**, arXiv:2107.08517.
57. Fragkou, E.; Chini, E.; Papadopoulou, M.; Papakostas, D.; Katsaros, D.; Dustdar, S. Distributed Federated Deep Learning in Clustered Internet of Things Wireless Networks with Data Similarity-based Client Participation. *IEEE Internet Comput. Mag.* 2024, *To appear*.
58. Kalra, S.; Wen, J.; Cresswell, J.C.; Volkovs, M.; Tizhoosh, H.R. Decentralized federated learning through proxy model sharing. *Nat. Commun.* **2021**, *14*, 2899. [CrossRef]
59. Zhang, Y.; Xiang, T.; Hospedales, T.M.; Lu, H. Deep mutual learning. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 18–23 June 2018; pp. 4320–4328.
60. Li, T.; Sanjabi, M.; Smith, V. Fair Resource Allocation in Federated Learning. *arXiv* **2019**, arXiv:1905.10497.
61. He, C.; Li, S.; So, J.; Zhang, M.; Wang, H.; Wang, X.; Vepakomma, P.; Singh, A.; Qiu, H.; Shen, L.; et al. FedML: A Research Library and Benchmark for Federated Machine Learning. *arXiv* **2020**, arXiv:2007.13518.
62. Beltrán, E.T.M.; Gómez, Á.L.P.; Feng, C.; Sánchez, P.M.S.; Bernal, S.L.; Bovet, G.; Pérez, M.G.; Pérez, G.M.; Celdrán, A.H. Fedstellar: A platform for decentralized federated learning. *Expert Syst. Appl.* **2024**, *242*, 122861. [CrossRef]
63. Zhu, H.; Xu, J.; Liu, S.; Jin, Y. Federated learning on non-IID data: A survey. *Neurocomputing* **2021**, *465*, 371–390. [CrossRef]
64. Disabato, S.; Roveri, M. Tiny Machine Learning for Concept Drift. *IEEE Trans. Neural Netw. Learn. Syst.* **2022**, *14*, 11. [CrossRef]
65. Chen, Y.; Chai, Z.; Cheng, Y.; Rangwala, H. Asynchronous Federated Learning for Sensor Data with Concept Drift. In Proceedings of the IEEE International Conference on Big Data (BigData), Orlando, FL, USA, 15–18 December 2021; pp. 4822–4831.
66. Lu, J.; Liu, A.; Dong, F.; Gu, F.; Gama, J.; Zhang, G. Learning under Concept Drift: A Review. *IEEE Trans. Knowl. Data Eng.* **2019**, *31*, 2346–2363. [CrossRef]

67. Hoefler, T.; Alistarh, D.; Ben-Nun, T.; Dryden, N.; Peste, A. Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks. *J. Mach. Learn. Res.* **2021**, *22*, 1–124.
68. Shafique, M.A.; Theocharides, T.; Reddy, V.J.; Murmann, B. TinyML: Current Progress, Research Challenges, and Future Roadmap. In Proceedings of the ACM/IEEE Design Automation Conference (DAC), San Francisco, CA, USA, 5–9 December 2021; pp. 1303–1306.
69. Frankle, J.; Carbin, M. The Lottery Ticket Hypothesis: Finding sparse, trainable neural networks. In Proceedings of the International Conference on Learning Representations (ICLR), New Orleans, LA, USA, 6–9 May 2019.
70. Leon, J.D.D.; Atienza, R. Depth Pruning with Auxiliary Networks for Tinyml. In Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Singapore, 7–13 May 2022; pp. 3963–3967.
71. Liu, H.; Song, P.; Qie, Y.; Li, Y. Real-time Prediction Method of Remaining Useful Life Based on TinyML. In Proceedings of the IEEE International Conference on Real-Time Computing and Robotics (RCAR), Guiyang, China, 17–22 July 2022; pp. 693–698.
72. Li, Y.; Li, Z.; Zhang, T.; Zhou, P.; Feng, S.; Yin, K. Design of a Novel Neural Network Compression Method for Tiny Machine Learning. In Proceedings of the International Conference on Electronic Information Technology and Computer Engineering, Xiamen, China, 22–24 October 2021.
73. Ghamari, S.; Ozcan, K.; Dinh, T.; Melnikov, A.; Carvajal, J.; Ernst, J.; Chai, S.M. Quantization-Guided Training for Compact TinyML Models. *arXiv* **2021**, arXiv:2103.06231.
74. Heim, L.; Biri, A.; Qu, Z.; Thiele, L. Measuring what Really Matters: Optimizing Neural Networks for TinyML. *arXiv* **2021**, arXiv:2104.10645
75. Zemlyanikin, M.; Smorkalov, A.; Khanova, T.; Petrovicheva, A.; Serebryakov, G. 512KiB RAM Is Enough! Live Camera Face Recognition DNN on MCU. In Proceedings of the IEEE/CVF International Conference on Computer Vision Workshop (ICCVW), Seoul, Korea, 28 October 2019; pp. 2493–2500.
76. Ren, P.; Xiao, Y.; Chang, X.; Huang, P.Y.; Li, Z.; Chen, X.; Wang, X. A comprehensive survey of neural architecture search: Challenges and solutions. *ACM Comput. Surv.* **2021**, *54*, 1–34. [CrossRef]
77. Buciluǎ, C.; Caruana, R.; Niculescu-Mizil, A. Model Compression. In Proceedings of the ACM Conference on Knowledge Discovery and Data Mining (KDD), Philadelphia, PA, USA, 20–23 August 2006; pp. 535–541.
78. Ba, J.; Caruana, R. Do Deep Nets Really Need to be Deep? In Proceedings of the Neural Information Processing Systems (NIPS), Lake Tahoe Nevada, 5–10 December **2013**.
79. Hinton, G.E.; Vinyals, O.; Dean, J. Distilling the Knowledge in a Neural Network. *arXiv* **2015**, arXiv:1503.02531
80. Gou, J.; Yu, B.; Maybank, S.J.; Tao, D. Knowledge Distillation: A Survey. *Int. J. Comput. Vis.* **2020**, *129*, 1789–1819. [CrossRef]
81. Deng, J.; Dong, W.; Socher, R.; Li, L.J.; Li, K.; Fei-Fei, L. Imagenet: A large-scale hierarchical image database. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Miami, FL, USA, 20–25 June 2009; pp. 248–255.
82. Fragkou, E.; Lygnos, V.; Katsaros, D. Transfer Learning for Convolutional Neural Networks in Tiny Deep Learning Environments. In Proceedings of the Pan-Hellenic Conference on Informatics (PCI), Athens, Greece, 25–27 November 2022; pp. 145–150.
83. Ahmed, K.; Hassan, M. tinyCare: A tinyML-based Low-Cost Continuous Blood Pressure Estimation on the Extreme Edge. In Proceedings of the IEEE International Conference on Healthcare Informatics (ICHI), Rochester, MN, USA, 11–14 June 2022; pp. 264–275.
84. T'Jonck, K.; Kancharla, C.R.; Vankeirsbilck, J.; Hallez, H.; Boydens, J.; Pang, B. Real-Time Activity Tracking using TinyML to Support Elderly Care. In Proceedings of the International Scientific Conference on Electronics (ET), Sozopol, Bulgaria, 15–17 September 2021; pp. 1–6.
85. Antonini, M.; Pincheira, M.; Vecchio, M.; Antonelli, F. A TinyML approach to non-repudiable anomaly detection in extreme industrial environments. In Proceedings of the EEE International Workshop on Metrology for Industry 4.0& IoT (MetroInd4.0 & IoT), Trento, Italy, 7–9 June 2022; pp. 397–402.
86. Andrade, P.; Silva, I.; Signoretti, G.; Silva, M.; Dias, J.; Marques, L.; Costa, D.G. An Unsupervised TinyML Approach Applied for Pavement Anomalies Detection Under the Internet of Intelligent Vehicles. In Proceedings of the IEEE International Workshop on Metrology for Industry 4.0 & IoT (MetroInd4.0&IoT), Rome, Italy, 7–9 June 2021; pp. 642–647.
87. A TinyML-based system for gas leakage detection. In Proceedings of the International Conference on Modern Circuits and Systems Technologies (MOCAST), Bremen, Germany, 8–10 June 2022; pp. 1–5.
88. Rana, A.; Dhiman, Y.; Anand, R. Cough Detection System using TinyML. In Proceedings of the International Conference on Computing, Communication and Power Technology (IC3P), Visakhapatnam, India, 7–8 January 2022; pp. 119–122.
89. Fang, K.; Xu, Z.; Li, Y.; Pan, J. A Fall Detection using Sound Technology Based on TinyML. In Proceedings of the International Conference on Information Technology in Medicine and Education (ITME), Wuyishan, China, 19–21 November 2021; pp. 222–225.
90. Nicolas, C.; Naila, B.; Amar, R.C. TinyML Smart Sensor for Energy Saving in Internet of Things Precision Agriculture platform. In Proceedings of the International Conference on Ubiquitous and Future Networks (ICUFN), Barcelona, Spain, 5–8 July 2022; pp. 256–259.
91. Giordano, M.; Baumann, N.; Crabolu, M.; Fischer, R.; Bellusci, G.; Magno, M. Design and Performance Evaluation of an Ultralow-Power Smart IoT Device with Embedded TinyML for Asset Activity Monitoring. *IEEE Trans. Instrum. Meas.* **2022**, *71*, 2510711. [CrossRef]
92. Maltoni, D.; Lomonaco, V. Continuous Learning in Single-Incremental-Task Scenarios. *Neural Netw.* **2018**, *116*, 56–73. [CrossRef]

93. Lesort, T.; Lomonaco, V.; Stoian, A.; Maltoni, D.; Filliat, D.; Díaz-Rodríguez, N. Continual learning for robotics: Definition, framework, learning strategies, opportunities and challenges. *Inf. Fusion* **2020**, *58*, 52–68. [CrossRef]
94. Pellegrini, L.; Graffieti, G.; Lomonaco, V.; Maltoni, D. Latent Replay for Real-Time Continual Learning. In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Macau, China, 4–8 November 2019; pp. 10203–10209.
95. Lopez-Paz, D.; Ranzato, M. Gradient Episodic Memory for Continual Learning. In Proceedings of the Neural Information Processing Systems (NIPS), Long Beach, CA, USA, 4–9 December 2017.
96. Cha, H.; Lee, J.; Shin, J. Co^2L: Contrastive continual learning. In Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV), Montreal, BC, Canada, 11–17 October 2021; pp. 9496–9505.
97. Smith, J.; Tian, J.; Hsu, Y.C.; Kira, Z. A Closer Look at Rehearsal-Free Continual Learning. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), New Orleans, LA, USA, 19–20 June 2022; pp. 2410–2420.
98. Ravaglia, L.; Rusci, M.; Nadalini, D.; Capotondi, A.; Conti, F.; Benini, L. A TinyML Platform for On-Device Continual Learning with Quantized Latent Replays. *IEEE J. Emerg. Sel. Top. Circuits Syst.* **2021**, *11*, 789–802. [CrossRef]
99. Hinton, G.E. The Forward-Forward Algorithm: Some preliminary investigations. *arXiv* **2022**, arXiv:2212.13345.
100. Vita, F.D.; Nawaiseh, R.M.A.; Bruneo, D.; Tomaselli, V.; Lattuada, M.; Falchetto, M. μ-FF: On-Device Forward-Forward Training Algorithm for Microcontrollers. In Proceedings of the IEEE International Conference on Smart Computing (SMARTCOMP), Nashville, TN, USA, 26–30 June 2023; pp. 49–56.
101. Mocanu, D.C.; Mocanu, E.; Stone, P.; Nguyen, P.H.; Gibescu, M.; Liotta, A. Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science. *Nat. Commun.* **2017**, *9*, 2383. [CrossRef]
102. Chouliaras, A.; Fragkou, E.; Katsaros, D. Feed Forward Neural Network Sparsificationwith Dynamic Pruning. In Proceedings of the Pan-Hellenic Conference on Informatics (PCI), Volos, Greece, 26–28 November 2021; pp. 12–17.
103. Fragkou, E.; Koultouki, M.; Katsaros, D. Model reduction of feed forward neural networks for resource-constrained devices. *Appl. Intell.* **2023**, *53*, 14102–14127. [CrossRef]
104. Yosinski, J.; Clune, J.; Bengio, Y.; Lipson, H. How transferable are features in deep neural networks? In Proceedings of the Neural Information Processing Systems (NIPS), Montreal Canada, 8–13 December 2014.
105. Ren, H.; Anicic, D.; Runkler, T.A. TinyOL: TinyML with Online-Learning on Microcontrollers. In Proceedings of the International Joint Conference on Neural Networks (IJCNN), Virtual, 18–22 July 2021; pp. 1–8.
106. Disabato, S.; Roveri, M. Incremental On-Device Tiny Machine Learning. In Proceedings of the International Workshop on Challenges in Artificial Intelligence and Machine Learning for Internet of Things (AIChallengeIoT), Virtual (online), 16 November 2020.
107. Aloufi, R.; Haddadi, H.; Boyle, D. Emotion Filtering at the Edge. In Proceedings of the Workshop on Machine Learning on Edge in Sensor Systems, New York, NY, USA, 10 November 2019.
108. Sudharsan, B.; Breslin, J.G.; Tahir, M.; Ali, M.I.; Rana, O.F.; Dustdar, S.; Ranjan, R.; Dustdar, S. OTA-TinyML: Over the Air Deployment of TinyML Models and Execution on IoT Devices. *IEEE Internet Comput. Mag.* **2022**, *26*, 69–78. [CrossRef]
109. Li, J.; Kuang, R. Split Federated Learning on Micro-controllers: A Keyword Spotting Showcase. *arXiv* **2022**, arXiv:2210.01961.
110. Banbury, C.R.; Zhou, C.; Fedorov, I.; Navarro, R.M.; Thakker, U.; Gope, D.; Reddi, V.J.; Mattina, M.; Whatmough, P.N. MicroNets: Neural Network Architectures for Deploying TinyML Applications on Commodity Microcontrollers. *Proc. Mach. Learn. Syst.* **2021**, *3*, 517–532.
111. Sievers, B.; Hauschild, S.; Hellbrück, H. Inference and Performance Analysis of Convolutional Neural Networks used for Human Gesture Recognition on IoT-Devices. *Computing* **2021**, *2*, 3.
112. de Prado, M.; Rusci, M.; Capotondi, A.; Donze, R.; Benini, L.; Pazos, N. Robustifying the Deployment of tinyML Models for Autonomous Mini-Vehicles. *Sensors* **2021**, *21*, 1339. [CrossRef]
113. Pullini, A.; Rossi, D.; Loi, I.; Tagliavini, G.; Benini, L.M. Wolf: An Energy-Precision Scalable Parallel Ultra Low Power SoC for IoT Edge Processing. *IEEE J. -Solid-State Circuits* **2019**, *54*, 1970–1981. [CrossRef]
114. David, R.; Duke, J.; Jain, A.; Janapa Reddi, V.; Jeffries, N.; Li, J.; Kreeger, N.; Nappier, I.; Natraj, M.; Wang, T.; et al. Tensorflow lite micro: Embedded machine learning for tinyml systems. *Proc. Mach. Learn. Syst.* **2021**, *3*, 800–811.
115. Ray, P.P. A review on TinyML: State-of-the-art and prospects. *J. King Saud Univ.-Comput. Inf. Sci.* **2021**, *34*, 1595–1623. [CrossRef]
116. Capotondi, A.; Rusci, M.; Fariselli, M.; Benini, L. CMix-NN: Mixed Low-Precision CNN Library for Memory-Constrained Edge Devices. *IEEE Trans. Circuits Syst. Ii Express Briefs* **2020**, *67*, 871–875. [CrossRef]
117. Jahanshahi, A. TinyCNN: A Tiny Modular CNN Accelerator for Embedded FPGA. *arXiv* **2019**, arXiv:1911.06777
118. Reddi, V.J.; Plancher, B.; Kennedy, S.; Moroney, L.; Warden, P.; Agarwal, A.; Banbury, C.R.; Banzi, M.; Bennett, M.; Brown, B.; et al. Widening Access to Applied Machine Learning with TinyML. *arXiv* **2021**, arXiv:2106.04008.
119. Kopparapu, K.; Lin, E. TinyFedTL: Federated Transfer Learning on Tiny Devices. *arXiv* **2021**, arXiv:2110.01107.
120. Zheng, J.; Xu, J.; Du, H.; Niyato, D.; Kang, J.; Nie, J.; Wang, Z. Trust Management of Tiny Federated Learning in Internet of Unmanned Aerial Vehicles. *IEEE Internet Things J.* **2024**, *11*, 21046–21060. [CrossRef]
121. Huang, H.; Zhang, L.; Sun, C.; Fang, R.; Yuan, X.; Wu, D. Distributed pruning towards tiny neural networks in federated learning. In Proceedings of the IEEE International Conference on Distributed Computing Systems (ICDCS), Hong Kong, China, 18–21 July 2023; pp. 190–201.
122. Zhu, J.; Cao, J.; Saxena, D.; Jiang, S.; Ferradi, H. Blockchain-empowered Federated Learning: Challenges, Solutions, and Future Directions. *ACM Comput. Surv.* **2022**, *55*, 1–31. [CrossRef]

123. Ma, X.; Xu, D. TORR: A Lightweight Blockchain for Decentralized Federated Learning. *IEEE Internet Things J.* **2023**, *11*, 1028–1040. [CrossRef]
124. Ghanem, M.C.; Dawoud, F.A.S.; Gamal, H.; Soliman, E.; Sharara, H.; El-Batt, T. FLoBC: A Decentralized Blockchain-Based Federated Learning Framework. In Proceedings of the Fourth International Conference on Blockchain Computing and Applications (BCCA), Tartu, Estonia, 15–16 November 2021; pp. 85–92.
125. Wang, H.; Mao, D.; Chen, Z.; Rao, H.; Li, Z. Blockchain-Based Decentralized Federated Learning Model. In Proceedings of the International Conference on Information Science, Parallel and Distributed Systems (ISPDS), Guangzhou, China, 14–16 July 2023; pp. 622–625.
126. Li, J.; Shao, Y.; Wei, K.; Ding, M.; Ma, C.; Shi, L.; Han, Z.; Poor, V. Blockchain Assisted Decentralized Federated Learning (BLADE-FL): Performance Analysis and Resource Allocation. *IEEE Trans. Parallel Distrib. Syst.* **2022**, *33*, 2401–2415. [CrossRef]
127. Wang, T.; Dong, Z.Y. Blockchain Based Clustered Federated Learning for Non-Intrusive Load Monitoring. *IEEE Trans. Smart Grid* **2024**, *15*, 2348–2361. [CrossRef]
128. Ranathunga, T.; Mcgibney, A.; Rea, S.; Bharti, S. Blockchain-Based Decentralized Model Aggregation for Cross-Silo Federated Learning in Industry 4.0. *IEEE Internet Things J.* **2023**, *10*, 4449–4461. [CrossRef]
129. Jin, Y.; Jiao, L.; Qian, Z.; Zhou, R.; Pu, L. Orchestrating Blockchain with Decentralized Federated Learning in Edge Networks. In Proceedings of the IEEE International Conference on Sensing, Communication, and Networking (SECON), Madrid, Spain, 11–14 September 2023; pp. 483–491.
130. Yapp, A.Z.H.; Koh, H.S.N.; Lai, Y.T.; Kang, J.; Li, X.; Ng, J.S.; Jiang, H.; Lim, W.Y.B.; Xiong, Z.; Niyato, D.T. Communication-efficient and Scalable Decentralized Federated Edge Learning. In Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), Montreal, BC, Canada, 19–27 August 2021.
131. Riahi, A.; Mohamed, A.; Erbad, A. BC-FL Location-Based Disease Detection in Healthcare IoT. In Proceedings of the International Wireless Communications and Mobile Computing (IWCMC), Marrakesh, Morocco, 19–23 June 2023; pp. 1684–1689.
132. Liu, S.; Liu, Z.; Xu, Z.; Liu, W.; Trian, J. Hierarchical Decentralized Federated Learning Framework with Adaptive Clustering: Bloom-Filter-Based Companions Choice for Learning non-IID Data in IoV. *Electronics* **2023**, *12*, 3811. [CrossRef]
133. Banbury, C.; Reddi, V.J.; Torelli, P.; Holleman, J.; Jeffries, N.; Kiraly, C.; Montino, P.; Kanter, D.; Ahmed, S.; Pau, D.; et al. MLPerf Tiny Benchmark. *arXiv* **2021**, arXiv:2106.07597.
134. Xiao, C.; Zhong, P.; Zheng, C. Enhancing Adversarial Defense by k-Winners-Take-All. *arXiv* **2019**, arXiv:1905.10510.
135. Marias, G.F.; Georgiadis, P.; Flitzanis, D.; Mandalas, K. Cooperation enforcement schemes for MANETs: A survey. *Wirel. Commun. Mob. Comput.* **2006**, *6*, 319–332. [CrossRef]
136. Zhang, P.; Kamel Boulos, M.N. Generative AI in medicine and healthcare: Promises, opportunities and challenges. *Future Internet* **2023**, *15*, 286. [CrossRef]
137. Kenthapadi, K.; Lakkaraju, H.; Rajani, N. Generative AI meets Responsible AI: Practical Challenges and Opportunities. In Proceedings of the ACM Conference on Knowledge Discovery and Data Mining (KDD), Long Beach, CA, USA, 6–10 August 2023; pp. 5805–5806.
138. Thakur, D.; Guzzo, A.; Fortino, G.; Piccialli, F. Green Federated Learning: A new era of Green Aware AI, *arXiv* **2024**, arXiv:2409.12626.
139. Bandi, A.; Adapa, P.; Kuchi, Y.E. The Power of Generative AI: A Review of Requirements, Models, Input–Output Formats, Evaluation Metrics, and Challenges. *Future Internet* **2023**, *15*, 260. [CrossRef]