Gist - Optimizing Segmentation for Decentralized Federated Learning on Tiny Devices

Navidreza Asadi*
Technical University Munich
Munich, Germany

Halil Ibrahim Bengü*
Technical University of Munich
Munich, Germany

Lars Wulfert*
Fraunhofer IMS
Duisburg, Germany

Hendrik Wöhrle
University of Duisburg-Essen
Duisburg, Germany

Wolfgang Kellerer
Technical University Munich
Munich, Germany

Abstract

We introduce Gist, a decentralized federated learning framework for tiny microcontrollers. Rather than considering all model parameters as equally important, we let devices get the "gist" of the updates. Our contribution has three pillars: (1) it segments model parameters by their importance, identifying the most impactful updates; (2) it shares the segments probabilistically, ensuring rapid propagation of important knowledge while maintaining model diversity; and (3) it aggregates updates using a success-based scheme, giving more weight to information from better-performing peers. We implement and validate Gist through various simulation experiments, realistic large-scale emulation, and deployment on a physical cluster of ESP32-S3 microcontrollers. Across three models and two tasks, Gist outperforms existing baselines, achieving higher accuracy and faster convergence, especially in larger networks consisting of hundreds of devices.

ACM Reference Format:
Navidreza Asadi, Halil Ibrahim Bengü, Lars Wulfert, Hendrik Wöhrle, and Wolfgang Kellerer. 2025. Gist - Optimizing Segmentation for Decentralized Federated Learning on Tiny Devices. In *Federated Learning and Edge AI for Privacy and Mobility (FLEdge-AI '25), November 4–8, 2025, Hong Kong, China.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3737899.3768527

1 Introduction

Internet of Things (IoT) devices, particularly microcontrollers, are often resource-constrained with limited processing power, memory, and connectivity options [1, 2, 10]. Added to this, the data generated by these devices is often sensitive and private, and changes over time. These make it difficult to share data with a central server for training, demanding a decentralized approach to learning directly on device. Federated Learning (FL) enables collaborative machine learning on decentralized data, thereby preserving data privacy [5, 12, 25, 33]. However, the reliance of conventional FL on a central aggregation server introduces single point of failure and limits scalability. Decentralized FL (DFL) addresses these limitations by direct parameter sharing among devices, making it particularly suitable for IoT networks. Nevertheless, DFL imposes significant computation-communication overhead, a critical challenge for resource-constrained microcontrollers that form the backbone of many IoT applications, to the extent that the vanilla DFL is barely feasible on such devices.
Challenges. Existing approaches to mitigate this overhead, such as segmented gossip [9] often rely on random segmentation, which is inefficient. While importance-based model pruning is explored in centralized FL [18, 30], these methods typically hinder convergence. The application of more nuanced, importance-driven segmentation in DFL remains largely unexplored. On-device tasks like partial updates and model segmentation must be optimized to stay within hardware limits [17]. Additionally, consistent updates without a central server requires reliable decentralized communication. Further, data on IoT devices is often non-*i.i.d.* (independent and identically distributed) and unbalanced. DFL systems must be robust to handle data heterogeneity [21].
Approach. To bridge this gap, we propose Gist, a dynamic segmentation method tailored to microcontroller-based DFL networks. Our approach integrates three key mechanisms: parameter importance, probabilistic segment sharing, and success-based aggregation. Instead of treating all model parameters equally, our method prioritizes segments containing more impactful updates. These prioritized segments are then shared more frequently through a probabilistic mechanism, ensuring a balance between exploiting important updates and exploring the wider parameter space. Furthermore, our success-based aggregation scheme weights incoming

*Equal contribution

ACM ISBN 979-8-4007-1976-9/2025/11
https://doi.org/10.1145/3737899.3768527

model updates based on the sender's performance, enhancing the quality of the aggregated model.

Contribution. Our key contributions are:

▹ A novel dynamic segmentation method for DFL that prioritizes model updates based on parameter magnitude.

▹ A probabilistic sharing mechanism that balances exchange of important segments with model diversity.

▹ A success-based aggregation strategy that weights updates by the sender's performance.

▹ Implementation and evaluation across simulation, emulation, and physical hardware on three models and two tasks, demonstrating its effectiveness and practicality in resource-constrained environments.

2 Related Work

Two main mitigation strategies have gained prominence for DFL in IoT: model segmentation to reduce transmission size and gossip-based communication for efficient information propagation. We examine prior work in these areas, focusing specifically on approaches suitable for resource-constrained IoT environments where our proposed system operates.

Gossip Communication. Gossip learning is a decentralized approach where devices exchange model parameters with random peers to gradually propagate updates across the network, mitigating the communication overhead of broadcasting to all nodes. However, without a central server, the dynamic nature of DFL networks makes neighbor selection challenging [20, 34]. Several approaches try to address this. [28] proposes a method with a predefined communication structure where nodes exchange beliefs about model parameters only with their immediate neighbors. GossipFL [27] uses a semi-decentralized model with a coordinator that generates a gossip matrix to guide peer-to-peer model exchanges, optimizing bandwidth. CHOCO-GOSSIP [11] reduces communication by allowing nodes to exchange compressed or quantized updates [3], which is advantageous in large-scale, resource-constrained environments.

Segmentation Strategies. Model segmentation divides the global model into smaller parts to reduce communication and computational load. *While some centralized FL methods use structured or sketched (compressed) updates [13], DFL requires peer-to-peer aggregation of segments.* One approach, FIARSE [30], dynamically creates submodels by selecting parameters with the highest magnitude, assuming they are the most important [22]. This risks omitting learned features from local data. Combo [9] proposes a segmented gossip protocol where the model is divided into equal, non-overlapping segments. Nodes pull different segments from various peers to reconstruct a full model replica. This reduces communication overhead but introduces synchronization delays, making it less practical for real-world applications. Other methods focus on partial updates. For instance, [29] proposes exchanging partitioned gradient updates. To improve convergence, AdaStair and AdaLoss [31] introduce adaptive learning rate adjustments based on training rounds or model loss, respectively. However, these methods often rely on static segmentation or heuristics that may not adapt well to the dynamic nature of data and model updates in DFL. Static partitioning can be inefficient, while magnitude-based selection risks discarding crucial weight updates. A key challenge remains in dynamically identifying and prioritizing the most impactful segments for exchange to maximize learning efficiency while minimizing communication overhead.

Our proposed method addresses this gap by introducing a probabilistic, importance-aware segmentation strategy that adapts to model dynamics, advocating for the most relevant information to be efficiently propagated through the network. Table 1 compares prior art in DFL with our proposed method. It highlights the unique features of our approach, such as serverless operations[1], gossip communication, partial updates, asynchronous aggregation, selective segmentation, and probabilistic sharing.

3 Gist Design

Our proposed method specifically addresses the challenge of selective parameter sharing by prioritizing more impactful model parameters during communication and weighting aggregation based on sender accuracy, ultimately improving convergence performance compared to traditional random segmentation approaches.

3.1 Segmentation & Aggregation: A Primer

Following local training on each device, the model parameters are partitioned into sparse *segments*. These segments are then exchanged among devices and aggregated to progressively build an aggregated model. A segment is a sparse selection of parameters from different layers of a model. Our

Table 1: Comparison of FL Solutions

Methods	Server-less	Gossip Comm.	Partial Update	Async Agg.	Selective Segment.	Prob. Sharing
FedAvg [21]	✗	✗	✗	✗	✗	✗
GossipFL [27]	✗	✓	✗	✗	✗	✗
FedLAMA [16]	✗	✗	✓	✓	✓	✗
FIARSE [30]	✗	✗	✓	✗	✓	✗
P2P [28]	✓	✗	✗	✓	✗	✗
CHOCO [11]	✓	✓	✗	✓	✗	✗
AKO [29]	✓	✓	✓	✓	✗	✗
Combo [9]	✓	✓	✓	✗	✗	✗
AdaStair [31]	✓	✓	✓	✓	✗	✗
YOGA [18]	✗	✗	✓	✗	✓	✗
Gist	✓	✓	✓	✓	✓	✓

[1]Not to be confused with serverless akin to Function as a Service (FaaS).

framework supports two segmentation strategies: random and importance-based, a key contribution of GIST.

Our aggregation process integrates received parameter segments with the device's local model, moving toward global convergence. The receiver device starts the aggregation by aligning each received segment to its known position using the assigned bitmap. Integration is facilitated by a division tensor, as illustrated in Fig. 1. This step is executed once a device accumulates a predefined number of segments.

Let $w^t \in \mathbb{R}^n$ denote the local model at round t, where n is the number of all model parameters. The device receives $N \in \mathbb{N}$ model segments from neighboring devices, each consisting of a binary bitmap and an associated value vector:

$$\{(\mathbf{b}_i, \mathbf{s}_i)\}_{i=1}^{N}, \quad \mathbf{b}_i \in \{0,1\}^n, \quad \mathbf{s}_i \in \mathbb{R}^d$$

Each bitmap $\mathbf{b}_i$ indicates which parameters were sent, containing exactly $d = \lceil \frac{n}{S} \rceil$ ones, where $S \in \mathbb{N}$ is the total number of model segments:

$$\sum_{j=1}^{n} b_{ij} = d, \qquad \forall i \in \{1,\dots,N\} \tag{1}$$

Let $\pi_i : \{1,\dots,d\} \longrightarrow \{j \in \{1,\dots,n\} \mid b_{ij} = 1\}$ denote the strictly increasing mapping that returns the index of the k-th one in bitmap $\mathbf{b}_i$. The aggregated parameter tensor $\mathbf{r} \in \mathbb{R}^n$ is computed by assigning the values from $\mathbf{s}_i$ to their corresponding parameter indices in $\mathbf{b}_i$:

$$r_j = \sum_{i=1}^{N} \sum_{k=1}^{d} \delta(j = \pi_i(k)) \cdot s_{ik}, \qquad \forall j \in \{1,\dots,n\} \tag{2}$$

$\delta(\cdot)$ denotes the Kronecker delta; only transmitted parameters contribute to the sum. This assigns s_{ik} to the position of the k-th one in the bitmap b_i, and is added to the corresponding index $j = \pi_i(k)$ in the total received tensor. Any parameters added to a segment are chosen randomly from the local model. For aggregation normalization, we define the division tensor $\phi \in \mathbb{R}^n$ that counts how many times each parameter index j was received:

$$\phi_j = \sum_{i=1}^{N} b_{ij}, \qquad \forall j \in \{1,\dots,n\} \tag{3}$$

Finally, the model update is performed by computing a weighted average of the accumulated received values and the local model parameters, normalized by the number of total contributions (including the local one):

$$w_j^{t+1} = \frac{r_j + w_j^t}{\phi_j + 1}, \qquad \forall j \in \{1,\dots,n\} \tag{4}$$

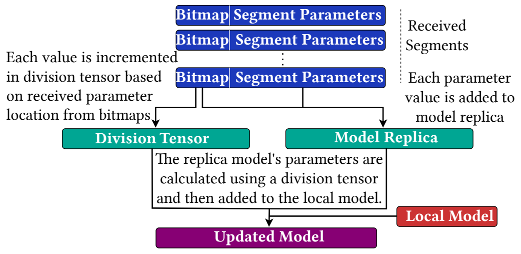

Bitmap
Segment Parameters
Received Segments
Each value is incremented in division tensor based on received parameter location from bitmaps
Each parameter value is added to model replica
Division Tensor
Model Replica
The replica model's parameters are calculated using a division tensor and then added to the local model.
Local Model
Updated Model

Figure 1: Aggregation on the receiver microcontroller.

3.2 Our Method

We introduce a DFL approach that segments the model based on parameter importance, prioritizing the propagation of *more impactful parameters*. Our method enhances model accuracy by synergistically combining importance-based segmentation, probabilistic segment selection, and success-based aggregation. This offers a fine-grained alternative to sharing entire model updates and a more dynamic approach than layer-level parameter ranking. The essential operations of the proposed method are illustrated in Fig. 2. The process begins by partitioning parameters into segments based on importance thresholds, which are derived from the desired number of segments. During communication, a probabilistic selection mechanism is employed, giving more important segments a higher probability of being shared. This strategy is balanced to ensure that less important segments are still shared periodically, mitigating model bias. Following the local training step, model accuracies on a common test set D_{test} are shared alongside the segments. This enables success-based aggregation, where received segments are weighted by the accuracy of their source nodes.

Importance-Based Segmentation. In GIST, the parameters of a local model are partitioned into S segments according to their magnitude, which acts as a proxy for importance. We start by calculating the percentile of the local model and creating S thresholds $\tau \in \mathbb{R}^S$ which are later used to divide the model parameters into S segments. First, the absolute value w^t of the local model at the round t is calculated for each element $\widetilde{w}_i^t = ||w_i^t||, \forall i \in \{1,\dots,n\}$. Subsequently, the values are ordered in non-descending order. For $i=1,\dots,S$ we set the target $q_i = \frac{100 \cdot i}{S}$ and determine

$$p_i = \frac{q_j}{100}(n-1), \qquad k_i = \lfloor p_i \rfloor + 1, \qquad \sigma = p_i - \lfloor p_i \rfloor.$$

The threshold τ_{q_i} is obtained by linear interpolation,

$$\tau_{q_i} = \begin{cases} \widetilde{w}_{(k)}^t, & k_i = n, \\ \widetilde{w}_{(k)}^t + \sigma\left(\widetilde{w}_{(k+1)}^t - \widetilde{w}_{(k)}^t\right), & 1 \le k_i < n. \end{cases} \tag{5}$$

Each parameter is assigned to exactly one segment. Let $d_i = |\mathcal{S}_i|, i=1,\dots,S$. We define

$$s_i = \left\{ j \in \{1,\dots,n\} \;\middle|\; ||w_j^t|| \ge \tau_i \land (i=S \lor ||w_j^t|| < \tau_{i+1}) \right\}, \tag{6}$$

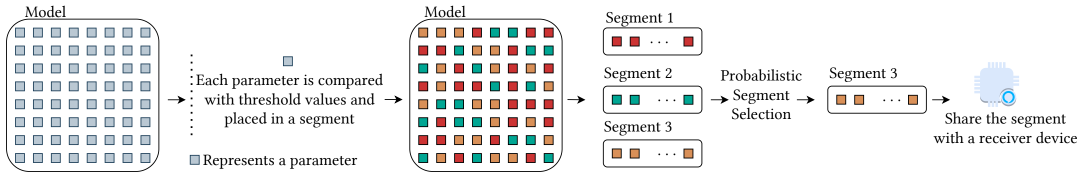

Model
Each parameter is compared with threshold values and placed in a segment
Represents a parameter
Model
Segment 1
Segment 2
Segment 3
Probabilistic Segment Selection
Segment 3
Share the segment with a receiver device

Figure 2: Segment creation and exchange in Gist.

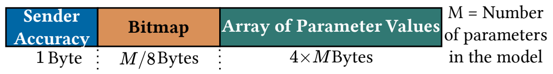

Sender Accuracy
Bitmap
Array of Parameter Values
M = Number of parameters in the model
1 Byte
M/8 Bytes
4×M Bytes

Figure 3: Contents of a model update message.

Hence d_i is the number of parameters that meet the i-th threshold and are not counted in any other segment. Parameters with magnitudes below the smallest threshold are not transmitted. This deliberate omission speeds up convergence and stabilizes training, which is particularly beneficial in DFL with non-*i.i.d.* data, where unstable convergence is often seen. For the transmission we define the bitmap entries as $b_{ij} = \mathbf{1}_{\{\tau_i \leq \| w_j^t \| < \tau_{i+1}\}}, \forall j \in \{1, \ldots, n\}$. An entry b_{ij}=1 indicates that the magnitude of the j-th model parameter lies within the interval $[\tau_i, \tau_{i+1})$ and therefore belongs to segment i, or otherwise b_{ij}=0.

Probabilistic Communication. The probability of sharing a segment is proportional to its importance, calculated via the softmax over the average parameter values within each segment (Eq. 7). $\overline{s_i}$ is the average of parameters in segment i.

$$P(i) = \frac{\exp(\overline{s_i})}{\sum_j \exp(\overline{s_j})} \tag{7}$$

Success-Based Aggregation. The aggregation process, therefore, modifies the formulation from Section 3.1. Let $a_i \in [0, 1]$ be the accuracy associated with segment i from a peer, and let a_{local} denote the accuracy of the local model. The received tensor and division tensor are computed as:

$$r_j = \sum_{i=1}^{N} \sum_{k=1}^{d} a_i \cdot \delta(j = \pi_i(k)) \cdot s_{ik}, \qquad \forall j \in \{1, \ldots, n\} \tag{8}$$

$$\phi_j = \sum_{i=1}^{N} a_i \cdot b_{ij}, \qquad \forall j \in \{1, \ldots, n\} \tag{9}$$

The received tensor is then computed by including the local model, weighted by its accuracy:

$$w_j^{t+1} = \frac{r_j + a_{local} \cdot w_j^t}{\phi_j + a_{local}}, \qquad \forall j \in \{1, \ldots, n\} \tag{10}$$

Finally, the local model is updated for the next round w^{t+1}. This accuracy-weighting scheme allows updates from more reliable peers to have a greater influence on the local model. The key steps of our method are outlined in Algorithm 1, comprising three main stages: importance-based segmentation, probabilistic communication, and success-based aggregation.

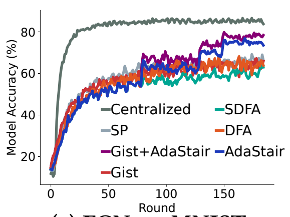
Algorithm 1 Gist - Importance-based Segmentation, Probabilistic Communication and Success-Based Aggregation

Input: $|s_{rec}|$: Total received segments, w_i: Initial parameters of device i, D_i: Local dataset of device i, D_{test}: Local test dataset, k: Number of devices to share with per round, τ_r: Model receive threshold for aggregation, S: Number of segments, T: Total communication rounds, E: Number of epochs, η: Learning rate

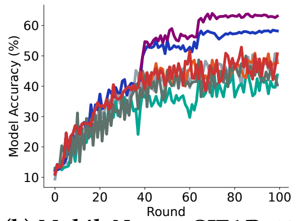
```
for $t$ to $T$ do                              ▷ Runs in parallel on each device
    $D_i^t$ = (split $D_i$ into batches of size $Z$)
    for $m$ to $E$ do
        for $\delta \in D_i^t$ do
            $w_i^t = w^t - \eta \nabla f(w^t, \delta)$
        end for
    end for
    for $i$ to $S$ do                          ▷ Segment Creation
        $q_i = \frac{100 \cdot i}{S}$
        $\tau_i$ = Percentile($|w^t|$, $q_i$)
        for $j$ to $n$ do
            $s_i$ = Equation 6
            $b_{ij} = 1_{\{\tau_i \leq \| w_j^t \| < \tau_{i+1}\}}$
        end for
    end for
    $a_{local}$ = Accuracy($w_i^t, D_{\text{test}}$)
    for $i$ to $k$ do                          ▷ Communication
        $P(i) = \frac{\exp(\overline{s_i})}{\sum_j \exp(\overline{s_j})}$
        $n$ = SelectRandomDevice()
        Share($a_{local}$, $b$, $P(i)$, $n$)
    end for
    if $|s_{rec}| > \tau_r$ then               ▷ Aggregation
        for $j$ to $n$ do
            $r_j = \sum_{i=1}^{N} \sum_{k=1}^{d} a_i \cdot \delta(j = \pi_i(k)) \cdot s_{ik}$
            $\phi_j = \sum_{i=1}^{N} a_i \cdot b_{ij}$
            $w_j^{t+1} = \frac{r_j + a_{local} \cdot w_j^t}{\phi_j + a_{local}}$
        end for
    end if
end for
```

4 Implementation

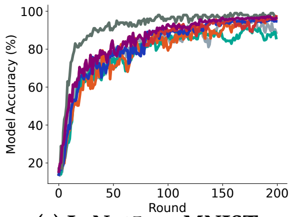
Gist operates across multiple deployment environments, from simulation to real microcontrollers, while maintaining the same foundational algorithms [1]. Each device acts as an autonomous entity. Upon startup, a device generates the parameters of the local neural network model. In our simulation environment, devices are always synchronized



(a) FCN on MNIST



(b) MobileNet on CIFAR-10



(c) LeNet5 on MNIST

Figure 4: Accuracy comparison over different models and datasets.

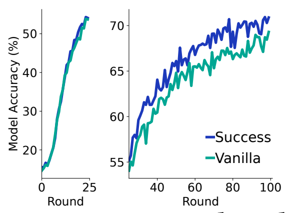


Figure 5: Success-based vs. Vanilla aggregation.

in their communication rounds. On the contrary, real and emulated devices operate independently, leading them to be at different training and communication rounds at times. For emulation and real microcontrollers, we implemented the operations and algorithms at a low level using `C`. We build our implementation on top of AIfES [32]. This includes implementation of the proposed segmentation, aggregation, data manipulation, communication, flattening of tensor objects and careful handling of memory; required to meet hardware limitations. We implemented the simulation environment in `Python` and integrated PyTorch libraries [6, 19, 23]. For simulation, we utilize high-level functions that simplify accessing, flattening, and loading weights back into the model. In our `C` implementation, computational operations are executed using allocated tensors, where the parameters included in the bitmaps are updated. Data types safety and race-condition prevention are ensured through thread locks, and aggregation is done incrementally to minimize memory consumption. This means the received segments are summed in a receiver array before the aggregation happens.

▹ **Communication.** At the beginning of each run, the device starts a service delegated to a single thread to listen for all incoming messages after registering to the network interface. **Service Discovery.** While initializing, the nodes broadcast their service to the neighboring nodes as part of the service discovery. We implement this in a blocking fashion, meaning that all nodes should pass this phase upon startup and before starting training and aggregating results. Each device sends a message after binding to the target Transmission Control Protocol (TCP) socket, and receiver devices send back a message with their device ID. Microcontrollers then save available nodes in the network for later communication between each other via direct TCP connections.

Messaging Scheme. A device trains its model locally and then segments the resulting parameters before each communication round into multiple parts with the selected segmentation method. All of the methods share these selected segments with a random subset of discovered devices. As Fig. 3 depicts, a message packet transmitted between devices consists of two main components: a header and the payload comprising model parameters. The header includes the device accuracy from the sender device over the commonly shared test dataset and a bitmap array marking the locations of the shared parameters inside the segment. The payload contains flattened parameter values of the model, that are serialized strings of floating point values. Upon receiving the message packet, the device first parses the header to keep the accuracy rate of the sender device and the bitmap. The payload is then divided into individual float values, which are integrated together with other received segments according to their positions using the bitmap information. This approach allows each device to accurately integrate parameter updates and offer a structured and efficient way to merge them with the device's local model for the aggregation phase.

5 Evaluation

We evaluate Gist on different environments and under different conditions, and compare it with state-of-the-art methods.

▹ **Baselines.** We select several methods for evaluation. We implement them focusing on contributions to training, segmentation, and aggregation. The baselines include:

① **Centralized Method:** Aggregates all model parameters from all devices, averages them, and obtains a global model. We then load this global model back to each device. This setup imitates traditional FL methods with a central server and ensures efficient aggregation of all learned information.

② **Decentralized Federated Averaging (DFA) [26]** and ③ **Segmented DFA (SDFA):** DFA uses a basic decentralized setup without segmentation or adaptive mechanisms. It performs full model updates. We also implement Segmented DFA using uniformly distributed randomized segmentation as a custom version of DFA for better comparison.

④ **Segmented Pulling (SP) [9]:** Requests segments from other devices after segmentation. It uses a system for devices to pull specific segments of model parameters from their neighbors and create model replicas from received segments. We additionally implement and include AdaStair [31], a method that dynamically adjusts the local learning rate of a device based on the round that the device is in, and sends segments randomly. AdaStair complements our approach.

▹ **Models.** Gist and our implementation are model-agnostic and support various models. We select models that can be easily trained and run on microcontrollers, and are supported by the Machine Learning (ML) framework AIfES [32], including a model with Fully-Connected Network (FCN) layers and more complex LeNet5 [15] and MobileNet [8] models.

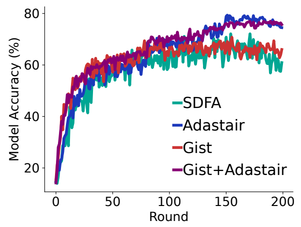


Figure 6: Accuracy on real tiny devices.

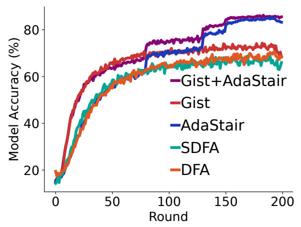
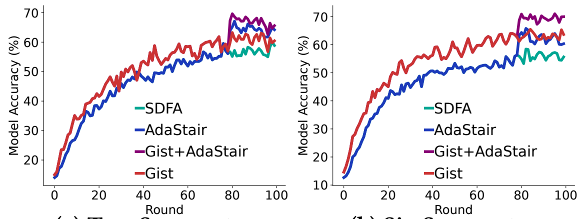


Figure 7: Emulation results over 200 devices.



(a) Two Segments **(b) Six Segments**

Figure 8: Impact of different number of segments.

▹ **Dataset.** We use multiple datasets deployable on tiny microcontroller devices, including MNIST [7] and CIFAR-10 [14], while adjusting the model inputs. We evaluate different data distribution scenarios for each dataset, including *i.i.d.*, uniform non-*i.i.d.*, and Dirichlet-driven distributions. We configure each device with 600 images available locally, and use 150 images from this dataset for training in each experiment.
▹ **Infrastructure.** We perform evaluation on three different experimental environments [1]:
① **Simulation:** A modular and expandable framework around a control loop for faster testing without physical hardware limitations. The goal is to imitate the real-life environment of a DFL system yet faster in testing and development.
② **Emulation:** A network of ESP32-S3 emulated devices was setup that we developed based on QEMU [4, 24] to test the practical feasibility of utilizing DFL on IoT hardware with the compiled firmware and realistic system constraints.
③ **Real Hardware:** We use up to ten ESP32-S3-DevKitC-1 boards for real-world validation.
▹ **Metric.** Test accuracy of the local models after each communication round is the primary evaluation metric used in the framework. For better comparison, we run each configuration with different seeds and compare their average.

5.1 Results

Accuracy on different models and datasets. We evaluate Gist in two different settings: with and without the addition of the Adastair method which is complementary to our proposal. Figure 4 shows accuracy through time over the course of maximum 200 asynchronous update rounds for a fully-connected model on MNIST, MobileNet on CIFAR-10, and LeNet5 on MNIST. Across the board, Gist achieves 4-5% higher accuracy over time and after convergence against the best method. When combined with Adastair, the gap grows larger to up to 20% on different infrastructures and tasks.
Accuracy on the network of real tiny devices. Figure 6 depicts convergence over time on a network of 10 microcontrollers. The results show similar trends comparative to the simulation experiments, where our segmentation proposal achieves the best accuracy on average and after convergence.
Performance at scale. To investigate the performance of different methods at scale while facing real-world constraints and condition, we performed DFL on our emulation setup with 200 interconnected devices. Figure 7 shows a clear cut between Gist and the rest when the number of participating devices increase by an order of magnitude, further highlighting its value in real-world scenarios where the number of devices may grow to a few hundreds to thousands.
Impact of our success-based aggregation. To understand the impact of our success-based aggregation mechanism, we evaluate the convergence trend with and without it. As in Fig. 5, it always outperforms the vanilla aggregation. The difference becomes more evident after the initial 25 rounds.
Fewer segments or more? We evaluate the methods that utilize segmentation strategies for two different segmentation counts, two and six segments (Fig. 8). While Segmented DecFedAvg and AdaStair perform better when devices send a larger portion of their model parameters, 50% in this case, Gist's performance improves with higher number of segments, demonstrating the efficiency of our importance-based segmentation algorithm.

6 Conclusion

This paper introduced Gist, a DFL method for resource-constrained tiny IoT devices. By synergistically combining our importance-based parameter segmentation, probabilistic sharing, and a success-based aggregation mechanism, Gist efficiently prioritizes the exchange of impactful model updates. This design ensures that information is propagated effectively to accelerate convergence, while maintaining model diversity by periodically sharing less important parameters. Our evaluation, spanning simulations of different models on different tasks, large-scale realistic emulation, and deployment on a physical cluster of ESP32-S3 microcontrollers, validates the effectiveness of our approach. Across various models and datasets, Gist consistently outperforms established baselines, achieving higher accuracy and faster convergence, particularly in scaled-up scenarios. The results confirm that our method is not only scalable but that its core components—notably the success-based aggregation and fine-grained, importance-aware segmentation—are key features. By enabling more efficient and robust collaborative learning on tiny devices, Gist represents a step toward the practical realization of decentralized tiny ML.

References

[1] Navidreza Asadi, Halil Ibrahim Bengü, Lars Wulfert, Hendrik Wöhrle, and Wolfgang Kellerer. 2025. Poster: Road to Tiny Reality: Digital Twins for Decentralized AI on Microcontrollers. In *Proceedings of the 31st Annual International Conference on Mobile Computing and Networking (MOBICOM '25)*. ACM. doi:10.1145/3680207.3765668

[2] Navidreza Asadi and Maziar Goudarzi. 2023. Variant parallelism: lightweight deep convolutional models for distributed inference on IoT devices. *IEEE Internet of Things Journal* 11, 1 (2023), 345–352.

[3] Tuncer Can Aysal, Mark J. Coates, and Michael G. Rabbat. 2008. Distributed Average Consensus With Dithered Quantization. *IEEE Transactions on Signal Processing* 56, 10 (2008), 4905–4918. doi:10.1109/TSP.2008.927071

[4] Fabrice Bellard. 2005. QEMU, a fast and portable dynamic translator.. In *USENIX annual technical conference, FREENIX Track*, Vol. 41. California, USA, 10–55.

[5] Enrique Tomás Martínez Beltrán, Mario Quiles Pérez, Pedro Miguel Sánchez Sánchez, Sergio López Bernal, Gérôme Bovet, Manuel Gil Pérez, Gregorio Martínez Pérez, and Alberto Huertas Celdrán. 2023. Decentralized federated learning: Fundamentals, state of the art, frameworks, trends, and challenges. *IEEE Communications Surveys & Tutorials* 25, 4 (2023), 2983–3013.

[6] Soumith Chintala, Gregory Chanan, Dmytro Dzhulgakov, Edward Yang, Nikita Shulga, and The PyTorch Team. 2025. PyTorch: Tensors and Dynamic neural networks in Python with strong GPU acceleration. https://github.com/pytorch/pytorch. GitHub repository, version as of July 23, 2025.

[7] Li Deng. 2012. The MNIST Database of Handwritten Digit Images for Machine Learning Research [Best of the Web]. *IEEE Signal Processing Magazine* 29, 6 (2012), 141–142. doi:10.1109/MSP.2012.2211477

[8] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. 2017. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. arXiv:1704.04861 [cs.CV] https://arxiv.org/abs/1704.04861

[9] Chenghao Hu, Jingyan Jiang, and Zhi Wang. 2019. Decentralized Federated Learning: A Segmented Gossip Approach. arXiv:1908.07782 [cs.LG] https://arxiv.org/abs/1908.07782

[10] Hossein Katebi, Navidreza Asadi, and Maziar Goudarzi. 2024. FullPack: Full Vector Utilization for Sub-Byte Quantized Matrix-Vector Multiplication on General Purpose CPUs. *IEEE Computer Architecture Letters* 23, 2 (2024), 142–145.

[11] Anastasia Koloskova, Sebastian U. Stich, and Martin Jaggi. 2019. Decentralized Stochastic Optimization and Gossip Algorithms with Compressed Communication. arXiv:1902.00340 [cs.LG] https://arxiv.org/abs/1902.00340

[12] Jakub Konečnỳ, H Brendan McMahan, Felix X Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. 2016. Federated learning: Strategies for improving communication efficiency. *arXiv preprint arXiv:1610.05492* (2016).

[13] Jakub Konečný, H. Brendan McMahan, Felix X. Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. 2017. Federated Learning: Strategies for Improving Communication Efficiency. arXiv:1610.05492 [cs.LG] https://arxiv.org/abs/1610.05492

[14] Alex Krizhevsky. 2009. Learning Multiple Layers of Features from Tiny Images. (2009), 32–33. https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf

[15] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324. doi:10.1109/5.726791

[16] Sunwoo Lee, Tuo Zhang, Chaoyang He, and Salman Avestimehr. 2022. Layer-wise Adaptive Model Aggregation for Scalable Federated Learning. arXiv:2110.10302 [cs.LG] https://arxiv.org/abs/2110.10302

[17] Jian Li, Tongbao Chen, and Shaohua Teng. 2024. A comprehensive survey on client selection strategies in federated learning. *Computer Networks* 251 (2024), 110663. doi:10.1016/j.comnet.2024.110663

[18] Jun Liu, Jianchun Liu, Hongli Xu, Yunming Liao, Zhiyuan Wang, and Qianpiao Ma. 2024. YOGA: Adaptive Layer-Wise Model Aggregation for Decentralized Federated Learning. *IEEE/ACM Transactions on Networking* 32, 2 (2024), 1768–1780. doi:10.1109/TNET.2023.3329005

[19] Sébastien Marcel and Yann Rodriguez. 2010. Torchvision the machine-vision package of torch. In *Proceedings of the 18th ACM international conference on Multimedia*. 1485–1488.

[20] Enrique Tomás Martínez Beltrán, Mario Quiles Pérez, Pedro Miguel Sánchez Sánchez, Sergio López Bernal, Gérôme Bovet, Manuel Gil Pérez, Gregorio Martínez Pérez, and Alberto Huertas Celdrán. 2023. Decentralized Federated Learning: Fundamentals, State of the Art, Frameworks, Trends, and Challenges. *IEEE Communications Surveys and Tutorials* 25, 4 (2023), 2983–3013. doi:10.1109/comst.2023.3315746

[21] H. Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. 2023. Communication-Efficient Learning of Deep Networks from Decentralized Data. arXiv:1602.05629 [cs.LG] https://arxiv.org/abs/1602.05629

[22] Hesham Mostafa and Xin Wang. 2019. Parameter Efficient Training of Deep Convolutional Neural Networks by Dynamic Sparse Reparameterization. arXiv:1902.05967 [cs.LG] https://arxiv.org/abs/1902.05967

[23] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems* 32 (2019).

[24] QEMU Project. 2025. QEMU: A Generic and Open Source Machine Emulator and Virtualizer. https://www.qemu.org/. Version 10.0.2, accessed July 23, 2025.

[25] Minh K Quan, Pubudu N Pathirana, Mayuri Wijayasundara, Sujeeva Setunge, Dinh C Nguyen, Christopher G Brinton, David J Love, and H Vincent Poor. 2025. Federated learning for cyber physical systems: a comprehensive survey. *IEEE Communications Surveys & Tutorials* (2025).

[26] Tao Sun, Dongsheng Li, and Bao Wang. 2023. Decentralized Federated Averaging. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 45, 4 (2023), 4289–4301. doi:10.1109/TPAMI.2022.3196503

[27] Zhenheng Tang, Shaohuai Shi, Bo Li, and Xiaowen Chu. 2023. GossipFL: A Decentralized Federated Learning Framework With Sparsified and Adaptive Communication. *IEEE Transactions on Parallel and Distributed Systems* 34, 3 (2023), 909–922. doi:10.1109/TPDS.2022.3230938

[28] Xinghan Wang, Anusha Lalitha, Tara Javidi, and Farinaz Koushanfar. 2022. Peer-to-Peer Variational Federated Learning Over Arbitrary Graphs. *IEEE Journal on Selected Areas in Information Theory* 3, 2 (2022), 172–182. doi:10.1109/JSAIT.2022.3189051

[29] Pijika Watcharapichat, Victoria Lopez Morales, Raul Castro Fernandez, and Peter Pietzuch. 2016. Ako: Decentralised Deep Learning with Partial Gradient Exchange. In *Proceedings of the Seventh ACM Symposium on Cloud Computing* (Santa Clara, CA, USA) *(SoCC '16)*. Association for Computing Machinery, New York, NY, USA, 84–97. doi:10.1145/2987550.2987586

[30] Feijie Wu, Xingchen Wang, Yaqing Wang, Tianci Liu, Lu Su, and Jing Gao. 2024. FIARSE: Model-Heterogeneous Federated Learning via Importance-Aware Submodel Extraction. arXiv:2407.19389 [cs.DC] https://arxiv.org/abs/2407.19389

[31] Lars Wulfert, Navidreza Asadi, Wen-Yu Chung, Christian Wiede, and Anton Grabmaier. 2023. Adaptive Decentralized Federated Gossip Learning for Resource-Constrained IoT Devices. In *Proceedings of the 4th International Workshop on Distributed Machine Learning* (Paris, France) *(DistributedML '23)*. Association for Computing Machinery, New York, NY, USA, 27–33. doi:10.1145/3630048.3630181

[32] Lars Wulfert, Johannes Kühnel, Lukas Krupp, Justus Viga, Christian Wiede, Pierre Gembaczka, and Anton Grabmaier. 2024. AIfES: A Next-Generation Edge AI Framework. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 46, 6 (2024), 4519–4533. doi:10.1109/TPAMI.2024.3355495

[33] Mang Ye, Xiuwen Fang, Bo Du, Pong C Yuen, and Dacheng Tao. 2023. Heterogeneous federated learning: State-of-the-art and research challenges. *Comput. Surveys* 56, 3 (2023), 1–44.

[34] Liangqi Yuan, Ziran Wang, Lichao Sun, Philip S. Yu, and Christopher G. Brinton. 2024. Decentralized Federated Learning: A Survey and Perspective. arXiv:2306.01603 [cs.LG] https://arxiv.org/abs/2306.01603