Client-Level Unlearning in Decentralized Learning

Robust Decentralized Learning

Razvan Dinu
Supervisors: Bart Cox, Jérémie Decouchant
EEMCS, Delft University of Technology, The Netherlands

A Thesis Submitted to the EEMCS Faculty of Delft University of Technology,
In Partial Fulfilment of the Requirements
For the **Bachelor of Computer Science and Engineering**
June 22, 2025

Name of the student: Razvan Dinu
Final project course: CSE3000 Research Project
Thesis committee: Bart Cox, Jérémie Decouchant, Anna Lukina

Client-Level Unlearning in Decentralized Learning

Razvan Dinu, Bart Cox, Jérémie Decouchant
Delft University of Technology

***Abstract*—Decentralized Learning is becoming increasingly popular due to its ability to protect user privacy and scale across large distributed systems. However, when clients leave the network, either by choice or due to failure, their past contributions remain in the model. This raises privacy concerns and may violate the right to be forgotten. In some applications, it is also undesirable to retain the outdated influence of clients that no longer reflect the current state of the system. While Machine Unlearning has seen significant progress in Federated Learning, similar solutions for Decentralized Learning are limited because there is no central server to orchestrate these operations. This work adapts and extends a state-of-the-art Federated Unlearning algorithm, QuickDrop, to operate in a decentralized setting. Our method uses synthetic data to reverse the influence of dropped clients and efficiently restore the model's generalization performance. It also supports unannounced client crashes and performs reliably in sparse network topologies. We evaluate the algorithm on MNIST and CIFAR-10 using different graph structures and show that it remains competitive with oracle baselines that require access to sensitive data. Finally, we discuss the limitations of our approach and suggest directions for future work in Decentralized Unlearning.**

I. Introduction

In traditional machine learning, models are trained in centralized environments where data is aggregated in one place [1]. While effective, this approach might raise significant concerns related to privacy, communication bottlenecks, and scalability. Distributed Learning offers a promising alternative, in which multiple nodes collaboratively train a model without the need to store all the data on a central server. Each node uses its local data and communicates only with a limited set of peer nodes.

A well-known subset of this field is Federated Learning (FL) [2, 3], which uses a central node to coordinate model updates. In contrast, Decentralized Learning (DL) (sometimes referred to as Gossip Learning [4]) operates on a peer-to-peer level without any central authority, which is what this research focuses on. This setting can increase robustness [5, 6] and privacy [7, 8], but it also introduces new obstacles such as synchronization, security, and adaptability [9–12]. A shared challenge is communication, where FL makes it more accessible and auditable, while DL increases its complexity, but greatly lowers cost for large systems.

DL also permits dynamic and non-permanent participation, commonly referred to as churn [13]. A critical issue that remains under-explored is how to handle clients that drop out permanently. When a client departs, either due to failure or by choice, its prior model contributions may continue to influence other nodes indirectly, possibly without that client's consent. This persistent influence raises both privacy and adaptability concerns.

Machine Unlearning (MU) [14, 15] has grown in interest with the introduction of privacy-oriented regulations such as the European Union's General Data Protection Regulation (GDPR) [16], which gives all users, and, in turn, clients of DL systems, *the right to be forgotten*. Additionally, one might also be interested in unlearning the influence of a client when it is paramount that no old data is used to train the model.

While MU has recently gained traction in centralized and federated contexts [17], few existing methods directly tackle unlearning in decentralized systems [10], which rely on impractical assumptions, such as restricting a client's influence to only its immediate neighbours. Moreover, the existing body of work does not sufficiently address the intersection of client crashes and unlearning. This research aims to bridge that gap by proposing a decentralized mechanism to remove the influence of dropped clients in a gossip-style learning framework.

This paper makes the following contributions:

1) We translate a current state-of-the-art (SOTA) machine unlearning algorithm from Federated to Decentralized Learning, overcoming the architectural differences between the two systems by fine-tuning its parameters. We focus on unlearning the influence of a particular client and report the generalization capabilities of the remaining model.
2) We improve on the existing SOTA by considering crashed clients and further tuning synthetic data generation.
3) We analyse the impact of the network topology, datasets and disconnection frequency w.r.t. the unlearning efficiency in DL.
4) We provide a complete implementation of our decentralized unlearning algorithm with support for different network topologies and crash recovery, evaluating its practical effectiveness against established theoretical benchmarks.

II. Background

A. Machine Unlearning

We define the dataset $D_f \subset D$, where D is the entire available training dataset, as the *forget* dataset (F-set), meaning the subset of data points that we wish to unlearn from the global model θ. The rest of the data $D \setminus D_f$ can be referred to as the *retain* set (R-set) [18]. The goal of efficient MU is to remove the influence of D_f on the model parameters θ, while retaining a high accuracy using just $D \setminus D_f$. In other words, after unlearning D_f, θ should maintain its global generalization accuracy while performing poorly on D_f.

MU can be classified into two main categories: exact [19] and approximate [20] unlearning. Exact unlearning implies that the influence of a client is fully unlearned, while in approximate unlearning, this influence is only minimized. The optimal degree of unlearning is achieved by retraining from scratch (resetting the model to its initial state), but this can lead to unnecessary increases in training time. Since in decentralized learning one node can affect all others in the network, directly or indirectly, exact unlearning is considered difficult to achieve [21], unless specific assumptions are made.

Unlearning can also be split depending on the contents of D_f:

1) **Class-level unlearning [22]:** The goal is to erase a specific class c, for example all instances of 0 for a digit dataset like MNIST. It follows that $D_f := \bigcup_{i \in N} D_i^c$, i.e. all instances of class c for each client $i \in N$, where N is the pool of clients.
2) **Sample-level unlearning [23]:** We unlearn a particular set of samples. Here, there is no formal definition of D_f, as it can contain any arbitrary samples from the original dataset D.
3) **Client-level unlearning:** This is specific to systems with more than one client, and $D_f := D_i$ for some client $i \in N$.

Our research focuses on client-level unlearning, as it is most applicable in practice to a DL environment. Because each client can only access a pre-established partition of the dataset D, when mentioning D or $D \setminus D_f$ we refer to the training partitions of each client. The test set is global and fixed, being created before clients receive their training data [24].

B. *Decentralized Learning*

One of the main differences between FL and DL is the way nodes communicate. In FL all nodes send direct updates to a central server, which tells the nodes what workflow to follow. This makes any communication between two nodes require at most two data transfers. However, in DL a node can only contact its direct neighbours, which increases the amount of data that needs to be sent through the network in order for two arbitrary clients to communicate. We consider the unlearning process successful if the global model has a lower testing accuracy on the dropped node's test set, comparable with the one of retraining from scratch. It should also be comparable in unlearning speed with the proposed baselines.

III. Related work

In current research, Federated Unlearning (FU) has been widely studied, while Decentralized Unlearning (DU) remains largely unexplored. However, not all FU algorithms can be directly translated to a decentralized setting due to the systems' design differences, especially in communication between nodes. This section surveys the SOTA algorithms for FU and DU. Table I gives a visual overview of what each presented algorithm lacks with respect to the problem at hand.

A. *Federated Unlearning*

Dhasade et al. [17] propose QuickDrop, a method for efficiently unlearning a client's influence without full retraining. Each node maintains a synthetic dataset [28], generated during each round alongside regular training, which enables fast gradient approximation. Unlearning is performed using client-level Stochastic Gradient Ascent (SGA) on the synthetic data of the dropped client, followed by recovery rounds using the one's personal synthetic dataset to partially restore its previous knowledge.

Since training typically uses Stochastic Gradient Descent (SGD), it seems straightforward to apply SGA for unlearning. However, the direct approach by Wu et al. [25] has two key limitations. First, it requires access to the full remaining dataset, which may be large and inefficient to process. Second, client-level unlearning is not directly supported and must be adapted using Elastic Weight Consolidation (EWC), which assigns importance weights to parameters and defines an unlearning loss for SGA. Moreover, applying SGA to another client's data is not feasible in DL, where data sharing is restricted. FL avoids this by performing the ascent steps on the central server.

Halimi et al. [26] present a related method using Projected Gradient Descent (PGD), where unlearning is approximated by maximizing local empirical loss within an $l2$-norm ball around a reference model, computed as the average of other clients' models. This constrains updates to prevent divergence and applies early stopping based on distance from the original model. The method achieves approximate unlearning based on empirical metrics such as efficacy, fidelity, and efficiency. However, since it requires centralized coordination and access to the global model and previous updates, it does not translate to decentralized learning.

Wu et al. [27] also propose a method using Knowledge Distillation (KD), in which the server tracks parameter updates. When a client requests unlearning, its updates are subtracted from the model, and the skew is corrected via distillation. This method is efficient since all operations are server-side, but it is incompatible with decentralized settings where updates cannot be stored centrally and communication costs would be high.

B. *Decentralized Unlearning*

To the best of our knowledge, the only successful attempt at DU is HDUS [10]. HDUS's approach is to use seed models without sensitive information, which are smaller models distilled from the model of a client and its neighbours, mimicking behaviour of the main model. The global model is trained via ensemble learning [29] based on the aforementioned seed models, which has the potential of improving robustness and reducing overfitting [30]. Unlearning the influence of a particular client is then equivalent to removing the seed model from the ensemble. This approach assumes that one node can only influence its direct neighbours, and that a client has sufficient non-sensitive data to efficiently train its neighbours' seed models. However, it does provide a promising framework for exact unlearning.

TABLE I: Comparison of related work highlighting the limitations in a decentralized setting. Columns refer to the unlearning algorithm itself, excluding the training process.

Algorithm	Decentralized Learning	Extends to Entire Network	Scalable to Large Networks	Storage Efficient	Computation Efficient	Adaptable to Client Crashes
QUICKDROP [17]	✗	✓	✓	✓	✓	✗
SGA [25]	✗	✓	✗	✓	✗	✗
PGD [26]	✗	✓	✗	✓	✗	✗
KD [27]	✗	✓	✗	✗	✗	✗
HDUS [10]	✓	✗	✓	✗	✗	✗
This Work	✓	✓	✓	✓	✓	✓

IV. SYSTEM MODEL

A. Nodes and Topologies

In DL, each node is an independent participant that holds a private portion of the training data and maintains a local copy of the model. During each training round, a node first performs local training on its data using techniques like SGD [11], then exchanges model weights with its direct neighbours in the network. After receiving updates, it averages its model with those of its peers to improve generalization. This process is repeated over many rounds. The main purpose of each node is to contribute to the collective learning process without sharing raw data, enabling privacy-preserving and scalable training across a distributed system. When defining the network of a DL experiment, we consider undirected graphs (Appendix A), where each client is a node in the graph. If client i sends its weights to client j during sharing, so does client j to client i, for any i, j in the network N. This simplifies the complexity of the algorithm, while aligning with other DL works [11, 24, 31, 32].

B. General Assumptions

When a client disconnects, we assume for simplicity that they do not leave any nodes isolated or cause deadlocks. While this assumption helps simplify the analysis, it is not a strict limitation in practice. In real-world decentralized systems, mechanisms such as peer sampling [24, 33] can be used to dynamically reconnect nodes and maintain network connectivity. Moreover, we assume that the network topology is static and does not change over time, with the exception of the chosen clients leaving the network for unlearning. We do not de-synchronize clients during learning because weight updates are done after each global round during regular DL training. We also assume that the global dataset is split into a global test set, shared between all clients, and a trainset which is split between the clients according to a known distribution. Because in practice not all clients hold the same distribution of data, we consider our system's clients' data to be non-IID (also referred to as heterogeneous [10, 30]), and assume a Dirichlet distribution ($\alpha = 0.1$) of data among clients [34]. Due to GPU memory constraints, we reduced the batch size from the default 256 [17, 28] to 128, 64 or even 8 in some experiments, which could marginally affect results.

V. METHODOLOGY

As Dhasade et al. [17] showed QUICKDROP to currently be the most efficient unlearning algorithm for FL, and since it can be translated to DL, this work closely follow theirs, but attempts to move it to a decentralized setting, with a stronger focus on client-level unlearning.

A. Synthetic Data Generation

The core of this algorithm stems from the ability of each client to condense their dataset into a smaller, synthetic dataset, that accurately represents their original data with minimal storage overhead. This is also known as Dataset Distillation (DD) [35]. To achieve this, the chosen method was inspired by QUICKDROP [17], which in turn took inspiration from the *gradient matching* algorithm of Zhao et al. [28]. This uses a distance function between the gradients of each layer of the neural network as its loss function:

$$S_i \leftarrow \text{opt-alg}_{S_i}\left(d\left(\nabla_\theta \mathcal{L}^{S_i}(\theta^i_{k,t}), \nabla_\theta \mathcal{L}^{D_i}(\theta^i_{k,t})\right), \varsigma_S, \eta_S\right) \quad (1)$$

where ς_S and η_S are the number of steps and learning rate, $\mathcal{L}^D(\theta)$ is the loss of model θ on dataset D, and $d(.,.)$ is the distance function proposed by Zhao et al. in their work [28].

Unlike QUICKDROP, we also leverage Differentiable Siamese Augmentation (DSA) [36], improving the masking of real data and increasing the quality of synthetic samples, as Zhao et al. [37] argue it can increase the efficiency of gradient matching. The preferred choice of gradient matching stems from the fact that the unlearning logic of the algorithm performs SGA. As argued in the description of QUICKDROP, the synthetic data should provide a gradient as close as possible to the real one, such that the ascent operation is in the desired direction. For the size of the synthetic dataset, we use a scale parameter s such that $|S^c_i| = \lceil |D^c_i| \times s \rceil$, i.e., a fraction s of the real samples from that class, but at least one.

B. General Workflow

We present the workflow of the algorithm in Algorithm 1, with a detailed explanation of each step below:

Tune synthetic data. In the proposed algorithm, each client $i \in N$ holds a synthetic dataset (S_i), which acts like a condensed version of their real dataset, without containing any sensitive data. This dataset is generated on the first global training round and tuned for all remaining rounds. Every few rounds, clients send their synthetic data to their neighbours for them to cache it in case they leave the network unexpectedly.

Perform SGA on synthetic data. Before a client $j \in N$ disconnects, they may send out an unlearning request to its direct neighbours, who then forward it recursively to the rest of the network each via their own neighbours. The disconnecting client passes its most recent synthetic dataset (S_j) as a payload of this message, which will be used by others for unlearning via SGA. Since learning is achieved using Decentralized Parallel Stochastic Gradient Descent (D-PSGD) [11, 24], this makes for a promising approach to effectively *reverse* the learning process, with a strong focus on removing the influence of the dropped node's data ($D_f = D_j$).

Augment synthetic datasets with real samples. After unlearning, each client takes its own synthetic dataset and prepares it for recovery by adding random samples from their real dataset to their synthetic data. The quantity of random samples is determined by the size of the synthetic dataset, such that the number of real samples in the set will be equal to the number of synthetic samples. Since this data can be sensitive, the real samples are only kept for the upcoming recovery step, after which the synthetic dataset is reverted to its previous state. We go a step further and augment the dataset with new random samples for each recovery step, allowing for better relearning of past knowledge.

Recovery rounds. Because only performing unlearning might not yield effective results [25], some recovery rounds are needed, in which we aim to find out how fast the model can reach a similar testing accuracy to its pre-unlearning state, while forgetting D_f. Normally, recovery is done by default when training resumes, but we accelerate them by using the previously augmented synthetic dataset.

VI. Experimental Setup

System Setup

All experiments are conducted on a machine using an AMD Ryzen Threadripper 1900X 8-core CPU and an RTX 3060 GPU with 12 GB of memory running CUDA 12.6.3. We use Python 3.12.3 with PyTorch and TensorFlow among other libraries for the implementation. All plots are done using Matplotlib.

The source code is made available in a TU Delft-hosted GitLab repository[1].

Decentralized Learning

Our environment is built using the DecentralizePy [24] DL framework, which allows for many configurations out of the box. In our configuration, nodes communicate via TCP and execute learning via D-PSGD [11], while weight sharing is done by Metropolis-Hastings averaging [38]. We execute the code on a single machine, running a number of processes equal to the number of clients, each client with one process. Typically, clients will announce their disconnection, in which case all others attempt to begin unlearning at the same round,

[1]See https://gitlab.tudelft.nl/cse3000-2025-robust-decentralized-learning/cse-3000-razvan-dinu/.

Algorithm 1 DL Training with Unlearning and Recovery

```
Input: Training data $\{D_i\}_{i=1}^N$, rounds $K$, local steps $T$, tuning
       steps $\varsigma_S$, learning rates $\eta_S, \eta_\theta, \eta'_\theta$, scale $s$
for each client $i = 1, \dots, N$ do
    for each class $c$ do
        if client $i$ has class $c$ then
            $S_i^c \leftarrow \lceil |D_i^c| \times s \rceil$ random samples from $D_i^c$
        else
            $S_i^c \leftarrow \emptyset$
    $S_i \leftarrow \bigcup_c S_i^c$
for $k = 0, \dots, K-1$ do
    for each client $i = 1, \dots, N$ do in parallel
        for $t = 0, \dots, T-1$ do
            Sample minibatches $B_i^D \sim D_i$, $B_i^S \sim S_i$
            Compute $\nabla\mathcal{L}_{D_i}(\theta_{k,t}^i)$, $\nabla\mathcal{L}_{S_i}(\theta_{k,t}^i)$
            Update $S_i$ using $d(\nabla\mathcal{L}_{D_i}, \nabla\mathcal{L}_{S_i})$ {Eq. (1)}
            if phase = train then
                $\theta_{k,t+1}^i \leftarrow \theta_{k,t}^i - \eta_\theta \nabla\mathcal{L}_{D_i}(\theta_{k,t}^i)$
            else if phase = unlearn$_j$ then
                $\theta_{k,t+1}^i \leftarrow \theta_{k,t}^i + \eta'_\theta \nabla\mathcal{L}_{S_j}(\theta_{k,t}^i)$
            else if phase = recover then
                $\theta_{k,t+1}^i \leftarrow \theta_{k,t}^i - \eta_\theta \nabla\mathcal{L}_{S_i}(\theta_{k,t}^i)$
    $\theta_{k+1} \leftarrow \sum_{i=1}^{N} \frac{|D_i|}{|D|} \theta_{k,T}^i$
Output: Synthetic datasets $\{S_i\}_{i=1}^N$, final model $\theta_{K,0}$
```

specified by the one that leaves. In case of an unexpected crash, we implement a timeout mechanism of 10 minutes: if after this time a neighbour does not send its weights at the end of a training round, we consider them crashed and begin unlearning on their data. This crash is simulated by ending the execution immediately.

We evaluate the algorithm on the MNIST [39] and CIFAR-10 [40] datasets, partitioned among clients using a Dirichlet distribution with $\alpha = 0.1$. The client to disconnect is chosen such that their number of training data points is close to the average. Each client uses a ConvNet [41] as its deep neural network. It consists of D duplicate blocks, where each block comprises a convolutional layer with W filters of size 3×3, followed by a normalization layer N, an activation function A, and a pooling layer P. This structure is represented as $[W, N, A, P] \times D$. The default ConvNet uses 3 such blocks, each containing 128 filters, InstanceNorm, ReLU activation, and AvgPooling. A linear classifier is applied after the final block. Images are reshaped to 32×32 size for standardization.

We consider three different *static* topologies (see Appendix A for visual representations):

- A **16-node 3-regular graph**, simulating a sparse DL environment.
- A **16-node 4-regular graph**, a more connected but still network-efficient DL environment.
- A **10-node fully-connected graph**, a less realistic DL

environment, but theoretically equivalent to FL when it comes to convergence rate, allowing for better comparisons with related work.

We primarily experiment on the 3-regular graph and use the others to show the influence of the connectivity of the graph and better benchmark against FL algorithms.

Hyperparameters

For all experiments, unless specified otherwise, we pretrain a model for $K = 400$ global training rounds and up to $T = 50$ local steps per round (stops early if a full epoch has been achieved). We use a batch size of 128 for training and unlearning. We use SGD optimizers with learning rates η_θ for training and 0.02 and η'_θ for unlearning 0.02 for MNIST and CIFAR-10, as well as a single optimization step for each ($\varsigma_\theta = \varsigma_{\theta'} = 1$). We perform 2 rounds of unlearning and 3 rounds of recovery. We found these to be a good balance between unlearning speed and forgetting efficiency for most experiments, but in Section VIII we argue how some topologies might require or allow for different unlearning hyperparameters.

Synthetic Data Generation

To generate synthetic data, we train the synthetic images along with the local model (i.e., for up to $T = 50$ for every global iteration k). This happens in the same training loop as the real data's. We use SGD as the optimization algorithm with learning rate $\eta_S = 0.1$ and one optimization step ($\varsigma_S = 1$). We set the scale parameter $s = 0.02$, such that the synthetic dataset is 2% of the real dataset of a client, allowing for minimal storage overhead. Synthetic samples are generated from random samples of real data, and we use the same distance function d as Zhao et al. [28] as the matching loss between the real and synthetic data. For DSA [36, 37] we use the *color*, *crop*, *cutout*, *flip*, *scale*, *rotate* methods for CIFAR-10 and exclude *flip* for MNIST, since flipping is not suitable for digit datasets, as argued by their work.

VII. Performance Evaluation

We report the average Top-1 accuracy of all clients on the global test set rounded to two decimals. For the F-set accuracy we consider the accuracy on the training dataset of the client that disconnects. We reconstruct this set locally using DecentralizePy's **mapping** module [24]. Since it is not realistic that other clients can access the data of others, we only use this set to assess the efficiency of the algorithm. The algorithm does not take this accuracy into account for learning. DecentralizePy views testing accuracy as the generalization capability of the model on the global dataset. In other words, the test set is global and the same for all clients. It is unchanged throughout execution, which remains the case for this project as well. As such, we report R-set accuracy as the testing accuracy of the model on the global test set, even if the remaining dataset $D \setminus D_f$ no longer holds samples of a particular class. Because of this, we expect higher F-set accuracies and lower R-set accuracies than what other FL SOTA algorithms report, which dynamically change the test set throughout learning.

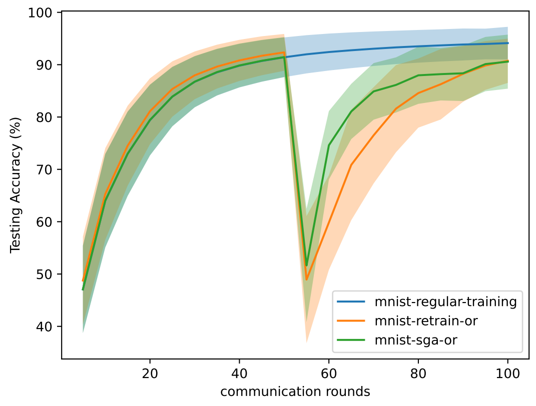


Fig. 1: Average global testing accuracies of the clients in the 16-node 3-regular network trained on the MNIST dataset. Comparison of 3 scenarios: regular training, Retrain-Or, and SGA-Or. Client 5 disconnects at iteration 50. Standard deviation is shown by the shaded area.

A. *Baselines*

We consider two baseline algorithms:

1) **Retraining from scratch.** Upon receiving an unlearning request, the Retrain-Or oracle resets all remaining local models and optimizers and retrains from scratch, thus acting like the dropped node never existed. The number of recovery rounds of Retrain-Or is equal to the number of training rounds prior to unlearning.
2) **SGA.** The SGA-Or oracle follows a similar approach to the proposed algorithm, but uses real training data instead of synthetic data. This is not applicable in practice for DL systems because it requires knowledge of the training dataset of the dropped node, but it provides a solid baseline for the effectiveness of using synthetic data.

Figure 1 exemplifies these algorithms, while other topologies and datasets being considered in Subsection VII-C.

B. *Quality of Synthetic Data*

Each client trains a dataset of synthetic images that should represent a condensed version of the original data as closely as possible. To assess the quality of this synthetic data, we employ two different comparison methods:

Training on synthetic data. We train a new DL model using only synthetic data and compare its global test accuracy to a model trained on real data. Despite the synthetic data being optimized for short unlearning rounds, its performance is comparable to that of real data, at just 2% of the size of the original dataset (see Fig. 2). It does have some difficulty in holding a stable high accuracy, which we attribute to the small dataset size.

Fréchet inception distance (FID) [42]. To measure the quality of the synthetic images, we also use the FID score. This metric compares the feature representations of real and

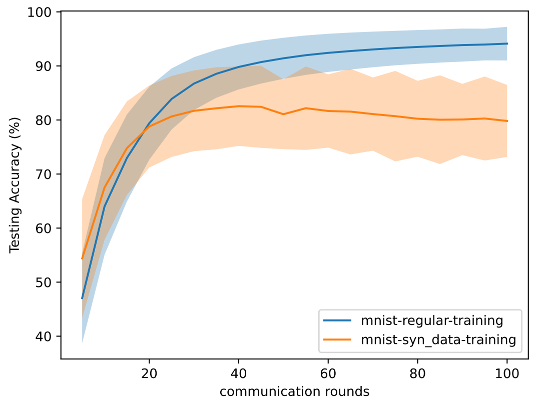


Fig. 2: Comparison on testing accuracies of a model trained on the real dataset and a model trained on the synthetic dataset, tested on the global test set. The original dataset is MNIST, and the topology is a 3-regular 16-node graph.

generated images, extracted using a pre-trained Inception v3 model. It calculates the distance between the two sets of features to estimate how similar they are. A lower FID score means the synthetic images are closer to the real ones in terms of visual features, with a score of 0.0 indicating perfect similarity. After training for 100 rounds, a 16-node 3-regular DL topology reaches an FID score of 1.57 for MNIST and 487.2 for CIFAR-10, using 1000 images from the real data to calculate it. Due to the high complexity and variety of CIFAR-10, the small training time and sample size might prove insufficient to fully assess the feature similarity between the real and synthetic datasets.

TABLE II: The mean total training and DD compute time for clients training on MNIST and CIFAR-10 after 115 iterations with one unlearning request. Calculated for a 3-regular 16-node network.

Dataset	**Total Train Time (s)**	**DD Compute Time (s)**	**DD Overhead**
MNIST	1320.01	1041.67	77.99%
CIFAR-10	1106.07	884.45	79.95%

C. *Performance of a Single Unlearning Request*

For the following experiments, we perform 10 iterations (global rounds), and client 8 disconnects at iteration 3, sending out an unlearning request to its neighbours, along with its synthetic data S_8. These neighbours then recursively propagate the request to the rest of the network via their neighbours.

We report the drop in general testing accuracy during unlearning and recovery in Figure 3. We compare F-set and R-set accuracies with the proposed baselines in Table III. We notice that our proposed method remains competitive with the oracles, proving its correctness. As we increase the degree of each node, we can notice the method recovering faster and better, while maintaining a low F-set accuracy. This leads to the observation that the more sparse a DL system is, the more difficult it is to both unlearn and recover in an efficient manner. However, for a fully connected network, the chosen unlearning hyperparameters prove to be too aggressive, as the model struggles to regain its past performance for CIFAR-10. Moreover, this ties in closely with the fact that DL systems have a slower convergence rate and higher variance in per-client testing accuracy. We also notice how R-set accuracies fail to reach the previous values for global accuracy, even for the retrain oracle. This is because of the strong non-IID-ness ($\alpha = 0.1$) of the data, which can lead to large parts of a class being deleted from the training data, as some nodes may be of higher importance to the global model than others. This, combined with a sparse network, makes correctly classifying these classes happen slower than before the client's disconnection.

D. *Performance of Multiple Unlearning Requests*

In our setup, we allow for an arbitrary number of clients to drop. We also go beyond QUICKDROP by allowing clients to crash unannounced, and implement a timeout mechanism, as presented in Section VI. We report the client-wise global testing accuracies of sequentially unlearning half of the clients in random order of a 4-regular 16-node network in Figure 4 such that the network remains connected. We observe that as more clients leave, the variance between them increases. This stems from the low connectivity of the network, which slows down convergence rate. However, most clients are still able to reach a high global accuracy despite half of them leaving, which shows that the recovery remains effective.

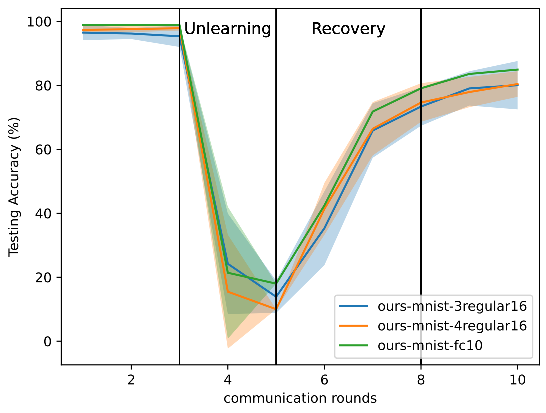
E. *Overhead During Training*

We report the following types of overhead during training for the proposed algorithm:

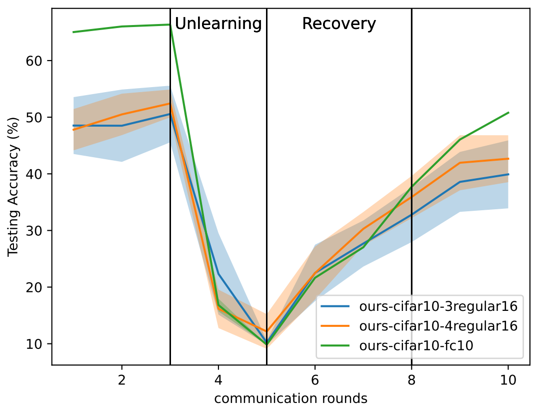
1) **Computation overhead of data distillation (DD).** Table II exemplifies the additional computation time needed for DD with the proposed hyperparameters. The total train time is calculated for each local training step, without initialization, communication or weight averaging, while the DD compute time is calculated only for tuning the synthetic dataset using the minibatch samples. While this does add some overhead, it is necessary for using synthetic data for fast and efficient unlearning and recovery.
2) **Time overhead of unlearning.** Our baselines remain competitive in terms of speed of unlearning and recovery, as shown by Table III. The increase in duration with respect to SGA-OR is due to the fact that the synthetic dataset is augmented for each recovery step. While it does add overhead to the unlearning operation, we prove it is worthwhile by the increase in R-set accuracy it provides over SGA-OR.

VIII. DISCUSSION

The results presented in Section VII showcase the impact of our research in the field of DL. We propose a novel approach



(a) MNIST



(b) CIFAR-10

Fig. 3: Average testing accuracy of the clients in the network before, during, and after unlearning and recovery for two datasets (MNIST (a) and CIFAR-10 (b)). Client 8 disconnects at iteration 3. Comparison of 3 topologies: 3-regular 16-node (*3regular16*), 4-regular 16-node (*4regular16*), and fully connected 10-node (*fc10*). Standard deviation is shown by the shaded area.

TABLE III: F-Set and R-Set accuracies, computation cost, and speedup of different DU methods for MNIST and CIFAR-10 datasets after unlearning and recovery. Comparison of 3 topologies: 3-regular 16-node (*3regular16*), 4-regular 16-node (*4regular16*), and fully connected 10-node (*fc10*). For RETRAIN-OR, we consider 400 rounds of retraining. We measure the time from when the unlearning request is triggered until the last recovery step ends. F-set accuracy should be low and R-set accuracy high. The best values are in bold.

Topology	**DU Approach**	**MNIST**				**CIFAR-10**			
		F-Set	**R-Set**	**Time (s)**	**Speed-up**	**F-Set**	**R-Set**	**Time (s)**	**Speed-up**
3regular16	RETRAIN-OR	94.16%	**97.19%**	2641.06	1×	54.61%	**54.93%**	1613.99	1×
	SGA-OR	**3.15%**	40.68%	**83.71**	**31.55×**	**5.66%**	30.84%	**46.67**	**34.58×**
	This Work	5.40%	73.54%	103.06	25.63×	8.37%	28.17%	66.42	24.30×
4regular16	RETRAIN-OR	94.83%	**97.91%**	2698.77	1×	59.96%	**57.78%**	1725.76	1×
	SGA-OR	**4.03%**	60.03%	**79.86**	**33.79×**	**3.41%**	33.46%	**43.91**	**30.30×**
	This Work	5.59%	74.62%	107.59	25.08×	7.97%	29.09%	72.05	23.95×
fc10	RETRAIN-OR	93.73%	**98.54%**	2369.61	1×	62.06%	**68.59%**	1470.90	1×
	SGA-OR	**8.93%**	77.59%	**63.66**	**37.22×**	**3.33%**	33.36%	**41.60**	**35.36×**
	This Work	9.05%	79.07%	76.50	30.98×	10.64%	37.71%	68.90	21.35×

for DU by improving on the existing FU SOTA, allowing for more real-world use cases (client crashes) and a more efficient gradient matching algorithm in theory. In this section we provide arguments towards the necessity of broader study in the area and make recommendations for the directions to take.

A. Limitations

While our proposed approach achieves competitive results against existing baselines and oracles, several limitations remain. Due to the fundamental differences between FL and DL, directly transferring hyperparameters from QUICKDROP did not lead to optimal performance. In FL, each node follows a synchronized and centrally-coordinated workflow, which simplifies parameter tuning. In contrast, DL involves asynchronous peer-to-peer communication, making it harder to ensure convergence and consistency. As such, our method required specific hyperparameter tuning for different topologies and datasets, limiting generalizability. More specifically, more unlearning and recovery steps were needed to reach similar effectiveness.

Moreover, although our method supports unlearning without full retraining and enables crashed client detection, scalability remains a concern. As the number of clients increases, so does the volume of synthetic data being shared and cached, despite it being only 2% of the size of their data partition. This leads to increased communication overhead, accentuated in dense topologies, which can become a bottleneck in resource-constrained environments. Currently, a client shares their data with all of its neighbours, who cache it in anticipation of potential unlearning requests. One could experiment with only sharing the synthetic data with a few of the neighbours.

Although DecentralizePy is synchronous by design, where

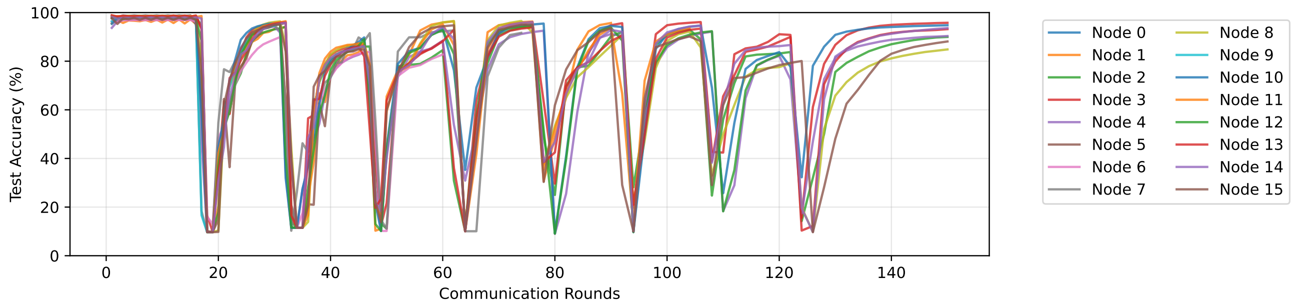


Fig. 4: Per-client global testing accuracy after unlearning clients 0, 9, 5, 6, 7, 1, 11, 12 every 15 iterations starting with round 15, on the MNIST dataset, with a 4-regular 16-node topology. Clients leave without notice, in which case neighbours initiate unlearning using the cached synthetic data of the departed client.

iteration $n+1$ begins only after a client and its neighbours complete iteration n, the propagation of unlearning requests is not. This asynchronous behaviour occurs only when a client crashes. For example, if client i crashes during iteration n, its neighbours may only react in iteration $n+1$ after completing their own local training. This is shown in Figure 4, where unlearning, marked by a sharp drop in test accuracy, begins at different times for different clients, especially as the network becomes more sparse. Since weight averaging continues during unlearning and recovery rounds to aid convergence and prevent overfitting, a recovering client may average with one that is unlearning, reducing the efficiency of the unlearning process.

Another challenge arises from the strongly non-IID nature of the data. When a client leaves the network, it may carry with it many unique samples of a class, causing entire classes to disappear. This makes both unlearning and recovery more difficult, and is particularly evident in sparse graphs, where fewer neighbours means less redundancy in the data.

B. Future work

Although our work focuses on client-level unlearning, the algorithm is, in theory, extensible to class-level and sample-level unlearning. This could prove useful for tasks involving sensitive categories or user-specific examples. Further research in this direction could help assess the adaptability and robustness of our method to these different types of unlearning requests.

Another direction is relearning, where a client who previously requested unlearning later rejoins the network. The current code does not support this functionality, but the method itself does, since synthetic data of dropped clients remains cached in their neighbours.

Scalability is also an important aspect for future exploration. Running our algorithm on larger DL networks would test its robustness, communication cost, and computation efficiency at scale. Exploring different topologies and their effects on convergence and unlearning quality may offer insights into how to optimize network structure dynamically during training or unlearning.

Additionally, our method could be tested on more complex models and real-world datasets. The current use of small CNNs and academic datasets like MNIST and CIFAR-10 may not fully reflect the challenges faced in real-world scenarios. The existing method of HDUS [10] was not evaluated in this work due to time constraints and the complexity of implementing it into DecentralizePy, but should be included in future comparisons to better understand the relative strengths and weaknesses of different approaches to DU.

We also recommend investigating alternative DD algorithms, such as those using *distribution matching* [43, 44]. These methods may offer better performance, higher convergence rates, or improve privacy when generating synthetic data.

IX. Conclusion

This work has addressed the under-explored problem of client-level unlearning in decentralized learning systems. We propose a method inspired by QuickDrop, adapting it to the decentralized setting through asynchronous peer-to-peer communication. Our algorithm allows for the effective removal of a dropped client's influence via SGA, followed by accelerated recovery rounds that restore generalization performance using only synthetic data.

Through experiments on the MNIST and CIFAR-10 datasets over three different topologies, we show that our method performs competitively with optimal baselines, which require impractical assumptions, such as access to raw training data or retraining from scratch. Moreover, our approach uniquely supports client crashes by using cached synthetic datasets, making it suitable for real-world decentralized systems where churn is inevitable.

Despite this, the algorithm's efficiency depends heavily on the quality of synthetic data, and performance can degrade in highly sparse and non-IID environments. Communication overheads increase with network size, and careful hyperparameter tuning is needed across different settings.

Nevertheless, this work provides a foundation for future research into privacy-preserving mechanisms in DL systems. Potential extensions include supporting other unlearning types, improving scalability, experimenting with larger models, and exploring other synthetic data generation algorithms. By enabling practical unlearning in decentralized contexts, our method takes a step toward more robust decentralized learning.

X. Responsible Research

This research was conducted responsibly in accordance with the Netherlands Code of Conduct for Research Integrity [45]. In this section, we reflect of several aspects of our work, highlighting measures towards transparency, integrity, reproducibility, replicability, and the responsible use of Artificial Intelligence (AI) throughout the research process and delivery.

A. Datasets

This research uses only publicly available datasets (MNIST [39], CIFAR-10 [40]) and code with an MIT License (DecentralizePy [24], DatasetCondensation [28]). The choice of datasets was backed by the need for ease and speed of development (MNIST), and effectiveness in benchmarking against other algorithms (CIFAR-10).

B. Results and Evaluation

DecentralizePy plays a major role in the *reproducibility* of the experiments we conduct, as it saves the configuration and command-line arguments used for each experiment along with its results. We save all results reported in this paper along in the aforementioned code repository. Its README file provides steps for reproducing each experiment. All experiments have been run at least three times to check that results have low variance. Additionally, we use pre-defined and configurable seeds for all randomness within our code. Moreover, we considered *replicability* in every step of our processes, by using the same non-skewed datasets which are often part of similar work. All design choices are outlined and explained in Sections V and VI.

Due to limited resources, experiments were run via SSH on an external machine, making some values, particularly timings in Tables II and III, sensitive to system and network conditions. Evaluation on CIFAR-10 was performed less frequently due to CUDA memory issues, which may have inflated speed compared to MNIST. These factors mean the reported speeds may vary across runs and are not fully representative. The extended timeout described in Section VI was necessary to avoid disconnections, as shorter timeouts proved unreliable. To provide more stable comparisons, we also report relative percentages for speed-ups and overheads.

C. Research Process

The research process was conducted responsibly, beginning with a thorough literature review. Section III presents the work performed in recent years in the areas of Federated and Decentralized Unlearning. We mention how, to the best of our knowledge, DU research is minimal, and attempt to translate the SOTA from FL to DL, while making some improvements. The process was also conducted with various ethical and correctness considerations in mind. Our research respects the privacy of users by making it more efficient for DL systems to respect users' rights to be forgotten (e.g. the GDPR [16]).

D. Use of Artificial Intelligence

AI was only used for correcting errors within the code, explaining new concepts or writing scripts for calculating reported values (e.g. mean unlearning times), via publicly available products like ChatGPT and GitHub Copilot. All generated content was verified by the authors. *No section of this paper* was written using AI.

References

[1] P. Kairouz et al. "Advances and Open Problems in Federated Learning". English. In: *Foundations and Trends® in Machine Learning* 14.1–2 (June 2021). Publisher: Now Publishers, Inc., pp. 1–210.

[2] C. Zhang, Y. Xie, H. Bai, B. Yu, W. Li, and Y. Gao. "A survey on federated learning". In: *Knowledge-Based Systems* 216 (Mar. 2021), p. 106775.

[3] B. Cox, L. Y. Chen, and J. Decouchant. "Aergia: leveraging heterogeneity in federated learning systems". In: *Proceedings of the 23rd ACM/IFIP International Middleware Conference*. 2022, pp. 107–120.

[4] I. Hegedűs, G. Danner, and M. Jelasity. "Gossip Learning as a Decentralized Alternative to Federated Learning". en. In: *Distributed Applications and Interoperable Systems*. Cham: Springer International Publishing, 2019, pp. 74–90.

[5] R. Wang et al. "MUDGUARD: Taming Malicious Majorities in Federated Learning using Privacy-Preserving Byzantine-Robust Clustering". In: *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 8.3 (2024), pp. 1–41.

[6] B. Cox, A. Mălan, L. Y. Chen, and J. Decouchant. "Asynchronous Byzantine federated learning". In: *arXiv preprint arXiv:2406.01438* (2024).

[7] A. Shankar, L. Y. Chen, J. Decouchant, D. Gkorou, and R. Hai. "Share Your Secrets for Privacy! Confidential Forecasting with Vertical Federated Learning". In: *arXiv preprint arXiv:2405.20761* (2024).

[8] A. Mălan, J. Decouchant, T. Guzella, and L. Chen. "CCBNet: Confidential Collaborative Bayesian Networks Inference". In: *arXiv preprint arXiv:2405.15055* (2024).

[9] R. Ormándi, I. Hegedűs, and M. Jelasity. "Gossip learning with linear models on fully distributed data". en. In: *Concurrency and Computation: Practice and Experience* 25.4 (2013). _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/cpe.2858, pp. 556–571.

[10] G. Ye, T. Chen, Q. V. H. Nguyen, and H. Yin. *Heterogeneous Decentralized Machine Unlearning with Seed Model Distillation*. arXiv:2308.13269 [cs]. Aug. 2023.

[11] X. Lian, C. Zhang, H. Zhang, C.-J. Hsieh, W. Zhang, and J. Liu. *Can Decentralized Algorithms Outperform Centralized Algorithms? A Case Study for Decentralized Parallel Stochastic Gradient Descent*. arXiv:1705.09056 [math]. Sept. 2017.

[12] S. Biswas et al. "Noiseless Privacy-Preserving Decentralized Learning". In: *Proceedings on Privacy Enhancing Technologies* (2025).

[13] D. Stutzbach and R. Rejaie. "Understanding churn in peer-to-peer networks". In: *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*. IMC '06. New York, NY, USA: Association for Computing Machinery, Oct. 2006, pp. 189–202.

[14] W. Wang, Z. Tian, C. Zhang, and S. Yu. *Machine Unlearning: A Comprehensive Survey*. arXiv:2405.07406 [cs]. July 2024.

[15] A. Ginart, M. Guan, G. Valiant, and J. Y. Zou. "Making AI Forget You: Data Deletion in Machine Learning". In: *Advances in Neural Information Processing Systems*. Vol. 32. Curran Associates, Inc., 2019.

[16] European Union. *General Data Protection Regulation*. Apr. 2016.

[17] A. Dhasade, Y. Ding, S. Guo, A.-M. Kermarrec, M. de Vos, and L. Wu. “QuickDrop: Efficient Federated Unlearning via Synthetic Data Generation”. In: *Proceedings of the 25th International Middleware Conference*. Middleware ’24. New York, NY, USA: Association for Computing Machinery, Dec. 2024, pp. 266–278.

[18] A. Hatua, T. T. Nguyen, F. Cano, and A. H. Sung. *Machine Unlearning using Forgetting Neural Networks*. arXiv:2410.22374 [cs]. Oct. 2024.

[19] A. K. Tarun, V. S. Chundawat, M. Mandal, and M. Kankanhalli. “Fast Yet Effective Machine Unlearning”. In: *IEEE Transactions on Neural Networks and Learning Systems* 35.9 (Sept. 2024), pp. 13046–13055.

[20] A. Sekhari, J. Acharya, G. Kamath, and A. T. Suresh. “Remember What You Want to Forget: Algorithms for Machine Unlearning”. In: *Advances in Neural Information Processing Systems*. Vol. 34. Curran Associates, Inc., 2021, pp. 18075–18086.

[21] Y. Li, J. Zhang, Y. Liu, and C. Chen. *Class-wise Federated Unlearning: Harnessing Active Forgetting with Teacher-Student Memory Generation*. arXiv:2307.03363 [cs]. Mar. 2025.

[22] A. Golatkar, A. Achille, and S. Soatto. *Eternal Sunshine of the Spotless Net: Selective Forgetting in Deep Networks*. arXiv:1911.04933 [cs]. Mar. 2020.

[23] Y. Cao and J. Yang. “Towards Making Systems Forget with Machine Unlearning”. In: *2015 IEEE Symposium on Security and Privacy*. ISSN: 2375-1207. May 2015, pp. 463–480.

[24] A. Dhasade, A.-M. Kermarrec, R. Pires, R. Sharma, and M. Vujasinovic. “Decentralized Learning Made Easy with DecentralizePy”. en. In: *Proceedings of the 3rd Workshop on Machine Learning and Systems*. Rome Italy: ACM, May 2023, pp. 34–41.

[25] L. Wu, S. Guo, J. Wang, Z. Hong, J. Zhang, and Y. Ding. “Federated Unlearning: Guarantee the Right of Clients to Forget”. In: *IEEE Network* 36.5 (Sept. 2022), pp. 129–135.

[26] A. Halimi, S. Kadhe, A. Rawat, and N. Baracaldo. *Federated Unlearning: How to Efficiently Erase a Client in FL?* arXiv:2207.05521 [cs]. Oct. 2023.

[27] C. Wu, S. Zhu, and P. Mitra. *Federated Unlearning with Knowledge Distillation*. arXiv:2201.09441 [cs]. Jan. 2022.

[28] B. Zhao, K. R. Mopuri, and H. Bilen. *Dataset Condensation with Gradient Matching*. arXiv:2006.05929 [cs]. Mar. 2021.

[29] Z.-H. Zhou. *Ensemble Methods: Foundations and Algorithms*. en. CRC Press, Feb. 2025.

[30] G. Ye, T. Chen, Y. Li, L. Cui, Q. V. H. Nguyen, and H. Yin. “Heterogeneous Collaborative Learning for Personalized Healthcare Analytics via Messenger Distillation”. In: *IEEE Journal of Biomedical and Health Informatics* 27.11 (Nov. 2023), pp. 5249–5259.

[31] A. Koloskova. “Optimization Algorithms for Decentralized, Distributed and Collaborative Machine Learning”. en. PhD thesis. EPFL, 2024.

[32] A. Koloskova*, T. Lin*, S. U. Stich, and M. Jaggi. “Decentralized Deep Learning with Arbitrary Communication Compression”. en. In: Sept. 2019.

[33] M. Jelasity, S. Voulgaris, R. Guerraoui, A.-M. Kermarrec, and M. van Steen. “Gossip-based peer sampling”. In: *ACM Trans. Comput. Syst.* 25.3 (Aug. 2007), 8–es.

[34] B. Cox, J. Galjaard, A. Shankar, J. Decouchant, and L. Y. Chen. “Parameterizing Federated Continual Learning for Reproducible Research”. In: *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer. 2023, pp. 478–486.

[35] T. Wang, J.-Y. Zhu, A. Torralba, and A. A. Efros. *Dataset Distillation*. arXiv:1811.10959 [cs]. Feb. 2020.

[36] S. Zhao, Z. Liu, J. Lin, J.-Y. Zhu, and S. Han. “Differentiable Augmentation for Data-Efficient GAN Training”. In: *Advances in Neural Information Processing Systems*. Vol. 33. Curran Associates, Inc., 2020, pp. 7559–7570.

[37] B. Zhao and H. Bilen. “Dataset Condensation with Differentiable Siamese Augmentation”. en. In: *Proceedings of the 38th International Conference on Machine Learning*. ISSN: 2640-3498. PMLR, July 2021, pp. 12674–12685.

[38] L. Xiao, S. Boyd, and S.-J. Kim. “Distributed average consensus with least-mean-square deviation”. In: *Journal of Parallel and Distributed Computing* 67.1 (Jan. 2007), pp. 33–46.

[39] Y. LeCun and C. Cortes. “The mnist database of handwritten digits”. In: 2005.

[40] A. Krizhevsky. “Learning Multiple Layers of Features from Tiny Images”. In: 2009.

[41] S. Gidaris and N. Komodakis. “Dynamic Few-Shot Visual Learning Without Forgetting”. In: 2018, pp. 4367–4375.

[42] T. Kynkäänniemi, T. Karras, M. Aittala, T. Aila, and J. Lehtinen. *The Role of ImageNet Classes in Fr\'echet Inception Distance*. en. Mar. 2022.

[43] B. Zhao and H. Bilen. *Dataset Condensation with Distribution Matching*. arXiv:2110.04181 [cs]. Dec. 2022.

[44] C.-Y. Huang, K. Srinivas, X. Zhang, and X. Li. *Overcoming Data and Model Heterogeneities in Decentralized Federated Learning via Synthetic Anchors*. arXiv:2405.11525 [cs]. Mar. 2025.

[45] *Netherlands Code of Conduct for Research Integrity | NWO*. en.

APPENDIX A
TOPOLOGIES USED

This Appendix provides a visual presentation of the topologies used throughout our experiments, outlined in Section VI.[2] An n-regular graph is a graph where each node has degree n.

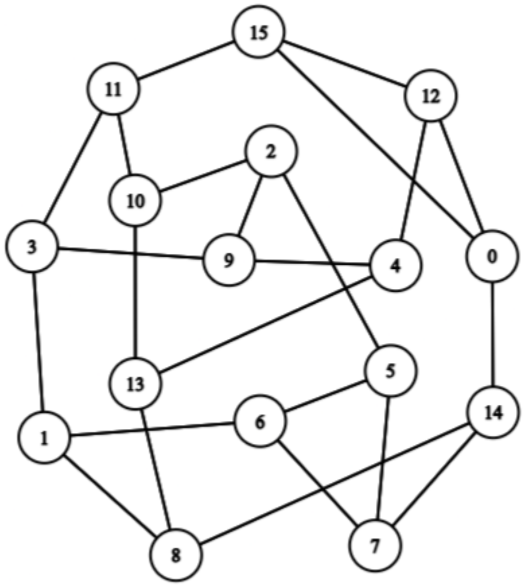


Fig. 5: The 16-node 3-regular graph we used.

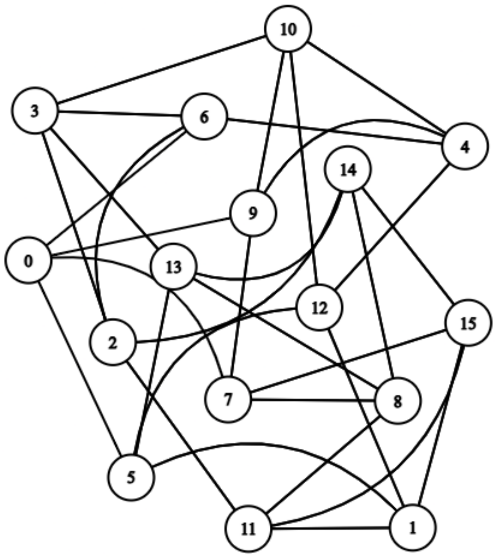


Fig. 6: The 16-node 4-regular graph we used.

[2]Drawn using CS Academy's Graph Drawer. See https://csacademy.com/app/graph_editor/

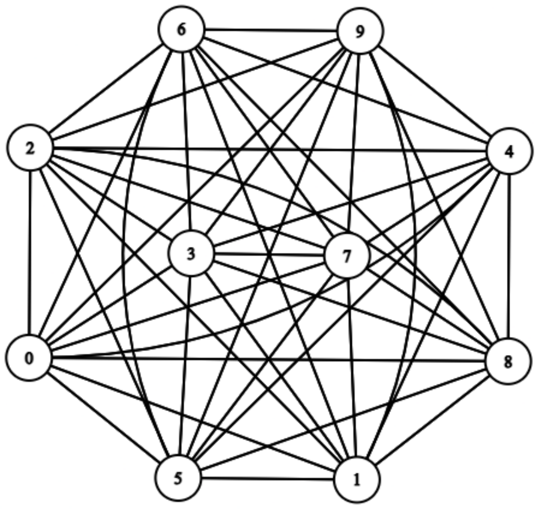


Fig. 7: The 10-node fully connected graph we used.