QuickDrop: Efficient Federated Unlearning via Synthetic Data Generation

Akash Dhasade
EPFL
Switzerland

Yaohong Ding
The Hong Kong Polytechnic University
China

Song Guo
The Hong Kong University of Science and Technology
China

Anne-Marie Kermarrec
EPFL
Switzerland

Martijn de Vos
EPFL
Switzerland

Leijie Wu
The Hong Kong Polytechnic University
China

ABSTRACT

Federated Unlearning (FU) aims to delete specific training data from an ML model trained using Federated Learning (FL). However, existing FU methods suffer from inefficiencies due to the high costs associated with gradient recomputation and storage. This paper presents QuickDrop, an original and efficient FU approach designed to overcome these limitations. During model training, each client uses QuickDrop to generate a compact synthetic dataset, serving as a compressed representation of the gradient information utilized during training. This synthetic dataset facilitates fast gradient approximation, allowing rapid downstream unlearning at minimal storage cost. To unlearn some knowledge from the trained model, QuickDrop clients execute stochastic gradient ascent with samples from the synthetic datasets instead of the training dataset. The tiny volume of synthetic data significantly reduces computational overhead compared to conventional FU methods. Evaluations with three standard datasets and five baselines show that, with comparable accuracy guarantees, QuickDrop reduces the unlearning duration by 463× compared to retraining the model from scratch and 65 – 218× compared to FU baselines. QuickDrop supports both class- and client-level unlearning, multiple unlearning requests, and relearning of previously erased data.

CCS CONCEPTS

• **Information systems** → *Data management systems*; • **Computing methodologies** → **Machine learning**.

KEYWORDS

Federated Unlearning, Machine Unlearning, Federated Learning, Privacy and Security, Dataset Distillation.

ACM Reference Format:
Akash Dhasade, Yaohong Ding, Song Guo, Anne-Marie Kermarrec, Martijn de Vos, and Leijie Wu. 2024. QuickDrop: Efficient Federated Unlearning via Synthetic Data Generation. In *24th International Middleware Conference (MIDDLEWARE '24), December 2–6, 2024, Hong Kong, Hong Kong*. ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3652892.3700764

This work is licensed under a Creative Commons Attribution International 4.0 License.
MIDDLEWARE '24, December 2–6, 2024, Hong Kong, Hong Kong
© 2024 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0623-3/24/12
https://doi.org/10.1145/3652892.3700764

1 INTRODUCTION

The vast amount of data produced by computing devices is increasingly used to train large-scale ML models that empower industrial processes and personal experiences [29]. However, this data is often privacy sensitive or very large in volume, making it prohibitively expensive to upload it to a central server [2, 43]. To sidestep this issue, federated learning (FL) is increasingly being applied to collaboratively train ML models in a privacy-preserving manner [30]. FL obviates the need to move the data to a central location by having participants only exchange model updates with a parameter server. In each round of FL, the parameter server aggregates all incoming trained models and then sends the aggregated global model back to participants.

Recent privacy regulations like the General Data Protection Regulation (GDPR) and California Consumer Privacy Act (CCPA) grant data owners with the "right to be forgotten" [5, 12]. In the realm of ML, this requires organizations to be able to remove the influence of personal data on the trained model upon request [45]. This is called *machine unlearning (MU)* [3]. For instance, hospitals that collaboratively trained a model using FL might have to unlearn particular data samples in response to patient requests. In FL, beyond the "right to be forgotten", removing data from the model proves essential for several other purposes. For instance, the ability to quickly eliminate outdated, manipulated, or erroneously included data enhances the security, responsiveness, and reliability of FL systems [10]. However, the distributed nature of FL and the inability of the parameter server to access training data directly makes *federated unlearning (FU)* a challenging task.

A naive method involves retraining the model from scratch, excluding target samples. This triggers many new training rounds, forcing clients to recompute gradients on the remaining data. Therefore, retraining is prohibitively expensive, with respect to the time and compute resources required. Alternatively, some methods store gradients from original FL training for *downstream unlearning* [27]. However, the linear scaling of storage costs with the number of clients and FL rounds leads to significant overhead. A more effective approach uses *stochastic gradient ascent (SGA)* on target samples [42], optimizing in the direction that maximizes the loss function. However, this approach updates the entire model, causing performance deterioration of the remaining samples. A subsequent

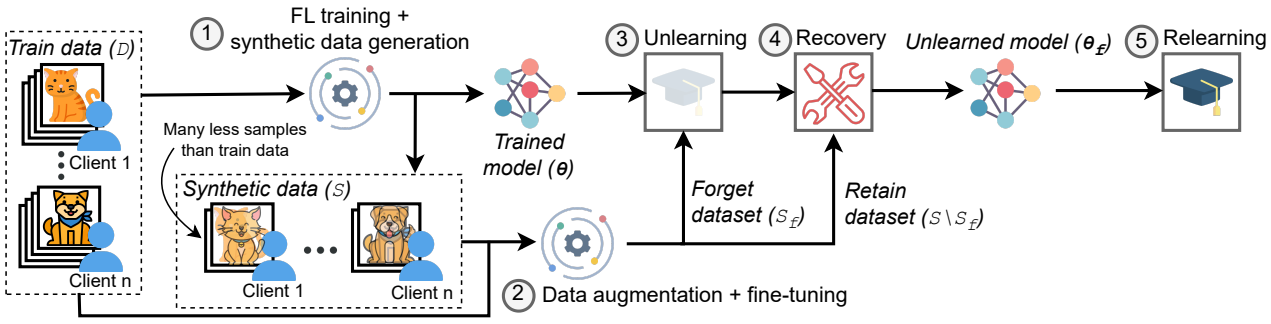

Figure 1: The workflow of QuickDrop, our efficient federated unlearning method using synthetic data.

recovery phase is necessary, involving model retraining with the remaining samples for a few rounds. While SGA is more efficient than full retraining, it remains computationally demanding, requiring clients to recompute numerous gradients for unlearning and recovery. Thus, existing FU approaches are inefficient and incur high costs for either gradient storage or recomputation.

To address this issue, we introduce QuickDrop, a novel FU approach that efficiently performs unlearning using SGA and synthetic data. QuickDrop circumvents the need to store or recompute expensive gradients by employing a technique of *gradient matching* [50] during FL training. This process synthesizes a compact, client-specific dataset in situ, acting as a compressed representation of original gradients. This synthetic dataset facilitates *fast gradient approximation* for downstream unlearning and significantly enhances efficiency. Furthermore, this dataset is merely 1% of the total volume of local datasets, resulting in minimal storage overhead.

The full workflow of QuickDrop is shown in Figure 1 and involves five main steps. Initially, clients engage in regular FL training to collaboratively train a model (step (1)). Simultaneously, each client generates a compact synthetic dataset through gradient matching, optimized for unlearning tasks. This synthetic dataset is then augmented with a few original samples and fine-tuned (step (2)), which improves the model accuracy during recovery. Unlearning rounds (step (3)) leverage SGA on the local synthetic dataset, maintaining efficiency due to its small volume. Recovery rounds (step (4)) also utilize the synthetic data, ensuring efficient downstream recovery compared to using original datasets. QuickDrop also supports the efficient relearning of unlearned knowledge (step (5)), again leveraging the synthetic data. In summary, our approach overcomes the limitations of traditional methods, offering a streamlined and efficient solution for FU.

Contributions. This paper makes the following contributions:

(1) We introduce QuickDrop, a novel and efficient federated unlearning approach that generates and uses synthetic datasets to unlearn specific knowledge from a trained FL model (Section 3).
(2) We formulate the problem of synthetic data generation for unlearning and show how such synthetic data can be generated in situ in FL training by adapting the technique of gradient matching [50] (Section 3.2).
(3) We implement and open-source QuickDrop, and evaluate its unlearning performance in terms of efficiency and accuracy on three standard datasets and five state-of-the-art (SOTA) baselines (Section 4). Under comparable accuracy guarantees, we find that QuickDrop reduces the duration of class unlearning by **463**× compared to model retraining from scratch and **65-218**× compared to other SOTA FU approaches.

2 BACKGROUND AND PROBLEM SETUP

We first provide background on machine and federated unlearning in Section 2.1, then formulate the problem in Section 2.2, and finally outline existing unlearning algorithms in Section 2.3.

2.1 Machine and Federated Unlearning

Machine unlearning (MU) was first proposed by Cao and Yang [6]. Consider a set of trained weights $\boldsymbol{\theta}$ on a training dataset D. The purpose of MU is to remove the influence of a specific subset of $D_f \subset D$ on the model parameters $\boldsymbol{\theta}$. The subset D_f is generally referred to as the *forget dataset*, and its complement $D \backslash D_f$ is called the *retain dataset*. Let the unlearning algorithm $\mathcal{U}$ be defined as $\boldsymbol{\theta}_f = \mathcal{U}(\boldsymbol{\theta}, D_f)$, where $\boldsymbol{\theta}_f$ is the unlearned model. Unlearning thus aims to obtain a model $\boldsymbol{\theta}_f$ that is equivalent in performance to a model trained only on $D \backslash D_f$. In other words, the unlearned model $\boldsymbol{\theta}_f$ should perform well on $D \backslash D_f$ while performing relatively less well on D_f.

Federated unlearning (FU) is a MU technique where knowledge is removed from a trained model in a distributed and collaborative manner. Generally, FU is more challenging than MU for the following two reasons. First, client data is only available on the client's side and cannot be moved to a central server. This mandates active participation by clients to perform unlearning. It also implies that the parameter server cannot conduct any fine-grained operations at the data level, rendering many existing MU techniques inapplicable in FL settings. Second, when the original model is also trained in a distributed, collaborative manner, *e.g.*, using FL, the parameter server does not always have access to intermediate, granular training information produced by clients. Some MU techniques rely on recorded training information to carry out an unlearning operation [19]. In FU settings, however, the parameter server might be unable to collect specific training information for unlearning, such as model updates per batch for each client.

2.2 Problem Setup

We consider a FL system containing N clients (*e.g.*, PCs or mobile devices), where each client $i \in N$ holds a local training dataset D_i. Clients collaboratively train a global FL model $\boldsymbol{\theta}$ using an FL algorithm (*e.g.*, FedAvg [30]). Once the global model $\boldsymbol{\theta}$ is trained, the parameter server may receive an unlearning request for the forget dataset D_f. The characterization of D_f defines the type of unlearning performed. We distinguish between the following three types of unlearning:

- **Class-level unlearning.** This type of unlearning erases the knowledge of a target class. Consequently, D_f encompasses the entire data of a class. Denoting by D_i^c the data of class c with the i-th client, we have $D_f := \cup_i D_i^c$, when the target class is c. The union is over all clients i which possess samples of class c. In essence, D_f for class-level unlearning is distributed across clients.
- **Client-level unlearning.** This type of unlearning erases the knowledge of a target client, *e.g.*, when exercising the right to be forgotten. Here, D_f contains the data of a single client. When the target client is i, $D_f := D_i$. In this case, D_f is concentrated on a single client.
- **Sample-level unlearning.** This type of unlearning erases the knowledge of one or more samples. Here, D_f contains arbitrary samples from one or more clients. Sample-level unlearning is the most general and difficult form of unlearning [34, 42].

Relearning. While the main objective of our work is to unlearn efficiently, it can be desirable to relearn the unlearned knowledge in certain situations, *e.g.*, when a client revokes its right to be forgotten. Where unlearning erases the knowledge in D_f from the trained model, relearning aims to add this knowledge back.

We remark that class- and client-level unlearning are the two most common use cases in federated settings [34, 42]; very few works address sample-level unlearning, even in the context of MU. In this work, we also specifically focus on class-level and client-level unlearning. We further discuss this aspect in Section 5.1.

2.3 Existing FU Algorithms and their Drawbacks

We now discuss existing FU approaches and their drawbacks before presenting the design of QuickDrop in Section 3.

Retraining from scratch. A naive way to unlearn D_f is to retrain the model from scratch while omitting samples from D_f. While this algorithm perfectly achieves the desired goal, complete retraining is prohibitively expensive as it initiates new FL training rounds on $D \backslash D_f$. Even executing a single unlearning request in such a way is highly compute- and time-intensive. We refer to this algorithm as Retrain-Or *i.e.*, as a retraining oracle due to its ideal achieved performance.

Gradient calibration. One way to speed up retraining from scratch is to reuse gradient information from the original training to avoid regenerating all gradients from scratch. However, these gradients must be adapted based on the forget dataset D_f and retain dataset $D \backslash D_f$, through a process referred to as *gradient calibration*. Algorithms employing gradient calibration like FedEraser [27] trade the central server's storage for unlearned model's construction time by leveraging historical parameter updates from FL training. However, the storage costs can grow quite large while the efficiency gains compared to retraining from scratch remain modest ($\sim 4\times$).

Table 1: Comparison of FU approaches and QuickDrop.

Algorithms	Class-unlearn.	Client-unlearn.	Relearn.	Storage Eff.	Computation Eff.
Retrain-Or	✓	✓	✓	✓	✗ (very low)
FedEraser [27]	✓	✓	✓	✗	✗ (low)
S2U [13]	✗	✓	✓	✓	✗ (low)
SGA [42]	✓	✓	✓	✓	✗ (medium)
FU-MP [39]	✓	✗	✗	✓	✗ (medium)
QuickDrop	✓	✓	✓	✓[1]	✓ (high)

[1] The exact storage overhead depends on the scale parameter.

Stochastic gradient ascent (SGA). Another FU approach involves performing SGA steps on D_f [42]. In each FL round, clients having data in D_f perform local SGA steps while the parameter server aggregates the received updates. However, SGA training introduces noise that affects the performance of remaining data. This noise necessitates subsequent recovery rounds during which clients engage in regular SGD training on the retain dataset $D \backslash D_f$. An unlearning request thus encompasses unlearning on D_f and recovery on $D \backslash D_f$ with each request updating the model with the entire dataset. Therefore, SGA remains inefficient when dealing with high dataset volumes or when executing many unlearning requests. Algorithm 1 provides the pseudo code for SGA.

S2U. Inspired by the observation that the up- or down-scaling of model updates can substantially influence the global model, S2U scales down the forgetting client's updates while scaling up the updates of remaining clients [13]. Its unlearning and recovery stages are integrated and conducted simultaneously, like Retrain-Or. S2U is specifically designed to only support client-level unlearning.

Model Pruning. FU-MP [39] uses model pruning by first measuring the class discrimination of channels in the model (*i.e.*, the relevance of different classes on the model channel) and then prunes the most relevant channel of the target class to unlearn it. While FU-MP is much more efficient than Retrain-Or, it only applies to class-level unlearning. Additionally, the pruned channels prevent relearning as pruning irreversibly modifies the model.

Except for FU-MP, all aforementioned FU approaches rely on some form of gradient information to perform unlearning. However, such approaches remain inefficient since gradients are expensive to store and recompute. Approaches such as S2U and FU-MP do not generalize to both client- and class-level unlearning. Table 1 provides a comparison of existing FU approaches and our work. The popularity of gradient-based approaches for FU motivates the following question: *what if one could succinctly compress all the gradient information such that it can be reused for fast gradient approximation for any downstream unlearning?* This directly leads to our design of QuickDrop, which, during FL training, derives synthetic samples that accumulate gradient information in a compact form, enabling efficient unlearning.

Algorithm 1: Unlearning using the SGA algorithm [42].

Input: Model parameters $\boldsymbol{\theta}$ trained via FL on training datasets $\{D_i\}_{i=1}^N$, forget set D_f, local update steps T and learning rate $\eta_{\boldsymbol{\theta}}$.

Server executes:
  Receive unlearning request for D_f
  Execute unlearning rounds using FedAvg (D_f, $\boldsymbol{\theta}$, `unlearn`)
  Execute recovery rounds using FedAvg ($D \backslash D_f$, $\boldsymbol{\theta}$, `recover`)
FedAvg (Z, $\boldsymbol{\theta}$, phase):
  Let $\{Z_i\}_{i=1}^N \leftarrow$ counterparts of Z with respective clients[1]
  $\boldsymbol{\theta}_{0,0} \leftarrow \boldsymbol{\theta}$
  for $k = 0, 1, \cdots$ *until convergence* **do**
    for *each client* $i = 1, \cdots, N$ *in parallel*[2] **do**
      Initialize $\boldsymbol{\theta}^i_{k,0} \leftarrow \boldsymbol{\theta}_{k,0}$
      for $t = 0, \cdots, T-1$ **do**
        Sample batch $B \sim Z_i$ and compute $\nabla \mathcal{L}^{Z_i}(\boldsymbol{\theta}^i_{k,t})$
        if phase *is* `unlearn` **then**
          $\boldsymbol{\theta}^i_{k,t+1} \leftarrow \boldsymbol{\theta}^i_{k,t} + \eta_{\boldsymbol{\theta}} \nabla \mathcal{L}^{Z_i}(\boldsymbol{\theta}^i_{k,t})$
        else if phase *is* `recover` **then**
          $\boldsymbol{\theta}^i_{k,t+1} \leftarrow \boldsymbol{\theta}^i_{k,t} - \eta_{\boldsymbol{\theta}} \nabla \mathcal{L}^{Z_i}(\boldsymbol{\theta}^i_{k,t})$
    $\boldsymbol{\theta}_{k+1,0} \leftarrow \sum_{i=1}^N \frac{|Z_i|}{|Z|} \boldsymbol{\theta}^i_{k,T}$

3 DESIGN OF QUICKDROP

QuickDrop unlearns using SGA, but it does so on a synthetically generated dataset rather than with the original dataset. This synthetic dataset is also used during recovery. In Section 3.1, we detail how the significantly smaller size of the synthetic data unlocks significant efficiency gains compared to standard SGA. In Section 3.2, we formulate the process of generating synthetic data for unlearning and describe the algorithm for synthetic data generation, which operates in situ with FL training. We discuss in Section 3.3 how the synthetic data can be fine-tuned to boost model accuracy during recovery. Finally, we summarize the end-to-end workflow of QuickDrop in Section 3.4.

3.1 Synthetic Data Generation for Efficient Unlearning

As detailed in Algorithm 1, SGA executes unlearning rounds on D_f followed by recovery rounds on $D \backslash D_f$. To significantly reduce the volume of data involved when executing an unlearning request, we generate synthetic samples that condense critical gradient information from the original training into a small synthetic dataset. This process is also known as dataset distillation (DD) [41]. In our FL setting, each client $i \in N$ independently synthesizes a synthetic dataset S_i from its local dataset D_i such that $|S_i| \ll |D_i|$. More specifically, each client locally generates a synthetic per-class counterpart S_i^c of its original per-class data D_i^c. Per-class generation of S_i enables QuickDrop to perform both class- and client-level unlearning. Similar to the training samples used by FL, generated synthetic samples *never leave the device*. The unlearning algorithm $\mathcal{U}$ can thus be modified as $\boldsymbol{\theta}_f = \mathcal{U}(\boldsymbol{\theta}, S_f)$, where S_f is the synthetic counterpart of the forget dataset D_f. In other words, once the synthetic data is generated, QuickDrop can serve any unlearning requests by executing unlearning rounds on S_f and recovery rounds on $S \backslash S_f$. To exemplify, when perform class-level unlearning for class c, we have $S_f := \cup_i S_i^c$ where client i has class c. Similarly, when performing client-level unlearning of the i-th client, we have $S_f := S_i$. Since the synthetic data is orders of magnitude smaller in volume, the unlearning task can executed very efficiently as only a few unlearning and recovery rounds are required to unlearn and recover the knowledge, respectively.

3.2 Formulating the Distillation Task

The goals of standard DD differ from our task of synthetic data generation for unlearning. We first formally describe standard DD before formulating the task in the context of unlearning. Suppose we are given a training dataset D from a distribution P_D containing m_D pairs of training images and class labels $D = \{(\boldsymbol{x}_i, y_i)\}|_{i=1}^{m_D}$ where $\boldsymbol{x}_i \in \mathcal{X} \subset \mathbb{R}^d$, $y_i \in \{0, \ldots, C-1\}$ and C is the number of classes. Let ϕ be a learnable function (*e.g.*, a deep neural network) with parameters $\boldsymbol{\theta}$ that correctly predicts labels of images. One can learn the parameters of this function by minimizing an empirical loss over the training set:

$$\boldsymbol{\theta}^D = \arg\min_{\boldsymbol{\theta}} \mathcal{L}^D(\boldsymbol{\theta}) \tag{1}$$

where $\mathcal{L}^D(\boldsymbol{\theta}) = \frac{1}{m_D} \sum_{(\boldsymbol{x},y) \in D} \ell(\phi_{\boldsymbol{\theta}}(\boldsymbol{x}), y)$, $\ell(\cdot,\cdot)$ is a task-specific loss (*i.e.*, cross-entropy) and $\boldsymbol{\theta}^D$ is the minimizer of $\mathcal{L}^D$. The generalization performance of the obtained model $\phi_{\boldsymbol{\theta}^D}$ can be written as $\mathbb{E}_{\boldsymbol{x},y \sim P_{\mathcal{D}}}[\ell(\phi_{\boldsymbol{\theta}^D}(\boldsymbol{x}), y)]$. DD seeks to generate a small set of condensed synthetic samples with their labels, $S = \{(\boldsymbol{s}_i, y_i)\}|_{i=1}^{m_S}$ where $\boldsymbol{s}_i \in \mathbb{R}^d$ and $y_i \in \mathcal{Y}$ such that $m_S \ll m_D$. Similar to Eq. (1), one can train ϕ with these synthetic samples as follows:

$$\boldsymbol{\theta}^S = \arg\min_{\boldsymbol{\theta}} \mathcal{L}^S(\boldsymbol{\theta}) \tag{2}$$

where $\mathcal{L}^S(\boldsymbol{\theta}) = \frac{1}{m_S} \sum_{(\boldsymbol{s},y) \in S} \ell(\phi_{\boldsymbol{\theta}}(\boldsymbol{s}), y)$ and $\boldsymbol{\theta}^S$ is the minimizer of $\mathcal{L}^S$. The goal of standard DD is to obtain S such that the generalization performance of $\phi_{\boldsymbol{\theta}^S}$ is as close as possible to $\phi_{\boldsymbol{\theta}^D}$, *i.e.*,

$$\mathbb{E}_{\boldsymbol{x},y \sim P_{\mathcal{D}}}[\ell(\phi_{\boldsymbol{\theta}^D}(\boldsymbol{x}), y)] \simeq \mathbb{E}_{\boldsymbol{x},y \sim P_{\mathcal{D}}}[\ell(\phi_{\boldsymbol{\theta}^S}(\boldsymbol{x}), y)] \tag{3}$$

over the real data distribution $P_{\mathcal{D}}$.

3.2.1 Formulating DD for FU. While the above DD formulation may work for FU, generating general-purpose synthetic samples is a compute intensive process, requiring many optimization iterations [46]. To significantly reduce the computational overhead for clients, we reformulate the synthetic data generation task to specifically target unlearning instead of general-purpose DD. We refer to the outcome of $\mathcal{U}(\boldsymbol{\theta}, D_f)$ by $\boldsymbol{\theta}_{D_f}$ and denote the outcome of $\mathcal{U}(\boldsymbol{\theta}, S_f)$ by $\boldsymbol{\theta}_{S_f}$. The goal of DD in QuickDrop is to generate synthetic data such that the generalization performance of the unlearned model $\phi_{\boldsymbol{\theta}_{S_f}}$ is as close as possible to unlearned model $\phi_{\boldsymbol{\theta}_{D_f}}$, *i.e.*,

$$\mathbb{E}_{\boldsymbol{x},y \sim P_{\mathcal{D}}}[\ell(\phi_{\boldsymbol{\theta}_{S_f}}(\boldsymbol{x}), y)] \simeq \mathbb{E}_{\boldsymbol{x},y \sim P_{\mathcal{D}}}[\ell(\phi_{\boldsymbol{\theta}_{D_f}}(\boldsymbol{x}), y)] \tag{4}$$

over the global data distribution $P_{\mathcal{D}}$.

[1] Only clients with non-empty Z_i are required to participate.
[2] Clients can also be sub-sampled.

3.2.2 DD with Gradient Matching. We aim to generate synthetic samples such that the impact of forgetting the target synthetic data is similar to the effect of forgetting the target original data. When using SGA as the unlearning algorithm $\mathcal{U}$, intuitively, the synthetic data should be such that the gradients are as close as possible to the original data, thereby following the reverse trajectory along the optimization path when ascending. Additionally, generating such synthetic data must be local to the client, avoiding any transfer of private local information. Inspired by the dataset condensation algorithm of Zhao *et al.* [50] in centralized settings, we achieve this by matching gradients obtained over training and synthetic data over several time steps during the execution of FL training, locally at each client. Precisely, each client obtains its synthetic data S_i by minimizing the following:

$$\min_{S_i} \sum_{k=0}^{K-1} \sum_{t=0}^{T-1} d(\nabla_{\boldsymbol{\theta}} \mathcal{L}^{S_i}(\boldsymbol{\theta}^i_{k,t}), \nabla_{\boldsymbol{\theta}} \mathcal{L}^{D_i}(\boldsymbol{\theta}^i_{k,t})) \tag{5}$$

where $d(.;.)$ is a function that measures the distance between the gradients for $\mathcal{L}^{S_i}$ and $\mathcal{L}^{D_i}$ w.r.t $\boldsymbol{\theta}$, K denotes the total number of FL global rounds and T represents the number of local steps within each round. At each time step, the synthetic data samples are updated by running a few steps of an optimization algorithm `opt-alg`, *e.g.*, stochastic gradient descent (SGD):

$$S_i \leftarrow \texttt{opt-alg}_{S_i}(d(\nabla_{\boldsymbol{\theta}} \mathcal{L}^{S_i}(\boldsymbol{\theta}^i_{k,t}), \nabla_{\boldsymbol{\theta}} \mathcal{L}^{D_i}(\boldsymbol{\theta}^i_{k,t})), \varsigma_S, \eta_S) \tag{6}$$

where ς_S and η_S are the number of steps and the learning rate. In other words, the synthetic data absorbs the gradient information along the optimization trajectory of the FL model $\boldsymbol{\theta}^i_{k,t}$. The synthetic data, therefore, can be interpreted as a *compact compression* of all the gradient information, which is reused for fast gradient approximation in downstream unlearning.

We detail this procedure to generate synthetic data in Algorithm 2. Each client initializes its per-class synthetic dataset S^c_i by randomly picking samples from the original dataset D^c_i (lines 2-7). In FL settings, clients typically have a highly unbalanced number of samples per class (non-IIDness) [52]. Thus, we set the number of synthetic samples per class proportionately to the original per-class dataset size through the scale parameter s as $|S^c_i| = \lceil |D^c_i|/s \rceil$. The ceil function ($\lceil . \rceil$) ensures at least one sample per class in synthetic datasets if the class exists within D_i. Increasing s decreases the number of synthetic samples but also reduces the accuracy after recovery since samples contain less information. We set the scale parameter to 100, resulting in a synthetic dataset which is 1% in volume of the original dataset. We found this value of s to yield a reasonable balance between efficiency and effectiveness. While this incurs some storage overhead for clients, it is marginal compared to the volume of the original dataset. We also experiment with different values of s in Section 4.4.2.

Synthetic data generation happens in situ with FL training, resulting in a negligible compute overhead for clients (also see section 4.4.1). During each local update step, clients sample mini-batch pairs B^{D_i} and B^{S_i} from their original and synthetic datasets, respectively (line 12). They evaluate respective gradients on the current model parameter $\boldsymbol{\theta}^i_{k,t}$ (lines 13 and 14). While the clients use the gradient on the original data ($\nabla \mathcal{L}^{D_i}$) to perform the FL local update

Algorithm 2: Generating synthetic data during FL training in QuickDrop.

Input: Training datasets $\{D_i\}_{i=1}^N$, FL global rounds K, local update steps T, number of local update steps ς_S and learning rate η_S for synthetic samples, model learning rate $\eta_{\boldsymbol{\theta}}$ and scale parameter s.

1 ▷ Initialization
2 **for** *each client* $i = 1, \cdots, N$ **do**
3 **if** *client* i *has class* c **then**
4 $S^c_i \leftarrow$ Randomly pick $\lceil |D^c_i|/s \rceil$ samples from D^c_i
5 **else**
6 $S^c_i \leftarrow \emptyset$
7 $S_i \leftarrow \cup_c S^c_i$
8 **for** $k = 0, \cdots, K-1$ **do**
9 **for** *each client* $i = 1, \cdots, N$ *in parallel* **do**
10 Initialize $\boldsymbol{\theta}^i_{k,0} \leftarrow \boldsymbol{\theta}_{k,0}$
11 **for** $t = 0, \cdots, T-1$ **do**
12 Sample minibatch pairs $B^{D_i} \sim D_i$ and $B^{S_i} \sim S_i$
13 Compute $\nabla \mathcal{L}^{D_i}(\boldsymbol{\theta}^i_{k,t})$ on B^{D_i}
14 Compute $\nabla \mathcal{L}^{S_i}(\boldsymbol{\theta}^i_{k,t})$ on B^{S_i}
15 Update S_i using $d(\nabla \mathcal{L}^{D_i}, \nabla \mathcal{L}^{S_i})$ ▷ eq. (6)
16 ▷ Update FL model using the above gradient
17 $\boldsymbol{\theta}^i_{k,t+1} \leftarrow \boldsymbol{\theta}^i_{k,t} - \eta_{\boldsymbol{\theta}} \nabla \mathcal{L}^{D_i}(\boldsymbol{\theta}^i_{k,t})$
18 $\boldsymbol{\theta}_{k+1,0} \leftarrow \sum_{i=1}^N \frac{|D_i|}{|D|} \boldsymbol{\theta}^i_{k,T}$

Output: $\{S_i\}_{i=1}^N, \boldsymbol{\theta}_{K,0}$

(line 17), the gradient is also matched with the gradient on the synthetic data ($\nabla \mathcal{L}^{S_i}$) (line 15). The synthetic samples corresponding to the mini-batch B^{S_i} are then updated through the execution of `opt-alg` for ς_S steps. The server executes parameter aggregation like standard FL (line 18). By the end of FL training, QuickDrop generates per-client synthetic datasets $\{S_i\}_{i=1}^N$ along with the trained FL global model $\boldsymbol{\theta}_{K,0}$. Finally, we remark that QuickDrop employs a class-wise gradient matching, similar to [50], wherein the update of synthetic samples in B^{S_i} takes place class-wise. We omitted it from Algorithm 2 for presentation clarity.

3.3 Enhancing Recovery for SGA

Once the synthetic data is generated on each client, QuickDrop employs SGA to unlearn the forget dataset D_f through S_f. However, the model must still undergo the recovery phase on the retain dataset $D \backslash D_f$ to restore the performance on the remaining data. Recovery is akin to original FL training and hence requires high-quality samples. While recovery in QuickDrop can be conducted using $D \backslash D_f$, the volume of remaining data can be quite large, thus significantly affecting the efficiency. On the other hand, while we can use $S \backslash S_f$ to achieve efficient recovery, the restoration quality is not the same as when using $D \backslash D_f$. This is because the synthetic data generation was not targeted towards generalization. We address this issue through the following two amendments.

3.3.1 Data augmentation with original samples. We found that even including a few original samples in the synthetic datasets during recovery can significantly boost the restoration performance. In

our experimental setting, we randomly select samples from the original dataset with the same size as the synthetic dataset (*i.e.*, synthetic : selected = 1:1). Since the size of the synthetic dataset is 1% compared to the original dataset, the relative size of the mixed dataset is only 2% after combining selected samples. Therefore, its influence on storage overhead is still marginal.

3.3.2 Fine-tuning (optional). One can optionally enhance the recovery accuracy by conducting additional optimization steps on the previously generated synthetic data, referred to as fine-tuning. We remark that even without fine-tuning, QuickDrop already demonstrates excellent unlearning performance. For fine-tuning, we use the distillation algorithm of Zhao *et al.* [50] which targets generalization. More specifically, the algorithm performs gradient matching but across thousands of parameter initializations of the deep neural network ϕ, necessary to achieve good generalization. In section 4.4.1, we show that only 200 of such fine-tuning steps are sufficient to make the recovery accuracy nearly match that of the oracle (Retrain-Or). Fine-tuning is independently executed by each client on their synthetic dataset S_i.

3.4 End-to-end Workflow of QuickDrop

Finally, we summarize the end-to-end workflow of QuickDrop, also depicted in Figure 1. Our algorithm involves the following five steps:

(1) **FL training + synthetic data generation –** Clients initially train a global model via standard FL while also deriving synthetic data samples for unlearning. This procedure is described in Algorithm 2.
(2) **Data augmentation + fine-tuning –** Clients augment their synthetic sets with a very small number of original data samples and optionally conduct some fine-tuning steps to improve the quality of synthetic data for recovery.
(3) **Unlearning –** When an unlearning request arrives at the server, clients engage in SGA-based unlearning using their synthetic datasets. If the request concerns class-level unlearning, all clients containing the specific target class c are involved, *i.e.*, $S_f := \cup_i S_i^c$ where client i has class c. If the request concerns client-level unlearning, only the specific target client is involved, *i.e.*, $S_f := S_i$.
(4) **Recovery –** All clients with remaining data after removing the forget dataset engage in the recovery phase. These clients perform standard FL-like training (*i.e.*, using SGD) on their synthetic datasets.
(5) **Relearning –** Clients are able to efficiently relearn previously unlearned knowledge by performing standard FL-like training (*i.e.*, using SGD) on S_f.

4 EVALUATION

We conduct extensive evaluations to assess the computational efficiency and performance of QuickDrop using three standard datasets and against five FU baselines. Our evaluations cover the performance of a single unlearning request (Section 4.2), multiple unlearning requests (Section 4.3), the effect of fine-tuning and the scale factor (Section 4.4), the performance in larger networks (Section 4.5), client-level unlearning (Section 4.6), the performance of relearning (Section 4.7) and the computational overhead incurred by QuickDrop (Section 4.8).

4.1 Experimental Setup

We evaluate the performance of QuickDrop on three standard image classification datasets: MNIST [26], CIFAR-10 [25], and SVHN [31]. For all datasets, we use a ConvNet as the deep neural network backbone [14]. Its modular architecture contains D duplicate blocks, and each block has a convolutional layer with W (3×3) filters, a normalization layer N, an activation layer A, and a pooling layer P, denoted as $[W, N, A, P] \times D$. The default ConvNet (unless specified otherwise) includes 3 blocks, each with 128 filters, followed by InstanceNorm, ReLU, and AvgPooling modules. A linear classifier follows the final block.

All experiments are conducted on a machine equipped with an i5-10600K CPU and an RTX 2060 GPU. All source code is made available in a GitHub repository. [1]

Federated Learning. To generate local datasets with varying data heterogeneity, we adopt the Dirichlet distribution-based approach from previous works [24, 53]. The degree of non-IIDness is controlled by a parameter $\alpha \in [0, \infty)$, with lower values corresponding to higher heterogeneity. We fix $\alpha = 0.1$ for all our experiments involving non-IID data. This value is commonly used in literature and results in highly non-IID data partitions across clients. We also conduct experiments with IID distributions in Section 4.6. We conduct experiments with 10 and 20 total clients, except in Section 4.5, where we assess the performance of QuickDrop in a network with 100 clients. All clients participate in each round of FL training and unlearning unless mentioned otherwise.

Hyperparameters. In all experiments, we run $K = 200$ global rounds for FL training with 50 local steps per round *i.e.*, $T = 50$. We use a mini-batch size of 256 for FL training *i.e.*, to generate gradients on real images. We use SGD as the client local optimizer and a learning rate $\eta_\theta = 0.01$ for FL training. The learning rates are separately tuned for unlearning and recovery, which are 0.02 and 0.01, respectively. For all experiments with QuickDrop, we perform a single round of unlearning and two rounds for recovery, which we found to be the minimum for the model to sufficiently unlearn particular knowledge and restore the performance on the remaining data. All experiments use data augmentation for recovery (Section 3.3.1), but we disable fine-tuning for all experiments (*i.e.*, $F = 0$), except for the experiments reported in Section 4.4.1.

Synthetic data generation. For synthetic data generation, we set $\varsigma_S = 1$, $\eta_S = 0.1$ and use SGD as the `opt-alg` algorithm following [50]. When fine-tuning using the algorithm of Zhao *et al.* [50], we use their default hyperparameters while varying the number of fine-tuning steps F. More specifically, we vary the number of outer-loop steps while keeping the inner-loop steps fixed at 50. We initialize the synthetic samples $\{S_i\}_{i=1}^N$ as randomly selected real training samples from the original local client dataset. We found this to be more effective in our setting than initializing these samples from Gaussian noise. We use the same distance function d for gradient matching as in [50] and, except for the experiments in Section 4.4.2, fix the scale parameter $s = 100$ for all experiments.

[1] See https://github.com/sacs-epfl/quickdrop.

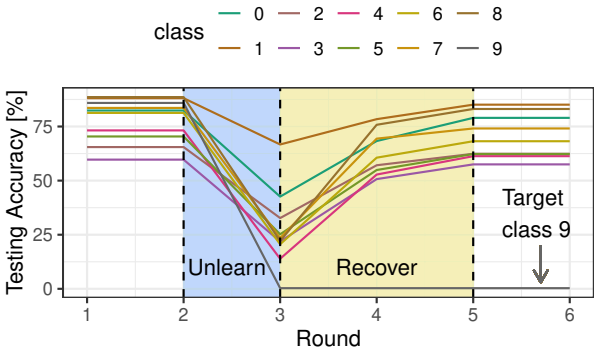

Figure 2: Class-wise testing accuracy on the CIFAR-10 dataset when unlearning class 9. QuickDrop successfully unlearns class 9 while retaining good performance on the remaining classes after recovery.

Baselines. We compare the performance of QuickDrop with the five FU baselines discussed in Section 2.3. These are Retrain-Or (the retraining oracle), SGA-Or (SGA on the original dataset) [42], FedEraser [27], FU-MP [39] and S2U [13]. Our baselines thus cover a diverse array of SOTA techniques on both client- and class-level unlearning. We focus on CV tasks to ensure compatibility with our selected baselines. Many of our baselines primarily demonstrate their performance on CV tasks, and some, like FU-MP, are specifically designed for CNNs due to their use of channel pruning techniques.

Metrics. We report testing accuracy as the Top-1 accuracy achieved on the specific testing data of each dataset (*e.g.*, class-wise testing accuracy when doing class-level unlearning). For a more fine-grained comparison, we also report the accuracy on the forget dataset, referred to in this section as the *F-Set*, and on the retain dataset, referred to as the *R-Set*. The goal of any FU approach is to match the accuracy of Retrain-Or (*i.e.*, oracle's performance) on both the F-Set and R-Set. We run each experiment five times and report averaged values.

4.2 Performance of a Single Unlearning Request

In this section, we will show the effectiveness of QuickDrop by evaluating the efficiency and effectiveness when doing a single unlearning request.

4.2.1 Unlearning a Single Class. We first quantify the change in testing accuracy of target and non-target classes after the unlearning and recovery stages, using the CIFAR-10 dataset and 10 clients. The network collaboratively unlearns from the model the knowledge corresponding to class 9 (digit "9") by performing one round of unlearning and two rounds of recovery. Figure 2 shows the testing accuracy for each class during six rounds in different colors when unlearning with QuickDrop. When QuickDrop starts the unlearning stage (round 2), we observe a rapid accuracy drop on the target class while the accuracy of non-target classes decreases as well. This is because SGA introduces some noise that affects non-target classes, even though the model parameters changed by SGA are mainly for unlearning the knowledge of the target class. We observe that even *a single unlearning round is sufficient to unlearn all knowledge of the target class* (its accuracy drops to 0.82%), and executing additional unlearning rounds is futile. Therefore, we execute a single unlearning round for QuickDrop in all remaining experiments. The recovery stage starts immediately after the unlearning stage (round 3). Figure 2 shows that the accuracies of non-target classes after two recovery rounds are almost restored to their original values. We observed that executing additional recovery rounds did not improve the performance of non-target classes. In the following section, we validate that the accuracies obtained by QuickDrop are consistent with those from Retrain-Or, demonstrating effective unlearning.

4.2.2 Accuracy against Baselines. Next, we compare the performance of QuickDrop with our baselines on CIFAR-10 when unlearning a single class. Table 2 shows the accuracy on the F-Set and R-Set for different FU approaches and after each stage (unlearning and recovery). We remark that there is no recovery stage for Retrain-Or. Table 2 shows that, after the unlearning stage, all approaches effectively eliminate the knowledge of the target class in the model as the F-Set accuracy matches that of Retrain-Or which achieves 0.81%. QuickDrop achieves 0.82% on the F-set, demonstrating the effectiveness of synthetic samples for unlearning. After recovery, all baselines restore the accuracy of the R-Set close to the values achieved with Retrain-Or. The accuracy of QuickDrop after recovery, 70.48%, is slightly lower than that of the baselines except for FedEraser. This is because the synthetic samples do not perfectly represent the original dataset, as they are optimized for unlearning. However, additional fine-tuning of synthetic datasets can close this gap at the cost of additional computation (see Section 4.4.1). Nonetheless, we conclude that QuickDrop effectively unlearns data samples with minimal impact on the performance of remaining samples. More importantly, QuickDrop unlearns significantly more efficiently than the baselines, thanks to the synthetic samples, as we show in Section 4.2.4.

4.2.3 Membership Inference Attack. To further assess the effectiveness of unlearning with QuickDrop, we conduct a membership inference attack (MIA) on the unlearned model, which follows related work on MU [9]. The MIA aims to determine whether a particular

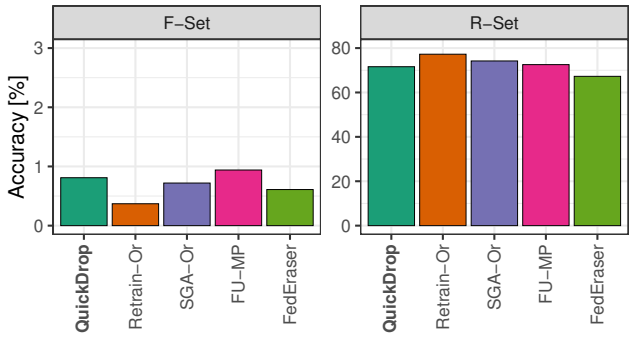

Figure 3: The membership inference attack (MIA) accuracy of all baselines after unlearning on the CIFAR-10 dataset with 10 clients and non-IID partitioning.

Table 2: The accuracy and computation cost of QuickDrop and FU baselines under class-level unlearning on the CIFAR-10 dataset, with non-IID data distributions ($\alpha = 0.1$) and in a 10-client network. The F-Set and R-Set denote the accuracy on the forget dataset and the retain dataset. The speedup is measured with respect to Retrain-Or.

Stage	Unlearning					Recovery					Unlearn. + Recovery	
	Accuracy		Computation Cost			Accuracy		Computation Cost			Total	
FU approach	F-Set	R-Set	Round	Time (s)	Data Size	F-Set	R-Set	Round	Time (s)	Data Size	Time (s)	Speedup
Retrain-Or	0.81%	74.95%	30	7239.58	45 000	—	—	—	—	—	7239.58	1×
FedEraser	0.02%	22.01%	10	2637.42	45 000	0.01%	69.67%	3	764.83	45 000	3402.25	2.12×
SGA-Or	0.75%	48.69%	2	495.17	5000	1.03%	74.83%	2	551.33	45 000	1046.50	6.92×
FU-MP	0.12%	11.58%	1	61.36	50 000	0.09%	73.96%	4	953.62	45 000	1014.98	7.13×
QuickDrop	0.82%	37.68%	1	5.03	100	0.85%	70.48%	2	10.58	900	**15.61**	**463.7**×

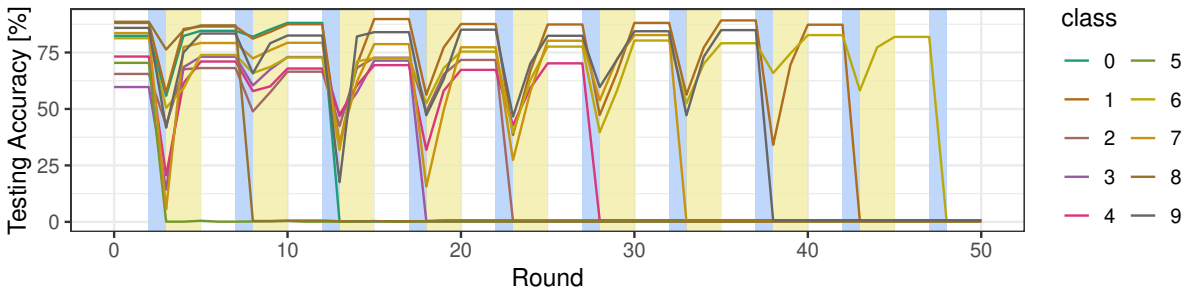

Figure 4: The accuracy of each class with sequential unlearning requests on CIFAR-10 and with $\alpha = 0.1$. We unlearn a random class every five rounds and highlight the unlearning and recovery stages in blue and yellow shades, respectively. The class unlearning order is [5, 8, 0, 3, 2, 4, 7, 9, 1, 6].

sample is included in the model's knowledge. We implement the MIA according to the settings in [16] and measure how often the attack model classifies a data sample from deleted data as a training sample of the unlearned model, serving as an alternative metric to test accuracy. Figure 3 shows these MIA accuracies on the F-Set and R-Set after the unlearning algorithm terminates. The performance of Retrain-Or can be considered optimal since the trained model has never seen the unlearned samples. For all approaches, the MIA accuracy on the F-Set is below 1%. We also observe that the MIA accuracy of QuickDrop on the R-Set (71.62%) is competitive with that of the baselines (67.28 − 74.21%) while the oracle achieves 77.25%. Thus, QuickDrop effectively unlearns knowledge from a trained model, evidenced by its MIA performance.

4.2.4 Computation Efficiency. Table 2 also shows the computational cost for unlearning and recovery in terms of rounds, time required, and the number of data samples involved in executing these rounds. The computation costs (round and time) correspond to the attainment of convergence in the specific phase (unlearning or recovery) for each baseline. We observe significant differences in computation cost between the evaluated FU approaches. Since DD reduces the number of samples for each client, the unlearning stage in QuickDrop only takes 5.03 s, and 10.58 s in the recovery stage; both stages are completed in just 15.61 s. This efficiency is because a round of unlearning and recovery with QuickDrop only involves 100 and 900 data samples, respectively. Although SGA-Or only needs two rounds each to unlearn and recover, it takes much longer (1046.50 s) in total than QuickDrop (15.61 s) since it must operate on the complete original data (5000 data samples in the unlearning stage and 45 000 samples in the recovery stage). While Retrain-Or is the simplest FU approach with the highest R-Set accuracy after unlearning, its computational time renders this approach infeasible in many scenarios, which is 6.92× higher than SGA-Or and 463.7× higher than QuickDrop. We note that FU-MP employs a different technique than other baselines, *i.e.*, model pruning, while its recovery stage is the same. Since model pruning only depends on information obtained from inference, which can be done relatively quickly, the unlearning is relatively fast (61.36 s). However, the recovery is still slow, resulting in a total time of 1014.98 s in comparison to 15.61 s for QuickDrop. Thus, QuickDrop remains 65× faster than FU-MP.

Although the computation efficiency of FedEraser improves over Retrain-Or, it stays relatively low (3402.25 s) due to the model update calibration step, which requires extra model training on all the remaining data of non-target classes. Consequently, QuickDrop also is significantly faster than FedEraser (218×). We note that FedEraser also incurs significant storage costs to retain historical model updates, which increases linearly with the number of clients and executed FL rounds. Thus, from Table 2, we conclude that QuickDrop achieves quick unlearning and recovery, boasting a speedup of 463.7× over Retrain-Or and 65 − 218× compared to other baselines.

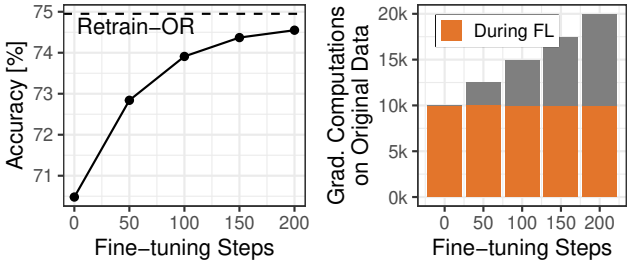

Figure 5: The accuracy on the R-Set after recovery (left) and the number of gradient computations performed on original data (right) when doing additional fine-tuning steps on CIFAR-10. The portion in orange corresponds to FL training, while the gray portion corresponds to new gradients computed for fine-tuning. QuickDrop nearly matches the accuracy of Retrain-Or at an extra gradient cost no higher than that of FL training.

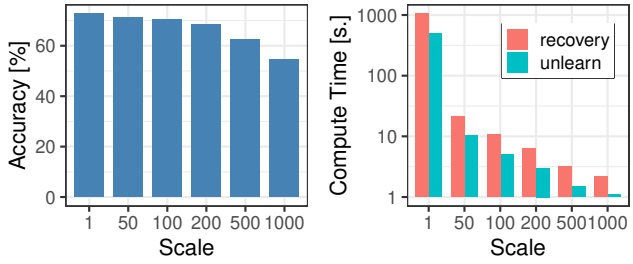

Figure 6: The accuracy of the R-Set after recovery (left) and the total compute time required for unlearning and recovery, for different scales s.

4.3 Performance of Multiple Unlearning Requests

In real-world settings, clients may continually launch unlearning requests. Therefore, we go beyond existing work on FU and evaluate the performance of QuickDrop with sequential class unlearning requests. Figure 4 shows the accuracies when sequentially unlearning all ten CIFAR-10 classes in random order. We observe that the unlearning phase for each target class results in low testing accuracy as before. Although the accuracies of non-target classes also drop after the unlearning stage, they are rapidly restored in the recovery stages while leaving the *low accuracy of the unlearned classes unaffected*. Therefore, Figure 4 shows the capability of QuickDrop in executing multiple unlearning requests.

4.4 Sensitivity Analysis

We now experiment with the most important parameters of QuickDrop and analyze their sensitivity on the achieved accuracy and efficiency. We first explore the impact of doing additional fine-tuning steps and then analyze the impact of the scale factor.

4.4.1 Impact of Fine-tuning. As discussed in Section 3.1, while the synthetic samples are optimized for unlearning, they might not perform sufficiently well during recovery. To offset this, QuickDrop allows clients to perform additional fine-tuning steps (F) to refine all synthetic samples using the algorithm of Zhao *et al.* [50]. Figure 5 (left) shows the accuracy of QuickDrop on the R-Set after the recovery stage when doing more fine-tuning (*i.e.*, increasing F from 0 to 200). We also show with a dashed horizontal line the accuracy of Retrain-Or (74.95%), which we consider optimal. We observe an increase in accuracy as F increases: from 70.48% at $F = 0$ to 74.55% at $F = 200$. More fine-tuning, however, comes at the cost of additional computation. Figure 5 (right) shows the total number of gradient computations performed on the original CIFAR-10 dataset for one client. This figure marks the portion of gradients generated during FL training in orange, while the portion of gradients in gray corresponds to the ones computed for fine-tuning. As F increases to 200, the number of gradients computed for fine-tuning (10k) match that of FL training (10k). Consequently, with an extra gradient cost no higher than that of FL training, QuickDrop clients can achieve performance parity with Retrain-Or through fine-tuning.

4.4.2 Impact of the Scale Parameter. The scale parameter s determines the ratio of original to synthetic samples per class and has a key impact on the computational efficiency and accuracy of QuickDrop. We explore the impact of this parameter by varying s from 1 to 1000, using the same setup as the experiment described in Section 4.2 (with the CIFAR-10 dataset and 10 clients). Figure 6 (left) shows the accuracy on the R-Set after recovery, for the considered values of s. As s increases, the accuracy on the R-set after recovery decreases. This is because increasing s results in lower number of synthetic samples and consequently more compression, inducing difficulty in unlearning. This decrease in accuracy is less noticeable when ranging s between 1 and 200. With $s = 1$ and $s = 100$, we observe an accuracy of 72.67% and 70.48%, respectively. The accuracy achieved when using only original samples is 74.83% (corresponding to SGA-Or). However, for $s > 200$, accuracy drops rapidly, reaching just 54.69% at $s = 1000$, where clients often have only one synthetic sample per class. Nonetheless, for this experiment setting, QuickDrop is able to retain high accuracies when $s \leq 100$.

Figure 6 (right) shows the total computation time (in logarithmic scale) incurred by all clients to execute the unlearning and recovery stages for different values of s. As s increases, the time to execute these stages decreases significantly. For instance, while the unlearning time for $s = 1$ is just over 8 minutes, it drops to only 5 seconds for $s = 100$ and as low as 1 second for $s = 1000$. Similarly, the recovery time decreases from 17.4 minutes for $s = 1$ to 10.6 seconds for $s = 100$. This decrease is because with an increasing value of s, clients will have less samples, thus decreasing the compute costs. Based on these results, we use $s = 100$ in all our experiments which achieves a good trade-off between accuracy and compute efficiency.

4.5 Performance in Larger Networks

We next analyze the unlearning performance of QuickDrop and other baselines for class-level unlearning in a 100-client network with the SVHN dataset [31]. Notably, our chosen network size surpasses the typical scale explored in related FU research [27, 42]. SVHN is a larger dataset compared to MNIST and CIFAR-10, containing 10 classes and more than 600 000 samples. In each round, the server selects a random 10% subset of clients to update the model

Table 3: The accuracy and computation cost of different FU approaches with 100 clients on the SVHN dataset for *class-level* unlearning. We report the performance when the algorithm terminates *i.e.*, after both unlearning and recovery.

Metric	Accuracy		Computation Cost	
FU approach	F-Set	R-Set	Time (s)	Speedup
Retrain-Or	0.34%	88.39%	10483.51	1×
FedEraser	0.38%	82.98%	2447.80	4.28×
SGA-Or	0.66%	86.47%	1276.13	8.21×
FU-MP	0.73%	85.63%	1927.43	5.43×
QuickDrop	0.81%	84.96%	32.09	**326.69**×

Table 4: The accuracy of QuickDrop and other baselines for *client-level* unlearning on CIFAR-10 (20 clients), with non-IID ($\alpha = 0.1$) and IID data distributions. We report the performance after unlearning and recovery.

Distribution	Non-IID ($\alpha = 0.1$)		IID	
FU approach	F-Set	R-Set	F-Set	R-Set
Retrain-Or	10.48%	73.69%	70.81%	71.64%
FedEraser	16.57%	69.85%	65.29%	66.04%
S2U	19.72%	70.25%	70.63%	71.28%
SGA-Or	9.58%	72.63%	69.32%	70.25%
QuickDrop	11.57%	70.89%	68.59%	68.48%

during training and recovery, while the participation rate for unlearning is 100%. Table 3 shows for each approach the accuracy on the F-Set and R-Set when unlearning class 9. Even with 100 clients, QuickDrop effectively unlearns class knowledge, achieving F-set accuracy of 0.81% while Retrain-Or achieves 0.34%. Furthermore, QuickDrop shows competitive accuracy on the R-Set (84.96%) compared to the baselines (82.98 − 86.47%), despite the large number of clients and samples in the training dataset. While Retrain-Or achieves slightly higher R-Set accuracy (88.39%), QuickDrop shows a massive 326.69× speedup over Retrain-Or.

4.6 Client-level Unlearning

Our previous evaluations focused on class-level unlearning. We now evaluate the effectiveness of QuickDrop on client-level unlearning where the goal is to erase the data samples of a specific *target client* from the trained model. Support for client-level unlearning is essential to adhere to privacy regulations such as the right to be forgotten [12]. We remark that FU-MP cannot perform client-level unlearning as it is specifically designed for class-level unlearning. We report evaluations on the CIFAR-10 dataset using two different data distributions: Non-IID (with $\alpha = 0.1$) and IID (uniform distribution). The target unlearning client is randomly selected.

Table 4 shows the performance of QuickDrop against baselines after the unlearning and recovery stages terminate. We first discuss the non-IID scenario. We observe that QuickDrop achieves 11.57% on the F-Set, remaining close to the oracle (10.48%) while the baselines achieve 9.58 − 19.72%. Note that the F-Set accuracies are higher than with class-level unlearning (see Table 2). This is because even though we unlearned the data samples of a particular client, some features associated with the data of the target client might still be embedded in the model's knowledge through other clients. Therefore, even after unlearning, some forgotten samples are correctly classified. Concerning the accuracy on the R-Set, Table 4 shows that QuickDrop (70.89%) remains very competitive with the baselines (69.85 − 72.63%) while the oracle achieves the highest (73.69%). These results are consistent with the accuracies obtained for class-level unlearning on non-IID data (Table 2).

In IID settings, we observe even higher accuracies on the F-Set after unlearning and recovery. For example, Retrain-Or achieves 70.81%, while QuickDrop reaches 68.59%. This is expected since, with an IID distribution, each client holds a similar type and quantity of data. Therefore, when we unlearn the target client, much of its contributed knowledge is still represented by the remaining data (R-Set) in the system, and the departure of the target client will barely impact the model performance. We also explored the effect of different α values ($\alpha = 1, 10$), alongside the reported $\alpha = 0.1$ and IID cases. Our findings were consistent with previous observations *i.e.*, as heterogeneity decreases (larger α), the impact of forgetting on final accuracy diminishes. Nevertheless, QuickDrop remains competitive with the baselines and offers substantial computational efficiency through synthetic data.

4.7 Performance of Relearning

This section provides additional accuracy results for single-class unlearning requests with two datasets and a network size of 20 clients. Complementing the previously presented CIFAR-10 results with 10 clients from Table 2, we include the remaining combinations in the left (CIFAR-10 with 20 clients) and the right (MNIST with 20 clients) partitions of Table 5. All these experiments follow the same setup as the results shown in Table 2. We further attach the results of the relearning stage in each table partition to show the effectiveness of different methods in relearning the eliminated class. The approach used in the relearning stage is the same for different baselines: we adopt traditional SGD-based model training to update the unlearned model with the forget dataset D_f. Note that QuickDrop still uses the synthetic data in the relearning stage while other baselines use the original data; QuickDrop thus can still keep its superior computation efficiency.

We observe that the performance after unlearning and recovery stages follows a similar trend as Table 2. In particular, almost all algorithms achieve low accuracy on F-Set, similar to that of Retrain-Or, demonstrating forgetting of the target class on both datasets. Concerning the R-Set accuracy on CIFAR-10 dataset, QuickDrop (65.78%) remains competitive with the baselines (67.38% − 70.04%) while the oracle achieves 71.48%. This gap is much lower on the MNIST dataset, where QuickDrop achieves 94.26% while the baselines and the oracle obtain 93.52% − 95.63%.

Table 5 also reports the accuracy on the F-Set and R-Set after relearning. Ideally, we want these accuracies to be high since we attempt to restore the model to the state before unlearning. Table 5 shows that all evaluated FU approaches successfully relearn the previously eliminated knowledge again. On the MNIST dataset,

Table 5: The accuracies on the F-Set and R-Set after the unlearning + recovery and *relearning* stages, for the CIFAR-10 and MNIST datasets with 20 clients and $\alpha = 0.1$. In each scenario, the goal is to match the performance of Retrain-Or on both the F-Set and the R-Set.

Distribution	CIFAR-10 (20 clients, $\alpha = 0.1$)				MNIST (20 clients, $\alpha = 0.1$)			
Stage	Unlearning + Recovery		Relearning		Unlearning + Recovery		Relearning	
FU approach	F-Set	R-Set	F-Set	R-Set	F-Set	R-Set	F-Set	R-Set
Retrain-Or	0.68%	71.48%	78.65%	71.83%	0.47%	95.63%	96.82%	95.74%
FedEraser	0.22%	67.38%	70.48%	68.22%	0.23%	93.52%	95.86%	95.43%
SGA-Or	0.71%	70.04%	75.83%	69.75%	0.51%	95.03%	96.28%	95.18%
FU-MP	0.59%	69.82%	—	—	0.31%	94.83%	—	—
QuickDrop	0.69%	65.78%	74.39%	66.21%	0.44%	94.26%	96.37%	94.58%

QuickDrop achieves an accuracy of 96.37% on the F-Set and 94.58% on the R-Set, almost matching Retrain-Or (96.82% and 95.74%). At the same time, QuickDrop keeps its superiority in computation efficiency since the relearning stage uses the compact synthetic dataset (66.7× faster than Retrain-Or and 47.29× than SGA-Or). We are unable to relearn using FU-MP. This is because model pruning irreversibly modifies the model structure, as described in Section 2.3. In conclusion, QuickDrop demonstrates high versatility in not only efficiently unlearning target classes but also relearning at low computation cost.

4.8 Computational Overhead of QuickDrop

QuickDrop generates synthetic samples during the FL training process, which incurs computational overhead. While we are able to re-use the gradients on original data ($\nabla \mathcal{L}^{D_i}$) computed by FL, we are still required to compute gradients on the synthetic data ($\nabla \mathcal{L}^{S_i}$) and update the synthetic samples (line 14 and 15 in Algorithm 2) during FL training. Table 6 shows the total and DD compute time for the 3 different datasets used across our experiments. The total time corresponds to the time required for FL training in QuickDrop, which includes DD time. The DD time corresponds to the time spent in executing line 14 and line 15 in Algorithm 2. We also compute the overhead of DD as percentage of the total compute time. Table 6 shows that the compute overhead of DD ranges around 50%, *i.e.*, it doubles the FL training time. Although QuickDrop slows down FL training, this initial investment is necessary to unlock significant efficiency gains in downstream unlearning as we showed previously.

Table 6: The total and DD compute time, and the overhead of DD, for all three datasets used in our experiments.

Dataset	Total Compute Time (s)	DD Compute Time (s)	Overhead
MNIST	4735	2557	54%
CIFAR-10	5360	2948	55%
SVHN	9079	4204	46.3%

5 DISCUSSION

We now discuss certain aspects of our evaluation and algorithmic aspects of QuickDrop that were not addressed before. We also discuss two limitations of QuickDrop in Section 5.1.

Unlearning Metric. First, we highlight that developing new metrics for unlearning is an active area of research [38]. While accuracy might not fully capture true unlearning, our approach aligns with SOTA FU work that reports unlearning via the accuracy metric [13, 27, 39]. Additionally, we reported the MIA performance to provide another metric for assessing unlearning.

Privacy of QuickDrop. We point out that the synthetically generated data in QuickDrop does not leave the client's device. Clients only exchange model parameters with the server during the unlearning and recovery phases. Hence, the privacy guarantees for QuickDrop remain the same as standard FL.

Partial Client Participation. In FL with many clients, FL algorithms usually have the parameter server choose a subset of clients to participate in model training during each round [8]. QuickDrop supports such partial client participation during both FL training and unlearning. Our SVHN experiments use a 10% participation rate for FL training. Even though the clients participate and consequently synthesize data only in a few rounds, the quality of unlearning is good. This is because data synthesis aims to eliminate only *contributed* knowledge, *i.e.*, corresponding to the round in which the client participated. Thus, this goal is met even with partial participation. Partial participation during unlearning and recovery is directly applicable in QuickDrop.

Frequency of Unlearning Requests. The computational benefits of QuickDrop are closely tied to the frequency of unlearning requests, *i.e.*, the benefits in compute time are effectively realized as there are more unlearning requests in the system. These requests can arise from individual clients exercising their right to be forgotten or organisations needing to remove outdated or erroneous data. QuickDrop requires an upfront investment in generating the synthetic dataset, but these compute costs are then amortized over the subsequent unlearning requests. Existing research on machine unlearning studies performance with hundreds of requests [3] and highlights the importance of efficiently managing multiple requests [47]. Recent work also proposes a streaming unlearning setting involving a sequence of data removal requests [21]. We thus anticipate a similar or even greater demand for efficiently

managing multiple unlearning requests when performing federated unlearning.

5.1 Limitations

Finally, we discuss two limitations of our approach.

Sample-level Unlearning. In its current form, QuickDrop supports only class-level and client-level unlearning. These two levels of unlearning already cover many applications of machine unlearning in federated setups. One might want to perform sample-level unlearning, where the goal is to unlearn a subset of data samples of a particular client. In QuickDrop, clients locally generate class-wise synthetic data. Hence, one way to adapt QuickDrop to enable sample-level unlearning is to consider subsets of data within each class. One can generate synthetic samples for each subset and then unlearn at the granularity of these subsets. We consider this challenge beyond the scope of current work and leave the exploration of these ideas for future work.

Compute and Storage Overhead. QuickDrop prolongs the training time for the original FL training, and requires additional storage for the synthetic datasets. We have evaluated QuickDrop's compute overhead in Section 4.8, showing that this overhead is between 46.3% and 55% in terms of prolonged training time on our evaluated datasets. The storage overhead of QuickDrop, determined by the scale parameter s, is $\frac{1}{s}$ of the local training dataset size per client. In Section 4.4.2, we have experimentally shown that decreasing the scale increases the size of the synthetic dataset but allows for more accurate unlearning. We find $s = 100$ to yield a reasonable balance between efficiency and effectiveness and use this value across all our experiments. This results in 1% storage overhead for QuickDrop.

6 RELATED WORK

We now discuss related work in the domain of machine and federated unlearning (Section 6.1) and dataset distillation (Section 6.2).

6.1 Machine and Federated Unlearning

Following the introduction of MU in [6], several algorithms have been proposed [3, 11, 15, 17, 18, 20]. These works focus mainly on unlearning knowledge from simple classification models, *e.g.*, for logistic regression, but are unsuitable for more complex models, *e.g.*, deep neural networks. Some algorithms have other restrictions and can only be applied to specific model architectures or scenarios, *e.g.*, [4] only fits random forests model and [32] is only for Bayesian learning. In traditional machine learning scenarios, all the local data of clients will be uploaded to the server for centralized management, so the server has a high level of flexibility to conduct arbitrary operations on all data. Therefore, various unlearning techniques (*e.g.*, ensemble learning for data splitting [3] or gradient amnesia of arbitrary batch [19]) are designed to operate in settings where training data is readily available. Such operations at data-level are not possible in FU, thus FU is more difficult than MU as outlined in Section 2.1. Besides the FU methods covered in Section 2.3, other recent approaches include [23, 44]. These works are not adopted as baselines either because they employ a similar unlearning technique as one of our baselines (*e.g.*, [23] is built upon SGA) or are tailored to specific scenarios (*e.g.*, [44] focuses on recommendation systems). Finally, among the very few theoretical works on FU, [37] tackle both communication efficiency and provable exact unlearning by leveraging total variation stability. We refer the reader to [28] for a more comprehensive overview of FU.

6.2 Dataset Distillation

Standard DD aims to replace a large training dataset with a significantly smaller one that can achieve the same generalization performance as the original training data [41]. DD can potentially speed up downstream tasks such as continual learning [22] and neural architecture search [33]. DD also has been previously leveraged in one-shot FL to significantly reduce communication cost compared to multi-round FL [36, 51]. Early DD approaches are based on core-set selection *i.e.*, identifying a subset of influential samples during training [1, 35]. Another class of algorithms synthesizes a set of new samples from the original dataset. The approach described in [50] is to match the gradients of a model trained on the original and synthetic data. Follow-up work has introduced distillation techniques based on trajectory gradient matching [7], differential data augmentation functions [48], distribution matching [49] and feature alignment [40]. While existing DD methods achieve leading performance, synthesizing samples targeted at generalization is highly compute intensive [46]. QuickDrop, in contrast, formulates DD for unlearning and synthesizes samples at lower computational overhead.

7 CONCLUSION

We introduced QuickDrop, a novel and efficient federated unlearning method in which clients generate and use synthetic datasets for unlearning. These synthetic datasets are a compact representation of the gradient information generated during training. To unlearn specific samples, clients execute stochastic gradient ascent (SGA) with synthetic dataset instead of the original training data. Recovery on remaining samples also takes place via synthetic datasets. Empirical evaluations using three standard datasets and SOTA baselines confirm the effectiveness and efficiency of QuickDrop, demonstrating a remarkable acceleration in the unlearning process compared to existing federated unlearning approaches.

ACKNOWLEDGMENTS

This work has been funded by the Swiss National Science Foundation, under the project "FRIDAY: Frugal, Privacy-Aware and Practical Decentralized Learning", SNSF proposal No. 10.001.796. This work was also supported by fundings from the Key-Area Research and Development Program of Guangdong Province (No. 2021B0101400003), Hong Kong RGC Research Impact Fund (No. R5060-19, No. R5034-18, No. R5011-23), Areas of Excellence Scheme (AoE/E-601/22-R), General Research Fund (No. 152203/20E, 152244/21E, 152169/22E, 152228/23E), Collaborative Research Fund (No. 8730102).

REFERENCES

[1] Olivier Bachem, Mario Lucic, and Andreas Krause. Practical coreset constructions for machine learning. *arXiv preprint arXiv:1703.06476*, 2017.
[2] Aurélien Bellet, Rachid Guerraoui, Mahsa Taziki, and Marc Tommasi. Personalized and private peer-to-peer machine learning. In *ICAIS*, pages 473–481. PMLR, 2018.

[3] Lucas Bourtoule, Varun Chandrasekaran, Christopher A Choquette-Choo, Hengrui Jia, Adelin Travers, Baiwu Zhang, David Lie, and Nicolas Papernot. Machine unlearning. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 141–159. IEEE, 2021.

[4] Jonathan Brophy and Daniel Lowd. Machine unlearning for random forests. In *International Conference on Machine Learning*, pages 1092–1104. PMLR, 2021.

[5] California State Legislature. California consumer privacy act of 2018. California Legislative Information, 2018. Available online: https://leginfo.legislature.ca.gov/faces/billTextClient.xhtml?bill_id=201720180AB375.

[6] Yinzhi Cao and Junfeng Yang. Towards making systems forget with machine unlearning. In *2015 IEEE symposium on security and privacy*, pages 463–480. IEEE, 2015.

[7] George Cazenavette, Tongzhou Wang, Antonio Torralba, Alexei A Efros, and Jun-Yan Zhu. Dataset distillation by matching training trajectories. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4750–4759, 2022.

[8] Zachary Charles, Zachary Garrett, Zhouyuan Huo, Sergei Shmulyian, and Virginia Smith. On large-cohort training for federated learning. *Advances in neural information processing systems*, 34:20461–20475, 2021.

[9] Min Chen, Zhikun Zhang, Tianhao Wang, Michael Backes, Mathias Humbert, and Yang Zhang. When machine unlearning jeopardizes privacy. In *Proceedings of the 2021 ACM SIGSAC conference on computer and communications security*, pages 896–911, 2021.

[10] Vikram S Chundawat, Ayush K Tarun, Murari Mandal, and Mohan Kankanhalli. Zero-shot machine unlearning. *IEEE Transactions on Information Forensics and Security*, 2023.

[11] Min Du, Zhi Chen, Chang Liu, Rajvardhan Oak, and Dawn Song. Lifelong anomaly detection through unlearning. In *Proceedings of the 2019 ACM SIGSAC conference on computer and communications security*, pages 1283–1297, 2019.

[12] European Union. Regulation (eu) 2016/679 of the european parliament and of the council of 27 april 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing directive 95/46/ec (general data protection regulation). Official Journal of the European Union, 2018. OJ L 119, 4.5.2016, p. 1–88.

[13] Xiangshan Gao, Xingjun Ma, Jingyi Wang, Youcheng Sun, Bo Li, Shouling Ji, Peng Cheng, and Jiming Chen. Verifi: Towards verifiable federated unlearning. *arXiv preprint arXiv:2205.12709*, 2022.

[14] Spyros Gidaris and Nikos Komodakis. Dynamic few-shot visual learning without forgetting. In *CVPR*, pages 4367–4375, 2018.

[15] Antonio Ginart, Melody Guan, Gregory Valiant, and James Y Zou. Making ai forget you: Data deletion in machine learning. *Advances in neural information processing systems*, 32, 2019.

[16] Aditya Golatkar, Alessandro Achille, Avinash Ravichandran, Marzia Polito, and Stefano Soatto. Mixed-privacy forgetting in deep networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 792–801, 2021.

[17] Aditya Golatkar, Alessandro Achille, and Stefano Soatto. Eternal sunshine of the spotless net: Selective forgetting in deep networks. In *CVPR*, pages 9304–9312, 2020.

[18] Aditya Golatkar, Alessandro Achille, and Stefano Soatto. Forgetting outside the box: Scrubbing deep networks of information accessible from input-output observations. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XXIX 16*, pages 383–398. Springer, 2020.

[19] Laura Graves, Vineel Nagisetty, and Vijay Ganesh. Amnesiac machine learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 11516–11524, 2021.

[20] Chuan Guo, Tom Goldstein, Awni Hannun, and Laurens Van Der Maaten. Certified data removal from machine learning models. *arXiv preprint arXiv:1911.03030*, 2019.

[21] Varun Gupta, Christopher Jung, Seth Neel, Aaron Roth, Saeed Sharifi-Malvajerdi, and Chris Waites. Adaptive machine unlearning. *Advances in Neural Information Processing Systems*, 34:16319–16330, 2021.

[22] Raia Hadsell, Dushyant Rao, Andrei A Rusu, and Razvan Pascanu. Embracing change: Continual learning in deep neural networks. *Trends in cognitive sciences*, 24(12):1028–1040, 2020.

[23] Anisa Halimi, Swanand Kadhe, Ambrish Rawat, and Nathalie Baracaldo. Federated unlearning: How to efficiently erase a client in fl? *arXiv preprint arXiv:2207.05521*, 2022.

[24] Tzu-Ming Harry Hsu, Hang Qi, and Matthew Brown. Measuring the effects of non-identical data distribution for federated visual classification. *arXiv preprint arXiv:1909.06335*, 2019.

[25] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[26] Yann LeCun. The mnist database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*, 1998.

[27] Gaoyang Liu, Xiaoqiang Ma, Yang Yang, Chen Wang, and Jiangchuan Liu. Federaser: Enabling efficient client-level data removal from federated learning models. In *2021 IEEE/ACM 29th International Symposium on Quality of Service (IWQOS)*, pages 1–10. IEEE, 2021.

[28] Ziyao Liu, Yu Jiang, Jiyuan Shen, Minyi Peng, Kwok-Yan Lam, Xingliang Yuan, and Xiaoning Liu. A survey on federated unlearning: Challenges, methods, and future directions. *ACM Comput. Surv.*, 57(1), October 2024.

[29] Mohammad Saeid Mahdavinejad, Mohammadreza Rezvan, Mohammadamin Barekatain, Peyman Adibi, Payam Barnaghi, and Amit P Sheth. Machine learning for internet of things data analysis: A survey. *Digital Communications and Networks*, 4(3):161–175, 2018.

[30] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *AISTATS*, pages 1273–1282. PMLR, 2017.

[31] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. 2011.

[32] Quoc Phong Nguyen, Bryan Kian Hsiang Low, and Patrick Jaillet. Variational bayesian unlearning. *Advances in Neural Information Processing Systems*, 33:16025–16036, 2020.

[33] Pengzhen Ren, Yun Xiao, Xiaojun Chang, Po-Yao Huang, Zhihui Li, Xiaojiang Chen, and Xin Wang. A comprehensive survey of neural architecture search: Challenges and solutions. *ACM Computing Surveys (CSUR)*, 54(4):1–34, 2021.

[34] Nicolò Romandini, Alessio Mora, Carlo Mazzocca, Rebecca Montanari, and Paolo Bellavista. Federated unlearning: A survey on methods, design guidelines, and evaluation metrics. *arXiv preprint arXiv:2401.05146*, 2024.

[35] Ozan Sener and Silvio Savarese. Active learning for convolutional neural networks: A core-set approach. In *International Conference on Learning Representations*, 2018.

[36] Rui Song, Dai Liu, Dave Zhenyu Chen, Andreas Festag, Carsten Trinitis, Martin Schulz, and Alois Knoll. Federated learning via decentralized dataset distillation in resource-constrained edge environments. In *2023 International Joint Conference on Neural Networks (IJCNN)*, pages 1–10, 2023.

[37] Youming Tao, Cheng-Long Wang, Miao Pan, Dongxiao Yu, Xiuzhen Cheng, and Di Wang. Communication efficient and provable federated unlearning. *Proc. VLDB Endow.*, 17(5):1119–1131, May 2024.

[38] Eleni Triantafillou, Peter Kairouz, Fabian Pedregosa, Jamie Hayes, Meghdad Kurmanji, Kairan Zhao, Vincent Dumoulin, Julio Jacques Junior, Ioannis Mitliagkas, Jun Wan, et al. Are we making progress in unlearning? findings from the first neurips unlearning competition. *arXiv preprint arXiv:2406.09073*, 2024.

[39] Junxiao Wang, Song Guo, Xin Xie, and Heng Qi. Federated unlearning via class-discriminative pruning. In *Proceedings of the ACM Web Conference 2022*, pages 622–632, 2022.

[40] Kai Wang, Bo Zhao, Xiangyu Peng, Zheng Zhu, Shuo Yang, Shuo Wang, Guan Huang, Hakan Bilen, Xinchao Wang, and Yang You. Cafe: Learning to condense dataset by aligning features. In *CVPR*, pages 12196–12205, 2022.

[41] Tongzhou Wang, Jun-Yan Zhu, Antonio Torralba, and Alexei A Efros. Dataset distillation. *arXiv preprint arXiv:1811.10959*, 2018.

[42] Leijie Wu, Song Guo, Junxiao Wang, Zicong Hong, Jie Zhang, and Yaohong Ding. Federated unlearning: Guarantee the right of clients to forget. *IEEE Network*, 36(5):129–135, 2022.

[43] Qiang Yang, Yang Liu, Tianjian Chen, and Yongxin Tong. Federated machine learning: Concept and applications. *ACM TIST*, 10(2):1–19, 2019.

[44] Wei Yuan, Hongzhi Yin, Fangzhao Wu, Shijie Zhang, Tieke He, and Hao Wang. Federated unlearning for on-device recommendation. In *Proceedings of the Sixteenth ACM International Conference on Web Search and Data Mining*, pages 393–401, 2023.

[45] Haibo Zhang, Toru Nakamura, Takamasa Isohara, and Kouichi Sakurai. A review on machine unlearning. *SN Computer Science*, 4(4):337, 2023.

[46] Lei Zhang, Jie Zhang, Bowen Lei, Subhabrata Mukherjee, Xiang Pan, Bo Zhao, Caiwen Ding, Yao Li, and Dongkuan Xu. Accelerating dataset distillation via model augmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11950–11959, 2023.

[47] Zijie Zhang, Yang Zhou, Xin Zhao, Tianshi Che, and Lingjuan Lyu. Prompt certified machine unlearning with randomized gradient smoothing and quantization. *Advances in Neural Information Processing Systems*, 35:13433–13455, 2022.

[48] Bo Zhao and Hakan Bilen. Dataset condensation with differentiable siamese augmentation. In *International Conference on Machine Learning*, pages 12674–12685. PMLR, 2021.

[49] Bo Zhao and Hakan Bilen. Dataset condensation with distribution matching. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 6514–6523, 2023.

[50] Bo Zhao, Konda Reddy Mopuri, and Hakan Bilen. Dataset condensation with gradient matching. In *International Conference on Learning Representations*, 2021.

[51] Yanlin Zhou, George Pu, Xiyao Ma, Xiaolin Li, and Dapeng Wu. Distilled one-shot federated learning. *arXiv preprint arXiv:2009.07999*, 2020.

[52] Hangyu Zhu, Jinjin Xu, Shiqing Liu, and Yaochu Jin. Federated learning on non-iid data: A survey. *Neurocomputing*, 465:371–390, 2021.

[53] Zhuangdi Zhu, Junyuan Hong, and Jiayu Zhou. Data-free knowledge distillation for heterogeneous federated learning. In *ICML*, pages 12878–12889. PMLR, 2021.