Asynchronous Federated Unlearning

Ningxin Su, Baochun Li
Department of Electrical and Computer Engineering
University of Toronto

***Abstract*—Thanks to regulatory policies such as the General Data Protection Regulation (GDPR), it is essential to provide users with *the right to erasure* regarding their own private data, even if such data has been used to train a neural network model. Such a *machine unlearning* problem becomes even more challenging in the context of federated learning, where clients collaborate to train a global model with their private data. When a client requests its data to be erased, its effects have already gradually permeated through a large number of clients, as the server aggregates client updates over multiple communication rounds. All of these affected clients need to participate in the retraining process, leading to prohibitive retraining costs with respect to the wall-clock training time.**

In this paper, we present the design and implementation of KNOT, a new clustered aggregation mechanism custom-tailored to asynchronous federated learning. The design of KNOT is based upon our intuition that, with asynchronous federated learning, clients can be divided into clusters, and aggregation can be performed within each cluster only so that retraining due to data erasure can be limited to within each cluster as well. To optimize client-cluster assignment, we formulated a lexicographical minimization problem that could be transformed into a linear programming problem and solved efficiently. Over a variety of datasets and tasks, we have shown clear evidence that KNOT outperformed the state-of-the-art federated unlearning mechanisms by up to 85% in the context of asynchronous federated learning.

I. INTRODUCTION

As one of the emerging paradigms in distributed machine learning, federated learning (FL) assumes that edge devices, also known as clients, collaboratively train a global model with their local private data, which are further aggregated by a central server. Compared to centralized training of machine learning models, such a design allows clients to keep and process their data locally, rather than sending them to the central server. With the recent introduction of data protection laws such as the General Data Protection Regulation (GDPR) in the European Union, however, it is essential to provide users with *the right to erasure* regarding their own private data, even if such data have already been used to train the global model in federated learning. In the context of centralized training, *the right to erasure* has led to recent research on *machine unlearning* [1].

Erasing the effects of select data samples in a trained model, however, is quite challenging to achieve, especially if we do not wish to incur the exorbitant costs of retraining from scratch. Without a doubt, it is even more challenging to accomplish machine unlearning in the context of federated learning: erasing data samples from one client requires a large number of clients to engage in a retraining process, due to server aggregation in each communication round. One can conceive of the effects of one data sample gradually propagating to models used for local training at more clients, as federated learning progresses. Therefore, machine unlearning in the federated learning setting, called *federated unlearning*, requires mechanisms that are even more carefully designed to minimize their retraining costs.

In federated unlearning, the primary objective is to minimize the time it takes to complete the retraining process, when a subset of the clients request the erasure of some of their data samples. In FedEraser [2], an approximation algorithm has been proposed as an alternative retraining mechanism, such that fewer rounds are needed for the retraining process. It is not straightforward, however, to evaluate the effectiveness of such approximation algorithms for retraining, and FedEraser proposed to use the accuracy of membership inference attacks for such evaluations. Though such approximation algorithms in the literature could reduce the retraining overhead as compared to the naive mechanism of retraining from scratch, they have not been able to reduce the number of clients involved in such a retraining process. As the effects of one data sample to be erased permeate through the entire group of clients as server aggregation progresses, most — if not all — of the clients may need to participate in the retraining process.

In the conventional context of *synchronous* federated learning, as most of the clients participate in the time-consuming retraining process, the overall performance — as measured by the wall-clock time of converging to a target accuracy — will be severely affected. If we wish to improve the overall performance even with the possibility of retraining, we may consider minimizing the number of clients that participate in the retraining process. Towards this objective, we should consider operating the FL training session in an *asynchronous* fashion: the server does not need to wait for all its selected clients to report their model updates, and proceeds with its aggregation process as soon as the model update from a minimum number of clients arrives. It has been shown in the literature (*e.g.*, PORT [3] and FedBuff [4]) that the performance of such an asynchronous paradigm is far superior to synchronous FL, especially in cases where clients are heterogeneous in their training capabilities.

In asynchronous FL, different clients progress at different speeds naturally in their local training. Intuitively, if we allow some clients to move forward towards global convergence while retraining only a small subset of the clients as data samples are erased, the inherent overhead of retraining can be substantially mitigated. A simple yet effective mechanism is to divide all clients into a small number of clusters, and aggregate client updates within the confines of each cluster only. The immediate effect of such *clustered aggregation* is that, if any client requests its data samples to be erased, only clients within the same cluster need to be retrained, while other unaffected clusters may continue with normal FL training.

In this paper, we introduce the new paradigm of *asynchronous federated unlearning*, and propose KNOT, our new federated unlearning mechanism that uses clustering to mitigate the negative effects of unlearning. But how should clients be assigned to each of the clusters? In asynchronous FL, fast clients are known to contribute more to the training process; as such, the local training speed of each client should be considered, and fast clients should be assigned to the same clusters. On the other hand, due to the inherent non-i.i.d. data heterogeneity in federated learning, model updates from different clients are known to be naturally clustered [5] with respect to their similarity to one another. Thus, one may consider assigning clients with similar model updates to the same clusters as well.

The main contribution of this paper is to transfer such an intricate problem of assigning clients to clusters to a solvable optimization problem. With a precise definition of a pair-wise *match rating* between clients and pre-determined clusters, the problem of minimizing the wall-clock time to process retraining is equivalent to minimizing the match ratings of all clients, which we further formulate as an integer lexicographical minimization problem. With empirical evidence, we show that solving such an optimization problem offers superior performance than simply clustering the clients randomly.

However, solving integer optimization problems at any scale in an online fashion is likely to be impractical. As a highlight of this paper, our analysis shows that our lexicographical minimization problem has a separable convex objective function and a totally unimodular constraint matrix, and can thus be solved directly using an off-the-shelf *linear* program solver [6].

Highlights of our original contributions in this paper are as follows. *First*, to our best knowledge, we are the first to consider federated unlearning in the asynchronous FL setting. *Second*, we proposed KNOT, a new federated unlearning mechanism that is designed to mitigate the retraining overhead by assigning clients to clusters, and performing aggregation within each cluster only. *Third*, we addressed the key challenge of assigning clients to clusters by formulating it as an integer lexicographical minimization problem, and our theoretical analyses show that such a problem can be efficiently solved using a linear program solver. *Finally*, we implemented KNOT, as well as the state-of-the-art mechanisms in the literature, in a scalable framework for FL research, and evaluated its performance with a variety of image classification and language modeling tasks. We show that KNOT has substantially outperformed the state-of-the-art [2], [7] with respect to the wall-clock time for converging to a target accuracy in the federated unlearning process.

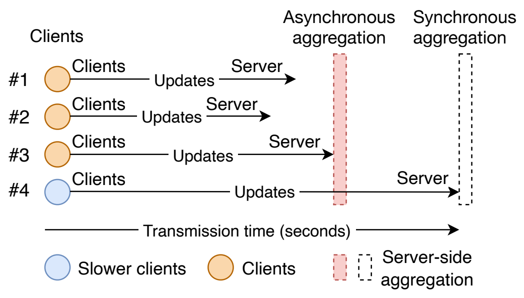


Fig. 1. In asynchronous FL, clients spend different amounts of time for training and communication, and the server only needs to wait for a subset of clients to report back.

II. Preliminaries and Related Work

As an emerging distributed machine learning paradigm, federated learning (FL) keeps private data at the clients to protect their privacy [8]. In each communication round, the server sends the initial model to a selected subset of clients, and clients use their local data to train the received model and send updates back to the server. The server then uses an aggregation algorithm to aggregate all received updates into a new model, in preparation for the next communication round.

Asynchronous federated learning. In practice, clients have different computing power and communication capabilities, leading to different amounts of time for them to finish training. In order to speed up the FL training session, asynchronous aggregation on the server has recently been proposed [4], [9]. The gist of asynchronous aggregation is simple: the server only needs to wait for a minimum number of faster clients before aggregation proceeds. For example, in Fig. 1, the slowest client, #4, is not waited for by an asynchronous server before aggregation, which may speed up the overall convergence.

Asynchronous FL naturally favors faster clients in each communication round, who contribute more to the convergence of the global model in the training session. The disadvantage of asynchronous aggregation, however, is that slower clients will inevitably be training based on models that are out-of-date and *stale*, and as their updates are used for aggregation, the accuracy of the aggregated model can be negatively affected. Still, it has been shown [3], [4] that the benefits of asynchronous aggregation outweigh its shortcomings and that the overall wall-clock time it takes to converge is substantially improved over synchronous aggregation.

Machine unlearning. With the recent emergence of regulatory policies that require corporations to provide the right to erase private data from users, machine unlearning has garnered increasing research attention. As illustrated in Fig. 2, with the machine unlearning process, a new neural network model should be produced by *unlearning* all the effects of any private data that are requested to be erased. Existing research on machine unlearning aims to produce the unlearned model with the highest possible efficiency, without taking the naïve approach of retraining an initial model from scratch.

Towards the goal of making the machine unlearning process more efficient, the first machine unlearning mechanism

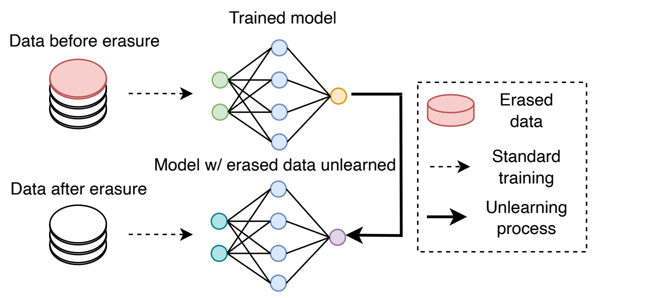


Fig. 2. The primary goal of the machine unlearning process is to produce a new neural network model with the erased data unlearned, equivalent to a model initialized and then trained without using the erased data.

was proposed by Cao *et al.* [10], which was asymptotically faster than retraining from scratch by transforming learning algorithms into a summation form. Subsequent works in the literature focused on *approximating* the model after unlearning the erased data [11]–[14], sometimes with theoretical guarantees that an adversary cannot extract information about the erased data [15]. Towards the objective of reducing the overhead of retraining, Ginart *et al.* [16] proposed two clustering algorithms to divide data into clusters, whereas other similar mechanisms were proposed more recently [1], [17].

Federated unlearning. The vanilla machine unlearning process may be applied in the context of federated learning, which is referred to as *federated unlearning* in the literature. As clients will not share their private data with the server, the federated unlearning process can only be performed on the clients. In addition, as the server aggregates client updates across multiple communication rounds, the effects of data from one client will eventually propagate to models trained by all the participating clients. When any data needs to be erased, all the clients may need to retrain from scratch over a time-consuming retraining process. In response to this challenge, FedEraser [2] designed an approximation algorithm to improve the efficiency of such a retraining process. Alternative approximation algorithms have been proposed more recently, using knowledge distillation [18] or class-discriminative pruning [19]. A common technique to evaluate the effectiveness of these approximation algorithms was to use membership inference attacks to show that any remaining contributions of the erased data cannot be detected by the attackers.

However, these existing approximation mechanisms may have violated regulatory policies such as GDPR, which stipulates that any effects of data must be erased completely without a trace. In response, Liu *et al.* [7] proposed to improve the efficiency of the retraining phase, which may be the only law-compliant way to properly perform federated unlearning. It used the second-order AdaHessian optimizer [20] to speed up the retraining process. However, it was still retraining from scratch without any further optimization in the federated unlearning process. In this paper, we proposed an orthogonal mechanism such that an optimized unlearning process, custom-tailored for federated learning, can be carried out.

III. Random Clustered Aggregation: Design and Empirical Observations

The ultimate objective in federated unlearning is to erase the effects of one or more pre-specified data samples from a trained model. Through rounds of model propagation and redistribution, the influence of a client's local data samples will continuously affect the other clients that the server selects in the future rounds. As such effects ripple throughout the system, the only way to erase all the effects of any of the local data samples from client A is to *retrain from scratch*. However, the computation costs of retraining from scratch *on all affected clients* are prohibitively high during such a vanilla federated unlearning process.

Random clustered aggregation: an orthogonal approach to retraining algorithms. In the specific scenario of asynchronous FL, faster clients tend to go through more rounds of communication as their updates are aggregated and selected more frequently than slower clients. It has been clearly shown that when clients are heterogeneous in their training speeds, asynchronous FL outperforms synchronous FL by a substantial margin. Yet, the effects of local data samples from faster clients tend to propagate more quickly than slower clients as well, which makes federated unlearning more complex and challenging. To our best knowledge, potential mechanisms of mitigating the exorbitant costs of retraining from scratch in the context of asynchronous FL have not yet been explored in the literature.

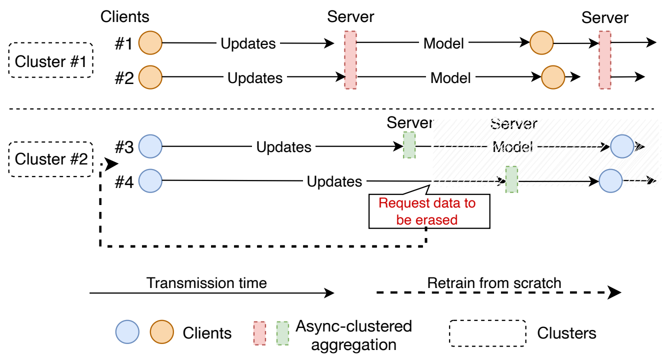


Fig. 3. An example of asynchronous clustered aggregation where four clients have been assigned to two clusters. If client #4 requests to erase the effects of its data from the server, only clients in cluster #2 need to be retrained, while clients in cluster #1 may proceed normally with their FL training process.

In contrast to existing work that sought to replace the naïve mechanism of retraining from scratch with approximation algorithms (*e.g.*, [2]), we propose to take a decidedly different approach that is *orthogonal* to approximation algorithms. Intuitively, if we can reduce the ripple effect of contributions from any particular client in the FL training process, the number of clients that must participate in the retraining process will be substantially reduced. With all existing works in the literature, as soon as a client's contributions have propagated throughout the entire pool of participating clients, all clients will need to participate in the retraining process when federated unlearning commences.

Instead, we propose to construct a "firewall" between *clusters* of clients in the system, such that server aggregation is carried out *within each cluster only*. With such a mechanism, which we refer to as *clustered aggregation*, the ripple effect of one client's contributions will only affect other clients in the same cluster, and clients outside the cluster will no longer need to be retrained. Fig. 3 shows how clients can be clustered and how the server aggregates the updates within each cluster only. It is worth noting that our new mechanism of clustered aggregation is orthogonal to any innovations in the retraining mechanism: it is complementary to both the naïve mechanism of retraining from scratch and all its alternatives using approximation algorithms.

The need for asynchrony. It turns out, however, that potential benefits from such a clustered aggregation mechanism come with a caveat: it only works effectively in the specific context of asynchronous FL, where client contributions are aggregated asynchronously without waiting for the slower clients. In the event that clients within one of the clusters need to roll back to a previous round and start their federated unlearning process, all the clients within the other clusters can proceed with their federated learning process normally. This phenomenon, where some clusters move forward in time while some other clusters roll back at the same time, is only feasible with asynchronous FL.

Criteria for terminating the training session. When is the right time for clustered aggregation to terminate, and for the server to begin aggregating across the clusters? In our random clustered aggregation mechanism, we consider two thresholds: one for the required validation accuracy, and one for the standard deviation across a recent history of such validation accuracies. As long as the highest validation accuracy across clusters is higher than the first threshold, and the standard deviation of recent accuracies is lower than the second, the training session will terminate, and the server will then perform its final round of aggregation to produce converged global model. Due to non-i.i.d. data distributions across the clients' local data, such an aggregation process is inevitably less efficient with respect to converging to the best possible accuracy as quickly as we can. It is therefore critically important to design a suitable clustering algorithm to minimize such a drag on training efficiency in a federated learning session, which we will soon elaborate in great detail in the next section.

Proof-of-concept experimental evaluations. Though suitable strategies for distributing clients across clusters are open to further investigation, we wish to first conduct some preliminary experiments to verify that clustered aggregation can indeed improve training performance in asynchronous FL, with some of the clients erasing their data. For the sake of simplicity, though random clustered aggregation can work with any retraining algorithm, we retrain from scratch when clients request erasing their data.

In our proof-of-concept experiments, we use the `MNIST` dataset to train the `LeNet-5` model, with 100 clients to be randomly assigned to 5 clusters. We select 30 clients in each communication round, and 15 clients as the minimum number of clients to be aggregated asynchronously. The data distribution of local datasets is non-i.i.d., sampled with the symmetric Dirichlet distribution with a concentration of 5. To simulate the heterogeneity across clients, we use a heavy-tailed Pareto distribution to simulate the asynchronous clients' training time, with its positive parameter $\alpha = 1$.

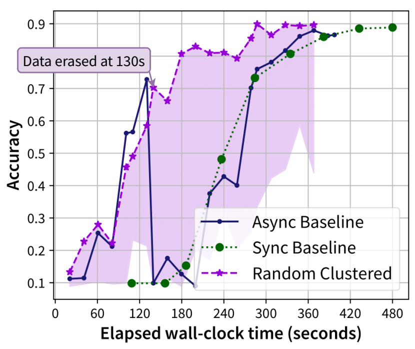


Fig. 4. A comparison study between synchronous FL baseline, asynchronous FL baseline, and random clustered aggregation using the `MNIST` [21] dataset, with full retraining from scratch after data erasure.

With 5 clusters, we proceed to compare the training performance of random clustered aggregation with two baseline algorithms: *FedBuff* [4] as our asynchronous FL baseline, and Federated Averaging (FedAvg) [8] as our synchronous baseline. Our results have been shown in Fig. 4, where we shaded the range of validation accuracies within each of the clusters. From these results, we confirmed the fact that synchronous FL with FedAvg was much slower than asynchronous FL with *FedBuff* before clients request to erase their data at the same wall-clock time (around 140 seconds). Once data erasure occurred, we observed a substantial reduction in global validation accuracy with the asynchronous FL baseline. This clearly showed the ripple effect of the erased data, in that all affected clients needed to be retrained from scratch.

In stark contrast to both synchronous and asynchronous baselines, only validation accuracies of some clusters experienced a substantial reduction with random clustered aggregation, while other clusters could be allowed to continue with their training process normally. As a result, global accuracies were not affected until convergence. After the criteria for terminating the training session is satisfied, the server produces the global model in the final round, which occurred at 360 seconds with a final accuracy of 90%, and outperformed both baselines. Further evaluations have shown that the number of clusters had no material effects on the training performance; therefore, we will continue to use 5 clusters in our later experiments in Section V.

IV. Knot: Algorithm and Analysis

Our preliminary experiments in Section III have clearly shown the effectiveness of our proposed clustered aggregation mechanism with respect to expediting the retraining process in the context of asynchronous FL. How clients are to be assigned to the clusters, however, remains to be determined. In our experiments so far, clients are assigned to the clusters randomly, but this is not the best approach in terms of maximizing the performance for two intuitive reasons. *First*, fast clients may be mixed with slower ones in the same cluster, making it less likely for training in a cluster to evolve quickly through more rounds with fast clients only. This is especially the case when a staleness bound is imposed in asynchronous FL, and fast clients need to wait for slow ones to reach their staleness bound. *Second*, random assignments may lead to similar non-i.i.d. data distributions within the cluster, which may suffer from the same challenges from such data heterogeneity as conventional FL without clustering.

In this section, we present KNOT, our optimized clustering aggregation mechanism for asynchronous federated unlearning. Towards the design of KNOT, we first wish to formulate an optimization problem to find a client-cluster assignment that offers better performance than assigning clients to clusters randomly. By solving such an optimization problem, clients will then be distributed more effectively and reasonably to each cluster, with the hope of contributing as much as possible towards minimizing the wall-clock time of asynchronous federated unlearning.

A. Optimizing Client-Cluster Assignment

First, we wish to discuss what types of clients should be clustered together. In asynchronous FL, clients are likely heterogeneous with respect to their local resources, leading to different completion times in a communication round. Naturally, if we assign faster clients to some clusters and slow clients to others, clusters with faster clients may converge faster over more communication rounds. This leads us to consider *local training times of the clients* as the first factor that affects client-cluster assignment.

We consider assigning clients $\{C_k\}_{k\in\mathcal{K}}$ to clusters $\{L_n\}_{n\in\mathcal{N}}$, where $\mathcal{K} = \{1, 2, \ldots, K\}$ and $\mathcal{N} = \{1, 2, \ldots, N\}$ are the corresponding indexing sets, and K and N correspond to the total number of clients and clusters, respectively. We define the training time T_k as the wall-clock time that elapses after the server sends the model and until it receives a parameter update from client C_k. A client with more computational resources may have a small T_k. In our optimization problem, we prefer to cluster the clients based on their training times.

The next factor that is likely to influence the convergence speed in a training session is *model disparity*. Due to non-i.i.d. data distributions, we use model disparity in our formulation to help assign clients with similar data in one cluster. The model disparity S_k measures how much client C_k's data diverges from the global model. We introduce cosine similarity $\Theta(u, v) = \frac{u\cdot v}{\|u\|\|v\|}$ which measures the angle between two vectors u and v. Starting with a random initial model with model parameters ω_0, we perform one training round on each client C_k to obtain its local update parameter Δ_k^1. These local updates are aggregated to form a global model with parameter ω_1. Intuitively, the vector $\omega_0 - \omega_1$ represents the consensus of the overall clients while Δ_k^1 reflects the influence of the individual client. Thus, a small angle between these two vectors implies that client C_k is a "good" client whose data looks congruent to the overall global model. Now, we can define the model disparity as $S_k = \frac{1-\Theta(\omega_0-\omega_1,\, \Delta_k^1)}{2}$, where we transform Θ such that $S_k \in [0, 1]\ \forall k \in \mathcal{K}$. Note that the smaller S_k is, the smaller the cosine similarity, and the more representative client C_k is concerning all clients.

In short, we aim to distribute the clients based on how "good" it is; namely, a good client should have a relatively short training time and a low model disparity. To achieve this goal, we set a *target training time* $\widetilde{T}_n$ and a *target model disparity* $\widetilde{S}_n$ for each cluster as anchor points and assign clients based on their T_k and S_k values. Suppose the training times of all clients span the range from S_* to S^*, we divide the ranges evenly by letting $\widetilde{T}_n = S_* + \frac{S^*-S_*}{N-1}\cdot(n-1)$ and $\widetilde{S}_n = \frac{n}{N}$. Next, to numerically represent the difference between client C_k and cluster L_n, we define the *match rating* as $d_{kn} = \|\ [a(\widetilde{T}_n - T_k), b(\widetilde{S}_n - S_k)]\ \|_2$, which is a weighted l^2 norm for the matrix scaled by hyper-parameters a and b.

The match rating allows us to tell whether a client is suitable for a cluster or not. For an arbitrary client C_k, $d_{k1} \le d_{k2}$ means that cluster L_1 is a better match for the client than cluster L_2 is, and we should assign it to L_1. At first glance, one may think that it suffices to choose the cluster that results in the lowest match rating for each client. However, our problem is more subtle because we need to ensure that each cluster has sufficient client data to train a decent cluster model that can contribute appropriately to the overall model after aggregation. Ideally, the resulting clusters should have sizes that are most beneficial for their effective convergence in the training session. For instance, suppose the difference in d_{k1} and d_{k2} is insignificant, and that L_1 is large while L_2 only has a few clients. In this case, it might be better in practice to assign the client to cluster L_2 so that when data erasure is requested, only a smaller fraction of the client needs to participate in retraining. For this reason, the match rating is not the only standard for our assignment. All decisions are interdependent, making our problem a more complicated one that involves more constraints.

B. Formulating the Optimization Problem

To begin formulating our problem, we represent a client-cluster assignment using a vector $x \in \{0, 1\}^{KN}$, $x = (x_{11}, \ldots, x_{KN})$, where $x_{kn} = 1$ if client C_k is assigned to cluster L_n and 0 otherwise. Then the vector $f = (d_{11}x_{11}, \ldots, d_{kn}x_{kn}, \ldots, d_{KN}x_{KN})$ provides a full description of our assignment, where the match rating of each pair is multiplied by the value which indicates whether the pair is assigned.

For our optimal clustering mechanism in KNOT, we aim to minimize the deviation of clients in the same cluster by keeping all d_{kn} values of the assigned client-cluster pairs to be as small as possible. Note that we do not seek to minimize the average match ratings of all assigned pairs. Instead, we emphasize reducing extreme values while keeping all coordinates as small as possible. More specifically, we want to minimize the largest value of vector f, then minimize the second largest value, and so on. This objective leads us to formulate our problem as a lexicographical minimization problem. We present the basic definitions in the following:

Definition 1: For $\alpha \in \mathbb{R}^n$, let $\langle\alpha\rangle = (\widetilde{\alpha}_1, \widetilde{\alpha}_2, \ldots, \widetilde{\alpha}_n)$ be α sorted in non-increasing order.

Definition 2: For $\alpha \in \mathbb{R}^n$, $\beta \in \mathbb{R}^n$, we say that α is *lexicographically smaller than* β, denoted as $\alpha \prec \beta$, if $\exists n_0 \in \{1, \ldots, n\}$ such that $\widetilde{\alpha}_{n_0} < \widetilde{\beta}_{n_0}$ and $n < n_0 \implies \widetilde{\alpha}_n = \widetilde{\beta}_n$. In addition, α is lexicographically no greater than β if $\alpha \prec \beta$ or two vectors have the same entries. That is, $\alpha \preceq \beta$ if $\alpha \prec \beta$ or $\widetilde{\alpha}_i = \widetilde{\beta}_i$, $\forall i \in \{1, 2, \ldots, n\}$.

Definition 3: Given a collection F of real vectors of the same length, we say that $\alpha_0 \in F$ is the lexicographical minimum of F if $\alpha_0 \preceq \alpha, \forall \alpha \in F$. For a function $f : \mathbb{R}^n \to \mathbb{R}^n$; $n \in \mathbb{R}$, $\underset{x}{\text{lexmin}}\ f = x_*$ if $f(x_*) \preceq f(x), \forall x$.

Intuitively, the process of finding $\underset{x}{\text{lexmin}}\ f$ accounts for looking for a vector x that minimizes the largest coordinate in the resulting vector $f(x)$, then minimizing the second largest value while keeping the largest coordinate unchanged. This procedure continues until the smallest value is minimal. Thus, the match rating of each client would be kept to the minimum as we desire.

We are now ready to formulate our lexicographical optimization problem.

$$\underset{x}{\text{lexmin}}\ f = (d_{11}x_{11}, \ldots, d_{kn}x_{kn}, \ldots, d_{KN}x_{KN}) \quad (1)$$

$$\text{s.t.} \sum_{n=1}^{N} x_{kn} \leq c_1, \forall k \in \mathcal{K} \quad (2)$$

$$\sum_{n=1}^{N} x_{kn} \geq 1,\ \forall k \in \mathcal{K} \quad (3)$$

$$\sum_{k=1}^{K} x_{kn} \geq c_2,\ \forall n \in \mathcal{N} \quad (4)$$

$$\sum_{k=1}^{K} x_{kn} \leq c_3,\ \forall n \in \mathcal{N} \quad (5)$$

$$x_{kn} \in \{0, 1\},\ \forall n \in \mathcal{N},\ \forall k \in \mathcal{K} \quad (6)$$

where the lexicographical minimum of function f corresponds to the desired client-cluster assignment.

Constraints (2)–(4) are non-trivial as they place more restrictions on the valid client-cluster assignments. For instance, consider the scenario where most clients have the smallest d_{kn} when $n = n^*$. In this case, most clients would be distributed to cluster L_{n^*}, which would result in one large cluster with other small clusters. It is difficult to judge if it is ideal or not because retraining on a large cluster would, to a certain extent, make KNOT lose its significance and appeal. We aim to make the training times for clients within a cluster as comparable as possible. However, if a cluster only has a small number of clients, over-fitting may cause the cluster model to be inadequate.

Another potential problem arises when the same client is assigned to multiple clusters in order to meet the lower bound on the number of clients per cluster. When this client requests erasing its own data, retraining is required for all the clusters it belongs to. To avoid this situation, we should also consider an upper bound on the number of clusters a client can be assigned.

For the above reasons, we need constraints (2)–(3) to make sure the number of clusters a client can belong to is between 1 and c_1; similarly, constraints (4)–(5) require the number of clients in each cluster to be between c_2 and c_3. In our forthcoming experiments in Section V, c_1, c_2, c_3 are set to 1, $K/2N$, and $K/2$, respectively.

C. Transforming into an LP Problem

Due to the complexity of solving the integer lexicographical optimization problem that we proposed, we need a method to transfer the original problem to a linear programming (LP) one, so that it can be more easily solved using off-the-shelf LP solvers. According to R. R. Meyer [6], a specific class of integer linear problem with a separable convex objective function and a totally unimodular (TU) constraint matrix can be solved as an LP problem. In this section, we will show that our problem belongs to the aforementioned class. Namely, we will transform the objective function to a separable convex one (Part A) and show that the corresponding matrix is TU (Part B).

Part A: Separable convex objective. A separable convex function is one that can be represented as a sum of multiple convex functions. To replace the original lexicographical objective function with a separable convex one, for any arbitrary vector $\alpha \in \mathbb{R}^n$, we consider the following function:

$$\phi(\alpha) = \sum_{m=1}^{n} n^{\alpha_m} \quad (7)$$

In particular, the exponential function is convex everywhere, and that the function $\phi(\alpha)$ preserves $\preceq$ when α is integer-valued, as stated in the following lemma [6]:

Lemma: $\forall \alpha,\ \beta \in \mathbb{R}^n.\ \alpha \preceq \beta \iff \phi(\alpha) \leq \phi(\beta)$.

We will transform each match rating d_{kn} in the objective function to an integer-valued D_{kn} by scaling and rounding. Let $d^* = \max_{k \in \mathcal{K} n \in \mathcal{N}} d_{kn}$ and $d_* = \min_{k \in \mathcal{K} n \in \mathcal{N}} d_{kn}$, and define $D_{kn} = \lceil \frac{d_{kn} - d_*}{d^* - d_*} \cdot 100 \rceil$. From our experiments, we observed that D_{kn} is sufficient to classify clients despite its reduced level of accuracy. In our problem, the length of vector $n = K \cdot N$ and $\alpha_m = d_{kn}x_{kn}$. Since $\alpha \preceq \beta$ is equivalent to $\phi(\alpha) \leq \phi(\beta)$, we can apply function (7) and simplify our original objective function to the following form:

$$\min_{x} \sum_{k \in \mathcal{K}} \sum_{n \in \mathcal{N}} (KN)^{D_{kn}x_{kn}}$$

s.t. (2), (3), (4), (5), (6).

Part B: Totally unimodular matrix. Before proving that our constraint matrix is totally unimodular (TU), let's explicitly formulate constraints (2), (3), (4), (5) in the form $Ax \leq b$.

Let M_{ij} refer to the entry on the i-th row and the j-th column of a matrix M. Let $A_1 \in \mathbb{Z}^{K \times KN}$ be the client indicator matrix and consider each column as representing a client-cluster pair. For each column j, $A_{1,ij} = 1$ indicates that client C_i is in the client-cluster pair j; otherwise, $A_{1,ij} = 0$. Similarly, $A_2 \in \mathbb{Z}^{N \times KN}$ is the cluster indicator matrix, where $A_{2,ij}$ indicates whether or not cluster L_i is in the client-cluster pair j. Now, we can rewrite our problem as follows:

$$\begin{aligned} \min_{x} \sum_{k \in \mathcal{K}} \sum_{n \in \mathcal{N}} & (KN)^{D_{kn} x_{kn}} \\ \text{s.t. } & Ax \le b \\ & 0 \le x \le 1, x \text{ integer} \end{aligned} \tag{8}$$

where

$$A = \begin{bmatrix} A_1 \\ -A_1 \\ A_2 \\ -A_2 \end{bmatrix}; b = \begin{bmatrix} c_1 \\ -1 \\ c_3 \\ -c_2 \end{bmatrix}$$

A matrix $M \in \mathbb{R}^{m \times n}$ is TU if every square sub-matrix M' of M has $\det(\mathrm{M}') \in \{-1, 0, 1\}$. This is an important property because it ensures that the inverse of any square sub-matrix is integral, thus all extreme points of the feasible region are integral. It is also known that a matrix M with all entries in $\{-1, 0, 1\}$ is TU if there exists a partition of every subset $R \subseteq \{1, 2, \ldots, m\}$ into $\mathcal{I}_1$ and $\mathcal{I}_2$ such that for each column j,

$$|\sum_{i \in \mathcal{I}_1} m_{ij} - \sum_{i \in \mathcal{I}_2} m_{ij}| \le 1 \tag{9}$$

Since matrix A contains only 0's and 1's, it suffices to provide a partition strategy for any subset of rows of A such that inequality (9) is satisfied.

Let $R \subset \{1, \ldots, 2K + 2N\}$ be given and we will partition R as follows. For each $k \in \{1, \ldots, K\}$, if $k \in R$, put k in $\mathcal{I}_1$ and $K + k$ in $\mathcal{I}_1$, if $K + k \in R$; otherwise, put $K + k$ in $\mathcal{I}_2$ if $K + k \in R$. This strategy ensures that $0 \le \sum_{\substack{i \in \mathcal{I}_1 \\ i \le 2K}} A_{ij} - \sum_{\substack{i \in \mathcal{I}_2 \\ i \le 2K}} A_{ij} \le 1$.

Next, for each $n \in \{1, \ldots, N\}$, if $2K + n \in R$, put $2K + n$ in $\mathcal{I}_2$, and if $2K + N + n \in R$, put $2K + N + n$ in $\mathcal{I}_2$; otherwise, put $2K + N + n$ in $\mathcal{I}_1$ if $2K + N + n \in R$. Similarly, we have $-1 \le \sum_{\substack{i \in \mathcal{I}_1 \\ i > 2K}} A_{ij} - \sum_{\substack{i \in \mathcal{I}_2 \\ i > 2K}} A_{ij} \le 0$. Adding up these two inequalities, we see that inequality (9) is satisfied. Thus, we have proved that matrix A is TU.

From an integer LP to an LP problem. We have shown that problem (8) has i) a separable convex objective function, ii) a TU constraint matrix, and iii) an integer matrix b. By the result in [6], the optimal solution for problem (8) is the same as the x-coordinates of the optimal solution of the following LP problem:

$$\begin{aligned} \min_{x, \lambda} \sum_{k \in \mathcal{K}} \sum_{n \in \mathcal{N}} & \lambda_{kn}^0 + (KN)^{D_{kn} x_{kn}} \lambda_{kn}^1 \\ \text{s.t. } & x_{kn} = \lambda_{kn}^1, \forall k \in \mathcal{K}, \forall n \in \mathcal{N} \\ & \lambda_{kn}^0 + \lambda_{kn}^1 = 1, \forall k \in \mathcal{K}, \forall n \in \mathcal{N} \\ & \lambda_{kn}^0, \lambda_{kn}^1, x_{kn} \in \mathbb{R}^+, \forall k \in \mathcal{K}, \forall n \in \mathcal{N} \\ & \text{Constraints (2), (3), (4) and (5).} \end{aligned}$$

where λ_{kn}^0, λ_{kn}^1 are newly introduced variable used in the λ-representation.

Thus, we have illustrated a method for efficiently solving our original integer lexicographical minimization problem.

V. Experimental Evaluation

A. Implementation and Preparation

We have implemented Knot in Plato, a new open-source research framework for asynchronous FL that we built from scratch. One of the key features in Plato has been the support for asynchronous FL, with the ability to measure the elapsed wall-clock time in a FL training session while achieving stellar *scalability* and *reproducibility*, so that state-of-the-art FL mechanisms can be fairly and accurately compared in the same real-world or emulated environment. In this work, we have made the following improvements to Plato.

Achieving stellar scalability. To scale up the number of clients with limited CPU and GPU (CUDA) memory, Plato launches a limited number of processes, which depends only on resource availability. For example, to train a `MNIST` model, one client uses around 1 GB of CUDA memory. With an NVIDIA A4500 GPU of 20 GB CUDA memory, approximately 20 clients can train at the same time. In this case, Plato will launch 20 processes only, and if more clients are selected in each round, they will run in consecutive batches. With a sufficient amount of time, Plato is scalable to an *unlimited* number of clients. Such scalability is the key to the success of completing our large-scale experiments later in this section.

Measuring wall-clock training times. Better scalability, however, does come with additional complexities when the wall-clock training times are measured in asynchronous FL. As one example, it may occur that faster clients in the next batch waiting to be launched on the GPU may finish even sooner than slower clients in the previous batch, if the clients were to be launched simultaneously. This can be handled correctly in our implementation by *simulating* the wall-clock time in each communication round, by asking all the clients who reported in real-time but are still considered training in simulated time to be kept in a priority queue, sorted by their finish times.

Improving reproducibility. In asynchronous FL experiments, random number generators are used for sampling participating clients and local datasets. For fair comparisons across different asynchronous FL mechanisms, we have carefully devised techniques to improve the reproducibility of our experiments by seeding, saving and restoring our random number generators, and by protecting random number generation from the effects of third-party frameworks. For further reproducibility, all our source code has already been made available as open-source to the research community.

Datasets and models. We have chosen four datasets, `CIFAR-10` [22], `Federated EMNIST` [23], `Purchase-100` [24] and `Tiny Shakespeare` [8] in our forthcoming experiments. They span a variety of tasks, including image classification (`CIFAR-10` and `Federated EMNIST`), behavioral pattern recognition (`Purchase-100`), and natural language processing

TABLE I
FOUR TRAINING TASKS WE TESTED: PARAMETER SETTINGS.

Parameter	CIFAR-10	FEMNIST	Purchase-100	Tiny-Shakespeare
K	100	250	100	70/50
# selected	20	200	60	50/30
# minimum	15	150	30	25
# erased	1/2	1/2	1/2	2/4
Samplers	non–i.i.d.	non–i.i.d.	non–i.i.d.	i.i.d.
Models	VGG-16	LeNet-5	MLP	GPT-2

(Tiny Shakespeare). It is worth noting that the Federated EMNIST (or FEMNIST) dataset is specifically designed for FL experiments, by pre-processing and partitioning the dataset into 3597 clients using metadata in the original EMNIST dataset. Important hyperparameters that we used in our experiments are listed in Table I, including the total number of clients, the number of clients selected per round, the minimum number of clients aggregated in asynchronous FL, and the number of clients requesting data erasures. Similar to Section III, we use a heavy-tailed Pareto distribution to simulate the asynchronous clients' training time, with its positive parameter $\alpha = 1$. We run our experiments on an in-house server with three NVIDIA RTX A4500 GPUs with 20 GB of CUDA memory each, 40 CPU cores, and a total of 256 GB memory.

B. KNOT: *Performance Evaluation*

Performance with benchmark datasets. To evaluate the performance of KNOT, we choose to compare it with (1) synchronous FedAvg as the synchronous baseline algorithm; (2) *FedBuff* [4] as the asynchronous baseline algorithm; (3) the *rapid retraining* algorithm proposed by Liu *et al.* [7], a state-of-the-art exact retraining algorithm (labeled as INFOCOM 2022 in our figures) without using approximations; and (4) random clustered aggregation without optimization. Though KNOT can work with any retraining algorithm, we evaluate it with the naïve algorithm of retraining from scratch so that its potential performance advantage is not attributed to the choice of the retraining algorithm.

We begin our experiments with three widely-used benchmark datasets: CIFAR-10, FEMNIST and Purchase-100. Our results over these datasets are shown in Figs. 5a to 5c, respectively. From these results, we can make a number of important observations. First, with client heterogeneity, *FedBuff* outperformed synchronous FedAvg by a very substantial margin. When training with the CIFAR-10 dataset, for example, to reach an accuracy of 75%, it took FedAvg 4426 seconds and *FedBuff* 2036 seconds only, reflecting an improvement of 54%. Second, random clustered aggregation — which serves as the foundation for KNOT — performed exceptionally well. With our three datasets, it outperformed *FedBuff* by 73%, 45%, and 61%, respectively.

Last but not least, thanks to optimized client-cluster assignments, KNOT enjoyed a minor performance advantage over random clustered aggregation, as it outperformed *FedBuff* by 85%, 52%, and 64%, respectively, over our datasets. In addition, we observed that it took KNOT a negligible overhead of less than a second to solve its optimization problem using Mosek 9, which is an off-the-shelf LP solver. This has clearly shown the benefits of transforming our lexicographical minimization problem to an equivalent linear programming problem, and solving the latter to optimality.

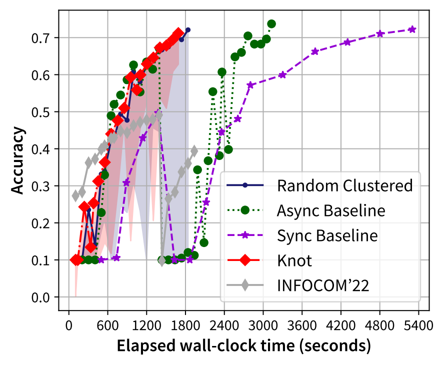
Real-world performance with the GPT-2 model. In order to evaluate the performance of KNOT when training a much larger model in a real-world language modeling task, we used a common dataset, Tiny Shakespeare, which has 40,000 lines of Shakespeare from a variety of Shakespeare's plays. We train a distilled variant of the celebrated GPT-2 model, which is a transformer from the HuggingFace framework [25]. Our results have been shown in Fig. 6a. Again, we observed that KNOT was able to outperform its competitors, achieving a 33% faster wall-clock training time compared to *FedBuff* and a small but performance advantage over random clustering, reaching a perplexity of 37.5 (lower perplexities are better in language modeling tasks).

Comparisons with FedEraser. FedEraser [2] represents one of the state-of-the-art retraining algorithms that used approximations, rather than naïve retraining from scratch. Though it requires less computation during the retraining process, it may not be considered to be in full compliance with regulatory policies such as GDPR. As an orthogonal approach, KNOT can work with FedEraser or any other approximation algorithm; it is nevertheless interesting to compare FedEraser's performance to KNOT with naïve retraining. For this reason, we conducted more experiments with exactly the same settings — including random seeds to ensure reproducibility — using FedEraser, and included our results in Table II. It can be observed that though FedEraser outperformed the *FedBuff* baseline by a small margin in FEMNIST and Purchase-100, it performed 2% *worse* (marked by ↓) in CIFAR-10, and failed to work correctly with the much larger GPT-2 model (which has not been evaluated in [2]). Overall, even with naïve retraining, KNOT outperformed FedEraser by **85%**, **47%**, and **56%**, respectively, in our benchmark datasets.

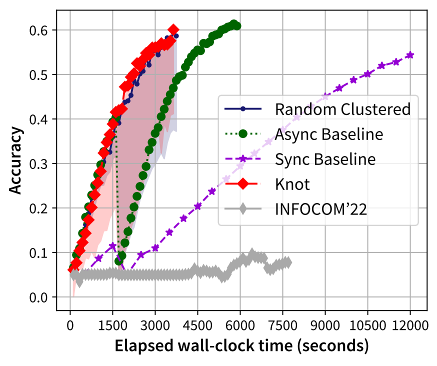
TABLE II
KNOT VS. ITS COMPETITORS: A COMPARISON REGARDING THE WALL-CLOCK TIMES NEEDED TO CONVERGE TO A TARGET ACCURACY (OR PERPLEXITY).

Mechanism	CIFAR-10	FEMNIST	Purchase-100	Tiny-Shakespeare
Accuracy (%)	75	60	63.5	37.5
FedBuff (secs)	2036	4272	2381	16293
FedEraser	2%↓	10%↑	19%↑	–
Random	73%↑	45%↑	61%↑	27%↑
KNOT	85%↑	52%↑	64%↑	33%↑

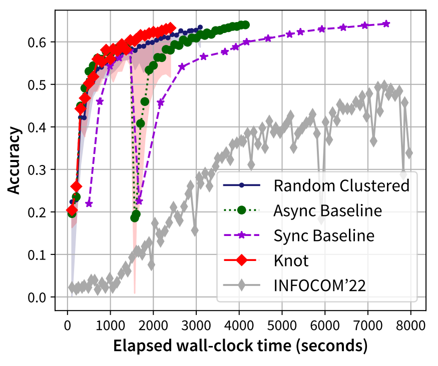
Rapid retraining. In contrast to FedEraser, the rapid retraining algorithm [7] (labeled as INFOCOM'22), is indeed an exact retraining algorithm. Unfortunately, we discovered that its performance was even worse than FedEraser across all of our datasets: it *failed to converge*, and could not be included in Table II as a result. In addition, with the Tiny Shakespeare



(a) CIFAR-10

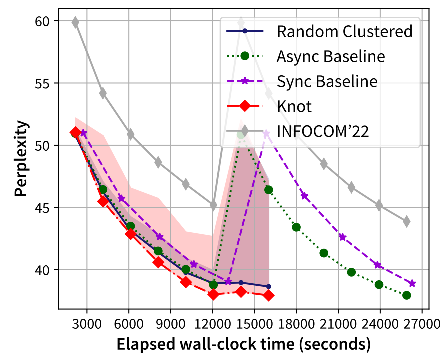


(b) FEMNIST

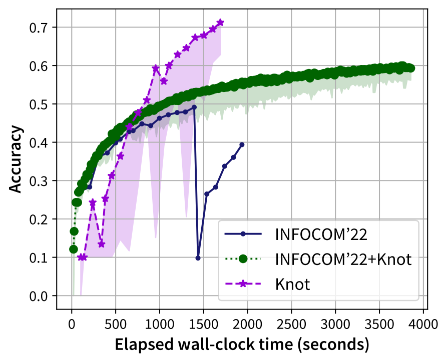


(c) Purchase-100

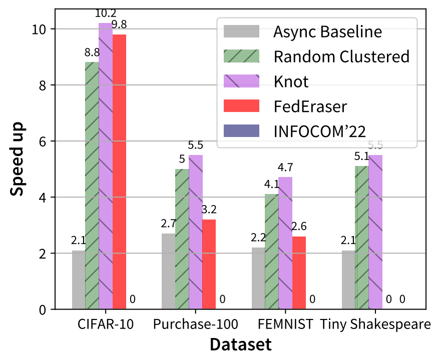
Fig. 5. KNOT vs. its leading competitors over three benchmark datasets.



(a) Tiny Shakespeare (converging to lower perplexity values)



(b) Combining KNOT with INFOCOM'22 on CIFAR-10.



(c) The unlearning efficiency when data from 2 clients (4 in Tiny Shakespeare) is to be erased.

Fig. 6. (a) KNOT vs. its leading competitors over a large-scale language modeling task using a distilled GPT-2 model and the Tiny Shakespeare dataset. (b) Using KNOT with the retraining algorithm in [7]. (c) Speedup comparisons over the synchronous baseline: KNOT vs. its competitors with two clients (four in the language modeling task) erased.

dataset, rapid retraining required a memory footprint that was approximately 200% higher than KNOT, while producing much worse performance as shown in Fig. 6a. Nevertheless, as it is an exact retraining algorithm, it would be interesting to combine KNOT with rapid retraining, and evaluate KNOT's performance with this new variant. Our results over CIFAR-10, plotted in Fig. 6b, showed that the use of KNOT substantially improved the convergence behavior of rapid retraining; yet it was still much worse than using KNOT with naïve retraining. It turned out that naïve retraining is not so "naïve" anyway! This experiment also showed that KNOT is indeed orthogonal to retraining algorithms, and can be combined with them easily.

Varying the number of clients requesting data erasures. In order to explore whether the number of clients requesting data erasures has an effect on KNOT's performance, we also designed experiments to erase the data on a varying number of clients under the same conditions. Fig. 6c shows our results — with respect to the speedup over the synchronous FedAvg baseline — with two clients requesting data erasure (rather than one in our previous experiments), and four clients when training GPT-2 with the Tiny Shakespeare dataset. It can be easily observed that KNOT and random clustered aggregation still performed substantially better than FedEraser, and rapid retraining failed to converge.

VI. CONCLUDING REMARKS

While existing works on federated unlearning focused on improving its performance with more efficient retraining algorithms, we sought to take a decidedly different and orthogonal approach in this paper with the introduction of clustered aggregation. The efficacy of our approach hinges upon the intuition that the retraining process can be constrained to within a cluster only, if server aggregation is only performed within each cluster, and training sessions in the other clusters can be carried out *asynchronously*. As an original highlight of this paper, we are the first to propose *asynchronous federated unlearning*, taking advantage of the well-recognized performance benefit of asynchronous FL. KNOT, our optimization-based clustered aggregation mechanism, pushes the performance envelope further by not only formulating the client-cluster assignment problem as a lexicographical minimization problem, but also proving that it can be solved efficiently using a linear program solver. With our scalable and reproducible implementation, we have shown that the wall-clock training time with KNOT is up to 85% better than FedEraser, a state-of-the-art on approximation algorithm, even when retraining from scratch. Last but not the least, our experiments were designed to be reproducible, and we have provided public access to our source code at https://github.com/TL-System/plato/tree/main/examples/knot.

References

[1] L. Bourtoule, V. Chandrasekaran, C. A. Choquette-Choo, H. Jia, A. Travers, B. Zhang, D. Lie, and N. Papernot, "Machine Unlearning," in *Proc. IEEE Symposium on Security and Privacy (SP)*, 2021, pp. 141–159.

[2] G. Liu, X. Ma, Y. Yang, C. Wang, and J. Liu, "FedEraser: Enabling Efficient Client-Level Data Removal from Federated Learning Models," in *Proc. IEEE/ACM 29th Int'l Symposium on Quality of Service (IWQoS)*, 2021, pp. 1–10.

[3] N. Su and B. Li, "How Asynchronous can Federated Learning Be?" in *Proc. IEEE/ACM 30th Int'l Symposium on Quality of Service (IWQoS)*, 2022.

[4] J. Nguyen, K. Malik, H. Zhan, A. Yousefpour, M. Rabbat, M. Malek, and D. Huba, "Federated Learning with Buffered Asynchronous Aggregation," in *Proc. Int'l Conference on Machine Learning (ICML)*, 2021.

[5] H. Wang, Z. Kaplan, D. Niu, and B. Li, "Optimizing Federated Learning on Non-IID data with Reinforcement Learning," in *Proc. IEEE Conference on Computer Communications (INFOCOM)*, 2020, pp. 1698–1707.

[6] R. R. Meyer, "A Class of Nonlinear Integer Programs Solvable by a Single Linear Program," *SIAM Journal on Control and Optimization*, vol. 15, no. 6, pp. 935–946, 1977.

[7] Y. Liu, L. Xu, X. Yuan, C. Wang, and B. Li, "The Right to be Forgotten in Federated Learning: An Efficient Realization with Rapid Retraining," in *Proc. IEEE Conference on Computer Communications (INFOCOM)*, 2022.

[8] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-Efficient Learning of Deep Networks from Decentralized Data," in *Proc. 20th Int'l Conference on Artificial Intelligence and Statistics (AISTATS)*, vol. 54, April 2017, pp. 1273–1282.

[9] C. Xie, S. Koyejo, and I. Gupta, "Asynchronous Federated Optimization," in *Proc. NeurIPS Workshop on Optimization for Machine Learning (OPT)*, 2020.

[10] Y. Cao and J. Yang, "Towards Making Systems Forget with Machine Unlearning," in *Proc. IEEE Symposium on Security and Privacy*, 2015, pp. 463–480.

[11] A. Golatkar, A. Achille, and S. Soatto, "Eternal Sunshine of the Spotless Net: Selective Forgetting in Deep Networks," in *Proc. IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020, pp. 9304–9312.

[12] ——, "Forgetting Outside the Box: Scrubbing Deep Networks of Information Accessible from Input-Output Observations," in *Proc. European Conference on Computer Vision (ECCV)*, 2020, pp. 383–398.

[13] Z. Izzo, M. A. Smart, K. Chaudhuri, and J. Zou, "Approximate Data Deletion from Machine Learning Models," in *Proc. 24th Int'l Conference on Artificial Intelligence and Statistics (AISTATS)*, 2021, pp. 2008–2016.

[14] S. Neel, A. Roth, and S. Sharifi-Malvajerdi, "Descent-to-Delete: Gradient-Based Methods for Machine Unlearning," in *Proc. 32nd Int'l Conference on Algorithmic Learning Theory*, 2021, pp. 931–962.

[15] C. Guo, T. Goldstein, A. Hannun, and L. Van Der Maaten, "Certified data removal from machine learning models," *arXiv preprint arXiv:1911.03030*, 2019.

[16] A. Ginart, M. Guan, G. Valiant, and J. Zou, "Making AI Forget You: Data Deletion in Machine Learning," *Advances in Neural Information Processing Systems (NeurIPS 2019)*, 2019.

[17] J. Brophy and D. Lowd, "Machine Unlearning for Random Forests," in *Proc. Int'l Conference on Machine Learning (ICML)*, 2021, pp. 1092–1104.

[18] C. Wu, S. Zhu, and P. Mitra, "Federated Unlearning with Knowledge Distillation," *arXiv preprint arXiv:2201.09441*, 2022.

[19] J. Wang, S. Guo, X. Xie, and H. Qi, "Federated Unlearning via Class-Discriminative Pruning," in *Proc. ACM Web Conference (WWW)*, 2022, pp. 622–632.

[20] Z. Yao, A. Gholami, S. Shen, M. Mustafa, K. Keutzer, and M. Mahoney, "ADAHESSIAN: An Adaptive Second Order Optimizer for Machine Learning," in *Proc. AAAI Conference on Artificial Intelligence (AAAI)*, vol. 35, no. 12, 2021, pp. 10 665–10 673.

[21] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-Based Learning applied to Document Recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[22] A. Krizhevsky, "Learning Multiple Layers of Features from Tiny Images," University of Toronto, Tech. Rep., 2009. [Online]. Available: https://www.cs.toronto.edu/~kriz/cifar.html

[23] S. Caldas, S. M. K. Duddu, P. Wu, T. Li, J. Konečný, H. B. McMahan, V. Smith, and A. Talwalkar, "LEAF: A Benchmark for Federated Settings," *arXiv preprint arXiv:1812.01097*, 2018.

[24] C. O. Sakar, S. O. Polat, M. Katircioglu, and Y. Kastro, "Real-Time Prediction of Online Shoppers' Purchasing Intention using Multilayer Perceptron and LSTM Recurrent Neural Networks," *Neural Computing and Applications*, vol. 31, no. 10, pp. 6893–6908, 2019.

[25] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. von Platen, C. Ma, Y. Jernite, J. Plu, C. Xu, T. L. Scao, S. Gugger, M. Drame, Q. Lhoest, and A. M. Rush, "Transformers: State-of-the-Art Natural Language Processing," in *Proc. 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, Oct. 2020, pp. 38–45.