LightSecAgg: Rethinking Secure Aggregation in Federated Learning

Abstract

Secure model aggregation is a key component of federated learning (FL) that aims at protecting the privacy of each user's individual model, while allowing their global aggregation. It can be applied to any aggregation-based approaches, including algorithms for training a global model (e.g., FedNova, FedProx, FedOpt), as well as personalized FL frameworks (e.g., pFedMe, Ditto, Per-FedAvg). Model aggregation needs to also be resilient to likely user dropouts in FL system, making its design substantially more complex. State-of-the-art secure aggregation protocols essentially rely on secret sharing of the random-seeds that are used for mask generations at the users, in order to enable the reconstruction and cancellation of those belonging to dropped users. The complexity of such approaches, however, grows substantially with the number of dropped users. We propose a new approach, named LightSecAgg, to overcome this bottleneck by turning the focus from "random-seed reconstruction of the dropped users" to "one-shot aggregate-mask reconstruction of the active users". More specifically, in LightSecAgg each user protects its local model by generating a single random mask. This mask is then encoded and shared to other users, in such a way that the aggregate-mask of any sufficiently large set of active users can be reconstructed directly at the server via encoded masks. We show that LightSecAgg achieves the same privacy and dropout-resiliency guarantees as the state-of-the-art protocols, while significantly reducing the overhead for resiliency to dropped users. Furthermore, our system optimization helps to hide the runtime cost of offline processing by parallelizing it with model training. We evaluate LightSecAgg via extensive experiments for training diverse models (logistic regression, shallow CNNs, MobileNetV3, and EfficientNet-B0) on various datasets (FEMNIST, CIFAR-100, GLD-23K) in a realistic FL system, and demonstrate that LightSecAgg significantly reduces the total training time, achieving a performance gain of up to $12.7\times$ over baselines.

1 Introduction

Federated learning (FL) has emerged as a promising approach to enable distributed training over a large number of users, while protecting the privacy of each user [16, 17]. The key idea of FL is to keep users' data on their devices and instead train local models at each user. The locally trained models would then be aggregated via a server to update a global model, which is then pushed back to users. Due to model inversion attacks (e.g., [9, 23, 27]), a critical consideration in FL design is to also ensure that the server does not learn the locally trained model of each user during model aggregation. Furthermore, model aggregation should be robust to likely user droupouts (due to poor connectivity, low battery, unavailability, etc) in FL systems. As such, there has been a series of works that aim at developing secure aggregation protocols for FL that protect privacy of each user's individual model, while allowing their global aggregation amidst possible user dropouts (e.g., [4, 19, 12]).

State-of-the-art secure aggregation protocols essentially rely on two main principles: (1) pairwise random-seed agreement between users in order to generate masks that hide users' models while having an additive structure that allows their cancellation when added at the server; (2) secret sharing of the random-seeds, in order to enable the reconstruction and cancellation of masks belonging to dropped users (see the illustration in Figure 1). The main drawback of such approaches is that the number of mask reconstructions at the server substantially grows as more users are dropped, causing

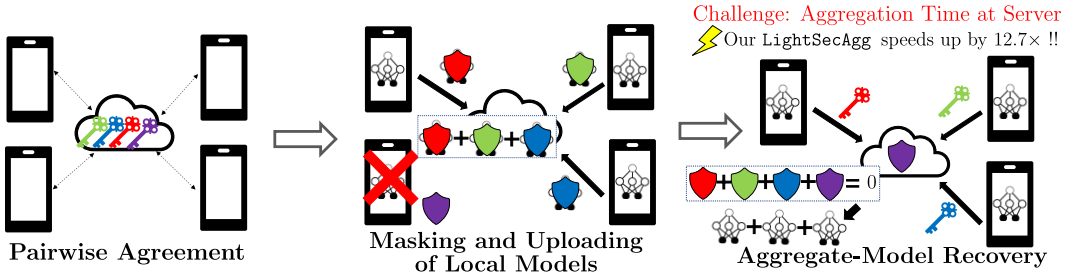


Figure 1: Illustration of secure aggregation in federated learning. (1) **Sharing key** ( ): users agree on pairwise seeds and secret share their keys. (2) **Masking model** ( ): each user masks its model by generating random masks, and uploads its masked model to the server. (3) **Unlocking masked models** ( ): The surviving users uploads the secretly shared keys to reconstruct the random masks. The server recovers the aggregate-model by canceling out the reconstructed aggregate-mask.

a major computational bottleneck. For instance, the execution time of the SecAgg protocol proposed in [4] is observed to be significantly limited by mask reconstructions at the server [3]. SecAgg+ [2], an improved version of SecAgg, reduces the overhead at the server by replacing the complete communication graph of SecAgg with a sparse random graph, such that secret sharing is only needed within a subset of users instead of for all users pairs. However, the number of mask reconstructions in SecAgg+, still increases as more users drop, eventually limiting the scalability of FL systems. There have also been several other approaches, such as [19, 12], to alleviate this bottleneck, however they either increase round/communication complexity, or compromise the dropout and privacy guarantees.

We propose a new perspective for secure model aggregation in FL, by turning the focus from "pairwise random-seed reconstruction of the dropped users" to "one-shot aggregate-mask reconstruction of the surviving users". Using this viewpoint, we propose a new protocol named LightSecAgg, that provides the same level of privacy and dropout-resiliency guarantees as the state-of-the-art while substantially reducing the aggregation (hence run-time) complexity. The main idea of LightSecAgg is that each user protects its local model using a locally generated random mask. This mask is then encoded and shared to other users, in such a way that the aggregate-mask of any sufficiently large set of surviving users can be directly reconstructed at the server. In sharp contrast to prior schemes, in this approach the server only needs to reconstruct *one* mask in the recovery phase, independent of the number of dropped users. We further propose system-level optimization methods to improve the run-time. In particular, we take advantage of the fact that the generation of random masks is independent of the computation of the local model, hence each user can parallelize these two operations via a multi-thread processing, which is beneficial to all evaluated secure aggregation protocols in reducing the total running time.

We run extensive experiments over Amazon EC2 cloud platform to empirically demonstrate the performance of LightSecAgg in a real-world FL setting with up to 200 users, and compare with two state-of-the-art protocols SecAgg and SecAgg+. To provide a comprehensive coverage of realistic FL settings, we train various machine learning models including logistic regression, convolutional neural network (CNN) [16], MobileNetV3 [11], and EfficientNet-B0 [21], for image classification over datasets of different image sizes: low resolution images (FEMNIST [5], CIFAR-100 [13]), and high resolution images (Google Landmarks Dataset 23k [24]). The empirical results show that LightSecAgg provides significant speedup for all considered FL training tasks in the running time over SecAgg and SecAgg+ and achieves a performance gain of up to $12.7\times$. The performance comparison implies that compared to baselines, LightSecAgg can even survive and speedup the training of large deep neural network models on high resolution image datasets. Breakdowns of speedup confirm that the primary gain lies in the complexity reduction at the server provided by LightSecAgg, especially for large number of users.

Related works. Beyond pairwise random masking technique used in [4, 2], there have been also other works towards making the secure aggregation more efficient (e.g., [12, 19, 26]). TurboAgg [19] utilizes a circular communication topology to reduce the communication and computation overhead but it incurs an additional round complexity. FastSecAgg [12] provides a secure aggregation with the overhead of $O(\log N)$ based on the Fast Fourier Transform multi-secret sharing. However, FastSecAgg provides lower privacy and dropout guarantees compared to the other state of the

arts. The idea of one-shot reconstruction of the aggregate-mask was also employed in a recent work [26], where the aggregated masks corresponding to each of the user dropout patterns was prepared by a trusted third party, encoded and distributed to users prior to model aggregation. The major advantages of `LightSecAgg` over the scheme in [26] are 1) not requiring a trusted third party; and 2) requiring much less amount of randomness generation and a much smaller storage cost at each user. Furthermore, there have also not been any system-level performance evaluations of [26] in FL experiments. We would finally like to emphasize that our proposed `LightSecAgg` protocol can be applied to any aggregation-based approaches (e.g., `FedNova` [22], `FedProx` [15], `FedOpt` [1]), as well as personalized FL frameworks (e.g., `pFedMe` [20], `Ditto` [14], `Per-FedAvg` [8]).

2 Problem Formulation

Federated learning (FL) is a distributed training framework for machine learning in mobile networks while preserving the privacy of users. The goal is to learn the parameter for the global model $\mathbf{x}$ with dimension d, using data held at mobile devices. This can be represented by minimizing a global objective function F: $F(\mathbf{x}) = \sum_{i=1}^{N} p_i F_i(\mathbf{x})$ where N is the total number of users, F_i is the local objective function of user i, and $p_i \geq 0$ is a weight parameter assigned to user i to specify the relative impact of each user such that $\sum_{i=1}^{N} p_i = 1$.[1]

Training is performed through an iterative process where mobile users interact through the central server to update the global model. At each iteration, the server shares the current state of the global model, denoted by $\mathbf{x}(t)$, with the mobile users. Each user i creates a local update, $\mathbf{x}_i(t)$. The local models of the N users are sent to the server and then aggregated by the server. Using the aggregated models, the server updates the global model $\mathbf{x}(t+1)$ for the next iteration. In FL, some users may potentially drop from the learning procedure due to unreliable communication connections. The goal of the server is to obtain the sum of the remaining surviving users' local models. This update equation is given by $\mathbf{x}(t+1) = \frac{1}{|\mathcal{U}(t)|} \sum_{i \in \mathcal{U}(t)} \mathbf{x}_i(t)$, where $\mathcal{U}(t)$ denotes the set of surviving users at iteration t. Then, the server pushes the updated global model $\mathbf{x}(t+1)$ to the mobile users.

As the local models carry extensive information about the local datasets stored at the users, e.g., the private data from the local models can be reconstructed by using a model inversion attack [9, 23, 27]. To address such privacy leakage from the local models, secure aggregation has been introduced in [4]. A secure aggregation protocol enables the computation of the aggregation operation while ensuring that the server learns no information about the local models $\mathbf{x}_i(t)$ beyond their aggregated model. In particular, our goal is to securely evaluate the aggregate of the local models $\mathbf{y} = \sum_{i \in \mathcal{U}} \mathbf{x}_i$, where we omit the iteration index t for simplicity. Since secure aggregation protocols build on cryptographic primitives that require all operations to be carried out over a finite field, we assume that the elements of $\mathbf{x}_i$ and $\mathbf{y}$ are from a finite field $\mathbb{F}_q$ for some field size q. We evaluate the performance of a secure aggregation protocol for FL through the following two key guarantees.

- **Privacy guarantee.** We consider a threat model where the users and the server are honest but curious. We assume that up to T users can collude with each other as well as with the server to infer the local models of other users. The secure aggregation protocol has to guarantee that nothing can be learned beyond the aggregate-model, even if up to T users cooperate with each other. We consider privacy in the information-theoretic sense. For every subset of users $\mathcal{T} \subseteq [N]$ of size at most T, we must have mutual information $I(\{\mathbf{x}_i\}_{i \in [N]}; \mathbf{Y} | \sum_{i \in \mathcal{U}} \mathbf{x}_i, \mathbf{Z}_{\mathcal{T}}) = 0$, where $\mathbf{Y}$ is the collection of information at the server, and $\mathbf{Z}_{\mathcal{T}}$ is the collection of information at the users in $\mathcal{T}$.
- **Dropout-resiliency guarantee.** In FL, users may get dropped or delayed at any time during protocol execution due to various reasons, e.g., poor wireless channel conditions, low battery, etc. We assume that there are at most D dropped users during the execution of protocol, i.e., there are at least $N - D$ surviving users after potential dropouts. The protocol has to guarantee that the server can correctly recover the aggregated models of the surviving users, even if up to D users drop.

We are interested in designing an efficient and scalable secure aggregation protocol that simultaneously achieves strong privacy and dropout-resiliency guarantees, scaling linearly with the number of users N, e.g., simultaneously achieves privacy guarantee $T = \frac{N}{2}$ and dropout-resiliency guarantee $D = \frac{N}{2} - 1$.

[1] For simplicity, we assume that all users have equal-sized datasets, i.e., $p_i = \frac{1}{N}$ for all $i \in [N]$.

3 Overview of Baseline Protocols SecAgg and SecAgg+

We first review the state-of-the-art secure aggregation protocols SecAgg [4] and SecAgg+ [2] as our main baselines. Essentially, SecAgg and SecAgg+ require each user to mask its local model using random keys before aggregation. In SecAgg, privacy of individual models is provided by pairwise random masking. Through a key agreement (e.g., Diffie-Hellman key agreement [7]), each pair of users $i, j \in [N]$ first agree on a pairwise random seed $a_{i,j} = \text{Key.Agree}(sk_i, pk_j) = \text{Key.Agree}(sk_j, pk_i)$ where sk_i and pk_i are the private and public keys of user i respectively. In addition, user i creates a private random seed b_i to prevent the privacy breaches that may occur if user i is only delayed instead of dropped, in which case the pairwise masks alone are not sufficient for privacy protection. User $i \in [N]$ then masks its model $\mathbf{x}_i$ by $\tilde{\mathbf{x}}_i = \mathbf{x}_i + \text{PRG}(b_i) + \sum_{j:i<j} \text{PRG}(a_{i,j}) - \sum_{j:i>j} \text{PRG}(a_{j,i})$, where PRG is a pseudo random generator, and sends it to the server. Finally, user i secret shares its private seed b_i as well as private key sk_i with the other users via Shamir's secret sharing [25]. From a subset of users who survived the previous stage, the server collects either the shares of the private key belonging to a dropped user, or the shares of the private seed belonging to a surviving user (but not both). Using the collected shares, the server reconstructs the private seed of each surviving user, and the pairwise seeds of each dropped user. The server then computes the aggregated models,

$$\sum_{i\in\mathcal{U}} \mathbf{x}_i = \sum_{i\in\mathcal{U}} (\tilde{\mathbf{x}}_i - \text{PRG}(b_i)) + \sum_{i\in\mathcal{D}} \left(\sum_{j:i<j} \text{PRG}(a_{i,j}) - \sum_{j:i>j} \text{PRG}(a_{j,i}) \right) \tag{1}$$

in which $\mathcal{U}$ and $\mathcal{D}$ represent the set of surviving and dropped users respectively. SecAgg protects model privacy against T colluding users, and is robust to D user dropouts as long as $N - D > T$.

We now illustrate SecAgg through a simple example. Consider a secure aggregation problem in FL, where there are $N = 3$ users with $T = 1$ privacy guarantee and dropout-resiliency guarantee $D = 1$. Each user $i \in \{1, 2, 3\}$, holds a local model $\mathbf{x}_i \in \mathbb{F}_q^d$. As shown in Figure 2, SecAgg is composed of the following three phases:

Offline pairwise agreement. User 1 and user 2 agree on pairwise random seed $a_{1,2}$. User 1 and user 3 agree on pairwise random seed $a_{1,3}$. User 2 and user 3 agree on pairwise random seed $a_{2,3}$. In addition, user $i \in \{1, 2, 3\}$ creates private random seed b_i. Then, user i secret shares b_i and its private key sk_i with the other users via Shamir's secret sharing. In this example, a 2 out of 3 secret sharing is used to tolerate 1 curious user.

Masking and uploading of local models. To provide the privacy of each individual model, user $i \in \{1, 2, 3\}$, masks its model $\mathbf{x}_i$ as follows:

$$\tilde{\mathbf{x}}_1 = \mathbf{x}_1 + \mathbf{n}_1 + \mathbf{z}_{1,2} + \mathbf{z}_{1,3}, \quad \tilde{\mathbf{x}}_2 = \mathbf{x}_2 + \mathbf{n}_2 + \mathbf{z}_{2,3} - \mathbf{z}_{1,2}, \quad \tilde{\mathbf{x}}_3 = \mathbf{x}_3 + \mathbf{n}_3 - \mathbf{z}_{1,3} - \mathbf{z}_{2,3}, \tag{2}$$

where $\mathbf{n}_i = \text{PRG}(b_i)$ and $\mathbf{z}_{i,j} = \text{PRG}(a_{i,j})$ are the random masks generated by a pseudo random generator. Then user $i \in \{1, 2, 3\}$, sends its masked local model $\tilde{\mathbf{x}}_i$ to the server.

Aggregate-model recovery. Suppose that user 1 drops in the previous phase. The goal of the server is to compute the aggregate of models $\mathbf{x}_2 + \mathbf{x}_3$. Note that

$$\mathbf{x}_2 + \mathbf{x}_3 = \tilde{\mathbf{x}}_2 + \tilde{\mathbf{x}}_3 + (\mathbf{z}_{1,2} + \mathbf{z}_{1,3} - \mathbf{n}_2 - \mathbf{n}_3). \tag{3}$$

Hence, the server needs to reconstruct masks $\mathbf{n}_2$, $\mathbf{n}_3$, $\mathbf{z}_{1,2}$, $\mathbf{z}_{1,3}$ to recover $\mathbf{x}_2 + \mathbf{x}_3$. To do so, the server has to collect two shares for each of b_2, b_3, sk_1, and then compute the aggregate model by (3). Since complexity of evaluating a PRG scales linearly with its size, the computation cost of the server for mask reconstruction is $4d$.

We note that SecAgg requires the server to compute a PRG function on *each* of reconstructed seeds to recover the aggregated masks, which incurs the overhead of $O(N^2)$ (see more details in Section 5) and dominates the overall execution time of the protocol [4, 3]. SecAgg+ reduces the overhead of mask reconstructions from $O(N^2)$ to $O(N \log N)$ by replacing the complete communication graph of SecAgg with a sparse random graph of degree $O(\log N)$ to reduce both communication and computation loads. Reconstructing pairwise random masks in SecAgg and SecAgg+ poses major bottlenecks in scaling to a large number of users. To overcome such computational bottleneck, we introduce the LightSecAgg protocol which enables the server to recover the aggregate-mask of all surviving users in one shot, while maintaining the same privacy and dropout-resiliency guarantees.

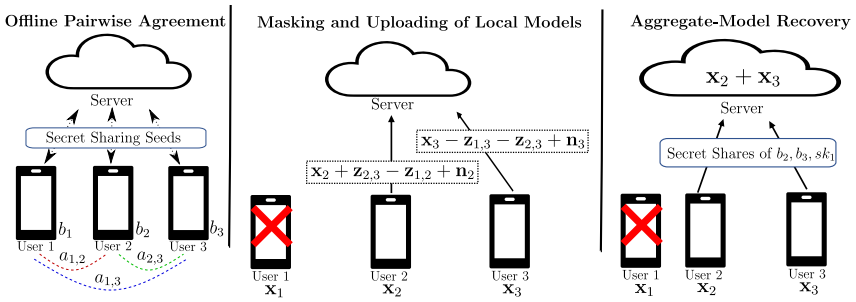


Figure 2: Illustration of SecAgg in the example of 3 users. Users first agree on pairwise random seeds, and secret share their private random seeds and private keys. The locals models are protected by the pairwise random masking. Suppose that user 1 drops. To recover the aggregate-mask, the server first reconstructs the private random seeds of surviving users and the private key of user 1 by collecting secret shares for each of them. Then, the server recovers $\mathbf{z}_{1,2}$, $\mathbf{z}_{1,3}$, $\mathbf{n}_2$ and $\mathbf{n}_3$, which incurs the computational cost of $4d$ at the server.

4 The Proposed LightSecAgg Protocol

Before providing a general description of LightSecAgg, we first illustrate its key ideas through the previous 3-user example. As shown in Figure 3, LightSecAgg has the following three phases:

Offline encoding and sharing of local masks. User $i \in \{1,2,3\}$ randomly picks $\mathbf{z}_i$ and $\mathbf{n}_i$ from $\mathbb{F}_q^d$. User $i \in \{1,2,3\}$ creates the masked version of $\mathbf{z}_i$ by encoding $\mathbf{z}_i$ and $\mathbf{n}_i$,

$$\tilde{\mathbf{z}}_{1,1} = -\mathbf{z}_1 + \mathbf{n}_1,\ \tilde{\mathbf{z}}_{1,2} = 2\mathbf{z}_1 + \mathbf{n}_1,\ \tilde{\mathbf{z}}_{1,3} = \mathbf{z}_1 + \mathbf{n}_1; \tag{4}$$

$$\tilde{\mathbf{z}}_{2,1} = -\mathbf{z}_2 + \mathbf{n}_2,\ \tilde{\mathbf{z}}_{2,2} = 2\mathbf{z}_2 + \mathbf{n}_2,\ \tilde{\mathbf{z}}_{2,3} = \mathbf{z}_2 + \mathbf{n}_2; \tag{5}$$

$$\tilde{\mathbf{z}}_{3,1} = -\mathbf{z}_3 + \mathbf{n}_3,\ \tilde{\mathbf{z}}_{3,2} = 2\mathbf{z}_3 + \mathbf{n}_3,\ \tilde{\mathbf{z}}_{3,3} = \mathbf{z}_3 + \mathbf{n}_3; \tag{6}$$

and user i sends $\tilde{\mathbf{z}}_{i,j}$ to each user $j \in \{1,2,3\}$. Thus, user i receives $\tilde{\mathbf{z}}_{j,i}$ for $j \in \{1,2,3\}$. In this case, this procedure provides robustness against 1 dropped user and privacy against 1 curious user.

Masking and uploading of local models. To make each individual model private, each user $i \in \{1,2,3\}$ masks its local model as follows:

$$\tilde{\mathbf{x}}_1 = \mathbf{x}_1 + \mathbf{z}_1, \quad \tilde{\mathbf{x}}_2 = \mathbf{x}_2 + \mathbf{z}_2, \quad \tilde{\mathbf{x}}_3 = \mathbf{x}_3 + \mathbf{z}_3, \tag{7}$$

and sends its masked model to the server.

One-shot aggregate-model recovery. Suppose that user 1 drops in the previous phase. To recover the aggregate of models $\mathbf{x}_2 + \mathbf{x}_3$, the server only needs to know the aggregated masks $\mathbf{z}_2 + \mathbf{z}_3$. To recover $\mathbf{z}_2 + \mathbf{z}_3$, the surviving user 2 and user 3 send $\tilde{\mathbf{z}}_{2,2} + \tilde{\mathbf{z}}_{3,2}$ and $\tilde{\mathbf{z}}_{2,3} + \tilde{\mathbf{z}}_{3,3}$,

$$\tilde{\mathbf{z}}_{2,2} + \tilde{\mathbf{z}}_{3,2} = 2(\mathbf{z}_2 + \mathbf{z}_3) + \mathbf{n}_2 + \mathbf{n}_3, \quad \tilde{\mathbf{z}}_{2,3} + \tilde{\mathbf{z}}_{3,3} = (\mathbf{z}_2 + \mathbf{z}_3) + \mathbf{n}_2 + \mathbf{n}_3, \tag{8}$$

to the server respectively. After receiving the messages from user 2 and user 3, the server can directly recover the aggregated masks via an one-shot computation as follows:

$$\mathbf{z}_2 + \mathbf{z}_3 = \tilde{\mathbf{z}}_{2,2} + \tilde{\mathbf{z}}_{3,2} - (\tilde{\mathbf{z}}_{2,3} + \tilde{\mathbf{z}}_{3,3}). \tag{9}$$

Then, the server recovers the aggregate-model $\mathbf{x}_2 + \mathbf{x}_3$ by subtracting $\mathbf{z}_2 + \mathbf{z}_3$ from $\tilde{\mathbf{x}}_2 + \tilde{\mathbf{x}}_3$. As opposed to SecAgg which has to reconstruct the dropped random seeds, LightSecAgg enables the server to reconstruct the desired aggregate of masks via a direct one-shot recovery. Compared with SecAgg, LightSecAgg reduces the server's computational cost from $4d$ to d in this simple example.

4.1 General Description of LightSecAgg

We formally present LightSecAgg, whose idea is to encode the local generated random masks in a way that the server can recover the aggregate of masks from the encoded masks via an one-shot computation with a cost that does not scale N. LightSecAgg has three design parameters: (1) $0 \leq T \leq N-1$ representing the privacy guarantee; (2) $0 \leq D \leq N-1$ representing the dropout-resiliency guarantee; (3) $1 \leq U \leq N$ representing the targeted number of surviving users. In particular, parameters T, D and U are selected such that $N - D \geq U > T \geq 0$.

LightSecAgg is composed of three main phases. First, each user first partitions its local random mask to $U - T$ pieces, and creates encoded masks via a Maximum Distance Separable (MDS) code

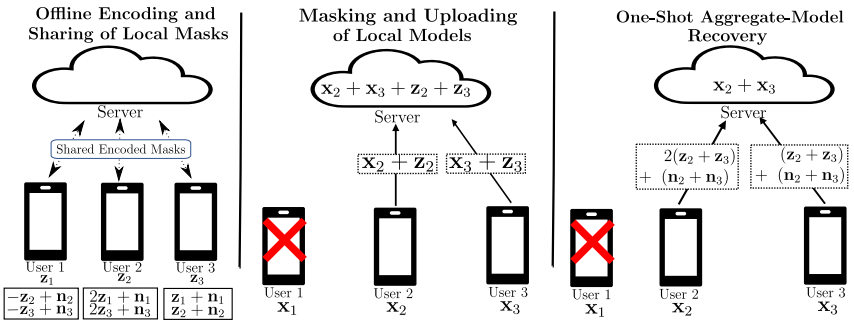


Figure 3: Illustration of LightSecAgg in the example of 3 users. Each user first generates a single mask. Each mask of a user is encoded and shared to other users. Each user's local model is protected by its generated mask. Suppose that user 1 drops during the execution of protocol. The server directly recovers the aggregate-mask in one shot. In this example, LightSecAgg reduces the computational cost at the server from $4d$ to d.

to provide robustness against D dropped users and privacy against T colluding users. Each user sends one of encoded masks to one of other users for the purpose of one-shot recovery. Second, each user uploads its masked local model to the server. Third, the server reconstructs the aggregated masks of the surviving users in order to recover their aggregate of models. Each surviving user sends the aggregated encoded masks to the server. After receiving U aggregated encoded masks from the surviving users, the server recovers the aggregate-mask and the desired aggregate-model. The pseudo code of LightSecAgg is provided in Appendix A. We now describe each of these phases in detail.

Offline encoding and sharing of local masks. User $i \in [N]$ picks $\mathbf{z}_i$ uniformly at random from $\mathbb{F}_q^d$ and mask $\mathbf{z}_i$ is partitioned to $U-T$ sub-masks $[\mathbf{z}_i]_k \in \mathbb{F}_q^{\frac{d}{U-T}}$, $k \in [U-T]$. With the randomly picked $[\mathbf{n}_i]_k \in \mathbb{F}_q^{\frac{d}{U-T}}$ for $k \in \{U-T+1, \ldots, U\}$, user $i \in [N]$ encodes sub-masks $[\mathbf{z}_i]_k$'s as

$$[\tilde{\mathbf{z}}_i]_j = ([\mathbf{z}_i]_1, \ldots, [\mathbf{z}_i]_{U-T}, [\mathbf{n}_i]_{U-T+1}, \ldots [\mathbf{n}_i]_U) \cdot W_j, \tag{10}$$

where W_j is j'th column of an MDS matrix (e.g., a Vandermonde matrix) $W \in \mathbb{F}_q^{U \times N}$ of dimension $U \times N$. Then, each user $i \in [N]$ sends $[\tilde{\mathbf{z}}_i]_j$ to user $j \in [N] \backslash \{i\}$. In the end of offline encoding and sharing of local masks, each user $i \in [N]$ has $[\tilde{\mathbf{z}}_j]_i$ from $j \in [N]$.

Masking and uploading of local models. To protect local models, each user i masks its local model as $\tilde{\mathbf{x}}_i = \mathbf{x}_i + \mathbf{z}_i$, and sends it to the server. Since some users may drop in this phase, the server identifies the set of surviving users, denoted by $\mathcal{U}_1 \subseteq [N]$. The server intends to recover $\sum_{i \in \mathcal{U}_1} \mathbf{x}_i$.

One-shot aggregate-model recovery. After identifying the surviving users in the previous phase, user $j \in \mathcal{U}_1$, is informed to send its aggregated encoded sub-masks $\sum_{i \in \mathcal{U}_1} [\tilde{\mathbf{z}}_i]_j$ to the server for the purpose of one-shot recovery. We note that each $\sum_{i \in \mathcal{U}_1} [\tilde{\mathbf{z}}_i]_j$ is an encoded version of $\sum_{i \in \mathcal{U}_1} [\mathbf{z}_i]_k$ for $k \in [U-T]$ using the MDS matrix W (see more details in Appendix B). Thus, the server is able to recover $\sum_{i \in \mathcal{U}_1} [\mathbf{z}_i]_k$ for $k \in [U-T]$ via MDS decoding after receiving a set of any U messages from the participating users, where this set is denoted by $\mathcal{U}_2$, $|\mathcal{U}_2| = U$. The server obtains the aggregated masks $\sum_{i \in \mathcal{U}_1} \mathbf{z}_i$ by concatenating $\sum_{i \in \mathcal{U}_1} [\mathbf{z}_i]_k$'s. Lastly, the server recovers the desired aggregate of models for the set of participating users $\mathcal{U}_1$ by subtracting $\sum_{i \in \mathcal{U}_1} \mathbf{z}_i$ from $\sum_{i \in \mathcal{U}_1} \tilde{\mathbf{x}}_i$.

5 Theoretical Analysis and Comparison with Prior Works

5.1 Theoretical Guarantees

We now state our main result for the theoretical guarantees of the LightSecAgg protocol.

Theorem 1. Consider a secure aggregation problem in federated learning with N users, the proposed LightSecAgg protocol can *simultaneously* achieve (1) privacy guarantee against up to any T colluding users, and (2) dropout-resiliency guarantee against up to any D dropped users, for any pair of privacy guarantee T and dropout-resiliency guarantee D such that $T + D < N$.

The proof of Theorem 1 is presented in Appendix B.

Remark 1. Theorem 1 provides a trade-off between privacy and dropout-resiliency guarantees, i.e., LightSecAgg can increase the privacy guarantee by reducing the dropout-resiliency guarantee and

vice versa. As SecAgg [4], LightSecAgg achieves the worst-case dropout-resiliency guarantee. That is, for any privacy guarantee T and the number of dropped users $D < N - T$, LightSecAgg ensures that any set of dropped users of size D in secure aggregation can be tolerated. Differently, SecAgg+ [2], FastSecAgg [12] and TurboAgg [19] relax the worst-case constraint to random dropouts, and provide probabilistic dropout-resiliency guarantee, i.e., the desired aggregate-model can be correctly recovered with high probability.

5.2 Complexity Analysis of LightSecAgg

We measure the storage cost, communication load, and computation load of LightSecAgg in unit of elements or operations in $\mathbb{F}_q$. Recall that U is a design parameter chosen such that $N - D \geq U > T$.

Offline storage cost. Each user i independently generates a random mask $\mathbf{z}_i$ of length d. Also, each user i stores a coded mask $[\tilde{\mathbf{z}}_j]_i$ of size $\frac{d}{U-T}$, for each $j = 1, \ldots, N$. Hence, the total offline storage cost at each user is $(1 + \frac{N}{U-T})d$.

Offline communication and computation loads. For each iteration of secure aggregation, before the local model is computed, each user prepares offline coded random masks and distributes them to the other users. Specifically, each user encodes U local data segments with each of size $\frac{d}{U-T}$ into N coded segments and distributes each of them to one of N users. Hence, the offline computation and communication load of LightSecAgg at each user is $O(\frac{dN \log N}{U-T})$ and $O(\frac{dN}{U-T})$ respectively.

Communication load during aggregation. While each user uploads a masked model of length d, in the phase of aggregate-model recovery, no matter how many users drop, each surviving user in $\mathcal{U}_1$ sends a coded mask of size $\frac{d}{U-T}$. The server is guaranteed to recover the aggregate-model of the surviving users in $\mathcal{U}_1$ after receiving messages from any U users. The total required communication load at the server in the phase of mask recovery is therefore $\frac{U}{U-T}d$.

Computation load during aggregation. For LightSecAgg, the major computation bottleneck is the decoding process to recover $\sum_{j \in \mathcal{U}_1} \mathbf{z}_j$ at the server. This involves decoding a dimension-U MDS code from U coded symbols, which can be performed with $O(U \log U)$ operations on elements in $\mathbb{F}_q$, hence a total computation load of $\frac{U \log U}{U-T}d$.

Table 1: Complexity comparison between SecAgg [4], SecAgg+ [2], and LightSecAgg. Here N is the total number of users, d is the model size, s is the length of the secret keys as the seeds for PRG ($s \ll d$).

	SecAgg	SecAgg+	LightSecAgg
Offline communication per user	$O(sN)$	$O(s \log N)$	$O(d)$
Offline computation per user	$O(dN + sN^2)$	$O(d \log N + s \log^2 N)$	$O(d \log N)$
Online communication per user	$O(d + sN)$	$O(d + s \log N)$	$O(d)$
Online communication at server	$O(dN + sN^2)$	$O(dN + sN \log N)$	$O(dN)$
Online computation per user	$O(d)$	$O(d)$	$O(d)$
Reconstruction complexity at server	$O(dN^2)$	$O(dN \log N)$	$O(d \log N)$

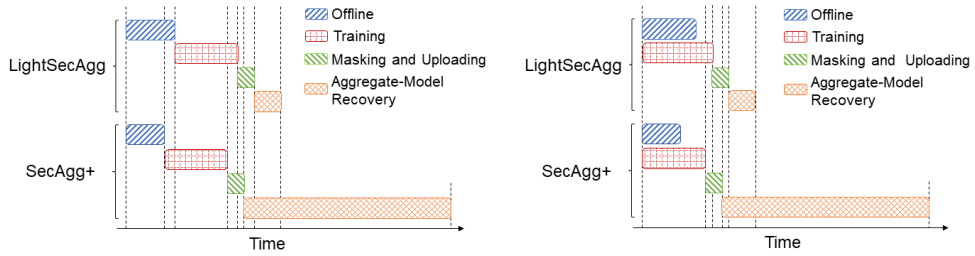
We compare the communication and computation complexities of LightSecAgg with baseline protocols. In particular, we consider the case where secure aggregation protocols aim at providing privacy guarantee $T = \frac{N}{2}$ and dropout-resiliency guarantee $D = pN$ simultaneously, for some $0 \leq p < \frac{1}{2}$. As shown in Table 1, by choosing $U = (1-p)N$, LightSecAgg significantly improves the computation efficiency at the server during aggregation. SecAgg and SecAgg+ incurs a total computation load of $O(dN^2)$ and $O(dN \log N)$ respectively at the server, while the server complexity of LightSecAgg almost stays constant with respect to N. The is expected to substantially reduce the overall aggregation time for a large number of users, which has been bottlenecked by the server's computation in SecAgg [4, 3]. More detailed discussions, as well as a comparison with another recently proposed secure aggregation protocol [26], which achieves similar server complexity as LightSecAgg, are carried out in Appendix C.

6 Experimental Analysis

We evaluate the performance of LightSecAgg and two baseline protocols, SecAgg and SecAgg+ in a realistic FL framework with up to $N = 200$ users to train various machine learning models.

Table 2: Summary of four implemented machine learning tasks and performance gain of `LightSecAgg` with respect to `SecAgg` [4] and `SecAgg+` [2]. All learning tasks are for image classification. FEMNIST and CIFAR-100 are low-resolution datasets, while images in GLD-23K are high resolution, which cost much longer training time for one mini-batch; LR and CNN are shallow models, but MobileNetV3 and EfficientNet-B0 are much larger models, but they are tailored for efficient edge training and inference.

No.	Dataset	Model	Model Size (d)	Gain	
				Non-overlapped	Overlapped
1	FEMNIST [5]	Logistic Regression	7,850	6.7×, 2.5×	8.0×, 2.9×
2	FEMNIST [5]	CNN [16]	1,206,590	11.3×, 3.7×	12.7×, 4.1×
3	CIFAR-100 [13]	MobileNetV3 [11]	3,111,462	7.6×, 2.8×	9.5×, 3.3×
4	GLD-23K [24]	EfficientNet-B0 [21]	5,288,548	3.3×, 1.6×	3.4×, 1.7×



(a) Non-overlapped (b) Overlapped

Figure 4: Timing diagram of `LightSecAgg` and `SecAgg+` [2] for a single iteration to train MobileNetV3 [11] with CIFAR-100 dataset [13]. `SecAgg` [4] is not included as it takes much longer than other two protocols.

6.1 Setup and System-level Optimization

Setup. We implement `LightSecAgg` on a distributed platform to train various machine learning models for image classification tasks, and examine its total running time of one global iteration with respect to the two baseline protocols. To model a FL framework, we use `m3.medium` instances over Amazon EC2 cloud, and implement communication using the `MPI4Py` [6] message passing interface on `Python`. To provide a comprehensive coverage of realistic FL settings, we train four machine learning models over datasets of different sizes, which is summarized in Table 2. The hyper-parameter settings are provided in Appendix D.

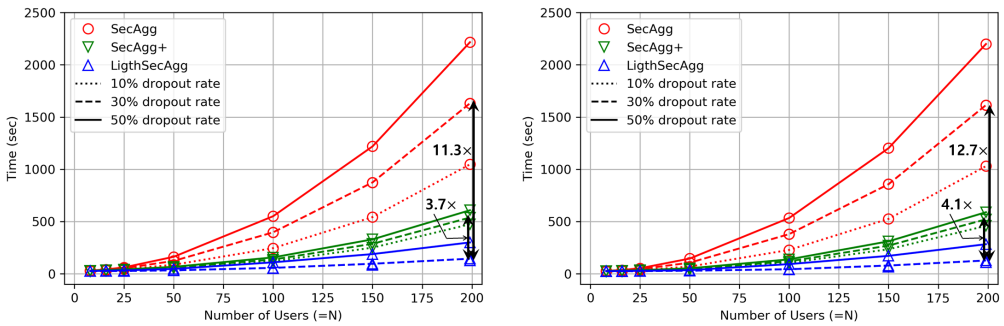
Dropout and Privacy Modeling. To model the dropped users, we randomly select pN users where p is the dropout rate. We consider the worst-case scenario in [4] where the selected pN users artificially drop after uploading the masked model. All three protocols provide privacy guarantee $T = \frac{N}{2}$ and resiliency for three different dropout rates, $p = 0.1$, $p = 0.3$, and $p = 0.5$.

System-level Optimization. We implement three protocols, `LightSecAgg`, `SecAgg`, and `SecAgg+`, and apply the following system-level optimizations to speed up their executions.

- **Parallelization of offline phase and model training.** In all three protocols, communication and computation time to generate and exchange the random masks in the offline phase can be *overlapped* with model training. That is, each user locally trains the model and carries out the offline phase simultaneously by running two parallel threads. Figure 4 demonstrates the timing diagram of two different implementations, referred to *non-overlapped* and *overlapped* implementations. While detailed analysis will be provided later, total running time is reduced in the overlapped case.
- **Amortized communication.** `LightSecAgg` can further speed up the communication time in the offline phase by parallelizing the transmission and reception of $[\tilde{\mathbf{z}}_i]_j$ via multi-threading. `SecAgg` and `SecAgg+` can also speed up the offline pairwise agreement by this parallelization.

6.2 Performance Evaluation

For the performance analysis, we measure the total running time for a single round of global iteration which includes model training and secure aggregation with each protocol while increasing the number of users N gradually for different user dropouts. Our results to train CNN [16] on the FEMNIST dataset [5] are demonstrated in Figure 5. Performance gain of `LightSecAgg` with respect to `SecAgg`



(a) Non-overlapped (b) Overlapped

Figure 5: Total running time of LightSecAgg versus the state-of-the-art protocols (SecAgg [4] and SecAgg+ [2]) to train CNN [16] on the FEMNIST dataset [5], as the number of users increases, for various dropout rates.

and SecAgg+ to train the other models is also provided in Table 2. More detailed experimental results are provided in Appendix D. We make the following key observations.

- Total running time of SecAgg and SecAgg+ increases monotonically with the dropout rate. This is because their total running time is dominated by the mask recovery at the server, which increases quadratically with the number of users.
- In the non-overlapped implementation, LightSecAgg provides a speedup up to $11.3\times$ and $3.7\times$ over SecAgg and SecAgg+, respectively, by significantly reducing the server's execution time.
- In the overlapped implementation, LightSecAgg provides a further speedup of up to $12.7\times$ and $4.1\times$ over SecAgg and SecAgg+, respectively. This is due to the fact that LightSecAgg requires more communication and computation cost in the offline phase than the baseline protocols and the overlapped implementation helps to mitigate this extra cost.
- LightSecAgg provides the smallest speedup in the most training-intensive task, training EfficientNet-B0 on GLD-23K dataset. This is due to the fact that training time is dominant in this task, and training takes almost the same time in LightSecAgg and baseline protocols. In particular, LightSecAgg provides significant speedup of the aggregate-model recovery phase at the server over the baseline protocols in all considered tasks.
- LightSecAgg incurs the smallest running time for the case of $p = 0.3$, which is almost identical to the case of $p = 0.1$. Recall that LightSecAgg can select design parameter U between $T = 0.5N$ and $N - D = (1 - p)N$. Within this range, while increasing U reduces the size of the symbol to be decoded, it also increases the complexity of decoding each symbol. The experimental results suggest that the optimal choices for the cases of $p = 0.1$ and $p = 0.3$ are both $U = \lfloor 0.7N \rfloor$, which leads to a faster execution than the case of $p = 0.5$ where U can only be chosen as $U = 0.5N + 1$.

7 Conclusion

This paper proposed LightSecAgg, a new approach for secure aggregation in federated learning. Compared with state-of-the-art, LightSecAgg reduces the overhead of model aggregation in FL by leveraging one-shot aggregate-mask reconstruction of the active users, while providing the same privacy and dropout-resiliency guarantees. In a realistic FL framework, extensive empirical results show that LightSecAgg can provide substantial speedup over baseline protocols for training diverse machine learning models with a performance gain up to $12.7\times$.

Broader Impact

Our protocol has the societal benefit of protecting user privacy in FL, where hundreds of users can jointly train a global machine learning model without revealing information about their individual datasets. One limitation is potential vulnerability to active/byzantine adversaries via model/data poisoning attacks. LightSecAgg can be combined with state-of-the-art byzantine resilient aggregation protocols in FL (e.g., [10, 18]) to address this limitation.

References

[1] Muhammad Asad, Ahmed Moustafa, and Takayuki Ito. Fedopt: Towards communication efficiency and privacy preservation in federated learning. *Applied Sciences*, 10(8):2864, 2020.

[2] James Henry Bell, Kallista A Bonawitz, Adrià Gascón, Tancrède Lepoint, and Mariana Raykova. Secure single-server aggregation with (poly) logarithmic overhead. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pages 1253–1269, 2020.

[3] Keith Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloé Kiddon, Jakub Konečný, Stefano Mazzocchi, Brendan McMahan, Timon Van Overveldt, David Petrou, Daniel Ramage, and Jason Roselander. Towards federated learning at scale: System design. In *Proceedings of Machine Learning and Systems*, volume 1, pages 374–388, 2019.

[4] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. Practical secure aggregation for privacy-preserving machine learning. In *proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1175–1191, 2017.

[5] Sebastian Caldas, Sai Meher Karthik Duddu, Peter Wu, Tian Li, Jakub Konečnỳ, H Brendan McMahan, Virginia Smith, and Ameet Talwalkar. Leaf: A benchmark for federated settings. *arXiv preprint arXiv:1812.01097*, 2018.

[6] Lisandro Dalcín, Rodrigo Paz, and Mario Storti. MPI for Python. *Journal of Parallel and Dist. Comp.*, 65(9):1108–1115, 2005.

[7] Whitfield Diffie and Martin Hellman. New directions in cryptography. *IEEE transactions on Information Theory*, 22(6):644–654, 1976.

[8] Alireza Fallah, Aryan Mokhtari, and Asuman Ozdaglar. Personalized federated learning with theoretical guarantees: A model-agnostic meta-learning approach. *Advances in Neural Information Processing Systems*, 33, 2020.

[9] Jonas Geiping, Hartmut Bauermeister, Hannah Dröge, and Michael Moeller. Inverting gradients - how easy is it to break privacy in federated learning? In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 16937–16947. Curran Associates, Inc., 2020.

[10] Lie He, Sai Praneeth Karimireddy, and Martin Jaggi. Secure byzantine-robust machine learning. *arXiv preprint arXiv:2006.04747*, 2020.

[11] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, et al. Searching for mobilenetv3. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1314–1324, 2019.

[12] Swanand Kadhe, Nived Rajaraman, O Ozan Koyluoglu, and Kannan Ramchandran. Fastsecagg: Scalable secure aggregation for privacy-preserving federated learning. *arXiv preprint arXiv:2009.11248*, 2020.

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

[14] Tian Li, Shengyuan Hu, Ahmad Beirami, and Virginia Smith. Ditto: Fair and robust federated learning through personalization. *arXiv: 2012.04221*, 2020.

[15] Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. Federated optimization in heterogeneous networks. *arXiv preprint arXiv:1812.06127*, 2018.

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

[17] H Brendan McMahan et al. Advances and open problems in federated learning. *Foundations and Trends® in Machine Learning*, 14(1), 2021.

[18] Jinhyun So, Başak Güler, and A Salman Avestimehr. Byzantine-resilient secure federated learning. *IEEE Journal on Selected Areas in Communications*, 2020.

[19] Jinhyun So, Başak Güler, and A Salman Avestimehr. Turbo-aggregate: Breaking the quadratic aggregation barrier in secure federated learning. *IEEE Journal on Selected Areas in Information Theory*, 2(1):479–489, 2021.

[20] Canh T. Dinh, Nguyen Tran, and Josh Nguyen. Personalized federated learning with moreau envelopes. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 21394–21405. Curran Associates, Inc., 2020.

[21] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International Conference on Machine Learning*, pages 6105–6114. PMLR, 2019.

[22] Jianyu Wang, Qinghua Liu, Hao Liang, Gauri Joshi, and H Vincent Poor. Tackling the objective inconsistency problem in heterogeneous federated optimization. *arXiv preprint arXiv:2007.07481*, 2020.

[23] Zhibo Wang, Mengkai Song, Zhifei Zhang, Yang Song, Qian Wang, and Hairong Qi. Beyond inferring class representatives: User-level privacy leakage from federated learning. In *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*, pages 2512–2520. IEEE, 2019.

[24] Tobias Weyand, Andre Araujo, Bingyi Cao, and Jack Sim. Google landmarks dataset v2-a large-scale benchmark for instance-level recognition and retrieval. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2575–2584, 2020.

[25] Andrew C Yao. Protocols for secure computations. In *23rd annual symposium on foundations of computer science (sfcs 1982)*, pages 160–164. IEEE, 1982.

[26] Yizhou Zhao and Hua Sun. Information theoretic secure aggregation with user dropouts. *arXiv preprint arXiv:2101.07750*, 2021.

[27] Ligeng Zhu and Song Han. Deep leakage from gradients. In *Federated Learning*, pages 17–31. Springer, 2020.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes]
   (c) Did you discuss any potential negative societal impacts of your work? [Yes]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes]
   (b) Did you include complete proofs of all theoretical results? [Yes]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes]
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes]
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [N/A]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes]
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [N/A]
   (c) Did you include any new assets either in the supplemental material or as a URL? [N/A]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]