GENERATIVE MODELS FOR EFFECTIVE ML ON PRIVATE, DECENTRALIZED DATASETS

Anonymous authors
Paper under double-blind review

ABSTRACT

To improve real-world applications of machine learning, experienced modelers develop intuition about their datasets, their models, and how the two interact. Manual inspection of raw data—of representative samples, of outliers, of misclassifications—is an essential tool in a) identifying and fixing problems in the data, b) generating new modeling hypotheses, and c) assigning or refining human-provided labels. However, manual data inspection is risky for privacy-sensitive datasets, such as those representing the behavior of real-world individuals. Furthermore, manual data inspection is impossible in the increasingly important setting of federated learning, where raw examples are stored at the edge and the modeler may only access aggregated outputs such as metrics or model parameters. This paper demonstrates that generative models—trained using federated methods and with formal differential privacy guarantees—can be used effectively to debug data issues even when the data cannot be directly inspected. We explore these methods in applications to text with differentially private federated RNNs and to images using a novel algorithm for differentially private federated GANs.

1 INTRODUCTION

Real-world systems increasingly depend on machine learning to make decisions, detect anomalies, and power products. Applications ranging from fraud detection to mobile phone keyboards use models trained on data that may be privacy sensitive. The data may contain, for example, financial, medical, or behavioral information about individuals. Institutions responsible for these applications of machine learning must balance data stewardship obligations—including minimizing the risks of data loss, theft, or abuse—with the practical needs of the "modeler" whose job is to develop and improve the machine learned models.

Modelers working with privacy-sensitive data face significant challenges in model development and debugging. Often, a modeler's first step would be to inspect individual examples in order to discover bugs, generate hypotheses, improve labeling, or alike. But direct inspection of data may be audited or disallowed by policy in some settings. In other settings—including federated learning (FL) (McMahan & Ramage, 2017; McMahan et al., 2017), which is the motivation and focus of this work—the data *cannot* be inspected. In federated learning, raw data remains distributed across a fleet of devices, such as mobile phones, while an orchestrating server coordinates training of a shared global model. Only the final model parameters and statistics are gathered and made available to the modeler. [1]

How can a modeler effectively debug when training data is privacy sensitive or decentralized? This paper explores how a novel class of auxiliary models—privacy-preserving federated generative models—can stand in for direct data examination during the process of debugging data errors during inference or training. By

[1] Despite these limitations, FL systems (Bonawitz et al., 2019) have proven effective for training and deploying real-world models, for example see Hard et al. (2018); Yang et al. (2018); Chen et al. (2019).

combining ideas from deep generative models, federated learning, and user-level differential privacy (DP), we show how some needs traditionally met with data inspection can instead be met by generating synthetic examples from a privacy-preserving federated generative model. These examples could be representative of all or a subset of the non-inspectable data, while at the same time preserving the privacy of individuals. Our contributions include:

- Identifying key challenges in implementing end-to-end workflows with non-inspectable data, e.g., for debugging a primary ML model used in a mobile application.
- Describing how auxiliary generative models can resolve these challenges, subject to the representative power of the generative models.
- Demonstrating how privacy preserving federated generative models—RNNs for text and GANs for images—can be trained to high enough fidelity to discover introduced data errors matching those encountered in real world scenarios. We also introduce a novel adaption of generative adversarial networks (GANs) to the federated setting with user-level DP guarantees.

2 Challenges of ML on non-inspectable data

A key aspect of deep learning is the utilization of large datasets. The behavior of a deep neural network is tightly coupled to the data it trains on. As Sculley et al. (2015) puts it, "ML is required in exactly those cases when the desired behavior cannot be effectively expressed in software logic without dependency on external data." The centrality of data to ML performance is recognized in the attention paid to data analysis, curation, and debugging. Textbooks devote chapters to practical methodologies for "Determining Whether to Gather More Data" (Goodfellow et al., 2016). The literature conveys practical lessons learned on ML system anti-patterns that increase the chance of data processing bugs (Sculley et al., 2015).

In general, the assumption of unfettered access to training or inference data is nearly universal (Riley, 2016; Zinkevich, 2017; Chakarov et al., 2016). But what to do when direct data access is precluded, e.g., when working with private and decentralized data, is largely an unaddressed question. In this section, we consider common scenarios where a modeler would typically use direct data access, allowing us to then develop alternative approaches in subsequent sections.

Model Improvement: Debugging and Debiasing When designing or improving end-to-end ML pipelines, the modeler derives considerable utility from their ability to examine individual examples of input data (Riley, 2016). For example, a ubiquitous first step in debugging a malfunctioning model is to randomly choose an input sample and observe its properties. Are the size, data types, and value ranges as expected? If text, are words or word pieces being properly separated and tokenized? A modeler is often working to identify mistakes in upstream processing, affecting all or most data, and readily apparent by looking at one or a few input samples. A modeler may also wish to inspect a sample from only a subset of the input data. Continuing the debugging example, perhaps a classifier model is exhibiting poor prediction behavior for a subset of classes, and the modeler would like to examine only those samples of input that induce misclassification.

Moving beyond debugging (where the modeler is aided by inspecting just one or a few examples), there are also scenarios where it is necessary to form an understanding of an overall data distribution via collection of examples. How well a model trained on proxy data will perform at inference on 'real' data is directly tied to how well the proxy dataset embodies the distribution of the real data. A host of problems arise in ML when a model is deployed for prediction on a data distribution that differs statistically from the distribution at training time.

For example, classes may be prevalent in real data observed at prediction time that were underrepresented in the training set (e.g., racial/ethnic/gender groups missing when training a face detector). These classes can be unrelated to the model's task but serve as a confounding factor in prediction if the model has erroneously fit an assumption about them during training (e.g., a smiling face detector that judges smiling vs. non-smiling based on gender or gender expression (Denton et al., 2019)). When entire groups are un- or underrepresented in training, the result can be biased predictions that induce quite insidious social effects (Angwin et al., 2016).

Consequently, the modeler requires the ability to inspect examples of classes missing at training but present at prediction time. This informs additional data collection to augment training data with examples representative of the underrepresented classes, thereby reducing bias. A related topic is ML interpretability. In recent work of Kim et al. (2018), data examples are a lingua franca by which higher level concepts are conveyed.

The interactions described above would be trivial were the prediction-time data inspectable. The challenge for the modeler arises when needing to perform these data interactions when the data is non-inspectable (private and decentralized), e.g., for models deployed to personal mobile devices. A need exists in the privacy-preserving ML space for improving models without accessing training or inference examples.

Data Labeling Many supervised learning problems are naturally 'self-labeling'; that is, they have the property that labels can be acquired at the same time that inputs are acquired. An example is predicting the next word a user will type in a keyboard app (e.g., as in Hard et al. (2018)): the input is a sequence of words and the label is the subsequent word, and both are acquired directly from a user's natural interaction with their keyboard. Other examples include Yang et al. (2018) and Chen et al. (2019). While self-labeling problems abound, there exist other problems where the label is not so easily acquired. Instead, developers must gather the collected inputs intended for training and pass them to separate human raters to apply labels. Only then does a set of input/label pairs exist to allow for supervised training. An example is the creation of the ImageNet dataset for visual object recognition (Deng et al., 2009).

FL is ideal for training models where the data is self-labeling: during the course of a user's device interaction, both their input data and the associated label can be acquired and stored in a local cache for later use in on-device training. But when a problem is not self-labeling the application of FL becomes tricky, as there is no privacy-preserving way for a human rater to examine an example input residing on a user's device. What is desired is a way to do FL where data labeling could still be performed by human raters, as this would expand the set of ML problems that can be afforded FL's privacy advantages.

3 DP Federated Generative Models

When faced with the challenges presented in the previous section, what can the modeler use in place of direct data inspection when working with private and decentralized data? We propose combining three technologies: generative models, federated learning (FL), and differential privacy. Deep generative models provide the capability to synthesize novel, high-quality examples, FL allows for training and evaluating against distributed data, and FL and DP both afford privacy protections to users' data.

Critically, none of the challenges presented require the inspection of any particular user's data[2]. As with ML, the goal is to discover something broadly meaningful. Thus, it is natural to consider the use of suitable synthetic examples in lieu of real user data. For this, we can leverage generative models based on deep neural networks (or 'deep generative models'). In contrast to discriminative models, generative models learn a joint distribution $p(x, y)$ and can be applied to data *synthesis*. Deep generative models have garnered a considerable amount of research interest, particularly for high dimensional spaces where explicit modeling is intractable. Via deep generative models, a neural network can be used to either approximate a likelihood function or serve as a mechanism for drawing samples (i.e., an implicit distribution). Deep generative models

[2] Note that debugging a problem that manifests only for a single user is out-of-scope, or at least requires different techniques than those presented here if the issue is to be discovered in a privacy-preserving way.

come in many classes (Kingma & Welling, 2013; Goodfellow et al., 2014; Kumar et al., 2019; Radford et al., 2019) and have been applied to a variety of rich domains including text, audio, and images.

To work with decentralized data, we train these generative models via federating learning, ensuring raw user data never leaves the edge device. Instead, a randomly chosen subset of devices download the current model, and each locally computes a model update based on their own data. Ephemeral model updates are then sent back to the coordinating server, where they are aggregated and used to update the global model. This process repeats over many rounds until the model converges. FL is borne of the mobile domain, where privacy is paramount and data is decentralized. Privacy for the mobile domain is a major motivator for this work, and we'll use FL extensively in this paper.

As with other machine learning models, deep generative models have a tendency to memorize unique training examples, leading to a valid concern that they may leak personal information. Differential privacy is a powerful tool for preventing such memorization. In particular, in this paper we'll emphasize the use of *user*-level DP, obtained in the FL context via a combination of per-user update clipping and post-aggregation Gaussian noising, following McMahan et al. (2018); Appendix A reviews user-level DP and FL. We assume this mechanism is implemented by the FL infrastructure, before the modeler has access to the trained models. This approach bounds the privacy loss to both the modeler and the devices that participate in FL rounds. In our experiments, we quantify the privacy obtained using (ϵ, δ) upper bounds. However, such upper bounds are often loose, and so can be complemented by direct measurement of memorization, as in Carlini et al. (2018); Song & Shmatikov (2019). Further, while often unrealistic as a threat model, membership inference attacks on GANs can also be used as a tool to empirically quantify memorization (Hayes et al., 2019; Hilprecht et al., 2019). Finally, we note that since the modeler has access to not only the auxiliary generative models, but also potentially other models trained on the same data, if an (ϵ, δ) guarantee is desired, the total privacy loss should be quantified across these models, e.g. using the strong composition theorem for DP (Mironov, 2017; Kairouz et al., 2017).

Combining these techniques, private generative models can provide an engine for producing high-quality examples that are statistically representative of a data distribution, where that distribution is an aggregation of many private local datasets, and without any of the synthesized examples revealing any personally-identifying piece of local data.

4 Related Work

In prior work, researchers have looked at training GANs with differential privacy (DP) in a centralized setting (Torkzadehmahani et al., 2019; Xie et al., 2018; Zhang et al., 2018; Frigerio et al., 2019; Beaulieu-Jones et al., 2018; Esteban et al., 2017). The focus is on how privacy protections like DP can be leveraged to afford quantifiable guarantees that the trained GAN will not synthesis examples that are too close to any real data. But this work does not involve decentralized data or FL, as would be the case when training a model on mobile devices.

Other work studies generative models for problems with decentralized data, but without privacy protections. An example is an application of GANs (Hardy et al., 2018) to decentralized data across multiple data centers. We consider such research thematically distant from the privacy focus of this paper.

One recent paper does combine all three constituent components (generative models, privacy measures, and federated learning) of our work here. The FedGP algorithm presented in Triastcyn & Faltings (2019) takes a similar approach to yielding a privacy-sensitive data synthesis engine, but differs in a few important ways. It is motivated by a different problem (synthesizing non-personally identifying labeled data for use in training a supervised ML network) than our desire to aid human ML modeler workflows on private, decentralized data. Also, where we use DP as our means of measuring and protecting against data release, it adopts a different measure of privacy (specifically, differential *average-case* privacy). Finally, in our algorithm for

GAN training (Algorithm 1), we train a single global generator and discriminator, and exchange the discriminator between the coordinating server and devices during a communication round. In FedGP, each device maintains a personalized version of a discriminator, and exchanges the generator during a communication round.

Perhaps the existing research closest in spirit to this paper is recent language modeling work in Chen et al. (2019) on using a character-level RNN trained via FL to generate common out-of-vocabulary (OOV) words (i.e., words that fall outside a finite vocabulary set). OOVs are interesting to the language modeler as they reflect natural temporal changes in word usage frequency (e.g., new slang terms, names in the news, etc.) that should be tracked. We use this model for debugging purposes in experiments in Section 5.

The next two sections present realistic scenarios together with experiments which demonstrate the use of an auxiliary federated generative model.

5 An Application to Debugging During Training with RNNs

DP Federated RNNs for Generating Natural Language Data Recurrent Neural Networks (RNNs) are a ubiquitous form of deep network, used to learn sequential content (e.g., language modeling). An interesting property of RNNs is that they embody both discriminative and generative behaviors in a single model. With regards to the former, an RNN learns a conditional distribution $p(x_{i+1}|x_i; D)$. Given a token x_i and its successor x_{i+1}, the RNN reports the probability that x_{i+1} succeeds x_i based on observing dataset D.

But an RNN is also a generative model: the successor token probability distribution can be sampled from to generate a next token, and this process can be repeated to generate additional tokens in the sequence. The repeated application of the RNN provides a sample from the joint distribution:

$$p(x_n, x_{n-1}, \ldots, x_2, x_1, x_o|D) = \prod_{i=0}^{n-1} p(x_{i+1}|x_i; D) \cdot p(x_0|D) \tag{1}$$

In this way, RNN word language models (word-LM) generate sentences and RNN character language models (char-LM) generate words. Training an RNN under FL and DP can be performed via the DP-FedAvg algorithm of McMahan et al. (2018) (restated with minor changes in Appendix A, Algorithm 2).

Experiment Consider a mobile keyboard application that uses a word-LM to offer next-word predictions to users based on previous words. The application takes as input raw text, performs preprocessing (e.g. tokenization and normalization), and then feeds the processed list of words as input to the word-LM. The word-LM has a fixed vocabulary of words V. Any words in the input outside of this vocabulary are marked as 'out-of-vocabulary' (OOV). Such word-LMs have been trained on-device via FL (Hard et al., 2018). We demonstrate how we can detect a bug in the training pipeline for the word-LM using DP federated RNNs.

Suppose a bug is introduced in tokenization which incorrectly concatenates the first two tokens in some sentences into one token. E.g., `'Good day to you.'` is tokenized into a list of words as [`'Good day'`, `'to'`, ...], instead of [`'Good'`, `'day'`, `'to'`, ...]. Such issues are not uncommon in production NLP systems. Because these concatenated tokens don't match words in vocabulary V, they will be marked as OOVs. Under normal circumstances OOVs occur at a relatively consistent rate, so this bug will be noticed via a spike in OOV frequency, a metric an NLP modeler would typically track (see Appendix B). Were this the cloud, some of the OOV tokens could be inspected, and the erroneous concatenation realized. But here the dataset is private and decentralized, so inspection is precluded.

We simulate this scenario using a dataset derived from Stack Overflow and hosted by Kaggle[3]. This data provides a realistic proxy for federated data, since we can associate all posts from a single author as a particular user device; see Appendix B.1 for further details on the data. We artificially introduce the concatenation bug in the dataset by concatenating the first two tokens in 10% of all of the sentences across users.

Two federated generative models are used in complementary fashion for debugging. The first is the 'primary' model, the DP word-LM that is being trained via FL for use in the next word prediction pipeline. While the bug is inhibiting its training for production use, we can still harness its generative properties for insight into the nature of the bug. The second is an 'auxiliary' model, a differentially private char-LM similar to networks used for addressing the on-device frequent OOV discovery problem (Chen et al., 2019). We assume we have access to both the word-LM and char-LM trained before the bug was introduced, as well as both models trained on buggy data — this emphasizes the importance of the continuous training of auxiliary models as part of a production FL workflow, as often the insights are gleaned not from the generated output of a single model, but from the comparison of the output of two generative models trained on different data (either different in time, as in this example, or different in selection criteria, as in Section 6).

Both the DP word- and char-LMs are trained independently via simulated FL (following the approach of McMahan et al. (2018)) on the bug-augmented and bug-free Stack Overflow dataset, producing four trained models in total. These models are now used to synthesize useful and complementary information about the pipeline's input data. From Figure 1, we notice that with the bug the sentences generated by the word-LM have a larger than normal OOV rate for the first position. In Table 1, we see that with the bug most of the top OOV words generated from the char-LM have spaces in them, indicating a tokenization bug in the pipeline. This bug can then be fixed by the modeler. See Appendix B for further details and expanded results.

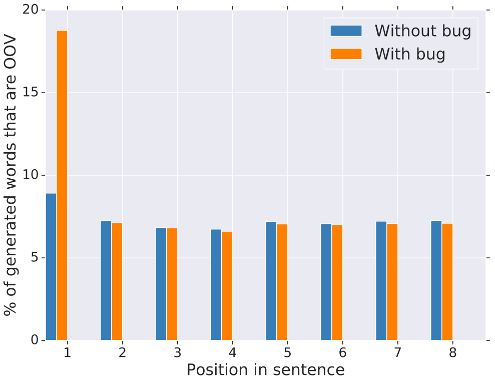


Figure 1: Percentage of samples generated from the word-LM that are OOV by position in the sentence with and without the bug.

Without bug (prob 10^{-3})		*With 10% bug (prob 10^{-3})*	
`regex`	5.50	`i have`	8.08
`jsfiddle`	3.90	`i am`	5.60
`xcode`	3.12	`regex`	4.45
`divs`	2.75	`you can`	3.71
`stackoverflow`	2.75	`if you`	2.81
`listview`	2.74	`this is`	2.77
`textbox`	2.73	`here is`	2.73
`foreach`	2.34	`jsfiddle`	2.70
`async`	2.27	`i want`	2.39
`iis`	2.21	`textbox`	2.28

Table 1: Top 10 generated OOV words by joint character probability (computed using Equation 1), with and without the bug. Number accompanying is joint probability. The model is trained with case-insensitive tokenization.

Privacy The dataset contains 342,477 unique users. Each round, 5,000 random users are selected[4]. The models trained for 2,000 rounds with noise scale of 1.0. The resulting ϵ is 9.22 for a δ of 2.92×10^{-6}.

[3] https://www.kaggle.com/stackoverflow/stackoverflow

[4] These experiments used sampling with replacement instead of without replacement due to a technical limitation of the simulation framework; however, this difference should have negligible impact on the utility of the models trained.

6 An Application to Debugging During Inference with GANs

DP Federated GANs for Generating Image Data Generative Adversarial Networks (GANs) (Goodfellow et al., 2014) are a state-of-the-art form of deep generative model, with numerous recent successes in particular in the image domain (Isola et al., 2016; Zhu et al., 2017; Karras et al., 2018; 2019; Brock et al., 2019). GANs work by alternately training two networks. One is a generator which maps a random input vector in a low-dimensional latent space into a rich, high-dimensional generative output like an image. The other is a discriminator, which judges whether an input image is 'real' (originating from a dataset of actual images) or 'fake' (created by the generator). Each network tries to defeat the other; the generator's training objective is to create content that the discriminator is unable to discern from real content, and the discriminator's training objective is to improve its ability to discern real content from generated content.

We can take the training framework of the GAN and adapt it to train under FL and DP, analogous to how RNN training was adapted to train under FL and DP in McMahan et al. (2018) via the DP-FedAvg algorithm. The difference here is that two sets of network model parameters are updated, by alternating minimization of two different loss functions. One key insight is that only the discriminator's training step involves the use of real user data which is private and restricted to the user's device; the generator training step does not require real user data, and thus can be computed at the coordinating server via a traditional (non-federated) gradient update. A second insight is that the generator's loss is a function of the discriminator. As observed by numerous earlier works involving GANs and DP, if the discriminator is trained under DP and the generator is trained only via the discriminator, then the generator also guarantees privacy at the same level of DP as the discriminator, via the post-processing property of DP (Dwork & Roth, 2014). Therefore, no additional computational steps (e.g., clipping, noising) are necessary at the application of the generator gradient update.

Algorithm 1 gives our algorithm for training a GAN under FL and DP, which we call DP-FedAvg-GAN. Note that the discriminator update resembles closely the update in standard DP-FedAvg, and then each round of communication concludes with a generator update taking place at the server. The discriminator is explicitly trained under DP. The generator is only exposed to the privatized discriminator during the course of training, and never directly exposed to real user data, so it obtains the same DP guarantees as the discriminator.

Experiment Our second experiment shows how auxiliary DP federated GANs can be used to monitor an on-device handwriting classification network being used for inference. Once trained, the federated GANs synthesize data that identifies the nature of a bug in on-device image preprocessing.

Consider the scenario of a banking app that uses the mobile phone's camera to scan checks for deposit. Internally, this app takes raw images of handwriting, does some pixel processing, and feeds the processed images to a pre-trained on-device CNN to infer labels for the handwritten characters. In the nomenclature of Section 1, this CNN is the 'primary model'. The modeler can monitor its performance via metrics like user correction rate (i.e., how often do users manually correct letters/digits inferred by the primary model), to get coarse feedback on accuracy. To simulate users' processed data, we use the Federated EMNIST dataset (Caldas et al., 2018). Figure 2a shows example images; see Appendix C.1 for further details on the data.

A new software update introduces a bug that incorrectly flips pixel intensities during preprocessing, inverting the images presented to the primary model (Figure 2b).[5] This change to the primary model's input data causes it to incorrectly classify most handwriting. As the update rolls out to an increasing percentage of the app's user population, the user correction rate metric spikes, indicating to the app developers a problem exists, but nothing about the nature of the problem.

Critically, the images in Figure 2 are never viewed by human eyes, but are fed to the primary model on device. If the data were public and the inference being performed on the app's server, some of the misclassified

[5] Such data issues are in fact common in the real world. For example, the Federated EMNIST dataset we use encodes pixel intensities with the opposite sign of the MNIST dataset distributed by TensorFlow (TFF documentation, 2019).

```
Server-orchestrated training loop:
  parameters: round participation fraction $q \in (0, 1]$, total number of users $N \in \mathbb{N}$, total number of rounds $T \in \mathbb{N}$, noise scale $z \in \mathbb{R}^+$, clip parameter $S \in \mathbb{R}^+$

  Initialize generator $\theta_G^0$, discriminator $\theta_D^0$, privacy accountant $\mathcal{M}$

  $\sigma = \frac{zS}{qN}$

  for each round $t$ from 0 to $T$ do
    $\mathcal{C}^t \leftarrow$ (sample of $qN$ distinct users)
    for each user $k \in \mathcal{C}^t$ in parallel do
      $\Delta_k^{t+1} \leftarrow$ UserDiscUpdate$(k, \theta_D^t, \theta_G^t)$

    $\Delta^{t+1} = \frac{1}{qN} \sum_{k \in \mathcal{C}^t} \Delta_k^{t+1}$
    $\theta_D^{t+1} \leftarrow \theta_D^t + \Delta^{t+1} + \mathcal{N}(0, I\sigma^2)$

    $\mathcal{M}$.accum_priv_spending($z$)

    $\theta_G^{t+1} \leftarrow$ GenUpdate$(\theta_D^{t+1}, \theta_G^t)$

  print $\mathcal{M}$.get_privacy_spent()
```

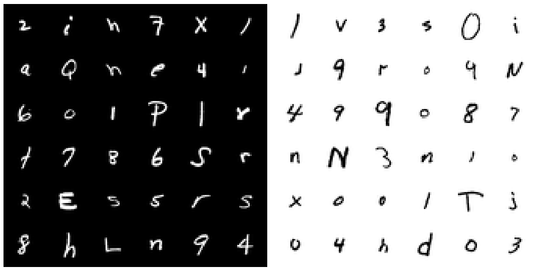
```
UserDiscUpdate$(k, \theta_D^0, \theta_G)$:
  parameters: number of steps $n \in \mathbb{N}$, batch size $B \in \mathbb{N}$, disc. learning rate $\eta_D \in \mathbb{R}^+$, clip parameter $S \in \mathbb{R}^+$, gen. input size $n_U \in \mathbb{N}$, gen. function $G(U; \theta_G)$, disc. loss function $\ell_D(\theta_D; b_{\text{real}}, b_{\text{fake}})$

  $\theta_D \leftarrow \theta_D^0$
  $\mathcal{B} \leftarrow$ ($k$'s data split into $n$ size $B$ batches)
  for each batch $b_{\text{real}} \in \mathcal{B}$ do
    $U \leftarrow$ (sample $B$ random vectors of dim. $n_U$)
    $b_{\text{fake}} \leftarrow G(U; \theta_G)$  // generated data
    $\theta_D \leftarrow \theta_D - \eta_D \nabla \ell_D(\theta_D; b_{\text{real}}, b_{\text{fake}})$
  $\Delta = \theta_D - \theta_D^0$
  return update $\Delta_k = \Delta \cdot \min\left(1, \frac{S}{\|\Delta\|}\right)$  // Clip

GenUpdate$(\theta_D, \theta_G^0)$:
  parameters: number of steps $n \in \mathbb{N}$, batch size $B \in \mathbb{N}$, gen. learning rate $\eta_G \in \mathbb{R}^+$, gen. input size $n_U \in \mathbb{N}$, gen. loss function $\ell_G(\theta_G; b, \theta_D)$

  $\theta_G \leftarrow \theta_G^0$
  for each generator training step $i$ from 1 to $n$ do
    $U \leftarrow$ (sample $B$ random vectors of dim. $n_U$)
    $b_{\text{fake}} \leftarrow G(U; \theta_G)$  // generated data
    $\theta_G \leftarrow \theta_G - \eta_G \nabla \ell_G(\theta_G; b_{\text{fake}}, \theta_D)$
  return $\theta_G$
```

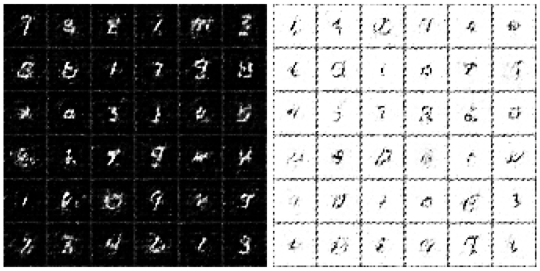
Algorithm 1: DP-FedAvg-GAN. This algorithm is closely based on DP-FedAvg (Appendix C), adapted for minimizing the two loss functions of GAN training..



(a) Expected. (Without Bug)

(b) Inverted. (Bug)

Figure 2: Examples of primary model CNN input, from EMNIST with letters and digits (62 classes).



(a) Trained on high-accuracy users.

(b) Trained on low-accuracy users.

Figure 3: DP Federated GAN generator output given an inversion bug on 50% of devices.

handwriting images could potentially be inspected, and the pixel inversion bug realized. But this cannot be done when the data is private and decentralized.

Instead, we propose training two DP federated GANs via Algorithm 1: one on a subset of the processed image data that tends to perform best when passed to the primary model, and one on a subset of processed image data that tends to perform worst. (Specifically, we do this by forming two distinct subpopulations of users who exhibit high and low classification accuracy, respectively, and training one GAN on each user subpopulation; full details in Appendix C.2.) By contrasting the images synthesized by the two GANs, we hope to understand what is causing degraded classification accuracy for some of the app's users. In our simulation of this scenario, we train for 1,000 federated rounds, with 10 users (drawn at random) participating in each round. See Appendix C.3 for training details. Figures 3a and 3b show examples of images generated by the two trained GANs' generators, when the bug is present on 50% of users' mobile phones.

Comparing the output of the two GANs, the presence of a pixel inversion bug is clear, enabling the modeler to locate and fix the issue. A natural follow-on experiment is to repeat the process of training the two GANs after the fix is deployed, which produces the results shown in Figure 6 in Appendix C.4. These images verify that the pixel inversion bug is fixed. However, potential new insights might also be gleaned by comparing the generated content of these new GANs. Perhaps it identifies another, more subtle bug, or a particular characteristic of handwriting common to users with many misclassified examples (indicating a missing user class in the primary model's training data). We propose it may be useful to continually train federated generative models like the GANs in this experiment, as a tool to monitor and debug.

Privacy at Scale We follow an approach as in McMahan et al. (2018) to determine the level of privacy afforded were the experiment scaled up to realistic population sizes. That is, in simulation we run our experiment with only 10 users per round and a small total population size (3,400) in order to validate the utility of our models and also determine a noise scale multiplier z that does not degrade the quality of the GAN's generated images. This simulation scenario will not afford good privacy protections (as indicated by the resulting very large ϵ bound). But it does indicate the factor necessary to scale z up to 1.0; this is the factor we'd have to apply to per-round participation count to achieve a real-world scenario that affords good privacy protections (as measured by ϵ and δ) while maintaining the quality of the GAN's generated images (since the relative noise in each average model update is unchanged). As the experiment shows good utility is achieved with a z of 0.01, the factor to apply is 100: a real-world use case would involve 1,000 users per round, out of a total population of 2 million mobile app users. These numbers are reasonable for a real-world system, e.g. (Bonawitz et al., 2019).

Table 2 gives DP hyperparameters and resulting ϵ and δ values, both for the simulation and the corresponding scenario with scaled-up, realistic population. The calculation of ϵ and δ is fully described in Appendix A.1. Note that three sets of numbers are presented, as ϵ and δ depend on the number of users meeting the selection criteria for a given use case. The cases are: (i) high-accuracy users when the bug exists (12.5% of overall population), (ii) low-accuracy users when the bug exists (62.5% of overall population), and (iii) either group of users when the bug is gone (25% of overall population). In all cases, for the realistic population we achieve single digit ϵ (between 1.48 and 2.38), indicating a reasonable amount of user privacy.

7 Conclusion

In this work we described ML modeler workflows that rely on data inspection and are precluded when direct inspection is impossible (i.e., in a private, decentralized data paradigm). We proposed a tool to overcome this limitation: differentially private, federated generative models that synthesize examples representative of the private data. We demonstrated application of two example model classes (DP federated RNNs and GANs), showing how they enable a modeler to debug natural language and image problems. In the GAN case, we presented a novel algorithm for training it under FL and DP. Section 2 mentioned several other modeler workflows (e.g., debiasing, labeling) that rely on data inspection and are inhibited in a private, decentralized paradigm. Applying DP federated generative models to these workflows will be pursued in future work.

To convert the concept proposed in this paper into a practical data science tool, a variety of further research is necessary. For federated generative models to be useful and broadly applicable, they should require minimal tuning; excessive experimentation needed to achieve convergence both destroys the value proposition to

	qN	N	z	ϵ	δ
simulation (Figure 3a)	10	425	0.01	1.00×10^7	2.35×10^{-3}
realistic scenario	$1,000$	$250,000$	1.00	2.38	4.0×10^{-8}
simulation (Figure 3b)	10	$2,125$	0.01	1.00×10^7	4.71×10^{-4}
realistic scenario	$1,000$	$1,250,000$	1.00	1.48	8.0×10^{-9}
simulation (Figure 6)	10	850	0.01	1.00×10^7	1.18×10^{-3}
realistic scenario	$1,000$	$500,000$	1.00	1.79	2.0×10^{-8}

Table 2: Privacy parameters for different scenarios. N is size of user subpopulation that meets selection criteria (not overall population size). All experiments use clip parameter S of 0.1 and 1,000 rounds. Simulations are with overall population of 3,400, and realistic scenarios are with overall population of 2,000,000.

the modeler and destroys the privacy budget under differential privacy analysis. They must be capable of synthesizing relevant content in scenarios where the 'signal-to-noise' ratio is low, e.g., when a bug is only present in a small amount of data (in absolute or relative terms). How to achieve this in the case of GANs, which are powerful but can suffer from mode collapse, is of special interest. Advances in these dimensions will greatly benefit the private, decentralized data science community. As an initial step to stimulate research in this area, we provide an open-source implementation[6] of our DP Federated GAN code (used to generate the results in Section 6). We share additional thoughts on open problems in Appendix D.

[6] Withheld for anonymity in review process. Code will be open-sourced post-review.

REFERENCES

Martin Abadi, Andy Chu, Ian Goodfellow, Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. Deep learning with differential privacy. In *23rd ACM Conference on Computer and Communications Security (ACM CCS)*, 2016.

Julia Angwin, Jeff Larson, Surya Mattu, and Lauren Kirchner. Machine bias: There's software used across the country to predict future criminals. and it's biased against blacks. *ProPublica*, 2016.

Raef Bassily, Adam Smith, and Abhradeep Thakurta. Private empirical risk minimization: Efficient algorithms and tight error bounds. In *Proceedings of the 2014 IEEE 55th Annual Symposium on Foundations of Computer Science*, FOCS '14, pp. 464–473, Washington, DC, USA, 2014. IEEE Computer Society. ISBN 978-1-4799-6517-5. doi: 10.1109/FOCS.2014.56. URL `http://dx.doi.org/10.1109/FOCS.2014.56`.

Brett K. Beaulieu-Jones, Zhiwei Steven Wu, Chris Williams, Ran Lee, Sanjeev P. Bhavnani, James Brian Byrd, and Casey S. Greene. Privacy-preserving generative deep neural networks support clinical data sharing. *bioRxiv*, 2018. doi: 10.1101/159756. URL `https://www.biorxiv.org/content/early/2018/12/20/159756`.

Keith Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloé Kiddon, Jakub Konečný, Stefano Mazzocchi, H Brendan McMahan, Timon Van Overveldt, David Petrou, Daniel Ramage, and Jason Roselander. Towards federated learning at scale: System design. In *SysML 2019*, 2019.

Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale GAN training for high fidelity natural image synthesis. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=B1xsqj09Fm`.

Sebastian Caldas, Peter Wu, Tian Li, Jakub Konečný, H Brendan McMahan, Virginia Smith, and Ameet Talwalkar. Leaf: A benchmark for federated settings. *arXiv preprint arXiv:1812.01097*, 2018.

N. Carlini, C. Liu, J. Kos, Ú. Erlingsson, and D. Song. The Secret Sharer: Measuring Unintended Neural Network Memorization & Extracting Secrets. *ArXiv e-prints*, February 2018. URL `https://arxiv.org/abs/1802.08232`.

Aleksandar Chakarov, Aditya V. Nori, Sriram K. Rajamani, Shayak Sen, and Deepak Vijaykeerthy. Debugging machine learning tasks. *CoRR*, abs/1603.07292, 2016. URL `http://arxiv.org/abs/1603.07292`.

Kamalika Chaudhuri, Claire Monteleoni, and Anand D. Sarwate. Differentially private empirical risk minimization. *J. Mach. Learn. Res.*, 12, July 2011.

Mingqing Chen, Rajiv Mathews, Tom Ouyang, and Françoise Beaufays. Federated learning of out-of-vocabulary words. *CoRR*, abs/1903.10635, 2019. URL `http://arxiv.org/abs/1903.10635`.

J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

Emily Denton, Ben Hutchinson, Margaret Mitchell, and Timnit Gebru. Detecting bias with generative counterfactual face attribute augmentation. 2019.

Cynthia Dwork. A firm foundation for private data analysis. *Commun. ACM*, 54(1):86–95, January 2011. ISSN 0001-0782. doi: 10.1145/1866739.1866758. URL `http://doi.acm.org/10.1145/1866739.1866758`.

Cynthia Dwork and Aaron Roth. *The Algorithmic Foundations of Differential Privacy*. Foundations and Trends in Theoretical Computer Science. Now Publishers, 2014.

Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating noise to sensitivity in private data analysis. In *TCC*, pp. 265–284. Springer, 2006.

Cristóbal Esteban, Stephanie Hyland, and Gunnar Rätsch. Real-valued (medical) time series generation with recurrent conditional gans, 2017. URL `http://arxiv.org/abs/1811.03850`.

Lorenzo Frigerio, Anderson Santana de Oliveira, Laurent Gomez, and Patrick Duverger. Differentially private generative adversarial networks for time series, continuous, and discrete open data. *CoRR*, abs/1901.02477, 2019. URL `http://arxiv.org/abs/1901.02477`.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger (eds.), *Advances in Neural Information Processing Systems 27*, pp. 2672–2680. Curran Associates, Inc., 2014. URL `http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf`.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. Deep learning. Book in preparation for MIT Press, 2016.

Klaus Greff, Rupesh Kumar Srivastava, Jan Koutník, Bas R. Steunebrink, and Jürgen Schmidhuber. LSTM: A search space odyssey. *CoRR*, abs/1503.04069, 2015. URL `http://arxiv.org/abs/1503.04069`.

Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron Courville. Improved training of wasserstein gans. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, pp. 5769–5779, USA, 2017. Curran Associates Inc. ISBN 978-1-5108-6096-4. URL `http://dl.acm.org/citation.cfm?id=3295222.3295327`.

Andrew Hard, Kanishka Rao, Rajiv Mathews, Françoise Beaufays, Sean Augenstein, Hubert Eichner, Chloé Kiddon, and Daniel Ramage. Federated learning for mobile keyboard prediction. *arXiv preprint arXiv:1811.03604*, 2018.

Corentin Hardy, Erwan Le Merrer, and Bruno Sericola. MD-GAN: multi-discriminator generative adversarial networks for distributed datasets. *CoRR*, abs/1811.03850, 2018. URL `http://arxiv.org/abs/1811.03850`.

Jamie Hayes, Luca Melis, George Danezis, and Emiliano De Cristofaro. Logan: Membership inference attacks against generative models. *Proceedings on Privacy Enhancing Technologies*, 2019(1):133–152, 2019.

Benjamin Hilprecht, Martin Härterich, and Daniel Bernau. Reconstruction and membership inference attacks against generative models. *arXiv preprint arXiv:1906.03006*, 2019.

Alex Ingerman and Krzys Ostrowski. Introducing tensorflow federated, 2019. URL `https://medium.com/tensorflow/introducing-tensorflow-federated-a4147aa20041`.

Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. *arxiv*, 2016.

Bargav Jayaraman and David Evans. Evaluating differentially private machine learning in practice. *USENIX Security Symposium*, Aug 2019. URL `http://par.nsf.gov/biblio/10099285`.

Peter Kairouz, Sewoong Oh, and Pramod Viswanath. The composition theorem for differential privacy. *IEEE Transactions on Information Theory*, 63(6):4037–4049, 2017.

Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of GANs for improved quality, stability, and variation. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=Hk99zCeAb.

Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

Been Kim, Martin Wattenberg, Justin Gilmer, Carrie Jun Cai, James Wexler, Fernanda Viegas, and Rory Abbott Sayres. Interpretability beyond feature attribution: Quantitative testing with concept activation vectors (tcav). In *ICML 2018: Proceedings of the 35th International Conference on Machine Learning*, 2018. URL http://proceedings.mlr.press/v80/kim18d/kim18d.pdf.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes, 2013.

Manoj Kumar, Mohammad Babaeizadeh, Dumitru Erhan, Chelsea Finn, Sergey Levine, Laurent Dinh, and Durk Kingma. Videoflow: A flow-based generative model for video. *CoRR*, abs/1903.01434, 2019. URL http://arxiv.org/abs/1903.01434.

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*, pp. 1273–1282, 2017.

H Brendan McMahan and Daniel Ramage. Federated learning: Collaborative machine learning without centralized training data, April 2017. URL https://ai.googleblog.com/2017/04/federated-learning-collaborative.html. Google AI Blog.

H Brendan McMahan, Daniel Ramage, Kunal Talwar, and Li Zhang. Learning differentially private recurrent language models. In *International Conference on Learning Representations (ICLR)*, 2018.

Ilya Mironov. Rényi differential privacy. In *2017 IEEE 30th Computer Security Foundations Symposium (CSF)*, pp. 263–275. IEEE, 2017.

Nicolas Papernot, Martín Abadi, Úlfar Erlingsson, Ian Goodfellow, and Kunal Talwar. Semi-supervised knowledge transfer for deep learning from private training data. In *Proceedings of the International Conference on Learning Representations*, 2017. URL https://arxiv.org/abs/1610.05755.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. In *OpenAI Technical report*, 2019.

Patrick Riley. The unofficial google data science blog: Practical advice for analysis of large, complex data sets, October 2016. URL http://www.unofficialgoogledatascience.com/2016/10/practical-advice-for-analysis-of-large.html.

Hasim Sak, Andrew W. Senior, and Françoise Beaufays. Long short-term memory based recurrent neural network architectures for large vocabulary speech recognition. *CoRR*, abs/1402.1128, 2014. URL http://arxiv.org/abs/1402.1128.

Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, Xi Chen, and Xi Chen. Improved techniques for training gans. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett (eds.), *Advances in Neural Information Processing Systems 29*, pp. 2234–2242. Curran Associates, Inc., 2016. URL http://papers.nips.cc/paper/6125-improved-techniques-for-training-gans.pdf.

Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

D. Sculley, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips, Dietmar Ebner, Vinay Chaudhary, Michael Young, Jean-François Crespo, and Dan Dennison. Hidden technical debt in machine learning systems. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett (eds.), *Advances in Neural Information Processing Systems 28*, pp. 2503–2511. Curran Associates, Inc., 2015. URL `http://papers.nips.cc/paper/5656-hidden-technical-debt-in-machine-learning-systems.pdf`.

Congzheng Song and Vitaly Shmatikov. Auditing data provenance in text-generation models, 2019.

TFF documentation. `tff.simulation.datasets.emnist.load_data` documentation, 2019. URL `https://www.tensorflow.org/federated/api_docs/python/tff/simulation/datasets/emnist/load_data`.

Reihaneh Torkzadehmahani, Peter Kairouz, and Benedict Paten. Dp-cgan: Differentially private synthetic data and label generation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pp. 0–0, 2019.

Aleksei Triastcyn and Boi Faltings. Federated generative privacy. *Proceedings of the IJCAI Workshop on Federated Machine Learning for User Privacy and Data Confidentiality (FML 2019)*, 2019. URL `http://infoscience.epfl.ch/record/269023`.

Yu-Xiang Wang, Borja Balle, and Shiva Kasiviswanathan. Subsampled r\'enyi differential privacy and analytical moments accountant. *arXiv preprint arXiv:1808.00087*, 2018.

Xi Wu, Fengan Li, Arun Kumar, Kamalika Chaudhuri, Somesh Jha, and Jeffrey F. Naughton. Bolt-on differential privacy for scalable stochastic gradient descent-based analytics. In *Proceedings of SIGMOD*, 2017.

Liyang Xie, Kaixiang Lin, Shu Wang, Fei Wang, and Jiayu Zhou. Differentially private generative adversarial network. *CoRR*, abs/1802.06739, 2018. URL `http://arxiv.org/abs/1802.06739`.

Timothy Yang, Galen Andrew, Hubert Eichner, Haicheng Sun, Wei Li, Nicholas Kong, Daniel Ramage, and Françoise Beaufays. Applied federated learning: Improving google keyboard query suggestions. *arXiv preprint 1812.02903*, 2018.

Xinyang Zhang, Shouling Ji, and Ting Wang. Differentially private releasing via deep generative model. *CoRR*, abs/1801.01594, 2018. URL `http://arxiv.org/abs/1801.01594`.

Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Computer Vision (ICCV), 2017 IEEE International Conference on*, 2017.

Martin Zinkevich. Rules of machine learning: Best practices for ml engineering., 2017. URL `http://martin.zinkevich.org/rules_of_ml/rules_of_ml.pdf`.

A User-Level Differential Privacy and Federated Learning

Differential privacy (DP) (Dwork et al., 2006; Dwork, 2011; Dwork & Roth, 2014) provides a well-tested formalization for the release of information derived from private data. Applied to machine learning, a differentially private training mechanism allows the public release of model parameters with a strong guarantee: adversaries are severely limited in what they can learn about the original training data based on analyzing the parameters, even when they have access to arbitrary side information. Formally, it says:

Definition 1 *Differential Privacy: A randomized mechanism $\mathcal{M}\colon \mathcal{D} \rightarrow \mathcal{R}$ with a domain $\mathcal{D}$ (e.g., possible training datasets) and range $\mathcal{R}$ (e.g., all possible trained models) satisfies (ϵ, δ)-differential privacy if for any two* ***adjacent*** *datasets $d, d' \in \mathcal{D}$ and for any subset of outputs $S \subseteq \mathcal{R}$ it holds that $\Pr[\mathcal{M}(d) \in S] \leq e^{\epsilon} \Pr[\mathcal{M}(d') \in S] + \delta$.*

The definition above leaves open the definition of adjacent datasets which will depend on the application. Most prior work on differentially private machine learning (e.g. Chaudhuri et al. (2011); Bassily et al. (2014); Abadi et al. (2016); Wu et al. (2017); Papernot et al. (2017)) deals with *example*-level privacy: two datasets d and d' are defined to be adjacent if d' can be formed by adding or removing a single training example from d.

However, recent work has presented an algorithm (DP-FedAvg, McMahan et al. (2018)) for differentially private machine learning with *user*-level privacy. Intuitively speaking, a model trained under DP-FedAvg should not change too much when one user's data is added, removed, or changed arbitrarily. Learning takes place via FL; since FL processes all of one user's data together, such guarantees are relatively easier to obtain than in the centralized setting.

We adopt an approach (Algorithm 2) for performing differentially private FL that follows closely from the DP-FedAvg algorithm. Similarly, our focus is on user-level differential privacy, where the privacy guarantee extends to all of a user's data (rather than, e.g., a single training example).

Specifically, following DP-FedAvg, we:

- enforce clipping of per-user updates so the total update has bounded L_2 norm.
- use a bounded-sensitivity estimator for computing the weighted average update to the global model.
- add Gaussian noise to the weighted average update.

A slight difference between our approach and that presented in DP-FedAvg is how we select devices to participate in a given federated round of computation. DP-FedAvg uses randomly-sized federated rounds, where users are selected independently with probability q. In this paper, we instead use fixed-size federated rounds, where the number of users selected to participate in the round, qN, is a constant (and N is the total number of mobile devices participating in training). I.e., q is the round participation fraction. This method of sampling has minor but non-trivial ramifications for the calculation of the overall privacy bound, which we discuss next.

A.1 Privacy Bounds

To obtain precise differential privacy guarantees, we use the analytical moments accountant of subsampled Rnyi differential privacy (RDP) method developed in Wang et al. (2018). In particular, the work of Wang et al. provides a tight upper bound on the RDP parameters for an algorithm that: (a) uniformly subsamples records from an underlying dataset, and then (b) applies a randomized mechanism to the subsample (Gaussian perturbation in our case). This is precisely what happens in one round of Algorithm 1 or 2. To track the overall privacy budget, we multiple the RDP orders of the uniformly subsampled Gaussian mechanism

```
Server-orchestrated training loop:
  parameters
    per-user example cap $\hat{w} \in \mathbb{R}^+$
    round participation fraction $q \in (0, 1]$
    total user population $N \in \mathbb{N}$
    noise scale $z \in \mathbb{R}^+$
    clip parameter $S \in \mathbb{R}^+$

  Initialize model $\theta^0$, moments accountant $\mathcal{M}$

  $n_k =$ number of examples for user $k$
  $w_k = \min\big(\frac{n_k}{\hat{w}}, 1\big)$ for all users $k$
  Set $W = \sum_{k \in d} w_k$ and $\sigma = \frac{zS}{qW}$

  for each round $t = 0, 1, 2, \ldots$ do
    $\mathcal{C}^t \leftarrow$ (sample without replacement $qN$ users from population)
    for each user $k \in \mathcal{C}^t$ in parallel do
      $\Delta_k^{t+1} \leftarrow$ UserUpdate($k, \theta^t$)

    $\Delta^{t+1} = \frac{1}{qW} \sum_{k \in \mathcal{C}^t} w_k \Delta_k^{t+1}$
    $\theta^{t+1} \leftarrow \theta^t + \Delta^{t+1} + \mathcal{N}(0, I\sigma^2)$
    $\mathcal{M}$.accum_priv_spending($z$)

  print $\mathcal{M}$.get_privacy_spent()
```

```
UserUpdate($k, \theta^0$):
  parameters
    number of local epochs $E \in \mathbb{N}$
    batch size $B \in \mathbb{N}$
    learning rate $\eta \in \mathbb{R}^+$
    clip parameter $S \in \mathbb{R}^+$
  parameter functions
    loss function $\ell(\theta; b)$

  $\theta \leftarrow \theta^0$
  for each local epoch $i$ from 1 to $E$ do
    $\mathcal{B} \leftarrow$ ($k$'s data split into size $B$ batches)
    for each batch $b \in \mathcal{B}$ do
      $\theta \leftarrow \theta - \eta \nabla \ell(\theta; b)$
  $\Delta = \theta - \theta^0$
  return update $\Delta_k = \Delta \cdot \min\left(1, \frac{S}{\|\Delta\|}\right)$   // Clip
```

Algorithm 2: DP-FedAvg with fixed-size federated rounds. This is the algorithm we use to train the word and character RNN LMs in Section 5.

by the number of rounds we execute Algorithm 1 or 2 and convert the resulting RDP orders to an (ϵ, δ) pair using Proposition 3 of Mironov (2017).

B DP FEDERATED RNNS: EXPERIMENTAL DETAILS

B.1 STACK OVERFLOW DATA

The Stack Overflow dataset contains questions and answers from the Stack Overflow forum grouped by user ids. The data consists of the text body of all questions and answers. The bodies were parsed into sentences, and any user with fewer than 100 sentences was expunged from the data. Minimal preprocessing was performed to fix capitalization, remove URLs etc. The corpus is divided into train, held-out and test parts. Table 3 summarizes the statistics on number of users and number of sentences for each partition. The dataset is available via the Tensorflow Federated open-source software framework (Ingerman & Ostrowski, 2019).

	Train	Held-out	Test
# Users	342K	38.8K	204K
# Users with both questions and answers	297K	34.3K	99.3K
# Sentences	136M	16.5M	16.6M
# Sentences that are questions	57.8M	7.17M	7.52M
# Sentences that are answers	78.0M	9.33M	9.07M

Table 3: Number of users and sentences in the Stack Overflow dataset.

B.2 RNN Model Architectures and Training

DP Word-LM The DP word-LM model is based on a coupled input forget gate CIFG-LSTM (Greff et al., 2015) with projection layer (Sak et al., 2014). It uses a 10K vocabulary, which is composed of the most frequent 10K words from the training corpus. The model architecture is based on an RNN with hidden size of 670 and an input embedding dimension of 96. The model is trained for 2,000 rounds. A server-side learning rate of 1.0, an on-device learning rate of 0.5, and Nesterov momentum of 0.99 are used. We use a noise scale of 1.0 and 5,000 devices per round.

DP Char-LM The model architecture used to train the DP char-LM is based on “$\text{FL}_{\text{L}}^{\text{M}}$” in Chen et al. (2019), which is also a CIFG-LSTM with projection using 3 layers, 256 RNN hidden units and 128 projected dimensions. The input embedding dimension is also 128. The character vocabulary has a size of 258, based on UTF-8 encoding with 256 categories from a single byte and additional start-of-word and end-of-word token. All other training and optimization parameters are the same as the word-LM model. Similar to Chen et al. (2019), once the char-LM is trained, Monte Carlo sampling is performed to generate OOV words.

B.3 Expanded Results

To demonstrate the effectiveness of using the OOV model to monitor the introduced system bug, the experiments are done in four comparison settings: (1) without bug; (2) 1% of sentences have first two words concatenated; (3) 10% sentences; (4) 100% sentences. Each task goes through 2,000 federated rounds, to meet the privacy guarantees in Section 6. We observed that models were converged after 2,000 rounds in all four settings.

DP Word-LM Figure 4 shows the percentage of the OOV words by their position of the sentence. It clearly shows that as the percentage of sentences affected increases, the word-LM reflects this in its generated output.

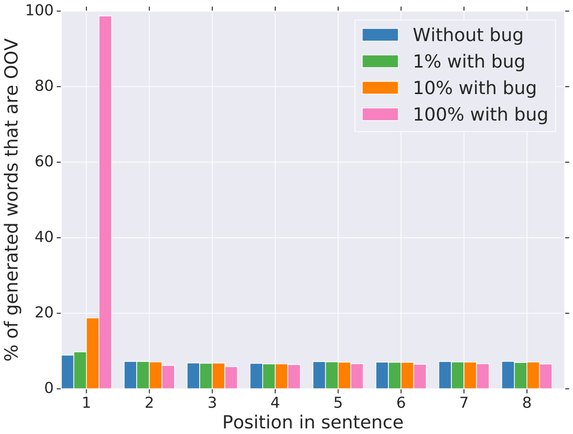


Figure 4: Percentage of samples generated from the word-LM that are OOV by position in the sentence (computed over 100,000 samples)

without bug	*1%*
`for UNK metrics to work fine , ago`	`this tries to load the textbox by get`
`i think your question is the correct format`	`if any user who was originally have an`
`question 1 UNK ? java 1 , UNK`	`which one is different at the same time`
`is the selector of the type of module`	`UNK currently building my ios app to integrate`
`basically , i do a bunch of articles`	`the only thing i can think of is`
`UNK data UNK UNK UNK @ 43 UNK`	`/ home / UNK / jmeter / ;`
`this is a blog post for huge UNK`	`the solution with our application is the current`
`one possibility is to send different limits to`	`there is another solution in their UNK feed`
`this is missing some functions UNK on index`	`( your client sends all the source properties`
`after that UNK start the above , then`	`thank you . . . in your worst`
10%	*100%*
`looks like it are pretty weather when i`	`UNK lg for this is computing 3 as`
`suppose one key is executed when you freeze`	`UNK it is that sql will stop over`
`delete disadvantage of the files and prediction .`	`UNK of UNK you need to call UNK`
`UNK a folder created on one system (`	`UNK ms might have this off ; &`
`please note that the button would mount a`	`UNK in the choice of sort of dragging`
`would it be displaying the same field i`	`UNK , p1 . UNK ( UNK ,`
`UNK just all the UNK files , such`	`UNK UNK editing the push ( ) controllers`
`UNK , it UNK matter why ( note`	`UNK info must have depend on the way`
`simply at UNK . UNK but put it`	`UNK i do it static ? UNK stop`
`i started many debug from the jboss`	`UNK trying to give more html privileges ?`

Table 4: Samples of phrases generated from the word-LM with 0% (without bug), 1%, 10% and 100% of the bug.

Table 4 shows samples of phrases generated from the word-LM in different experiment settings. We can see that the phrases have more OOVs (marked as UNK) as the percentage of sentences affected by the concatenation bug increases.

Table 5 shows the OOV rate observed in four experiment settings. It can be seen that OOV rate goes up as the percentage of sentences affected by the concatenation bug increases. Thus OOV rate can be used as an early abnormality indicator of the corpus.

without bug	*1%*	*10%*	*100%*
6.52%	6.63%	7.60%	17.89%

Table 5: OOV rate observed (during training) in different word-LM experiment settings.

DP Char-LM Table 6 shows the 20 most frequently occurring OOV words generated by the char-LM, for the four different experiment cases. These generated results provide actual information on the tokenized words being fed to the model from the decentralized dataset. Similar to the word-LM, as the ratio of the bug increases, the chance of generating content reflecting the bug (in this case, concatenated words) becomes higher. In 100% setting, all the top 20 words are concatenated words; the word-level joint probabilities of the concatenated words are significantly higher than their non-concatenated counterparts.

without bug (prob 10^{-3}*)*		*1% (prob* 10^{-3}*)*		*10% (prob* 10^{-3}*)*		*100% (prob* 10^{-3}*)*	
`regex`	5.50	`regex`	5.15	`i have`	8.08	`i have`	27.47
`jsfiddle`	3.90	`jsfiddle`	3.73	`i am`	5.60	`i am`	16.50
`xcode`	3.12	`xcode`	2.86	`regex`	4.45	`this is`	9.92
`divs`	2.75	`listview`	2.64	`you can`	3.71	`you can`	9.04
`stackoverflow`	2.75	`textbox`	2.56	`if you`	2.81	`here is`	8.30
`listview`	2.74	`divs`	2.45	`this is`	2.77	`if you`	8.29
`textbox`	2.73	`stackoverflow`	2.44	`here is`	2.73	`i want`	8.17
`foreach`	2.34	`iis`	2.27	`jsfiddle`	2.70	`is there`	7.51
`async`	2.27	`foreach`	2.24	`i want`	2.39	`how can`	5.78
`iis`	2.21	`linq`	2.20	`textbox`	2.28	`when i`	5.33
`onclick`	2.07	`async`	2.17	`xcode`	2.23	`however ,`	5.25
`linq`	2.02	`onclick`	2.05	`listview`	2.10	`i would`	4.72
`params`	2.00	`htaccess`	1.90	`stackoverflow`	2.10	`if i`	4.22
`htaccess`	1.90	`params`	1.89	`is there`	2.07	`for example`	4.20
`arraylist`	1.90	`mongodb`	1.86	`divs`	1.93	`i tried`	4.14
`mongodb`	1.83	`npm`	1.77	`async`	1.79	`edit :`	4.07
`xaml`	1.79	`xaml`	1.74	`linq`	1.72	`i think`	3.86
`npm`	1.66	`arraylist`	1.67	`foreach`	1.71	`the problem`	3.77
`dataframe`	1.61	`enum`	1.58	`iis`	1.70	`i don't`	3.61
`nginx`	1.55	`nginx`	1.55	`however ,`	1.69	`i need`	3.49

Table 6: Top 20 generated OOV words by joint character probability (computed using Equation 1), for 0% (without bug), 1%, 10% and 100% of examples affected by bug. Number accompanying is joint probability.

C DP Federated GAN: Experimental Details

C.1 Federated EMNIST Data

We use the Federated EMNIST dataset (Caldas et al., 2018) to represent the app's users' processed data in our GAN experiments in Section 6. This dataset contains 28x28 gray-scale images (Figure 2a) of handwritten letters and numbers, grouped by writer. This has the realistic property that all data grouped to a given

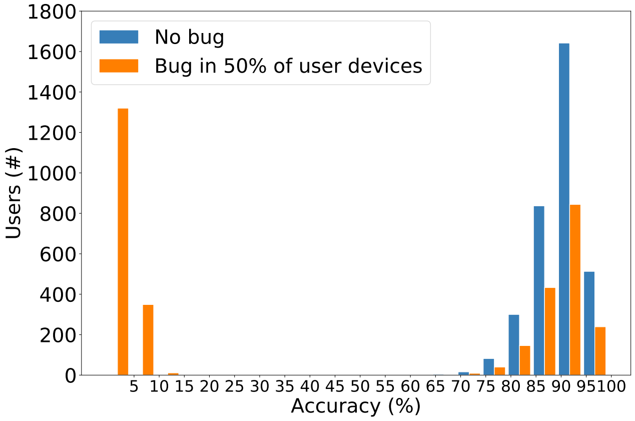


Figure 5: User Accuracy Histogram.

user exhibits the same personally-identifying handwriting properties; our solution should generate examples while obfuscating these properties. The dataset contains 3,400 users. It is available via the Tensorflow Federated open-source software framework (Ingerman & Ostrowski, 2019).

C.2 User Selection

The modeler ideally wishes to train one GAN on the exact subset of data affected by the bug and the other GAN on the exact subset of data unaffected by the bug, in order to best discern the difference and identify the bug. Of course, to achieve this perfectly they would need to know what the actual bug is a priori. As they don't, the modeler must settle for focusing the two GANs to train on the subsets of the data with the highest and lowest likelihood (respectively) of being related to the bug.

A question is how to define these two subsets of the overall decentralized dataset. In the experiments in Section 6, we filter these two subsets at the level of the *user*. We define criteria for a user's app experience to be considered as 'poor', and if a user meets this criteria then their device belongs to the 'poorly performing' subpopulation. When their device is selected to participate in a round, their examples will be used to calculate updates to the GAN discriminator for this subpopulation. (The equivalent is true for identifying members of the 'strongly performing' subpopulation.)

The criteria we use is a user's classification accuracy, i.e., the percentage of examples in their local dataset that were correctly classified by the primary model (the CNN). Figure 5 shows a histogram of classification accuracy for the app's users, and Table 7 gives the 25th and 75th percentiles when no bug exists. We set accuracy thresholds of 'poor' (and 'strong') performance based on the 25th (and 75th) percentile, i.e., when no bug exists we aim to capture the 25% of users who experience the worst (and best) classification accuracy.

When a bug exists and performance of the primary model drops, the 'poorly performing' subpopulation swells and the 'strongly performing' subpopulation shrinks (note the shift in histogram in Figure 5 with and without the bug). That is, subpopulation size varies depending on the degree of impact of the bug. Table 8 gives the subpopulation sizes that resulted for the experiments we ran.

Of course, this is not a perfect manner for identifying which data contains the bug (although as the bug is unknown, no manner could be perfect a priori). The poorly performing subpopulation will always contain

	Accuracy
75th Percentile of Users	93.9%
25th Percentile of Users	88.2%

Table 7: User Accuracy Percentiles (Without Bug).

	Without Bug	Bug in 50% of user devices
High-Accuracy Users	850	425
Low-Accuracy Users	850	2125

Table 8: Subpopulation sizes for simulations in Section 6. Total population size is 3,400 user devices.

the 25% of the overall population that doesn't classify well for reasons unrelated to any bug (e.g., users with irregular handwriting, etc.), and their data will be training the discriminator along with the data actually affected by the bug. The assumption is that either the generative model has the capacity to model both modes of data, or the unaffected data will be dwarfed by the affected data and the generative model will focus on the larger mode of data. This assumption held for this experiment, but further research on the 'sensitivity' of selection criteria is necessary (see Section 7).

We plan on investigating different selection criteria in future work.

C.3 Federated GAN Architecture and Training

Models and Losses Our generator and discriminator network architectures are borrowed from a popular GAN code tutorial[7] for MNIST. For the generator and discriminator loss functions, $l_G()$ and $l_D()$, we used the Improved Wasserstein loss formulation presented in Gulrajani et al. (2017). We used a gradient penalty coefficient of 10.0.

Hyperparameters Table 9 gives the various hyperparameters used with the DP-FedAvg-GAN algorithm to yield a generator that produced the images displayed in Figures 3 and 6.

[7] https://github.com/tensorflow/gan

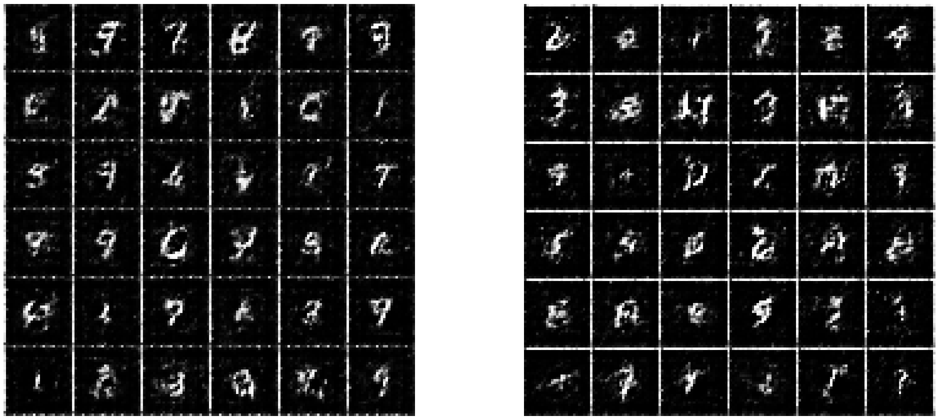
generator training steps n	generator batch size B	generator learning rate η_G	generator input size n_U
6	32	0.005	128
discriminator training steps[8] n	discriminator batch size[8] B	discriminator learning rate η_D	L_2 clip parameter S
≤ 6	≤ 32	0.0005	0.1
participation count qN	total users N	noise scale z	number of rounds T
10	*see Table 8*	0.01	$1,000$

Table 9: DP Federated GAN Experimental Hyperparameters.

Network Update Cadence For simplicity, Algorithm 1 shows a 1:1 ratio of discriminator:generator training updates. This is not required. The discriminator could go through multiple federated updates before an update to the generator is calculated, just as is the case with non-federated GANsGoodfellow et al. (2014). In these experiments we used a 1:1 ratio of discriminator:generator training updates, as this update cadence empirically was determined to work well.

C.4 Additional Generated Results

Figure 6 shows the output of two DP Federated GANs trained on the worst and best-classifying devices, once the bug described in Section 6 is fixed.



(a) Trained on high-accuracy users.

(b) Trained on low-accuracy users.

Figure 6: DP Federated GAN generator output after the inversion bug is fixed.

[8] Each user in the Federated EMNIST set has a different number of examples. Some users have less than 6 × 32 examples total. When such a user is selected for a round by the coordinating server, the discriminator update will use all of its examples to train, but the batch size of the final batch could be less than 32 and the number of steps taken could be less than 6.

D Open Problems

The intent of this paper is to highlight the gaps in a modeler's data inspection toolbox when undertaking private, decentralized learning, and to present what appears to be a promising solution direction involving generative models. To say that open work remains is an understatement, and there are many interesting topics to consider in this new space.

Alternative Notions of Privacy In this paper we implemented user-level privacy via the mechanism of (ϵ, δ)-differential privacy (DP), and considered acceptable privacy to be achieved via single-digit bounds on ϵ. Other useful notions of privacy exist, e.g. Carlini et al. (2018) defines a concept called *exposure*, a more empirical measure of data disclosure that takes into account the network training process. Research like Carlini et al. (2018) and Jayaraman & Evans (2019) shows that for many problems a large gap exists between empirical measures and DP's measure of privacy risk. While DP is tighter, empirical concepts like exposure may prove to be more practically useful for a given application; even if the ϵ bound is larger than single-digits, *exposure* can indicate that data memorization is not taking place and privacy protected. Exploring how alternative concepts of privacy affect the utility of federated generative models could improve their usefulness. (This topic connects with the FedGP work in Triastcyn & Faltings (2019) exploring differential *average-case* privacy for federated GAN training.)

Sensitivity The GAN experiment in 6 demonstrated debugging when 50% of user devices have been affected. The RNN experiment in 5 demonstrated debugging when 10% of examples have been affected. In Appendix B we present RNN results with differing percentages of affected examples, but in general a question remains: how sensitive to the presence of a bug are such federated generative models? Is there a minimum percentage of users or examples that's required in a composite data distribution such that the model learns to generate examples from that part of the distribution? This question is of particular interest in the case of GANs, which are known to suffer from problems of mode collapse (Salimans et al., 2016).

Federated GAN Training We chose GANs for this paper as they had not (to our knowledge) previously been trained with FL and DP, and developing an algorithm to do so was an open question. Apart from the DP-FedAvg-GAN algorithm we present, the GAN used is relatively simplistic (see details in Appendix C.3). Exploring the space of GAN techniques to study which losses and model architectures are most amenable to privatization (e.g., scaling and noising for DP) and federation would be immensely useful.

Other Classes of Generative Models Other types of generative model exist, with interesting questions on how to federate them. For example, Variational Autoencoders (VAEs) involve an encoder mapping high-dimensional data examples to a lower-dimensional latent space, and a decoder mapping from latent space back to high-dimensional space. With decentralized user-private data, is a useful paradigm to have each user have a personalized, private encoder (specific to the particulars of their local data and how it maps to the latent space), with a common decoder serving as synthesis engine mapping latent space to examples?

Federating Contrastive or Triplet Losses A class of ML losses involve distinguishing examples (or users) from each other. An example is the triplet loss used for FaceNet in Schroff et al. (2015). The loss is the sum of a reward based on similarity to a positive example and a penalty based on similarity to a negative example. For such loss calculations, a compute node requires access to both positive and negative examples.

FL protects not only the privacy of users from the coordinating server, but also the privacy of users from each other; each user's data resides in a distinct silo. Computing a triplet loss on a user's device requires access to negative examples of data (i.e., from other users), which is not directly possible. DP federated generative models offer a potential solution. If one can be trained to embody a user population, it can then be shipped to users' devices to serve as an engine providing synthetic negative examples for the triplet loss calculation.