Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

A Study of Privacy Preservation in Machine Learning Systems

A Thesis

written by

Mohamed Suliman

under the supervision of **Prof. Douglas Leith**, and submitted to the
School of Computer Science and Statistics
in fulfilment of the requirements for the degree of

Doctor of Philosophy

at *Trinity College Dublin, the University of Dublin*

January 2026

Declaration

I, the undersigned, declare that this work has not previously been submitted as an exercise for a degree at this, or any other University, and that unless otherwise stated, is my own work. I, the undersigned, agree to deposit this thesis in the University's open access institutional repository or allow the library to do so on my behalf, subject to Irish Copyright Legislation and Trinity College Library conditions of use and acknowledgement. I consent to the examiner retaining a copy of the thesis beyond the examining period, should they so wish (EU GDPR May 2018).

Signature

Date

Acknowledgements

I would firstly like to thank my supervisor, Douglas Leith, for his constant support during these last 4 years. I am indebted to him for his guidance, both technical and personal, throughout my time under his supervision.

I would also like to thank my colleagues in the security and privacy team at IBM Research, and especially thank Anisa Halimi for voluntarily taking on the role as my de-facto mentor at IBM. I am incredibly fortunate to have had the pleasure of her mentorship and friendship. A further thank you to Swanand Kadhe and Nathalie Baracaldo for their insightful exchanges during our numerous early morning and late evening video calls.

A big thank you to Vaibhav, David, Dilina, and Sulthana for making the lab the best place to work and for their amazing friendship.

Finally, to my friends and family, and especially to my parents, for their unconditional love and support, without which this thesis would not have been possible.

Abstract

As machine learning systems become more popular, and the quantity and scope of the data used to train them increases, questions regarding individual privacy naturally follow. This thesis is presented as an investigation into two methods of privacy preservation in machine learning: federated learning (FL) and computational proof based methods of model provenance. Our study of FL involves developing attacks that break its promise of private machine learning, using as a case study one of the largest real world deployments of federated learning, Google's Gboard. We develop methods that can recover private user data and highlight the ineffectiveness of current defense strategies. We ultimately conclude that, using Gboard as a case study, that the privacy benefit of FL requires re-evaluation. By considering the wider system implementation of which FL is a part of, we trace out aspects that can be exploited by parties whose honest participation is required for FL to be truly private. We then devote further research into other methods of privacy preservation, focusing heavily on model provenance. Here we address recent literature that has challenged the integrity of proof of learning (PoL), a computational proof-based method of certifying a model's training trajectory and data. Data forging attacks are recent developments against computational proofs of model provenance, and we scrutinise their performance and find that they are detectable in practice under informed verifiers. In addition, we provide theoretical evidence to the computational infeasibility of developing data forging attacks that are not detectable, and ultimately affirm the use of computational proofs for model provenance.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The remarkable growth in performance of machine learning systems in recent years is in large part due to access to massive quantities of data. With this access comes questions regarding individual privacy. Can these systems exacerbate the risk of a privacy violation? Are current methods of privacy-preserving machine learning sufficient, and if not, what paths forward are there?

Before we attempt to answer these questions, a discussion on what exactly is privacy and what constitutes a privacy violation is needed. Here we defer to Nissenbaum [55], who developed the idea that privacy can be regarded as the honouring of data's *contextual integrity*. Loosely, this means that a piece of information, or a *secret*, remains "private" if it does not leave its intended recipients e.g., a therapist informing their patient's place of work of the existence or contents of their meetings, or both. This constitutes a privacy violation as the patient's *secret*, that they visit a psychotherapist, has been disclosed to parties outside the intended recipient. Importantly, violations can result in demonstrable harm to individual e.g., here the patient may lose their job if their employer finds out about their sessions.

We now return to the questions regarding privacy posed initially. Machine learning systems *in the wild* consist not only of a model but also include data collection, training, and inference pipelines that in their totality form an application with some stated goal. A privacy violation may occur at any particular stage during the application's lifetime, and we may recall examples for each stage from recent history. Regarding data collection, the political consulting group Cambridge Analytica used illegitimately obtained facebook data to train models and deliver targeted content to inflame online discourse surrounding the 2016 U.S presidential election and the Brexit referendum [13, 35]. By arranging a facebook survey under the pretense of academic research, they exploited facebook's Graph API to collect not just respondant personal data but also the data of their facebook friends who had not consented.

Additionally, machine learning models themselves can lead to privacy violations,

having been shown to *leak* their training data to varying degrees [34, 18, 15, 16, 17, 67, 93], ranging from simply revealing whether a particular record was used or not, and in some cases, allowing for its near exact reconstruction. The data subjects often have no control over how these models are disseminated in the real world, increasing the potential risk of their data being disclosed to someone outside its intended recipients. Finally, the larger systems that these models are part of, while their stated goals are may appear benign, can be abused to violate individual privacy i.e., using the general purpose reasoning capabalities of large language models (LLMs) to infer sensitive attributes from about individuals. For example, Staab et al. [69] show how LLMs can be used to reveal a user's location by prompting these models to analyse their public social media activity. Clearly therefore, solutions are needed to mitigate the real danger presented by these systems to individual privacy.

This thesis consists of two parts: the first presents a critical investigation into one of the proposed solutions for privacy preserving machine learning, that is, Federated Learning (FL). Introduced to minimise data sharing and prevent privacy violations during the data collection and training stages of machine learning systems, FL allows for the collaborative training of a machine learning model while keeping private data on-premises. In FL, only the *updates* to the model are shared among participants. Our investigation is centered around Google's GBoard, one of the largest real world deployments of FL, training a language model on user keyboard data e.g., typed web searches or text messages. Our work firstly exposes a major privacy risk within this FL deployment; namely by developing an attack that can recover the words and sentences typed by a user from the updates that are shared during FL. Defences to our attack include obfuscating the updates via random added noise, however a fundamental tradeoff exists between maintaining user privacy and training a useful model. Secondly, using GBoard as a case study, we highlight the large amounts of trust in third parties required for FL to be truly private, motivating a re-evaluation of the privacy benefit of federated learning. We ultimately conclude that FL, as a privacy-preserving technology in its current instantiation, falls short on delivering upon its promise of private machine learning. As such, we must devote further research into the reduction of trust in FL, or to look elsewhere.

The second part of this thesis looks elsewhere, concerning itself with model provenance, a concept directly linked to data privacy. Put simply, model provenance relates to how one may go about certifying that a machine learning model was trained with a given dataset. Model provenance is an important tool both for those conducting training and the subjects of their models' training data, allowing the former to comply with legally mandated data audits and the latter to be aware of which models are trained using their data, anticipating potential privacy disclosures.

In this part, we tackle methods that challenge the integrity of Proof of Learning

(PoL) [39], a proposed method of model provenance. PoL allows a 3rd party verifier to certify a model's training data by producing a computational proof to that effect. Recently, *data forging attacks* have emerged as a threat to computational proof based methods of data certification. These type of attacks claim to produce contradictory but computationally valid proofs that a given model can result from training on *distinct* datasets. This appears to jeopardise any attempt at data governance as privacy worrisome models may be adversarily allowed to falsely demonstrate compliance by counterfactually "proving" its training dataset consisted of only legally compliant data. In order to reconcile these apparent contradictions and address the threat to governance that data forging attacks impose, we take a critical view of these attacks, scrutinising their performance. Ultimately, we find that they are not, in their current instantiation, a real threat to governance. Their apparent danger is pacified and contradictions are resolved upon closer analysis. Further we provide compelling theoretical evidence that stronger data forging attacks that successfully realise their implications are computationally infeasible. Our results support PoL's concept of computational proof-based methods for model provenance, paving a potential path forward for data privacy when training machine learning models.

1.1 Contributions

This thesis consists of three chapters whose contributions are summarised and related publications are also cited.

Chapter 2 - Two Models are Better than One: Federated Learning Is Not Private For Google GBoard Next Word Prediction

We illustrate the effectiveness of the attacks against the next word prediction model used in Google's GBoard app, a widely used mobile keyboard app that has been an early adopter of federated learning for production use. We demonstrate that the words a user types on their mobile handset, e.g. when sending text messages, can be recovered with high accuracy under a wide range of conditions and that countermeasures such a use of mini-batches and adding local noise are ineffective. We also show that the word order (and so the actual sentences typed) can be reconstructed with high fidelity. This raises obvious privacy concerns, particularly since GBoard is in production use.

This chapter is based on the following publications:

Suliman, M., & Leith, D. J. (2022, April). Gradient Information From Google GBoard NWP LSTM Is Sufficient to Reconstruct Words Typed. In *2022 Cyber Research Conference-Ireland (Cyber-RCI)* (pp. 1-4). IEEE.

Suliman, M., & Leith, D. (2023, September). Two models are better than one: Federated learning is not private for google gboard next word prediction. In *Euro-*

pean Symposium on Research in Computer Security (pp. 105-122). Cham: Springer Nature Switzerland.

Chapter 3 - Re-evaluating the Privacy Benefit of Federated Learning

Federated Learning's (FL) main attractive privacy feature is data localisation; that is, its ability to train a model on private user data that does not require centralised collection by a third party. This, however, has been shown to not be guaranteed, under both honest-but-curious and actively malicious threat models in a growing body of attacks against FL. We observe that FL participants must place a disproportionate level of trust in the server that controls the model architecture, training algorithm, client participation, encryption, etc, giving rise to the question of what the added privacy benefit of FL really is compared to conventional approaches that also require trust. To illustrate the issues of concern, we use Google's Gboard as a case study. We develop attacks that can recover the words and sentences typed by its users even under Secure Aggregation within the constrictive honest-but-curious threat model, and expose system-level vulnerabilities that can de-anonymise FL participants. This highlights the often overlooked gap between theoretical privacy analysis and concrete implementations. Taking all this together, we argue FL and its added privacy benefit require a re-think.

This chapter is based on the following publication:

Suliman, M., Leith, D., & Halimi, A. (2023, September). Re-evaluating the Privacy Benefit of Federated Learning. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases* (pp. 470-477). Cham: Springer Nature Switzerland.

Author Note: Douglas Leith provided telemetry measurements of the Gboard application.

Chapter 4 - Towards a Re-evaluation of Data Forging Attacks in Practice

Data forging attacks provide counterfactual proof that a model was trained on a given dataset, when in fact, it was trained on another. These attacks work by forging (replacing) mini-batches with ones containing distinct training examples that produce nearly identical gradients. Data forging appears to break any potential avenues for data governance, as adversarial model owners may forge their training set from a dataset that is not compliant to one that is. Given these serious implications on data auditing and compliance, we critically analyse data forging from both a practical and theoretical point of view, finding that a key practical limitation of current attack methods makes them easily detectable by a verifier; namely that they cannot produce sufficiently identical gradients. Theoretically, we analyse the question of whether two distinct mini-batches can produce the same gradient. Generally, we find that while there may exist an infinite number of distinct mini-batches with real-valued training examples and labels that produce the same gradient, finding

those that are within the allowed domain e.g. pixel values between 0-255 and one hot labels is a non trivial task. Our results call for the reevaluation of the strength of existing attacks, and for additional research into successful data forging, given the serious consequences it may have on machine learning and privacy.

This chapter is based on the following publications:

Suliman, M., Kadhe, S., Halimi, A., Leith, D., Baracaldo, N., & Rawat, A. (2024, May). Data forging is harder than you think. In *Privacy Regulation and Protection in Machine Learning at ICLR.*

Suliman, M., Halimi, A., Kadhe, S., Baracaldo, N., & Leith, D. (2025). Towards a Re-evaluation of Data Forging Attacks in Practice. In *34th USENIX security symposium (USENIX Security 25)*

Chapter 2

Two Models are Better than One: Federated Learning Is Not Private For Google GBoard Next Word Prediction

2.1 Introduction

Federated Learning (FL) is a class of distributed algorithms for the training of machine learning models such as neural networks. A primary aim of FL when it was first introduced was to enhance user privacy, namely by keeping sensitive data stored locally and avoiding uploading it to a central server [51]. The basic idea is that users train a local version of the model with their own data, and share only the resulting model parameters with a central coordinating server. This server then combines the models of all the participants, transmits the aggregate back to them, and this cycle (i.e. a single FL 'round') repeats until the model is judged to have converged. A notable real-world deployment of FL is within Google's Gboard, a widely used Android keyboard application that comes pre-installed on many mobile handsets and which has $>$ 5 Billion downloads [29]. Within GBoard, FL is used to train the Next Word Prediction (NWP) model that provides the suggested next words that appear above the keyboard while typing [36].

In this chapter, we show that FL is not private for next word prediction. We present an attack that reconstructs the original training data, i.e. the text typed by a user, from the FL parameter updates with a high degree of fidelity. Both the FedSGD and FederatedAveraging variants of FL are susceptible to this attack. In fairness, Google have been aware of the possibility of information leakage from FL updates since the earliest days of FL, e.g. see footnote 1 in [51]. Our results demonstrate that

not only does information leakage indeed happen for real-world models deployed in production and in widespread use, but that the amount of information leaked is enough to allow the local training data to be fully reconstructed.

We also show that adding Gaussian noise to the transmitted updates, which has been proposed to ensure local Differential Privacy (DP), provides little defence unless the noise levels used are so large that the utility of the model becomes substantially degraded. That is, DP is not an effective countermeasure to our attack[1]. We also show that use of mini-batches of up to 256 sentences provides little protection. Other defences, such as Secure Aggregation (a form of multi-party computation MPC), Homomorphic Encryption (HE), and Trusted Execution Environments (TEEs), are currently either impractical or require the client to trust that the server is honest[2] in which case use of FL is redundant.

Previous studies of reconstruction attacks against FL have mainly focused on image reconstruction, rather than text as considered here. Unfortunately, we find that the methods developed for image reconstruction, which are based on used gradient descent to minimise the distance between the observed model data and the model data corresponding to a synthetic input, do not readily transfer over to text data. This is perhaps unsurprising since the inherently discrete nature of text makes the cost surface highly non-smooth and so gradient-based optimisation is difficult to apply successfully. In this paper we therefore propose new reconstruction approaches that are specialised to text data.

It is important to note that the transmission of user data to a remote server is not inherently a breach of privacy. The risk of a privacy breach is related to the nature of the data being sent, as well as whether it's owner can be readily identified. For example, sending device models, version numbers, and locale/region information is not an immediate concern but it seems clear that the sentences entered by users, e.g. when typing messages, writing notes and emails, web browsing and performing searches, may well be private. Indeed, it is not only the sentences typed which can be sensitive but also the set of words used (i.e. even without knowing the word ordering) since this can be used for targeting surveillance via keyword blacklists [6].

In addition, most Google telemetry is tagged with with an Android ID. Via other data collected by Google Play Services the Android ID is linked to (i) the handset hardware serial number, (ii) the SIM IMEI (which uniquely identifies the SIM slot)

[1]DP aims to protect the aggregate training data/model against query-based attacks, whereas our attack targets the individual updates. Nevertheless, we note that DP is sometimes suggested as a potential defence against the type of attack carried out here.

[2]Google's Secure Aggregation approach [10] is a prominent example of an approach requiring trust in the server, or more specifically in the PKI infrastructure which in practice is operated by the same organisation that runs the FL server since it involves authentication/verification of clients. We note also that Secure Aggregation is not currently deployed in the GBoard app despite being proposed 6 years ago.

and (iii) the user's Google account [48, 47]. When creating a Google account users are encouraged to supply a phone number and for many people this will be their own phone number. Use of Google services such as buying a paid app on the Google Play store or using Google Pay further links a person's Google account to their credit card/bank details. A user's Google account, and so the Android ID, can therefore commonly be expected to be linked to the person's real identity.

2.2 Background

2.2.1 Federated Learning Client Update

Algorithm 1 gives the procedure followed by FL participants to generate a model update. The number of local epochs E, mini-batch size B, and local client learning rate η can be changed depending on the FL application. When $E = 1$ and the mini-batch size is equal the size of the training dataset, then it is called FedSGD, and any other configuration corresponds to FedAveraging, where multiple gradient descent steps occur on the client.

Algorithm 1 Federated Learning Client Update

Input: Parameters of the global model θ_0, loss function $\mathcal{L}$, dataset D
Output: Parameters after training θ_1

$\theta_1 \leftarrow \theta_0$
$\mathcal{B} \leftarrow$ (split the dataset into batches of size b)
for local epoch i from 1 to E **do**
  for batch $B \in \mathcal{B}$ **do**
    $\theta_1 \leftarrow \theta_1 - \eta \nabla_\theta \mathcal{L}(B; \theta_1)$
  end for
end for

2.2.2 Threat Model

The threat model is that of a honest-but-curious adversary that has access to (i) the FL model architecture, (ii) the current global FL model parameters θ_0, and (iii) the FL model parameters, θ_1, after updating locally using the training data of an individual user. The FL server has, for example, access to all of these and so this threat model captures the situation where there is an honest-but-curious FL server.

We do not consider membership attacks against the global model, although knowledge of θ_0 allows such attacks, since they have already received much attention in the literature. Instead we focus on local reconstruction attacks i.e. attacks that aim to reconstruct the local training data of a user from knowledge of θ_0 and θ_1. In

the GBoard Next Word Prediction task the local training data is the text typed by the user while using apps on their mobile handset e.g. while sending text messages.

2.2.3 GBoard NWP Model

Figure 2.1 shows the LSTM architecture used by by Gboard for NWP. The Gboard LSTM RNN is a word level language model, predicting the probability of the next word given what the user has already typed into the keyboard. Input words are first mapped to a dictionary entry, which has a vocabulary of $V = 9502$ words, with a special `<UNK>` entry used for words that are not in the dictionary, and `<S>` to indicate the start of a sentence. The index of the dictionary entry is then mapped to a dense vector of size $D = 96$ using a lookup table (the dictionary entry is one-hot encoded and then multiplied by a $\mathbb{R}^{D \times V}$ weighting matrix W^T) and applied as input to an LSTM layer with 670 units i.e. the state C_t is a vector of size 670. The LSTM layer uses a CIFG architecture without peephole connections, illustrated schematically in Figure 2.1. The LSTM state C_t is linearly projected down to an output vector h_t of size D, which is mapped to a raw logit vector z_t of size V via a weighting matrix W and bias b. This extra linear projection is not part of the orthodox CIFG cell structure that was introduced in [31], and is included to accommodate the model's tied input and output embedding matrices [63]. A softmax output layer finally maps this to an $[0, 1]^V$ vector $\hat{y}_t$ of probabilities, the i'th element being the estimated probability that the next word is the i'th dictionary entry.

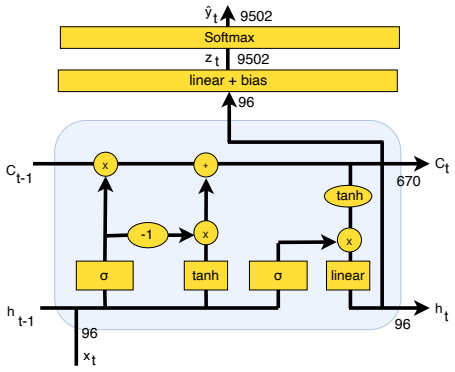


Figure 2.1: Schematic of LSTM architecture. LSTM layer takes as input dense vector x_t representing a typed word and outputs a dense vector h_t. This output is then mapped to vector z_t of size 9502 (the size of the dictionary) with the value of each element being the raw logit for the corresponding dictionary word. A softmax layer then normalises the raw z_t values to give a vector $\hat{y}_t$ of probabilities.

2.3 Reconstruction Attack

2.3.1 Word Recovery

In next word prediction the input to the RNN is echoed in it's output. That is, the output of the RNN aims to match the sequence of words typed by the user, albeit with a shift one word ahead. The sign of the output loss gradient directly reveals information about the words typed by the user, which can then be recovered easily by inspection. This key observation, first made in [88], is the basis of our word recovery attack.

Consider the training example $\hat{\mathbf{x}} = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_T)$ where each $\mathbf{x}_t$ is a one hot vector that encodes the token at timestep t i.e $[\mathbf{x}_t]_i = 1$ where i is the typed token, and elsewhere is 0. The cross entropy loss at timestep t is given by

$$\begin{aligned} \ell_t(\hat{\mathbf{x}}) &= -\sum_{i=1}^{V} [\mathbf{x}_{t+1}]_i \cdot \log([\hat{\mathbf{y}}_t]_i) \\ &= -\log[\hat{\mathbf{y}}]_{c_{t+1}}, \end{aligned} \tag{2.1}$$

where c_{t+1} is the index of the next word, and $\hat{\mathbf{y}}_t = \text{softmax}(\mathbf{z}_t) = \text{softmax}(\mathbf{W}^T\mathbf{h}_t + \mathbf{b})$. Here, $\mathbf{W} \in \mathbb{R}^{D \times V}$ and $\mathbf{b} \in \mathbb{R}^V$ represents the weight and bias parameters of the final layer of the model, and $\mathbf{h}_t$ is the output of the LSTM layer at timestep t.

The full loss for the example $\hat{\mathbf{x}}$ consisting of T timesteps is $\ell_{1:T}(\hat{\mathbf{x}}) = \sum_{t=1}^{T-1} \ell_t(\hat{\mathbf{x}})$. Differentiating with respect to a single component of the final layer bias $\mathbf{b}_i$ we have that,

$$\begin{aligned} \nabla \mathbf{b}_i := \frac{\partial \ell_{1:T}(\hat{\mathbf{x}})}{\partial \mathbf{b}_i} &= \sum_{t=1}^{T-1} \frac{\partial \ell_t(\hat{\mathbf{x}})}{\partial \mathbf{b}_i} \\ &= \sum_{t=1}^{T-1} \sum_{j=1}^{V} -[\mathbf{x}_{t+1}]_j \cdot \frac{\partial \log([\hat{\mathbf{y}}_t]_j)}{\partial \mathbf{b}_i} \\ &= \sum_{t=1}^{T-1} \sum_{j=1}^{V} -[\mathbf{x}_{t+1}]_j \cdot \frac{\partial \log([\hat{\mathbf{y}}_t]_j)}{\partial [\mathbf{z}_t]_i} \cdot \frac{\partial [\mathbf{z}_t]_i}{\partial \mathbf{b}_i} \\ &= \sum_{t=1}^{T-1} \sum_{j=1}^{V} -[\mathbf{x}_{t+1}]_j \cdot \frac{\partial \log([\hat{\mathbf{y}}_t]_j)}{\partial [\mathbf{z}_t]_i} \\ &= \sum_{t=1}^{T-1} \sum_{j=1}^{V} -\frac{[\mathbf{x}_{t+1}]_j}{[\hat{\mathbf{y}}_t]_j} \cdot \frac{\partial [\hat{\mathbf{y}}_t]_j}{\partial [\mathbf{z}_t]_i} \end{aligned} \tag{2.2}$$

We know that from the derivative of the softmax function,

$$\frac{\partial[\hat{\mathbf{y}}_t]_j}{\partial[\mathbf{z}_t]_i} = \begin{cases} [\hat{\mathbf{y}}_t]_j(1-[\hat{\mathbf{y}}_t]_j) & \text{if } i=j \\ -[\hat{\mathbf{y}}_t]_j \cdot [\hat{\mathbf{y}}_t]_i & \text{otherwise,} \end{cases} \tag{2.3}$$

therefore,

$$\begin{aligned} \frac{\partial \ell_{1:T}(\hat{\mathbf{x}})}{\partial \mathbf{b}_i} &= \sum_{t=1}^{T-1} -[\mathbf{x}_{t+1}]_i(1-[\hat{\mathbf{y}}_t]_i) - \sum_{\substack{j=1 \\ j\neq i}}^{V} [\mathbf{x}_{t+1}]_j \cdot [\hat{\mathbf{y}}_t]_j \\ &= \sum_{t=1}^{T-1} -[\mathbf{x}_{t+1}]_i + \Big(\sum_{j=1}^{V} [\mathbf{x}_{t+1}]_j\Big) \cdot [\hat{\mathbf{y}}_t]_i \\ &= \sum_{t=1}^{T-1} [\hat{\mathbf{y}}_t]_i - [\mathbf{x}_{t+1}]_i. \end{aligned} \tag{2.4}$$

From (2.4), we have that $\frac{\partial \ell_{1:T}(\hat{\mathbf{x}})}{\partial \mathbf{b}} = \sum_{t=1}^{T-1} \hat{\mathbf{y}}_t - \mathbf{x}_{t+1}$. It follows that for tokens k which do not appear in the training example $\hat{\mathbf{x}}$ i.e., at no timestep is $[\mathbf{x}_t]_k = 1$, $\frac{\partial \ell_{1:T}}{\partial \mathbf{b}_k} > 0$. Therefore, if $\frac{\partial \ell_{1:T}}{\partial \mathbf{b}_k} < 0$, then the token k was typed. Also, assuming that the neural net has been trained to have reasonable performance then $[\hat{\mathbf{y}}_t]_k$ will tend to be small for words k that do not appear next and large for words which do. Therefore for words k^* that appear in the text we expect that $\frac{\partial \ell_{1:T}}{\partial \mathbf{b}_{k^*}} < 0$. The above analysis focuses mainly on the bias parameters of the final fully connected layer, however similar methods can be applied to the W parameters [88]. The key aspect here that lends to the ease of this attack is that the outputs echo the inputs, unlike for example, the task of object detection in images. In that case, the output is just the object label.

Example: To execute this attack in practice, simply subtract the final layer parameters of the current global model θ_0 from those of the resulting model trained on the client's local data, θ_1, as shown in Algorithm 2. The indices of the negative values reveal the typed words. Suppose the client's local data consists of just the one sentence "learning online is not so private". We then train model θ_0 on this sentence for 1 epoch, with a mini-batch size of 1, and SGD learning rate of 0.001 (FedSGD), and report the the values at the negative indices in Table 2.1.

Algorithm 2 Word Recovery

Input: Final layer bias parameters of the global model $\mathbf{b}_0$, Final layer bias parameters of the client update $\mathbf{b}_1$

Output: Set of typed tokens S

1: $\mathbf{d} \leftarrow \mathbf{b}_1 - \mathbf{b}_0$

2: $W \leftarrow \{i \mid \mathbf{d}_i < 0\}$

word	i	$(\theta_1 - \theta_0)_i$
learning	7437	-0.0009951561
online	4904	-0.0009941629
is	209	-0.000997875
not	1808	-0.0009941144
so	26	-0.0009965639
private	6314	-0.0009951561

Table 2.1: Values of the final layer parameter difference at the indices of the typed words. Produced after training the model on the sentence "learning online is not so private", $E = 1, B = 1, \eta = 0.001$. These are the only indices where negative values occur.

2.3.2 Reconstructing Sentences

The attack described previously retrieves the words typed, but gives no indication of the order in which they occurred. To reveal this order, we *ask* the model by greedily decoding its output[3].

The basic idea is that after running multiple rounds of gradient descent on the local training data, the local model is "tuned" to the local data in the sense that when it is presented with the first words of a sentence from the local training data, the model's next word prediction will tend to match the training data and so we can bootstrap reconstruction of the full training data text. In more detail, the t'th input word is represented by a vector $\mathbf{x}_t \in \{0,1\}^V$, with all elements zero apart from the element corresponding to the index of the word in the dictionary. The output $\hat{\mathbf{y}}_{t+1} \in [0,1]^V$ from the model after seeing input words $\mathbf{x}_1, \ldots, \mathbf{x}_{T-1}$ is a probability distribution over the dictionary. We begin by selecting $\mathbf{x}_1$ equal to the start of sentence token `<S>` and $\mathbf{x}_2$ equal to the one of the words from our set of reconstructed words, then ask the model to generate $\hat{\mathbf{y}}_2 = Pr(\mathbf{x}_3|\mathbf{x}_1, \mathbf{x}_2; \theta_1)$. We set all elements of $\hat{\mathbf{y}}_2$ that are not in the set of reconstructed words to zero, since we know that these were not part of the local training data, re-normalise $\hat{\mathbf{y}}_2$ so that its elements sum to one, and then select the most likely next word as $\mathbf{x}_3 := \arg\max_i [\hat{\mathbf{y}}_2]_i$. We now repeat this process for $\hat{\mathbf{y}}_3 = Pr(\mathbf{x}_4|\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3; \theta_1)$, and so on, until a complete sentence has been generated.

This method generates as many sentences as there are extracted words. This results in a lot more sentences than were originally in the client's training dataset. In order to filter out the unnecessary sentences, we rank each generated sentence by its change in perplexity, from the initial global model θ_0 to the new update θ_1. The

[3]It is perhaps worth noting that we studied a variety of reconstruction attacks, e.g., using Monte Carlo Tree Search to perform a smart search over all words sequences, but found the attack method described here to be simple, efficient and highly effective

Log-Perplexity of a sequence $\mathbf{x}_1, \ldots, \mathbf{x}_T$, is defined as

$$PP_\theta(\mathbf{x}_1, \ldots, \mathbf{x}_T) = \sum_{i=1}^{T-1} -\log Pr(\mathbf{x}_{i+1}|\mathbf{x}_1, ..., \mathbf{x}_i; \theta),$$

and quantifies how 'surprised' the model is by the sequence. Those sentences that report a high perplexity for θ_0 but a comparatively lower one for θ_1 reveal themselves as having been part of the dataset used to train θ_1. Each generated sentence is scored by their percentage change in perplexity:

$$Score(\mathbf{x}_1, \ldots, \mathbf{x}_T) = \frac{PP_{\theta_0}(\mathbf{x}_1, \ldots, \mathbf{x}_T) - PP_{\theta_1}(\mathbf{x}_1, \ldots, \mathbf{x}_T)}{PP_{\theta_0}(\mathbf{x}_1, \ldots, \mathbf{x}_T)}.$$

By selecting the top-n ranked sentences, we select those most likely to have been present in the training dataset.

2.4 Performance Of Attacks Against Vanilla Federated Learning

2.4.1 Experimental Setup

We make use of the LSTM RNN extracted from Gboard as the basis of our experiments. The value of it's extracted parameters are used as the initial 'global' model θ_0; the starting point of the updates we generate. There are several variables that go into producing an update: the number of sentences in the dataset, n_k, the number of epochs E, the batch size B, and the local learning rate η. Note that when $E = 1$ and $B = n_k$, this corresponds to a FedSGD update, and that any other configuration corresponds to a FedAveraging update. Unless explicitly mentioned otherwise, we keep the client learning rate $\eta = 0.001$ constant for all our experiments. All sample datasets used consist of 4 word long sentences, mirroring the average length of sentences that the Gboard model was trained with [36].

To evaluate the effectiveness of our attack, the sample datasets we use are taken from a corpus of american english text messages [56], which includes short sentences similar to those the Gboard LSTM extracted from the mobile application was trained on. We perform our two attacks on datasets consisting of $n_k = 16, 32, 64, 128$, and 256 sentences.

We train model θ_0 on this training data for a specified number of epochs E with a mini-batch size of B, according to Algorithm 1 to produce the local update θ_1. We then subtract the final layer parameters of the two models to recover the words, and iteratively sample θ_1 according to the methodology described in Section 2.3.2 to

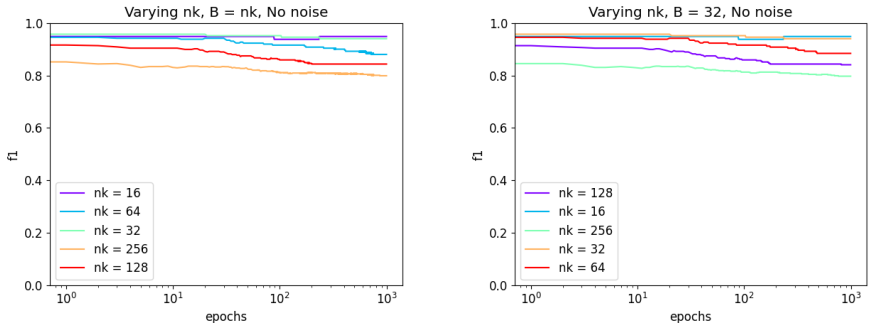
reconstruct the sentences, and take the top-n_k ranked sentences by their perplexity score.

2.4.2 Metrics

To evaluate performance, we use the F1 score which balances the precision and recall of word recovery with our attack. We also use a modified version of the Levenshtein ratio i.e. the normalised Levenshtein distance [50] (the minimum number of word level edits needed to make one string match another) to evaluate our sentence reconstruction attack. Ranging from 0 to 100, the larger the Levenshtein ratio, the closer the match between our reconstructed and the ground truth sentence.

2.4.3 Measurements

Word Recovery Performance. Figure 2.2 shows the measured performance of our word recovery attack for both the FedSGD and FedAveraging variants of FL for mini-batch/full batch training and as the dataset size and training time are varied. It can be seen that none of these variables have much of an effect on the F1 score achieved by our attack, which remains high across a wide range of conditions. Note that the maximum value of the F1 score is not 1 for this data but instead a smaller value related to how many of the words in the dataset are also present in the model's vocabulary. Some words, e.g. unique nouns, slang, etc, do not exist in the model's 9502 word dictionary, and our word recovery attack can only extract the `<UNK>` token in their place, limiting how many words we can actually recover.



(a) (b)

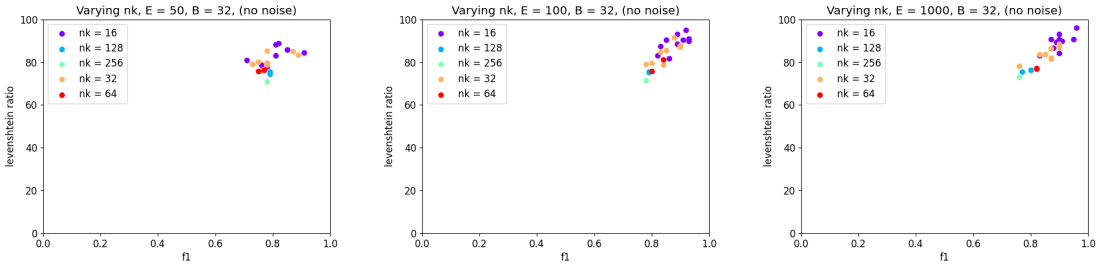
Figure 2.2: Word recovery performance over time with full batch and mini-batch training. The upper bound on the F1 score for the different datasets is related to how many words in the training set are also in the model dictionary. Figure 2.2a shows the F1 score over time with $B = n_k$ full batch training. At epoch 1 (FedSGD), the F1 is high and stays constant as you train for longer (FedAveraging). Using a mini-batch $B = 32$ (Figure 2.2b) has no effect on the attack (in the case where $n_k = 16, B = 16$).

Sentence Reconstruction Performance. Figure 2.3 shows the measured per-

formance of our sentence reconstruction attack. Figures 2.3a, 2.3b, and 2.3c show that as you train for more epochs (50, 100, and 1000 respectively) the quality of the reconstructed sentences improves. This is intuitive as the models trained for longer are more overfit to the data, and so the iterative sampling approach is more likely to return the correct next word given a conditioning prefix. However, longer training times are not necessary to accurately reconstruct sentences. It can be seen from Figure 2.4(b) that even in the FedSGD setting, where the number of epochs $E = 1$, we can sometimes still get high quality sentence reconstructions by modifying the model parameters θ_1 to be $\theta_1 + s(\theta_1 - \theta_0)$, with s being a scaling factor. Since $\theta_1 = \theta_0 - \eta\nabla\mathcal{L}(B;\theta_0)$ with FedSGD,

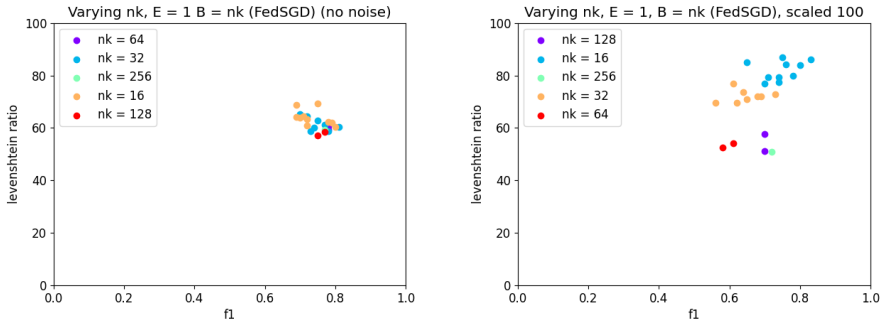
$$\theta_1 + s(\theta_1 - \theta_0) = \theta_0 - \eta(1+s)\nabla\mathcal{L}(B;\theta_0)$$

from which it can be seen that scaling factor s effectively increases the gradient descent step size.



(a) $E = 50$ epochs (b) $E = 100$ epochs (c) $E = 1000$ epochs

Figure 2.3: Sentence reconstruction performance. Each point corresponds to a different dataset colour coded by it's size. The y-axis gives the average Levenshtein ratio of the reconstructed sentences. The x-axis is the F1 score between the tokens used in the reconstructed sentences and the ground truth. The closer a point is to the top-right corner, the closer the reconstruction is to perfect.



(a) (b)

Figure 2.4: FedSGD sentence reconstruction performance without (a) and with (b) scaling.

2.5 Existing Attacks And Their Defences

2.5.1 Image Data Reconstruction

Information leakage from the gradients of neural networks used for object detection in images appears to have been initially investigated in [93], which proposed the Deep Leakage from Gradients (DLG) algorithm. An image is input to a neural net and the output is a label specifying an object detected in the image. In DLG, a synthetic input is applied to the neural net and the gradients of the model parameters are calculated. These gradients are then compared to the observed model gradients sent to the FL server and gradient descent is used to update the synthetic input so as to minimise the difference between its model gradients and the observed model gradients.

This work was subsequently extended by [88, 28, 77, 91, 85, 41] to improve the stability and performance of the original DLG algorithm, as well as the fidelity of the images it generates. In [85, 41], changes to the optimisation terms allowed for successful data reconstruction at batch sizes of up to 48 and 100 respectively. Analytical techniques of data extraction [8, 57] benefit from not being as costly to compute as compared to optimization based methods. Additionally, these analytical attacks extract the exact ground truth data, as compared to DLG and others who often settle to image reconstructions that include artefacts.

2.5.2 Text Data Reconstruction

Most work on reconstruction attacks has focused on images and there is relatively little work on text reconstruction. A single text reconstruction example is presented in [93], with no performance results. Probably the closest work to the present paper is [21] which applies a variant of DLG to text reconstruction from gradients of transformer-based models (variants of BERT [75]). As already noted, DLG tends to perform poorly with text data and the word recovery rate achieved in [21] is generally no more than 50%.

In our attack context, DLG can recover words but at much smaller scales than we have demonstrated, and takes longer to find these words. There is also no guarantee that DLG can recover words and place them in the correct order. In [93], it is noted that the algorithm requires multiple restarts before successful reconstruction. Additionally, DLG operates by matching the single gradient of a batch of training data, therefore it only works in the FedSGD setting, where $E = 1$. We show the results of DLG in Listing 2.1 on gradients of $B = 1$, and 2 4 word sentences. In the first example, it took DLG 1000 iterations to produce "`<S> how are venue`", and

1500 iterations to produce "`<S> how are sure cow`, `<S> haha where are Tell van`". These reconstructions include some of the original words, but recovery is not as precise as our attack, and takes orders of magnitude longer to carry out.

Listing 2.1: Original and reconstructed sentences by DLG

```
<S> how are you
<S> how are venue

<S> how are you doing
<S> how are sure cow
<S> where are you going
<S> haha where are Tell van
```

Work has also been carried out on membership attacks against text data models such as GPT2 i.e. given a trained model the attack seeks to infer one or more training data points. See for example [16, 17]. But, as already noted, such attacks are not the focus of the present paper.

2.5.3 Proposed Defences

Several defences have been proposed to prevent data leakage in FL.

Abadi et al. [2] proposed Differentially Private Stochastic Gradient Descent (DP-SGD), which clips stochastic gradient decent updates and adds Gaussian noise at each iteration. This aims to defend against membership attacks against neural networks, rather than the reconstruction attacks that we consider here. In [52] it was appled to train a next word prediction RNN motivated by mobile keyboard applications, again with a focus on membership attacks. Recently, the same team at Google proposed DP-FTRL [42] which avoids the sampling step in DP-SGD.

Secure Aggregation is a multi-party protocol proposed in 2016 by [10] as a defence against data leakage from the data uploaded by clients to an FL server. In this setting the central server only has access to the sum of, and not individual updates. However, this approach still requires clients to trust that the PKI infrastructure is honest since dishonest PKI infrastructure allows the server to perform a sybil attack (see Section 6.2 in [10]) to reveal the data sent by an individual client. When both the FL server and the PKI infrastructure are operated by Google then Secure Aggregation requires users to trust Google servers to be honest, and so offers from an attack capability point of view offers no security benefit. Recent work by Pasquini et al. [59] has also shown that by distributing different models to each client a dishonest server can recover individual model updates. As a mitigation they propose adding local noise to client updates to obtain a form of local differential privacy. We note that despite the early deployment of FL in production systems such as GBoard, to the best our knowledge, there does not exist a real-world deployment of secure aggregation. This is also true for homomorphically encrypted FL, and FL using

Trusted Execution Environments (TEEs).

2.6 Performance Of Our Attacks Against Federated Learning with Local DP

Typically, when differential privacy is used with FL noise is added by the server to the aggregate update from multiple clients i.e. no noise is added to the update before leaving a device. This corresponds to the situation considered in Section 2.4. In this section we now evaluate how local differential privacy, that is, noise added either during local training (DPSGD) or to the final model parameters θ_1, before its transmission to the coordinating FL server, affect the performance of both our word recovery and sentence reconstruction attacks.

Algorithm 3 Local DPSGD

Input: Parameters of the global model θ_0, loss function $\mathcal{L}$, dataset D
Output: Parameters after training θ_1

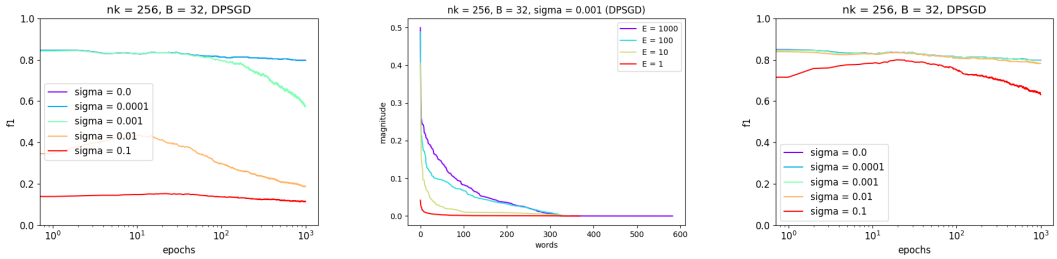
$\theta_1 \leftarrow \theta_0$
$\mathcal{B} \leftarrow$ (split the dataset into batches of size b)
for local epoch i from 1 to E **do**
  for batch $B \in \mathcal{B}$ **do**
    $\theta_1 \leftarrow \theta_1 - \eta(\nabla_\theta \mathcal{L}(B; \theta_1) + \mathcal{N}(0, \sigma))$
  end for
end for

Algorithm 3 outlines the procedure for DPSGD-like local training, where Gaussian noise of mean 0 and standard deviation σ is added along with each gradient update. Algorithm 4 details the typical FL client update procedure but adds Gaussian noise to the final model θ_1 before it is returned to the server. In our experiments, everything else as described in Section 2.4 is kept the same.

Algorithm 4 Local Single Noise Addition

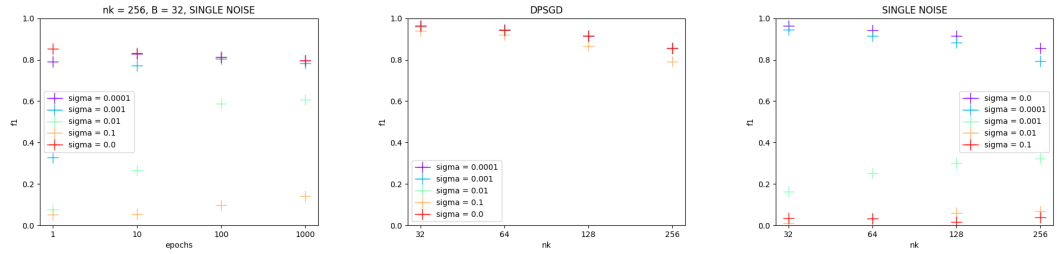
Input: Parameters of the global model θ_0, loss function $\mathcal{L}$, dataset D
Output: Parameters after training θ_1

$\theta_1 \leftarrow \theta_0$
$\mathcal{B} \leftarrow$ (split the dataset into batches of size b)
for local epoch i from 1 to E **do**
  for batch $B \in \mathcal{B}$ **do**
    $\theta_1 \leftarrow \theta_1 - \eta \nabla_\theta \mathcal{L}(B; \theta_1)$
  end for
end for
$\theta_1 \leftarrow \theta_1 + \mathcal{N}(0, \sigma)$



(a) (b) (c)

Figure 2.5: Word recovery behaviour when Gaussian noise is all to local FL updates: (a) vanilla word recovery performance, (b) disparity of magnitudes between those words that were present in the dataset and those 'noisily' flipped negative, (c) word recovery performance when filtering is used.



(a) (b) (c)

Figure 2.6: Word recovery results for two local DP methods. Figure 2.6a shows how added noise to the final model parameters affects the attack for different training times. With DPSGD-like training, noise levels of up to $\sigma = 0.1$ are manageable with our magnitude threshold trick, resulting in F1 scores close to those had we not added any noise at all. With FedSGD updates then for noise of up to $\sigma = 0.0001$ added to the final parameters, we can still recover words to a high degree of precision and recall.

2.6.1 Word Recovery Performance

Figure 2.5a shows the performance of our word recovery attack against DPSGD-like local training for different levels of σ. For noise levels of $\sigma = 0.001$ or greater, it can be seen that the F1 score drops significantly. What is happening is that the added noise introduces more negative values in the difference of the final layer parameters and so results in our attack extracting more words than actually occurred in the dataset, destroying its precision. However, one can eliminate most of these "noisily" added words by simple inspection. Figure 2.5b graphs the sorted magnitudes of the negative values in the difference between the final layer parameters of θ_1 and θ_0, after DPSGD-like training with $B = 32, n_k = 256,$ and $\sigma = 0.001$. We can see that the more epochs the model is trained for, the more words are extracted via our attack. Of the around 600 words extracted after 1000 epochs of training, only about 300 of them were actually present in the dataset. On this graph, those words with higher magnitudes correspond to the ground truth words. It can be seen that

we therefore can simply cutoff any words extracted beyond a specified magnitude threshold. This drastically improves the performance of the attack, see Figure 2.5c. It can be seen that even for $\sigma = 0.1$, we now get word recovery results close to those obtained when we added no noise at all.

Figure 2.6a shows the performance of our word recovery attack when noise is added to the local model parameters θ_1 in the FedAveraging setting, with $n_k = 256$, and $B = 32$. When $\sigma \geq 0.01$, it can be seen that the performance drops drastically[4], even when we use the magnitude threshold trick described previously. With FedSGD (Figures 2.6b and 2.6c), we see that with DPSGD-like training, these levels of noise are manageable, but for single noise addition, there are only so many words that are recoverable before being lost in the comparatively large amounts of noise added.

For comparison, in the FL literature on differential privacy, the addition of Gaussian noise with standard deviation no more than around 0.001 (and often much less) is typically considered, and is only added after the update has been transmitted to the coordinating FL server.

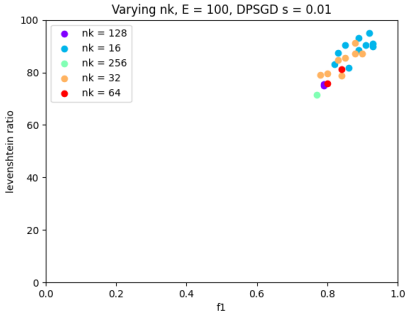
2.6.2 Sentence Reconstruction Performance

Figure 2.7 shows the measured sentence reconstruction performance with both DPSGD-like training (Figures 2.7a and 2.7b) and when noise is added to the final parameters of the model (Figures 2.7c and 2.7d). By removing the noisily added words and running our sentence reconstruction attack, we get results close to those had we not added any noise for up to $\sigma = 0.1$. For the single noise addition method, as these levels of noise are not calibrated, $\sigma = 0.1$ is enough to destroy the quality of reconstructions, however these levels of noise also destroy any model utility.

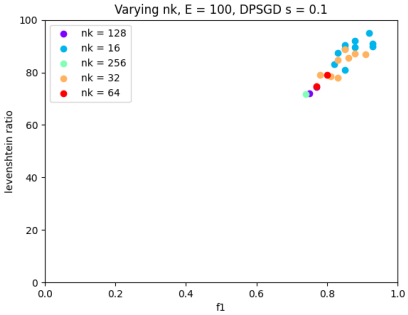
2.7 Additional Material

The code for all of the attacks here, the LSTM model and the datasets used are all publicly available on github `here`.

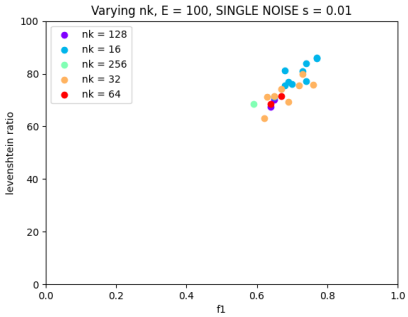
[4]Note that in DPSGD the added noise is multiplied by the learning rate η, and so this factor needs to be taken into account when comparing the σ values used in DPSGD above and with single noise addition. This means added noise with standard deviation σ for DPSGD corresponds roughly to a standard deviation of $\eta\sqrt{EB}\sigma$ with single noise addition. For $\eta = 0.001$, $E = 1000$, $B = 32$, $\sigma = 0.1$ the corresponding single noise addition standard deviation is 0.018.



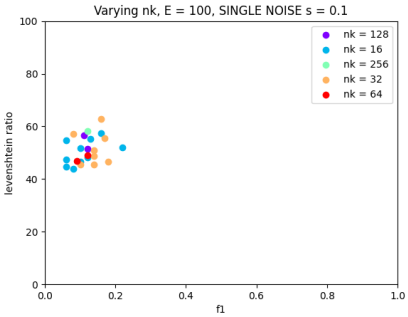
(a)



(b)



(c)



(d)

Figure 2.7: Sentence reconstruction performance with local DP for FedAveraging. The top two Figures (2.7a and 2.7b) show the reconstruction performance for different datasets colour coded by their size for DPSGD-like training, with $E = 100$, $B = 32$. Here we see no real effect in our results compared to the noise-free case. The bottom two figures show the effect of single noise addition on sentence reconstruction for the same setting.

2.8 Summary And Conclusions

In this paper we introduce two local reconstruction attacks against federated learning when used to train natural language text models. We find that previously proposed attacks (DLG and its variants) targeting image data are ineffective for text data and so new methods of attack tailored to text data are necessary. Our attacks are simple to carry out, efficient, and highly effective. We illustrate their effectiveness against the next word prediction model used in Google's GBoard app, a widely used mobile keyboard app (with > 5 Billion downloads) that has been an early adopter of federated learning for production use. We demonstrate that the words a user types on their mobile handset, e.g. when sending text messages, can be recovered with high accuracy under a wide range of conditions and that counter-measures such a use of mini-batches and adding local noise are ineffective. We also show that the word order (and so the actual sentences typed) can be reconstructed with high fidelity. This raises obvious privacy concerns, particularly since GBoard is in production use.

Secure multi-party computation methods such as Secure Aggregation and also methods such as Homomorphic Encryption and Trusted Execution Environments are potential defences that can improve privacy, but these can be difficult to implement in practice. Secure Aggregation requires users to trust that the server is honest, despite the fact that FL aims to avoid the need for such trust. Homomorphic Encryption implementations that are sufficiently efficient to allow large-scale production use are currently lacking.

On a more positive note, the privacy situation may not be quite as bad as it seems given these reconstruction attacks. Firstly, it is not the raw sentences typed by a user that are reconstructed in our attacks but rather the sentences after they have been mapped to tokens in a text model dictionary. Words which are not in the dictionary are mapped to a special `<UNK>` token. This means that the reconstructed text is effectively redacted, with words not in the dictionary having been masked out. This suggests that a fruitful direction for future privacy research on FL for natural language models may well lie in taking a closer look at the specification of the dictionary used. Secondly, we also note that changing from a word-based text model to a character-based one would likely make our attacks much harder to perform.

Chapter 3

Re-evaluating the Privacy Benefit of Federated Learning

3.1 Introduction

Since its introduction, Federated Learning (FL) [51] has been described as a privacy-preserving method of collaborative model training. Several implementations of FL exist from domains in healthcare such as medical imaging to smartphone virtual keyboards [37, 64, 54]. Indeed FL currently is a first choice methodology for machine learning applications that involve sensitive data. An inherent characteristic of FL and its main attractive privacy feature is data localisation. FL does not require the user data to leave their premises to be collected on a remote server for centralised training. Users train models locally and only share their individual updates. This allows the participants in FL to rest assured that their data exists solely on their own property, eliminating the possibility of a data breach on the remote server. Additionally, FL allows the participants to benefit from a large collection of data that otherwise would not have been available to them, resulting in richer, more useful models.

Recently, a growing body of work [79, 78, 26, 40, 19, 34, 85, 27, 65, 93, 88, 89, 33, 32, 9] has looked at the important question of what information about the original data is encoded in the model updates sent to the FL coordinating server, and whether it is possible to infer certain properties of, or to even reconstruct individual training data points. Within the constrictive honest-but-curious threat model, it has been shown [85, 27, 88, 78, 33] that it is possible to reconstruct private user data to a high degree of fidelity from observing individual model updates. Mitigations against these sort of attacks under this threat model come in the form of Secure Aggregation (SA) [10] and Differential Privacy (DP) [3, 81, 23]. With DP and SA applied, the central FL server only has access to the noisy aggregate of the

individual FL user updates. SA with Distributed Differential Privacy (DDP) [72] works by having each user add a small amount of noise to their update before it leaves their device, not enough to prevent data reconstruction attacks, but enough so that there exist strong privacy guarantees for the aggregated model as well as preserving utility. However, FL hardened with SA and DDP does not guarantee user privacy. Departing from the honest-but-curious threat model, [9, 60, 26, 89, 90] have shown that when a server acts maliciously, it is possible still to reconstruct private user data. These attacks exploit the inherent trust FL users must place in the coordinating server. They must trust (i) that the population of users selected by the server for the current FL round is large enough and that it consists of genuine devices with real data and not synthetic devices added maliciously, (ii) the Public Key Infrastructure (PKI) that is used to perform the secure aggregation, often maintained by the coordinating server, (iii) the models trained are not crafted to allow user data to be reconstructed (e.g. see Figure 3.1), (iv) the FL software implementation does not allow de-anonymisation or bypassing of aggregation. In [9] it is concluded that FL is only truly private when the server is trusted to honestly carry out the protocol, or when Local Differential Privacy (LDP) is employed. FL with LDP [73] is a scheme whereby the user adds sufficient privacy-preserving noise to their local update before sharing. Unfortunately, the amount of noise required to achieve LDP is such that the performance of the final learned model degrades as a consequence [80]. Mitigating this requires a large number of clients to participate, which falls into the hands of the coordinating server to orchestrate - a responsibility that the clients must again trust is carried out honestly.

With such considerations in mind, it is worth stepping back for a moment and asking what privacy benefits FL without LDP offers over simply sending raw client data to the central server and trusting that it is stored ephemerally (i.e., in RAM for a short period and immediately deleted after use) and only used for the purposes of model training in combination with data from sufficiently many other users. The levels of trust asked of users seem much the same for both approaches after all. The point of asking this question is not to suggest that we should abandon the pursuit of user privacy, but rather to highlight that FL, as currently formulated, has significant shortcomings that the research community needs to address. In particular, using FL in Google's Gboard mobile keyboard application [37] as a case study, we highlight the following open research questions:

(i) *What Classes of Model Can Be Trained Efficiently Yet Privately.* It is probably too much to ask that arbitrary machine learning models can be trained both efficiently (i.e., with few FL rounds yielding a model with good performance) and privately. For example, we show below that the Gboard LSTM model is susceptible to word and sentence reconstruction attacks and that apparently innocuous changes

such as adding a bias to the final layer can further boost the effectiveness of these attacks. Similarly, it is easy to intentionally craft models that allow local input data to be reliably reconstructed [26]. Of course, adding sufficient local noise can still provide protection, but at the cost of highly inefficient training. This raises the question of what restrictions should be introduced on the class of models used in FL, and how can these restrictions be enforced (e.g., by an FL service where a model may be supplied by a third party).

(ii) *Reducing The Need For User Trust.* Privacy differs from security in that a user chooses to release some information in return for a benefit, and accepts that the information release carries a degree of risk but would like to manage that risk. In the context of FL, the primary risk is data disclosure to the operator of the FL service and/or to the owner of the trained FL model. When either or both of these are in the business of analytics or advertising, user concerns regarding the risks of data disclosure are naturally heightened. In a threat model, these risks manifest themselves in the trust required of users. As already noted, FL currently requires users to place a great deal of trust in both the FL operator and the model owner. The level of trust required is currently too high and there is a need for practical, efficient FL methods that admit much lower trust requirements.

(iii) *Verifying FL System Implementations.* Production implementations of FL algorithms are embedded within larger software systems that include telemetry for monitoring system health, remote configuration to allow settings to be adjusted for testing and roll out of updates, authentication/attestation to protect against sybil and poisoning attacks. It is the privacy of this system as a whole that is of concern to users. Poor implementations can easily allow de-anonymisation of devices and users as well as creating new potential channels for attacks. For example, we show below that in Google's FL implementation in Gboard: (i) the FL attestation methods used to verify device integrity allow device fingerprinting, (ii) the FL telemetry allows device and user de-anonymisation, (iii) the FL protocol allows aggregation to be bypassed, (iv) assignment of a handset to an FL population need not be anonymous. This highlights that public documentation and support for independent evaluation of developed apps by the FL community is important both to verify privacy claims and to build confidence in users that apps employing FL are indeed safe to use.

3.2 Background

Data reconstruction attacks against FL under the honest-but-curious threat model began with the work of Melis *et al.* [53] who explored the question of leakage from gradients and Zhu *et al.* [88], that showed that is possible to reconstruct an exact user data point given its gradient. Further improvements [78, 40, 85, 27, 88, 92] to

their Deep Leakage from Gradients (DLG) algorithm increased it's performance on larger batch sizes, higher resolution images, and the stability of the optimisation procedure. Analytical techniques of data extraction [57] benefit from not being as costly to compute as compared to optimization based methods. While the works previously mentioned focus is on the reconstruction of image data, [21] applies a variant of DLG to text reconstruction from gradients of transformer-based models.

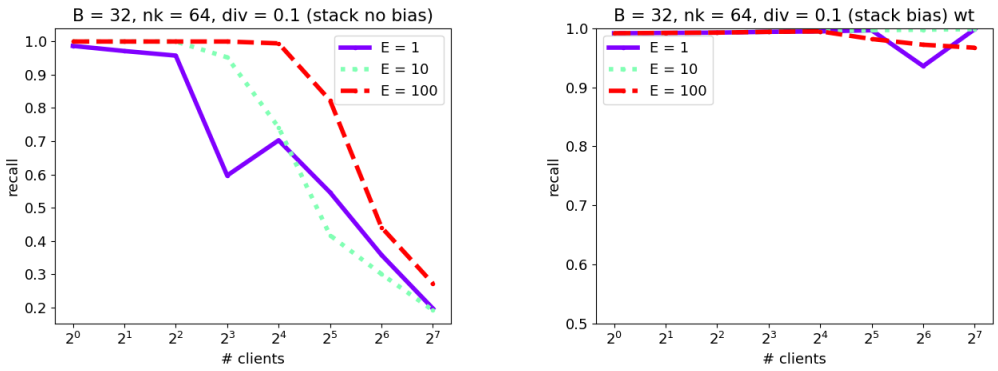
Gupta *et al.* [33] developed similar attacks against language models trained using FL. Gupta *et al.* however provided a unique defense against their attacks beyond SA and DP, that is, freezing the word embedding layers. Using pretrained word embeddings removes these gradients from individual user updates, thus defeating the attack, highlighting the effect of model architecture on the perceived privacy benefit of FL. Gupta *et al.* applied their attacks to only individual model updates and do not provide any results on how their attack performs on aggregate model updates.

The following set of works concern data reconstruction attacks that rely on either malicious modification of the model architecture before the FL training process, active modification of the model parameters during training, or the injection of sybil devices. Fowl *et al.* [26] introduced the idea of modifying the architecture of a model to improve data reconstruction fidelity. Their attack adds fully connected (FC) layers early in the model, whose gradients represent the inputs. Zhao *et al.* [90] extend this work to by adding a convolutional layer and 2 FC layers at the beginning of the model. Through sending customised convolutional kernels to each client in a particular FL round, their attack, termed *MANDRAKE*, can recover up to 80% of the original data with SA of up to 100 clients. This attack was further developed in [89] to acheive similar levels of privacy leakage up to 1000 clients. Pasquini *et al.*'s [60] attack targets the private data of a single client by modifying the parameters sent at each round to each of the non-targeted clients. Crucially, [26, 60] degrade with SA, requiring a greater number of neurons in the artificially inserted FC layers to acheive similar attack performance. This results in very large models (up to 600MB for 100 clients [89]), and can be readily identified by a vigilant client. Model parameter modification attacks [8, 60, 82] are not as easily spotted. Boenisch *et al.* [9] make use of this type of attack, as well as the introduction of sybil devices to avoid any privacy protection afforded by SA and DDP. These sybil devices are synthetic devices controlled by the FL server. Importantly, they can made to produce zero gradients, resulting in recovering the exact individual gradients of a single, targeted user. The fraction of benign and malicious sybil devices affects their attack, showing an inverse relationship between the fraction of benign devices and the reconstruction rate.

3.3 Gboard: A Case Study

3.3.1 Model Architecture and Privacy

In the previous chapter, we have shown that it is possible to reconstruct private client data under the honest-but-curious threat model from individual Gboard model updates. The attack, described in Section 2.3.1, exploits two aspects of the architecture, namely (i) the inputs are echoed by the outputs and (ii) there exists a final layer bias whose gradients reveal the typed words. Figure 3.1 shows how a simple change such as removing the bias can drastically impact the attack's performance. In our experiments[1], each client trains their local LSTM model on sentences taken from the stack exchange data dump [24] following the `FederatedAveraging` algorithm described in [51]. The server, then mounts the attack described in Section 2.3.1 on the difference between the initial shared model and and the final aggregate. This attack can also be performed on the gradients of the final layer weight matrix $\mathbf{W}$, as shown in [88]. However, this attack was originally developed for batch sizes of only 1 example. Crucially, the attack degrades when a greater number of clients is used in SA when no bias is present, however, when a bias is present, SA appears to provide no extra benefit in terms of privacy. Ultimately, architectural changes can both enhance and degrade privacy. Gupta *et al.* [33] propose the use of pre trained word embeddings, eliminating the gradients for the final layer. Additionally, removing a bias from the final layer can help in preventing this type of attack.



(a) No bias (b) With bias

Figure 3.1: Effect of model architecture on the performance of a word reconstruction attack against Gboard updates (see later for further details). Simple changes like adding a bias to the final layer can subvert the protection afforded by Secure Aggregation. Here, each client trains their local model using 64 sentences, a batch size of 32. We plot the word recall results for a varying number of local epochs .

[1]Our code for these FL experiments is available at `https://github.com/namilus/nwp-fedlearning`

Providing privacy guarantees to arbitrary models is difficult, and by using only audited models can FL clients begin to place trust in the protocol. General statements about the privacy of FL across unrestrained models have been shown to be untrue through the development of the line of research concerned with attacking federated learning. Therefore, we believe a closer consideration of the role of model architecture is required in the further development of FL.

3.3.2 Analysis Of Gboard FL Implementation

Much of the FL literature related to privacy has focussed on algorithmic details, such as secure aggregation and the addition of noise to aggregates and individual client uploads, and the associated analysis relies on assumptions such as the anonymity of client updates. However, care is needed in the implementation of these approaches to ensure that privacy goals are actually achieved. In addition, independent evaluation of developed apps is important both to verify privacy claims and to build confidence in users that the apps are indeed safe to use.

Production implementations of FL algorithms are embedded within larger software systems that include telemetry for monitoring system health, remote configuration to allow settings to be adjusted for testing and roll out of updates, authentication/attestation to protect against sybil and poisoning attacks. Unless a great deal of care is taken then these system components can easily allow de-anonymisation of devices and users as well as create new potential channels for attack. In this section, we study the Google FL implementation on Android as an exemplar since it is widely deployed and since Google is a large, well resourced organisation that has been working on FL for some years and so their FL implementation might be expected to represent current best practice.

We measure the actual data transmitted to FL servers by the Google GBoard app with a view to evaluating user privacy. In summary, we find that (i) the FL attestation methods used by Google to verify device integrity allow device fingerprinting, (ii) the FL telemetry collected by Google allows device and user de-anonymisation (the participation of an individual device/user in a specific FL execution round can be readily detected), (iii) the Google FL protocol allows aggregation to be bypassed, (iv) assignment of a handset to an FL population need not be anonymous (and so potentially might be targeted at specific users/devices).

Experimental Setup. Hardware and software used: Google Pixel 4a, Android 11 (similar behaviour observed for Android 12), Google Play Services ver. 22.09.20, Google Gboard ver. 12.4.06, rooted using Magisk. Device Settings: following factory reset, settings are left at their defaults.

Network traffic sent/received by the handset is transmitted via WiFi access point.

On this access point, the firewall settings redirect all WiFi HTTP/HTTPS traffic to `mitmdump` [20] so that the proxying is transparent to the handset. The mitmdump CA cert is installed on the handset as a system cert (this is why the handset needs to be rooted) so that SSL certificate verification checks on the handset pass. When a process running on the handset starts a new network connection, the mitmdump proxy pretends to be the destination server and presents a fake certificate for the target server. This allows mitmdump to decrypt the traffic. It then creates an onward connection to the actual target server and acts as an intermediary, relaying requests and their replies between the app and the target server while logging the traffic.

The WiFi access point runs a DNS resolver, and the firewall is configured to redirect handset DNS requests to this resolver. This resolves all requests as usual except for requests to `federatedml-pa.googleapis.com` which are resolved to the WiFi access point itself, which runs a custom FL server that responds to these requests.

The protocol used by the FL server to communicate with the handset is reconstructed partly from Google's Federated Compute code[2] and partly from inspection of the measured network traffic. In summary, Google apps on handset (Google Messages, Google GBoard) regularly connect to `federatedml-pa.googleapis.com` and send an `eligibility_eval_checkin_request` message over gRPC. Our custom FL server responds with an `eligibility_eval_checkin_response` message specifying `no_eligibility_eval_configured`. The handset then sends a `checkin_request` message and the server responds with a `checkin_response` that contains an `acceptance_info` message which specifies an `execution_phase_id`, `task_name`, `init_checkpoint` and `plan`. The `execution_phase_id` and `task_name` can be arbitrary names, we used "fake_phase_id" and "fake task name". The `init_checkpoint` specifies the initial model parameter values and the `plan` specifies the local dataset to use and a TensorFlow graph the handset should execute to update this checkpoint. In our tests, null values are sent by our server for simplicity (this turns out to be enough for present purposes).

Device Attestation Allows De-Anonymisation. In our experiments, we observe that the `eligibility_eval_checkin_request` messages regularly sent by the Google GBoard and Google Messages apps on the handset to Google server `federatedml-pa.googleapis.com` include Google SafetyNet[3] device attestation data. For example:

```
Headers:
:path: /google.internal.federatedml.v2.FederatedTrainingApi/Session
:authority: federatedml-pa.googleapis.com
content-type: application/grpc
```

[2] `https://github.com/google/federated-compute`

[3] See, e.g., `https://developer.android.com/training/safetynet/attestation`

```
Body:
eligibility_eval_checkin_request {
  attestation_measurement: "CgaqR5AG...AEIgA"
  <...>
}
```

Google does not provide any public documentation on the content of this binary attestation data but recent analysis [70] establishes that the attestation data is a protobuf containing encrypted device data. The data sent can readily be used for device fingerprinting e.g., it contains the names of cache folders which are randomly generated and so effectively act as device identifiers.

Further, Google Play Services on the handset also makes regular connections that also send this attestation data. For example, connections to Google server `android.googleapis.com/checkin` send this data alongside persistent device and user identifiers including the handset hardware serial number, the SIM IMEI, the user Google account email, the Google Android ID and a Google NID cookie [46]. This therefore allows de-aonynmisation of the individual device and user from the attestation data.

Telemetry Allows De-Anonymisation. Google apps send telemetry data via the private Clearcut logger API of Google Play Services. Google Play Services then batches up the data received and forwards it to Google server `play.googleapis.com/log/batch`. Google does not provide public documentation on the data sent to this server, nor on the data format used. However, recent work has reversed engineered a good deal of the message format [46], and we leverage that work here. Each message sent includes an NID cookie and an x-server-token authentication token (which act as device identifiers), followed by a sequence of telemetry protobufs. In our experiments, we observe that Gboard sends telemetry logging FL operations. For example:

```
POST https://play.googleapis.com/log/batch
Headers:
x-server-token : CAESKQDyiOh8EP...25qEp5
cookie : NID=511=DleS...YCwE
Body:
<...>
androidId: 4468978717649541595
<...>
logSourceName: BRELLA
logEntry:
<...>
timestamp: 1681397351854
inAppTrainingService: "com.google.android.inputmethod.latin"
<...>
runID: 40431939126563851
phaseID: "fake_phase_id"
```

It can be seen that data is sent by `com.google.android.inputmethod.latin` which is the Android name of the Google GBoard app. The telemetry data includes an event timestamp (to millisecond accuracy), a run ID that can be used to link together related telemetry events, and the phase ID string sent by our custom FL server. The Google Android ID, NID cookie, and x-server-token are sent alongside

this telemetry data, allowing it to be linked to the individual device and user (using the messages sent to `android.googleapis.com/checkin` noted above). Participation of an individual device and user in a specific FL execution round can therefore be detected.

Aggregation Can Be Bypassed. In the Google FL protocol, the `eligibility_eval_checkin_response` message from the FL server (sent in response to periodic `eligibility_eval_checkin_request` messages from the handset) can request the handset to execute an `eligibility_eval` plan and return the response. This includes an initial checkpoint (model parameter values), a local dataset to use, and a TensorFlow graph the handset should execute to update the checkpoint and generate a response. This response is not aggregated and is sent by the handset to the server in subsequent FL `checkin_request` messages.

Population Assignment Can Be Based On Identity. In the Google FL protocol, each FL model within a Google app on a handset is assigned to a population. There may be multiple models within the same app, e.g., in GBoard there are next word prediction, speed recognition, and emoji prediction models and each may be assigned to a separate population.

The population of a model within an app is specified by a string value. This value is assigned a name, e.g., `lstm_federated_training_population`, and defaults to a null value, e.g., "impossible',' but can be updated via the Google Play Services Heterodyne service. The Heterodyne service sends details of installed apps to server `www.googleapis.com/experimentsandconfigs` and the server responds with configuration values. For example:

```
POST https://www.googleapis.com/experimentsandconfigs/v1/getExperimentsAndConfigs
Headers:
x-server-token : CAESOQDyi0h8v...Hp4Q==
authorization : Bearer ya29.m.CvkBAZ8...kS1pjLU53
Body:
<...>
androidId: 4468978717649541595
cookie: "NID=511=DleS...YCwE"
<...>
<details of apps installed and their experiment tokens>
```

It can be seen that this request includes multiple device and user identifiers. The androidId itself is enough to link the message to the individual device and user (using the messages sent to `android.googleapis.com/checkin` noted above), but in addition, the authorization header is generated using the user's Google account and so is directly linked to the handset user. This means that the server has to hand information that allows the FL population values it assigns to be based on the individual device and user.

Production implementations of FL algorithms are embedded within larger software systems that include telemetry, remote configuration, device authentication/attestation etc. It is, of course, the privacy of the system as a whole that is of concern

to users. We show that poor implementations can easily allow de-anonymisation of devices and users as well as creating new potential channels for attacks. Our GBoard measurement study also highlights that public documentation and support for independent evaluation of developed apps by the FL community is important both to verify privacy claims and to build confidence in users that apps employing FL are indeed safe to use.

3.4 FL Currently Requires Too Much Trust

An honest FL participant may believe that they have preserved their privacy by using FL, yet their data can be readily recovered and tied to them by the coordinating server. This is especially true when our attacks are combined with previous attacks [90, 8] that target a single user via the injection of sybil devices. This is a direct result of the inordinate amount of trust clients are asked to place in the FL system. Clients must *trust* that the co-ordinating server is faithfully carrying out the FL protocol, clients must *trust* that the other clients are genuine, clients must *trust* that the cryptography behind the PKI in SA is not compromised, clients must *trust* the model architecture has not been designed maliciously, etc.

When the FL system and the model being trained are both operated by a reputable organisation then perhaps such trust can be justified. However, it requires strong governance and oversight of that organisation and the avoidance of potential conflicts of interest (such as the organisation also being a consumer of user data for analytics, advertising etc). When such a level of trust exists then it also begs the question of why not simply send raw client data to the central server and trust that it is stored ephemerally and only used for the purposes of model training in combination with data from sufficiently many other users i.e., the added privacy value of FL seems rather small.

When multiple parties are involved in the FL system, such as with Federated-Learning-as-a-Service (FLaaS) [44], establishing sufficient trust seems much harder. For example, suppose an organisation operates FLaaS for mobile apps. Then the models to be trained by FL on private client data may be supplied and used by multiple different app developers with a broad geographic spread and different regulatory regimes. As we already know from mobile app stores, users then have only a limited ability to establish developer bona fides and even powerful gatekeepers such as Google and Apple have difficulty regulating developer behaviour. Hence, even when the FLaaS provider is trusted, the overall FLaaS system need not be trustworthy.

Attacks against FL are executed under a presumed threat model. They can range from honest-but-curious to an actively malicious server. Attacks that operate

under the former are difficult to spot by a client, as from their perspective, FL is operating normally, and they are presented with a useful model at the end of training. Modifying the model architecture before training begins is still within the honest-but-curious model. No changes happen during the FL process as it is still performed faithfully by the server.

On the other hand, actively malicious servers use their power over the round to actively target individual clients or the entire population. They may insert synthetic devices, manipulate model weights, architecture, compromise the PKI, and training process actively during training. These different threat models affect what kind of attacks can potentially be mounted, and a value judgement must be made about how realistic a given threat model is. FL that is secure to even the most relaxed threat model, where the server can actively thwart even the ultimate training goal to reconstruct user data, may not provide any service at all, relinquishing any gained utility for the utmost guarantees of security and privacy.

We note, however, that some degree of trust is likely to be asked of FL users. Trying to ensure user privacy when the FL service is actively malicious is probably a hopeless endeavour – the server may insert synthetic devices, manipulate model weights, architecture, compromise the PKI, and training process actively during training and it seems hard to defend against all of these while still providing a useful FL service. The need is to greatly reduce the level of trust asked of users, and thereby provide a better privacy risk-benefit trade-off to them. Ultimately, it is a question of risk assessment on the side of the potential FL participant. They must weigh the potential risks that are incurred by the participation in FL and the benefit gained from the service. It has been made clear, however, through our work on Gboard and the growing body of attacks against FL, that the supposed *private* nature inherent to FL needs to be questioned, and any added privacy benefit to be made evident.

3.5 Conclusion

FL in its current form requires a large amount of trust in the central server on the part of the participating clients. By studying Google's Gboard FL application, we have demonstrated that certain aspects of FL, such as the choice of the model architecture, and their real-life implementation, require special attention. We have built upon the work showing that model architecture affects privacy by providing a real world, in production, example of architectural modification to aid data reconstruction. Additionally, we have also analyzed Google's FL implementation in Gboard and shown that there exist some privacy concerns in the telemetry. User and device de-anonymisation is possible, data aggregation can be bypassed, and the

central server maintains control of the population. We hope that the discussions provided here will motivate further research into FL and its added privacy benefit and serve as a potential future roadmap.

Chapter 4

Towards a Re-evaluation of Data Forging Attacks in Practice

4.1 Introduction

Data governance is becoming an increasingly important subject with the rise of indiscriminate scraping of web data to train machine learning models. Legislation such as the GDPR Framework [76, 74] and the CCPA [45] have been introduced to prevent the use of sensitive personal information during model training and reduce potential harm to the data owners that may result from the proliferation of these models. Multiple stakeholders exist: data owners want assurances that their copyrighted or private information has not been used to train models without their consent, while the model owners want to prove that their training data is legally compliant.

Data forging [71, 43, 7, 86] has recently emerged as a new attack against machine learning models, particularly against data governance. Data forging attacks appear to provide counterfactual "proof" that a model was trained using a particular dataset; when in reality, it was trained on another. In effect, these attacks *forge* one dataset into another *distinct* dataset that produces the same model. Given the true training set D, a forgery D' may be produced along with sufficient proof claiming that D' is the real training set, not D. Thus, data forging attacks appear to throw into jeopardy the question of ever determining exactly what data a model was trained on.

Data Forging. At a high level, given a machine learning model's true training dataset D, a data forging attack produces a claim that the model is actually the result of training on a *different* dataset D'. To achieve this, data forging attacks rely on the fact that supervised training uses iterative algorithms such as Stochastic Gradient Descent (SGD). These algorithms produce a sequence consisting of model

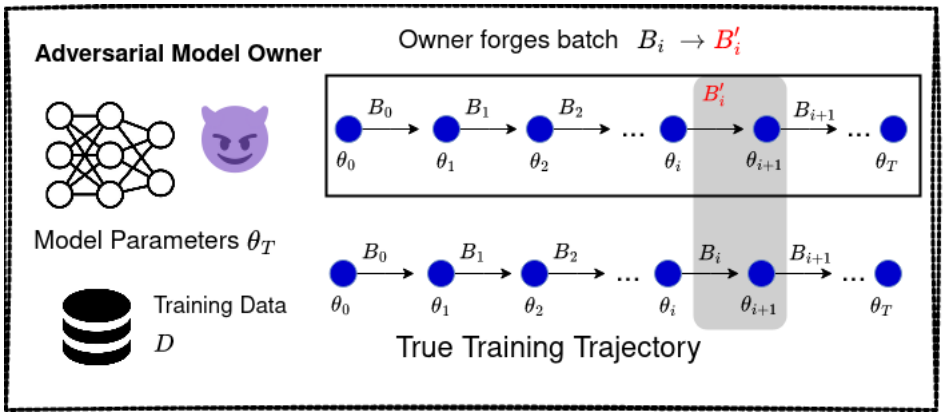


Figure 4.1: The adversarial model owner has access to the true execution trace denoting the sequence of model parameters and associated mini-batches. When performing a data forging attack, the model owner *forges* (*i.e.*, replaces) a batch B_i such that $(\mathbf{x}, \mathbf{y}) \in B_i$ with a different batch B_i' not containing $(\mathbf{x}, \mathbf{y})$ such that the resulting models are nearly identical.

parameters (or *checkpoints*) and associated mini-batches, starting with the random initialisation and ending with the final trained model parameters. Such a sequence, or *execution trace*, may be *verified* by a third party, who reproduces each checkpoint in the sequence, using the mini-batch and parameters from the previous step, and ensures that what they have recomputed and the current checkpoint are nearly identical, i.e., the ℓ_2 distance between the two sets of model parameters is below a given *small* non-zero error threshold ϵ. A non zero threshold may be allowed by the verifier due to noise that stems from hardware and software factors. Specifically, there are small numerical deviations between repeated recomputations of any particular checkpoint due to benign noise (see Section 4.2.2 for a detailed discussion on the source of these errors). Importantly, if an execution trace passes verification i.e., the recomputation of every step is below the verifier's chosen threshold, then that trace is valid "proof" for the verifier that the data used to train the model is what occurs in the mini-batches present in the trace and no other (provided the initial model θ_0 is random). This concept is formally known as Proof of Learning (PoL), introduced by [39]. An adversary performing a data forging attack replaces one or more mini-batches in the execution trace with *forged* (*i.e.*, different) mini-batches that produce nearly identical gradient updates, and falsely claims that *(i)* the model checkpoints present in the execution trace are actually the result of training on these forged mini-batches, not the original mini-batches that were replaced, and *(ii)* the observed recomputation error by the verifier is simply a result of hardware and software factors.

Privacy Implications. Existing works [71, 43, 86] have noted that data forging has clear implications for two large concepts within machine learning and privacy; namely *(i)* membership inference, and *(ii)* machine unlearning. Membership inference attacks (MIAs) [67, 15, 38] have been developed in order to determine whether

a particular data record was part of the model's training dataset. Suppose that the training example $(\mathbf{x}, \mathbf{y})$ is indeed a member of the model's training set; an adversary may refute this claim by providing a forged execution trace such that $(\mathbf{x}, \mathbf{y}) \notin B_i$, for all mini-batches B_i used during the execution trace (see Fig. 4.1). If such an execution trace passes verification, then the adversary has successfully refuted the membership inference claim as they have provided valid proof for the verifier that $(\mathbf{x}, \mathbf{y})$ was not used to train their model. This is because in none of the steps of the execution trace does $(\mathbf{x}, \mathbf{y})$ occur as a member of the current step's mini-batch.

Performing such an attack is also equivalent to claiming that $(\mathbf{x}, \mathbf{y})$ has been exactly "unlearnt". The concept of *exact unlearning* [11], refers to the idea of unlearning i.e., removing the influence of, a particular example $(\mathbf{x}, \mathbf{y})$ from the model, which involves retraining from scratch using the dataset $D' := D \backslash \{(\mathbf{x}, \mathbf{y})\}$. The forged execution trace that passes verification provides valid "proof" to the verifier that the final model parameters are the result of training on a dataset that does not contain $(\mathbf{x}, \mathbf{y})$; the definition of exact unlearning. Importantly, the adversary does not have to perform any actual re-training, resulting in two equally valid claims that appear to contradict each other; the model has been both trained and not trained using $(\mathbf{x}, \mathbf{y})$. Machine unlearning [83, 11, 14] has emerged as a response to GDPR's [76] *right to be forgotten* framework, and prior work [71, 86] note how data forging can undermine the concept, and in its wake, call for the research and development of auditable machine unlearning. We argue that the seriousness of these implications hinges on the performance of data forging attacks, i.e., *how identical are the model updates between original and forged mini-batches produced by these attacks?* This metric, known as approximation error, is given by the ℓ_2 norm between the forged model update and the original present in the execution trace.

Within the data forging literature, there exists a dichotomy of assumptions on how small this approximation error should be in order to fully realise the implications discussed previously. Prior data forging works [43, 71, 86] argue that their attacks produce approximation errors that are indistinguishable from benign reproduction errors. On the other hand, Baluta et al. [7] argue against the acceptance of any error, citing that floating point operations are deterministic and that there should be zero error when recomputing execution traces. Baluta et al. go on to prove that zero error, or *exact* data forging, is not possible using existing attack methods. As a result, they find that existing data forging attacks do not pose any risk to membership inference or machine unlearning as they are easily detectable by the presence of non zero error. While the zero error threshold scenario posed by Baluta et al. [7] is possible if the original hardware and software are available during verification (and may be necessary in certain high stakes scenarios such as medical data), it is unrealistic to always impose such a strict setting in practice. Particularly

when the original hardware and software is not available, it is not practical, and often impossible, to avoid benign hardware reproduction errors. We argue that data forging attacks fully realise their implications only if their approximation errors are of the same order of magnitude as these benign reproduction errors. Prior data forging works [43, 71, 86] do not justify the level of approximation error generated by their attacks, and largely ignore this pertinent question regarding their performance.
Contributions. We list our main contributions to data forging as the following:

1. We provide the first comprehensive analysis of the magnitude of reproduction errors for fully connected, convolutional, and transformer neural networks architectures trained on tabular, image, and textual data.

2. We find that the approximation errors produced by existing data forging attacks are too large, often several orders of magnitude larger than our observed reproduction errors. This makes existing attacks easily detectable by a verifier, nullifying their privacy impact in practice.

3. The current theoretical analysis of data forging in Baluta et al. [7] is restricted to the existing instantiations of data forging attacks and necessitates the investigation of broader potential attack strategies. To tackle this, we extend their analysis, addressing more general questions regarding the existence of distinct mini-batches that produce the same gradient and how they may be constructed. Focusing on fully connected neural networks, we formulate the problem of exact data forging as a system of linear equations, where each solution corresponds to a forged mini-batch. Of the infinite solutions that may exist, finding those that correspond to a distinct, valid mini-batch i.e., one that falls within the domain of allowed training examples (e.g., in the case of images, pixel values between $0 - 255$ and one hot labels) is a non-trivial task, and one that we conjecture may be computationally infeasible, even for relatively simple linear models such as logistic regression.

As a result, we call for a re-evaluation of existing attacks, and for further research into this nascent attack vector against machine learning.

4.2 Background

4.2.1 Execution Traces

Supervised machine learning is a process to learn a *model*, in particular, a parameterized function $f_\theta : \mathcal{X} \to \mathcal{Y}$. The parameters are typically optimized by applying iterative methods such as Stochastic Gradient Descent (SGD) to a training set. At

the i-th gradient descent step, the next set of model parameters θ_{i+1} is calculated from the previous set of parameters θ_i and a mini-batch B_i consisting of b training examples sampled from the dataset $D = \{(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})\}_{i=1}^{N}$. For mini-batch SGD, we have that $\theta_{i+1} := g(\theta_i, B_i) = \theta_i - \eta\nabla_\theta\mathcal{L}(B_i; \theta_i)$. Starting with a random initialisation θ_0, the final model θ_T is the result of performing T total gradient descent steps iteratively. This results in an *execution trace* (or simply, a trace) consisting of a sequence of tuples $\{(\theta_i, B_i)\}_{i=0}^{T}$, that capture the *training trajectory* of the final model θ_T (note that $B_T = \emptyset$).

4.2.2 Reproduction Errors

Recent work by Schlögl et al. [66] found that runtime optimisations of machine learning frameworks can result in non-deterministic numerical deviations in the outputs of neural networks. Schlögl et al. cite auto-tuning [30] as the root cause. This is a process whereby microbenchmarks executed just before inference determine what optimisations to apply to the given operation e.g., the results of the microbenchmarks determine which GPU kernel gives the best performance on a given hardware and input shape. A well known property of floating point arithmetic is that addition is not associative, that is, it is not necessarily the case that $(a+b)+c = a+(b+c)$, but may be off by a small error. These auto-tuning optimisations can lead to changes in the aggregation order of intermediate results from kernel to kernel, resulting in deviations in the final output of a neural network, even for the same input. In the context of execution traces for machine learning models, this means that given the same random initialisation θ_0 and sequence of mini-batches $B_0, B_1, B_2 \ldots, B_{T-1}$, the resulting parameters $\theta_1, \theta_2, \theta_3, \ldots, \theta_T$ will differ between repeated recomputations due to auto-tuning.

4.2.3 Data Forging

Thudi et al.'s [71] concepts of *forgeability* and data forging attempt to produce distinct execution traces for a given model θ_T using the same checkpoints, but with different e.g., *forged* mini-batches. At a high level, any mini-batch from an execution traces may be *forged* (*i.e.*, replaced) with another containing different training data, yet still produce a nearly identical gradient descent update. *Data forging attacks* against execution traces attempt to construct, for a given mini-batch B_i, a forged (*i.e.*, distinct) counterpart B_i' such that the next model in the execution trace, when calculated using B_i' is *nearly* identical to θ_{i+1} *i.e.*, the approximation error $\|\theta_{i+1} - g(\theta_i, B_i')\|_2$ is minimal. The attacks proposed in [71, 43] construct forged mini-batches by randomly sampling *other* training examples from the dataset. These attacks were introduced in the context of *deleting* a set $U \subset D$ of training examples

from an execution trace, and work by *greedily searching* over the set of training examples $D \backslash U$. A *greedy search* attack works in three steps *(i)* sample n data points uniformly from $D \backslash U$, *(ii)* sample M mini-batches $\{\hat{B}_1, \ldots, \hat{B}_M\}$ uniformly from the selected n data points, and *(iii)* out of the M mini-batches $\{\hat{B}_1, \ldots, \hat{B}_M\}$, select the mini-batch $\hat{B}_j$ that minimises $\|\theta_{i+1} - g(\theta_i, \hat{B}_j)\|_2$, and output the forged mini-batch $B'_i = \hat{B}_j$.

Thudi et al. introduced this type of forging approach, and Kong et al. [43] improve upon its efficiency, with similar performance. Recently, Zhang et al. [86] augmented this approach by choosing to instead replace data points $(\mathbf{x}, \mathbf{y}) \in U$ with their class-wise nearest neighbour from $D \backslash U$, with also similar performance.

4.3 Data Forging Threat Model

At its core, the danger of data forging, and why it must be taken seriously, ultimately stems from the uncertainty it casts over exactly what data was used to train a model. This can allow an adversary (*i.e.*, a malicious model owner) to claim their model was trained on one dataset, when in fact, it was trained on another. Motivating applications include models that were trained on copyrighted and/or sensitive information.

In Algorithm 5, we formulate the threat of data forging as game between an adversary $\mathcal{A}$ and a verifier $\mathcal{V}$. Our formulation captures the setup proposed in previous data forging works [43, 71, 7, 86]. We allow for a non zero error between recomputed and original checkpoints in contrast to the data forging game defined by [7], who required $\epsilon = 0$, a threat model which is much stricter than ours, and one that may not be realistic in all scenarios due to the existence of floating point errors (see Section 4.2.2 for a more detailed discussion).

The game plays as follows: the verifier begins by first running the training algorithm $\mathcal{T}$ e.g., stochastic gradient descent on dataset D for T steps, and produces execution trace $\{(\theta_i, B_i)\}_{i=0}^{T}$, saving every checkpoint. The verifier then challenges the adversary to forge a random batch B_t from the execution trace. In concrete data forging attack settings, batch B_t may contain legally problematic or sensitive information; data that the adversary must forge with approximation error less than the verifier's chosen threshold ϵ, in order to successfully claim the data in that batch was not used *i.e.,* falsely *delete* from their model's training data. The adversary has access to the full trace and the dataset, and is capable of instantiating any data forging attack algorithm in order to produce the corresponding forged mini-batch B'_t, which must satisfy the following initial conditions:

- $B'_t \neq B_t$. Adversaries participating in the game must produce a forged mini-

Algorithm 5 The data forgery game between $\mathcal{A}$ and $\mathcal{V}$.

Input: Adversary $\mathcal{A}$, Verifier $\mathcal{V}$, Training algorithm $\mathcal{T}$, Dataset D, error threshold ϵ, batch size b

Output: ACCEPT or REJECT

$\mathcal{V}$ produces execution trace $\{(\theta_i, B_i)\}_{i=0}^{T} \leftarrow \mathcal{T}(D, b)$.
$\mathcal{V}$ chooses a random index $t \in \{0, 1, 2, 3, ..., T-1\}$.
$\mathcal{A}$ forges mini-batch B_t into $B'_t \leftarrow \mathcal{A}(\{(\theta_i, B_i)\}_{i=0}^{T}, D, t)$.
if $(B'_t = B_t)$ or $(\exists(\mathbf{x}', \mathbf{y}') \in B'_t$ s.t. $(\mathbf{x}', \mathbf{y}') \notin \mathcal{X} \times \mathcal{Y})$ **then**
  return REJECT
end if
$\mathcal{V}$ calculates $\theta^{\mathcal{V}}_{t+1} \leftarrow g(\theta_t, B'_t)$ on their own hardware.
$\mathcal{V}$ calculates the error $\epsilon_{t+1} \leftarrow \|\theta_{t+1} - \theta^{\mathcal{V}}_{t+1}\|_2$
if $(\epsilon_{t+1} > \epsilon)$ **then**
  return REJECT
else
  return ACCEPT
end if

batch that is *different* from the original in at least one example.

- $(\mathbf{x}', \mathbf{y}') \in \mathcal{X} \times \mathcal{Y}, \quad \forall(\mathbf{x}', \mathbf{y}') \in B'_t$. This condition constrains each forged training example $(\mathbf{x}', \mathbf{y}')$ to be within the domain of the training context *e.g.* pixel values between $0-255$ and one hot labels. Otherwise, B'_t will be rejected and the adversary loses the game.

Finally, The verifier computes $\theta^{\mathcal{V}}_{t+1} := g(\theta_t, B'_t)$ using the forged mini-batch B'_t produced by the adversary and calculates the ℓ_2 error $\epsilon_{t+1} := \|\theta_{t+1} - \theta^{\mathcal{V}}_{t+1}\|_2$ between their recomputed checkpoint, and the one present in the trace. The game ends with a REJECT result if $\epsilon_{t+1} > \epsilon$, otherwise the game ends with ACCEPT *i.e.*, the adversary wins. The reproduction error threshold ϵ is a parameter of the game that defines how large the ℓ_2 between a recomputed model and the original model may be. We introduce the notion of ***stealthy data forging*** which refers to the production of a forged mini-batch B'_t that wins the above data forging game. Concretely, this means that attacks that produce a B'_t whose approximation error is less than or equal to the chosen threshold ϵ are deemed *stealthy*.

4.4 Are Current Data Forging Attacks Stealthy?

Previous data forging works [43, 71, 86] have employed data forging as a means of adversarially *deleting* data from a model's execution trace *i.e.*, for a given set of training examples $U \subset D$, these attacks forge the true dataset D into another D' such that $D' \cap U = \emptyset$. Kong et al. [43] note that if an adversary can delete training examples from an execution trace by replacing them with forged examples, they can

provably refute a membership inference claim against them. Additionally, [71, 86] observe that if an adversary performs the same replacement, they can also claim to have "unlearnt" those training examples. These implications of data forging are only fully realised if the attacks are *stealthy* i.e., the adversary wins the game with an approximation error lower than the error threshold chosen by the verifier.

Theoretically, Thudi et al. [71] have proven the existence of an adversary that can win the forging game defined in the previous section for any $\epsilon > 0$, given that they have access to the underlying data distribution $\mathcal{D}$, and have freedom over the choice of batch size b. In particular, they prove that in the limit $b \to \infty$, the probability that mini-batches from any datasets D and D' sampled i.i.d from $\mathcal{D}$ can be forged approaches 1 as $b \to \infty$ (see Theorems 2 and 3 from [71] regarding probabilistic forging). Of course, in practice, the adversary cannot sample indefinitely[1], from, nor do they have access to $\mathcal{D}$, however, Thudi et al. demonstrate the practicality of forging by developing a data forging attack algorithm that can win the forging game for $\epsilon \ll 1$ (see Sec. 4.2.3 for a description of their algorithm, as well as others in the literature). However, they do not consider the pertinent question regarding whether the approximation errors produced by their attack are comparable to reproduction errors induced by floating point noise.

In practice, an informed verifier will choose their error threshold ϵ to be no larger than observed reproduction errors. As a result, for a data forging attack to be stealthy, its approximation errors must be on the same order of magnitude as reproduction errors. Stealthiness, we argue, derives from the fact that the approximation errors would be indistinguishable from the regular noise that pervades floating point computation. In order to answer the question of whether existing attacks are stealthy, we first conduct experiments to determine the magnitude of reproduction errors, and compare this to the approximation errors produced by existing attacks. In the next section, we perform repeated recomputations of execution traces, and measure the ℓ_2 distance between repeated recomputations, in order to characterise the magnitude of reproduction errors. Subsequently, we compare the magnitude of our observed reproduction errors to the magnitude of the approximation errors produced by the data forging attacks developed in [43, 71, 86]. **Our findings suggest that existing data forging attacks are not stealthy** i.e., reproduction errors are often orders of magnitude smaller than approximation errors produced by existing data forging attacks.

[1] Additionally, there only exists a finite number of images consisting of discrete pixel values.

4.4.1 What is the Magnitude of Benign Reproduction Errors?

In order to quantify the magnitude of reproduction errors, we perform experiments where a model is trained on a dataset for T total steps, recording its execution trace. We then recompute this trace, on the same and also different hardware, and measure the distance between every recomputed checkpoint and the corresponding original checkpoint. We repeat the verification process 10 times. The magnitude of the error between repeated recomputations of the same checkpoint is called the *reproduction error* and is calculated using the ℓ_2 norm $\epsilon_{repr} := \arg\max_i \epsilon_{i+1}$ *i.e.*, the reproduction error is the largest observed ℓ_2 distance between recomputations. In Table 4.1, we summarise the models and datasets used in our experiments, which includes a fully connected neural networks (FCN) with 1 ReLU hidden layer consisting of 100 neurons, a small and large convolutional neural network, and a decoder-only transformer. These models are trained on the Adult, MNIST, CIFAR10, and IMDB datasets respectively, covering classification, object detection, and sentiment analyis tasks across tabular, image, and textual data modalities. We keep the learning rate fixed at $\eta = 0.01$. For all our experiments, we use TensorFlow v2.15.0, CUDA v12.2, and CUDNN v8.9. In addition to `float32` training, we also investigate the effect of `float16` and `bfloat16` mixed-precision training on reproduction errors.

Table 4.1: Models and Datsets.

Model / Dataset	# Total Params
FCN / Adult	$1,601$
LeNet / MNIST	$61,706$
Transformer / IMDB	$657,758$
VGGmini / CIFAR10	$5,742,986$

For a given execution trace (e.g., a trace produced for a given model, dataset, batch size b, and on a given hardware platform), our experiments proceed as follows:

1. First, we choose the hardware platform where the execution trace will be recomputed *i.e.*, this is either the same GPU it was generated on or a different one.

2. We recompute each checkpoint by loading θ_i from the execution trace and training on the corresponding mini-batch B_i, also taken from the execution trace.

3. We measure the distance between our recomputed checkpoint and the corresponding checkpoint θ_{i+1} present in the execution trace.

4. We repeat this process 10 times and report the largest observed distance for each recomputed checkpoint, as well as variance.

The question of what might affect the magnitude of reproduction errors boils down to the question of what might affect the results of the auto-tuning microbenchmarks that ultimately decide which floating point operations are done in which order. It is possible (however unlikely), that two training runs, starting with the same random initialisation and using the same sequence of mini-batches, may produce the exact same sequence of checkpoints (*i.e.*, zero reproduction error) simply by chance. In reality, reproduction errors occur because the likelihood that the floating point operations are done in the same order after auto-tuning between two different training runs is very small. What goes into the decision of gpu kernel can depend on a host of reasons such as the model architecture, dataset, the particular hardware and software, the other processes that are running on the machine at the time, etc. A comprehensive analysis of all these factors that affect reproduction error is beyond the scope of this work, however we pick the same models and datasets as previous works e.g. LeNet/MNIST and VGGmini/CIFAR10 as used in [7, 71, 43] (and additionally FCNs and decoder-only transformers), in order to determine a value for the magnitude of reproduction errors such that we may compare their attack's approximation errors to. Previous work by [62] considered the effect non determinism introduced by GPU training on the final accuracy of the trained models (see Appendix B our accuracy measurements). Our experiments regarding reproduction errors attempt to determine the magnitude of these errors as we vary model size and type, data modality, floating point precision, batch size, and GPU model.

Batch size, model size and data modality do not appear to greatly affect reproduction error. To characterise the effect of batch size on reproduction errors, we train our models with batch sizes $b = \{50, 100, 500, 1000, 2000\}$, covering 3 orders of magnitude. Figure 4.2 provides the largest observed reproduction errors during the recomputation of execution traces for each of the 4 considered model setups. We see that the reproduction error remains constant for all 4 setups, varying by only one order of magnitude at most. The largest observed reproduction error was for the transformer model, which was no larger than 1×10^{-6} in magnitude for $b = 50$. Additionally, the models considered span 4 orders of magnitude in terms of the number of parameters (see Table 4.1), and we find no correlation between model size and reproduction error; the transformer model consisting of 600K parameters produces larger reproduction errors than VGGmini which has 5.75M parameters. From our experiments the effect of data modality is unclear i.e., images and tabular data produce similar levels of reproduction errors whereas textual data produces slightly larger errors (larger by a single order of magnitude).

Mixed precision training can result in larger reproduction errors. Mixed

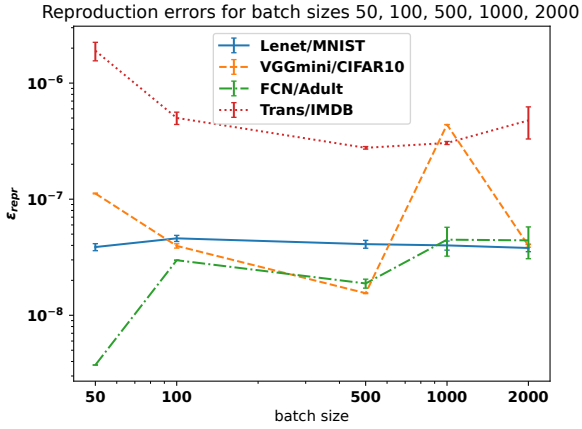


Figure 4.2: The observed largest reproduction error across multiple recomputations across batch size and model type and size.

precision training involves using lower precision floating point types such as `float16` and `bfloat16`[2] during training in order to increase performance on modern GPUs. Given their lower precision, it is possible that reproduction errors for lower precision floating point numbers can be larger than those for the default `float32`, as investigated previously in Section 4.4.1.

To evaluate this claim, we re-train our models using both `float16` and `bfloat16` specifications and provide the largest observed reproduction errors in Table 4.2. Generally, lower precision training appears to result in larger reproduction errors as compared to when training with the more precise `float32`. We find that these errors may be up to 3 orders of magnitude larger e.g. LeNet/MNIST for batch size $b = 100$ results in reproduction errors $\sim 1 \times 10^{-8}$ with `float32` training, however can be as large as $\sim 1 \times 10^{-5}$ with mixed precision `bfloat16` training.

Recomputation on different hardware can result in larger reproduction errors. We now attempt to characterise the magnitude of benign hardware errors when the recomputation is done on a different GPU than the one that produced the execution trace. In Table 4.3, we report the largest observed reproduction error for cross hardware recomputation. We consider two different GPUs in our experiments: a RTX 4090 and a Tesla V100. We find that recomputing execution traces on different hardware can produce slightly larger levels of error compared to when recomputation is done on the original hardware, In our cross-hardware experimentation, we found that the error increases only by a single order of magnitude. On both GPUs, auto-tuning is turned on.

In most machine learning applications, auto-tuning is turned on as it boosts performance, however, widely used machine learning frameworks such as TensorFlow

[2]Both formats take up 2 bytes, however `float16` uses 10 bits for the mantissa, whereas `bfloat16` uses 7.

Model/Dataset	Largest ϵ_{repr}
LeNet/MNIST	$8.96 \times 10^{-6}(\pm 8.64 \times 10^{-7})$
VGGmini/CIFAR10	$7.35 \times 10^{-5}(\pm 1.11 \times 10^{-6})$
FCN/Adult	$4.2 \times 10^{-6}(\pm 1.94 \times 10^{-7})$
Trans/IMDB	$4.74 \times 10^{-5}(\pm 2.83 \times 10^{-5})$

(a) `float16`

Model/Dataset	Largest ϵ_{repr}
LeNet/MNIST	$6.6 \times 10^{-5}(\pm 4.77 \times 10^{-6})$
VGGmini/CIFAR10	$5.97 \times 10^{-4}(\pm 2.54 \times 10^{-5})$
FCN/Adult	$1.22 \times 10^{-4}(\pm 5.14 \times 10^{-5})$

(b) `bfloat16`

Table 4.2: The largest observed ϵ_{repr} when the training and recomputation are done using mixed-precision.

[1] and PyTorch [61] provide an option to turn it off[3]. We found that turning auto-tuning off results in zero error when recomputing execution traces using the same hardware and software, as the floating point operations occur in a deterministic order. However, different hardware platforms may not share the same implementations of algorithms *i.e.*, they may differ in approach and aggregation order, resulting in unavoidable reproduction error, even if auto-tuning is turned off. Table 4.3b shows how when auto-tuning is off e.g., in deterministic training, there is zero error when the execution trace is generated and recomputed on the same hardware. However, when the generating and recomputing hardware differ, we found that the reproduction error is non zero, and during our experiments, was larger than when auto-tuning is turned on.

Discussion. In this section, we have provided our results on observed reproduction errors as we vary models (large and small), datasets, batch sizes, floating point precision, and recomputation hardware, however we refrain from making any statement about what directly results in the magnitude of the reproduction error for a given setup. This is due to the closed source nature of CUDA (NVIDIA's proprietary GPU driver code), making any grounded determinations as to what is going on "under the hood" of the GPUs extremely difficult. However, our results and methodology represent a principled approach for verifiers to calculate suitable error thresholds for the verification of execution traces. Using these results, in the next section we evaluate the *stealthiness* of data forging attacks.

[3]See `https://www.tensorflow.org/api_docs/python/tf/config/experimental/enable_op_determinism` and `https://pytorch.org/docs/stable/notes/randomness.html`

	RTX 4090	Tesla V100
RTX 4090	10^{-8}	10^{-7}
Tesla V100	10^{-7}	10^{-8}

(a) LeNet/MNIST $b = 100$

	RTX 4090	Tesla V100
RTX 4090	0	10^{-6}
Tesla V100	10^{-6}	0

(b) Deterministic LeNet/MNIST $b = 100$

	RTX 4090	Tesla V100
RTX 4090	10^{-8}	10^{-7}
Tesla V100	10^{-7}	10^{-8}

(c) LeNet/MNIST $b = 1000$

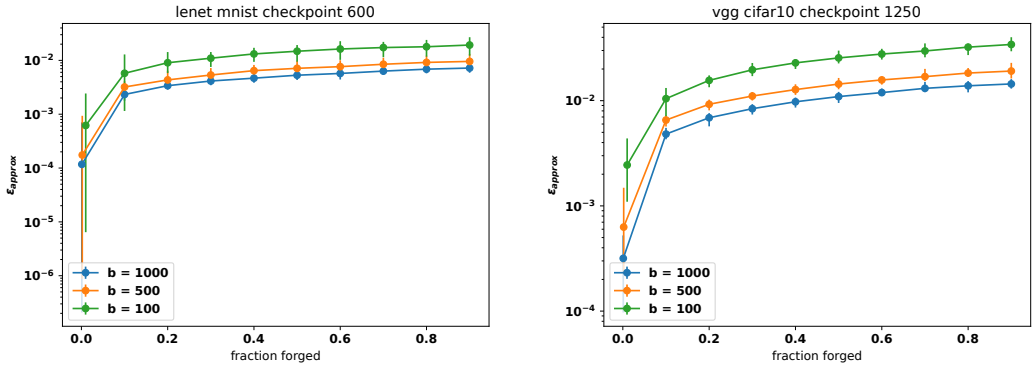
Table 4.3: The largest observed ϵ_{repr} when recomputation is done on different hardware. For a given execution trace setup (*i.e.*, each subtable), each row indicates the GPU the execution trace was generated on and the column indicates which GPU the trace was recomputed on. In Tab. 4.3b, the diagonal entries are zero because deterministic training results in zero error when recomputing on the same hardware.

4.4.2 Evaluating the Performance of Existing Data Forging Attacks: How Small are their Approximation Errors?

In this section, we perform data forging attacks against the execution traces we generated in the previous section, and report their approximation error. Our results highlight two important factors that affect the approximation error; namely *(i)* the forging fraction (*i.e.*, the number of examples in a mini-batch that need to be replaced), and *(ii)* the stage of training at which forging is taking place.

Smaller forging fractions result in smaller approximation errors. Recall that the goal of the adversary is to replace the original mini-batch B_i with a forged mini-batch B_i' such that $U \cap B_i' = \emptyset$. The difficulty of producing the forged mini-batch B_i' can be viewed as being proportional to the cardinality of $B_i \cap U$. If $\#(B_i \cap U) = 1$, *i.e.*, only one example needs to be forged, then the adversary can keep all the other $b-1$ examples the same in B_i' and only forge the single training example. For larger values of b, the effect of a single example on the average gradient $\nabla_\theta \mathcal{L}(B_i')$ becomes negligible; regardless of what the training example is replaced with, its effect on the average gradient is essentially "drowned out" by the other $b-1$ training examples that are shared between B_i and B_i'. Given a fixed number of training examples in B_i that needs to be replaced, for larger and larger batch sizes (*i.e.*, *smaller forging fractions* $\frac{\#(B_i \cap U)}{\#(B_i)}$), we expect the approximation error to get smaller. However, if $\#(B_i \cap U) = \#(B_i) = b$, then forging is much more difficult, as every example needs to be replaced. Figure 4.3 shows that this expected behaviour is indeed observed

when forging both LeNet/MNIST and VGGmini/CIFAR10 traces. We see that the attacks perform best (*i.e.*, produce the smallest approximation errors) when the forging fraction $\frac{\#(B_i \cap U)}{\#(B_i)} = \frac{1}{b}$, and degrade quickly with increased forging fractions. We also observe that Thudi et al. [71] and Zhang et al.[86]'s attacks have similar performance (see Appendix D for a comparison).



(a) LeNet/MNIST (b) VGGmini/CIFAR10

Figure 4.3: The average approximation error of Thudi et al.'s attack [71] (over several runs), as well as the min and max observed approximation error for different batch sizes and increasing forging fractions. Performance degrades quickly with increased forging fraction. Additionally, there is high variance when forging just one example. In both settings, we forge the middle checkpoint of the execution trace.

Forging performance is inconsistent across training. Data forging involves replacing all occurences of a given set training examples U in an execution trace with distinct examples that produce a similar gradient. Consequently, we postulate that the location of the mini-batches B_i such that $B_i \cap U \neq \emptyset$ can affect the peformance of these data forging attacks. Given a trace that captures E total epochs, mini-batches that contain a given example $(\mathbf{x}, \mathbf{y})$ will occur E times roughly uniformly across the execution trace. As a result, we must evaluate the performance of data forging attacks across training as every occurence of $(\mathbf{x}, \mathbf{y})$ in the trace must be forged. We run the following experiment to evaluate this:

1. For a given execution trace's training trajectory $\theta_0, \theta_1, \ldots, \theta_T$, we sample a true mini-batch B and calculate its model update $g(\theta_i, B)$ across the trajectory.

2. We then forge mini-batch B into B' across training at each checkpoint and measure the approximation error.

3. We repeat this several times for different sampled mini-batches in order to study the variance.

Firstly, we find that when the forging fraction is 1, approximation errors are 2 to 5 orders of magnitude larger than our largest observed reproduction errors for both

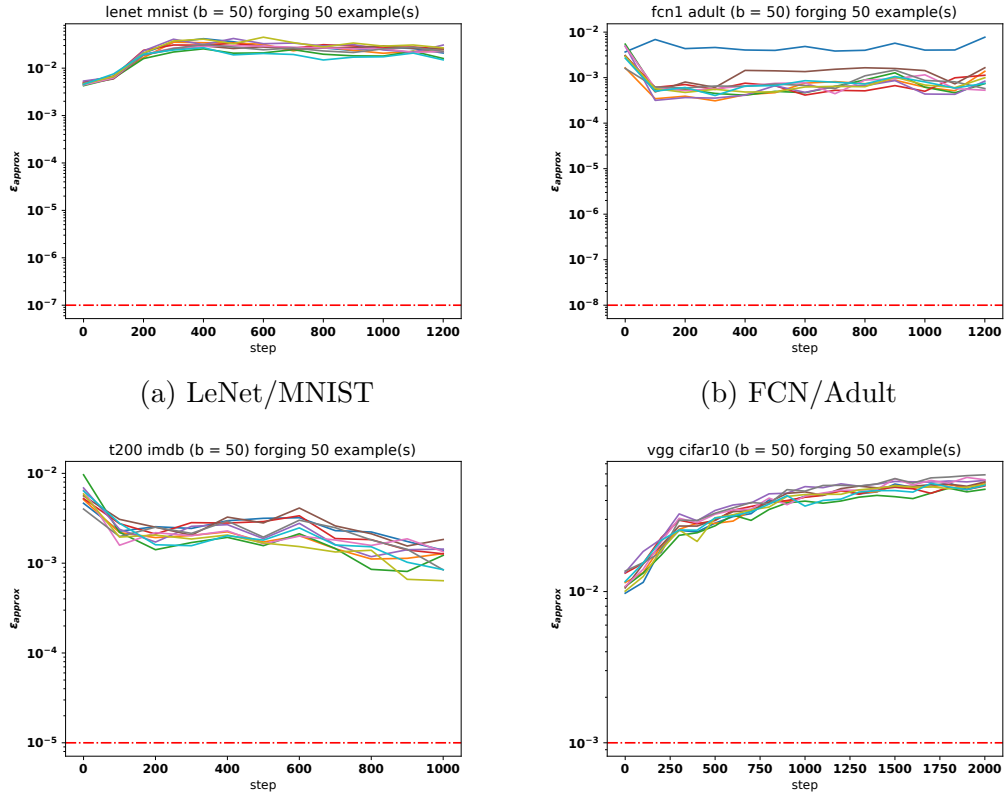
`float32` and `float16` formats. Approximation errors were closer to reproduction errors only for the `bfloat16` mixed precision regime, however they were still found to be at least one order of magnitude larger. In Figures 4.4, 4.7, and 4.9, we observe that the approximation errors are consistently orders of magnitude larger than the largest observed reproduction errors (shown as the red dotted line).

Now consider the case where the forging fraction is $\frac{1}{b}$. In Figures 4.5, 4.6, 4.8 we plot the approximation errors for a given training example $(\mathbf{x}, \mathbf{y})$ at uniform intervals across the execution trace (each line represents a different example). In this setting, we see that approximation errors can vary highly depending on both the example being forged and the position in the execution trace where it is being forged. For `bfloat` mixed precision training and a forging fraction of 1/100, we find that approximation errors of data forging attacks are closest to reproduction errors, being either the same magnitude or a single order of magnitude larger when forging single examples (see Fig. 4.8). However, the performance of the attack is not consistent across training, as seen in Figures 4.5, and 4.6, where approximation errors can vary from being only a single order of magnitude larger to nearly 5 orders of magnitude larger (see Fig. 4.5a).

We highlight the relationship between approximation error at a given checkpoint in training and the loss of the single example being forged at that same checkpoint in order to explain this observed difference in forging performance. Figures 4.10a,4.10b show for the LeNet/MNIST setup the approximation errors of data forging and the loss of the particular example being forged (each colour in both figures corresponds to the same training example). In these two figures we can see a direct correlation between the loss of the model on the particular example $(\mathbf{x}, \mathbf{y})$ being forged, and the approximation error. The smaller the loss, the smaller the norm of the example gradient $\nabla_\theta \ell(f_\theta(\mathbf{x}), \mathbf{y})$, the less the example contributes to the average gradient $\nabla_\theta \mathcal{L}(B)$. As a result, when forging such an example, data forging attacks find another example with a similarly small gradient norm and replace it. Since both gradient norms are small, replacing one with the other does not greatly affect the direction or norm of the average gradient, resulting in small approximation errors. However this is not the case when the norm of the example gradient is large, such as early in training. Data forging attacks cannot find a suitable replacement; resulting in larger approximation errors. We find this pattern holds across model architectures as well, see Figures 4.10c, and 4.10d for similar behaviour for a transfomer.

4.4.3 Is Data Forging Stealthy?

From our experimentation in Section 4.4.1, we found that the model type, batch size, floating point precision, and GPU hardware can affect the magnitude of benign



(a) LeNet/MNIST

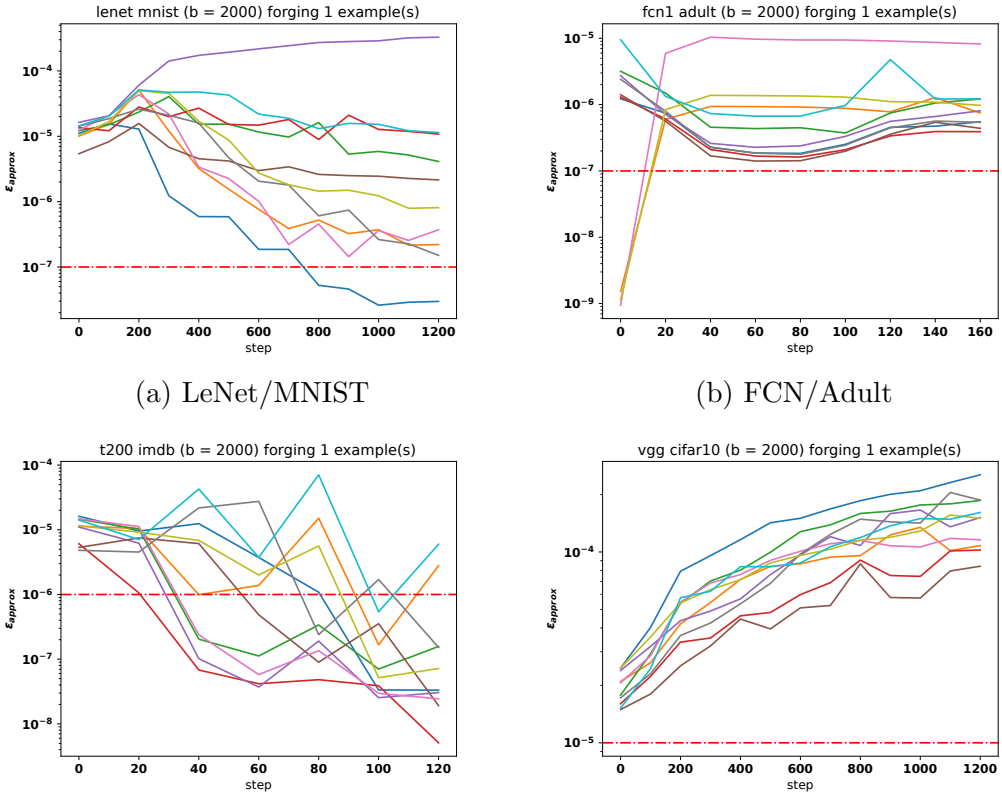
(b) FCN/Adult

(c) Trans/IMDB

(d) VGGmini/CIFAR10

Figure 4.4: Data forging approximation errors across stages of training where the forging fraction is 1 for `float32` training. Each line corresponds to a different true mini-batch B, and the y-axis reports the approximation error when trying to forge B at the given step on the x-axis.

reproduction errors, and in Section 4.4.2, we evaluated the performance data forging attacks. **From our evaluation of the existing data forging attacks, they cannot be classed as stealthy**. While factors such as forging fraction, the stage of training, and floating point precision can affect the performance of these attacks, overall they are not stealthy. In the case where the forging fraction is 1, data forging attacks consistently produce approximation errors that are several orders of magnitude larger than benign reproduction errors. When the forging fraction is $1/b$, we found that the performance is inconsistent across training. Depending on the value of the loss function for the particular example that is being forged, the approximation error can be close to reproduction error or be several orders of magnitude larger.



(a) LeNet/MNIST (b) FCN/Adult

(c) Trans/IMDB (d) VGGmini/CIFAR10

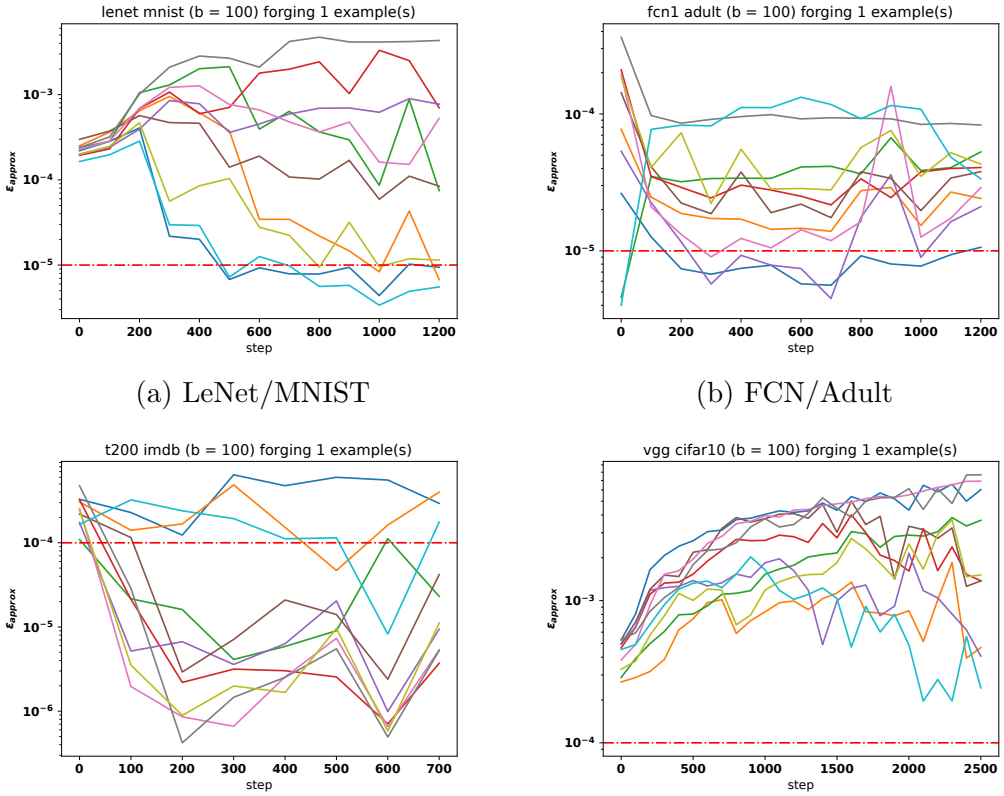
Figure 4.5: Data forging approximation errors across stages of training where the forging fraction is 1/2000 for `float32` training.

4.4.4 Discussion: Choosing An Error Threshold In Practice is Hard

From our experimentation, we have seen that existing attacks are not stealthy when the error threshold ϵ is chosen to be no larger than benign reproduction errors. When error thresholds are accurately chosen via thorough experimentation to determine the magnitude of reproduction errors, we have seen that data forging attacks are consistently not stealthy. As a result, we contend that such a method for determining the model's provenance *i.e.*, its true training data, is adequate. Existing forging attacks cannot produce forged execution traces that can fool an informed verifier.

However the approach is not perfect. Determining accurate error thresholds value can be tricky when it is not possible to perform cross hardware recomputation. Consider the following scenario that highlights the difficulty of correctly setting ϵ in practice: a verifier intends to verify a LeNet/MNIST ($b = 100$) execution trace on a RTX4090 GPU. They set $\epsilon = 10^{-8}$ for their error threshold as this is their observed level of reproduction error on their hardware[4]. Now suppose that the

[4]See the corresponding plot in Figure 4.2 for LeNet/MNIST ($b = 100$).



(a) LeNet/MNIST

(b) FCN/Adult

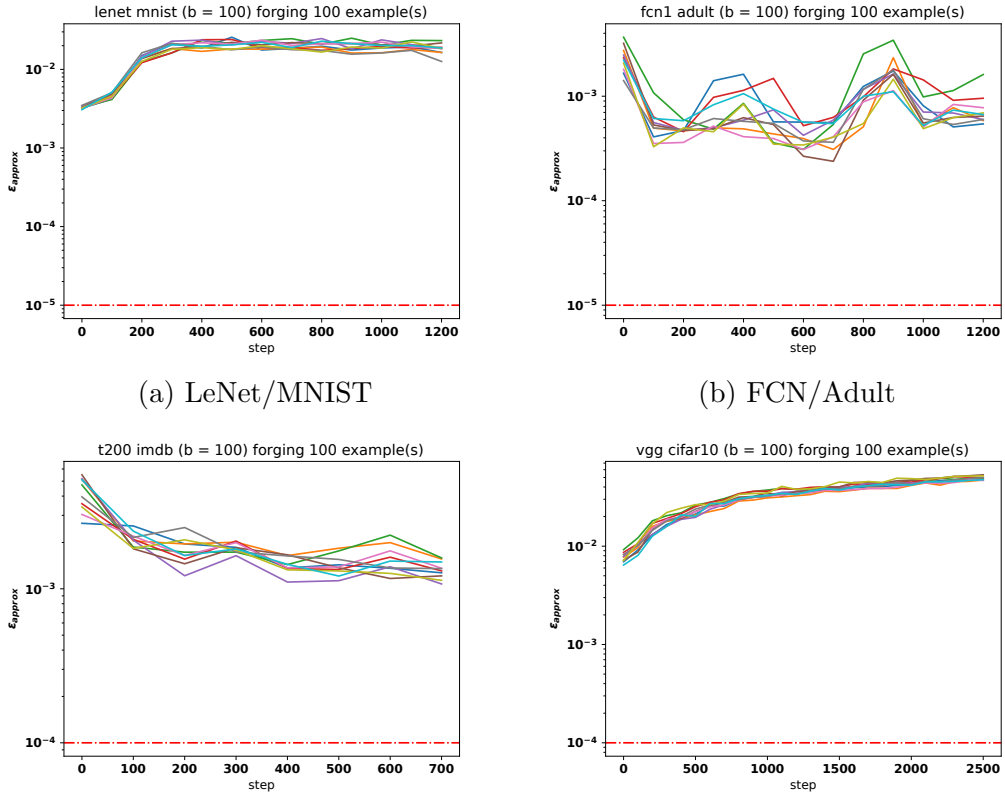
(c) Trans/IMDB

(d) VGGmini/CIFAR10

Figure 4.6: Data forging approximation errors across stages of training where the forging fraction is 1/100 for `float16` training.

model owner is honest and trained their model using a Tesla V100 GPU. From our experimentation, we found that the reproduction error of honest traces generated using a V100 and recomputed on a RTX4080 can be as large as 10^{-6} (See Table 4.3b). As a result, even the honest trace would not pass verification if $\epsilon = 10^{-8}$. One of the challenges of building robust verification schemes is that, as we have seen, reproduction errors can vary by several orders of magnitude depending on the hardware that the checkpoints were generated on, as well as the hardware they are reproduced on.

Within the data forging literature, there is a difference of opinion on what is an acceptable choice of ϵ. Baluta et al. [7] consider only the scenario of a strict verifier that always selects $\epsilon = 0$. They assert that given *(i)* the initial random seed(s) used during training, *(ii)* the original software, and *(iii)* the original hardware the computations were performed on, there should be zero error between recomputations of any given execution trace. They claim that since the entire computation sequence is deterministic, error at any step of reproduction of the execution trace must therefore point to foul play. While it is true that reproduction error can be eliminated given



(a) LeNet/MNIST

(b) FCN/Adult

(c) Trans/IMDB

(d) VGGmini/CIFAR10

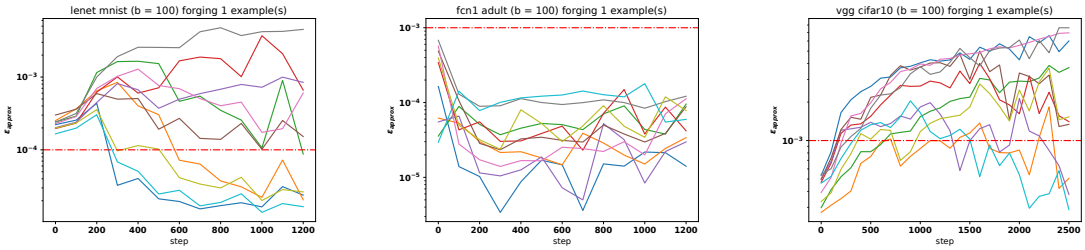
Figure 4.7: Data forging approximation errors across stages of training where the forging fraction is 1 for `float16` training.

these conditions[5] and may be necessary in high stakes applications such as medical machine learning models, we argue that always demanding $\epsilon = 0$ is unrealistic in practice, particularly when the original hardware is not available.

Due to the existence of reproduction error, prior data forging works [43, 71, 86] have recognised the practical necessity of a non-zero threshold during verification, and consider the $\epsilon > 0$ scenario. While their attacks produce non-zero approximation errors, the question of whether a verifier should accept the levels of approximation error generated by their attack algorithms is largely overlooked by the prior work. They provide no analysis of how large reproduction errors can be and whether their attack algorithms produce approximation errors that are comparable. In this work, we have conclusively determined that this is not the case; reproduction errors are orders of magnitude smaller than the approximation errors produced by existing attacks.

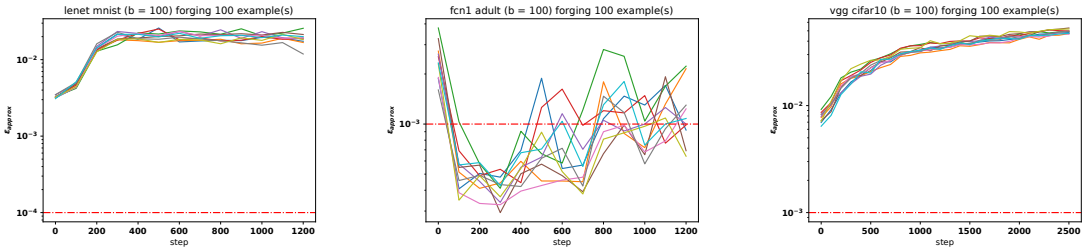
Verifiers are therefore met with a tradeoff when choosing their error threshold. Seeking to not reject honest traces, and being cognisant of the fact that there may exist other factors that affect reproduction errors; they set their threshold to be

[5]See Table 4.3b, where we observed zero reproduction error when the original random seeds, hardware, and software were used for the recomputation.



(a) LeNet/MNIST (b) FCN/Adult (c) VGGmini/CIFAR10

Figure 4.8: Data forging approximation errors across stages of training where the forging fraction is 1/100 for `bfloat16` training.



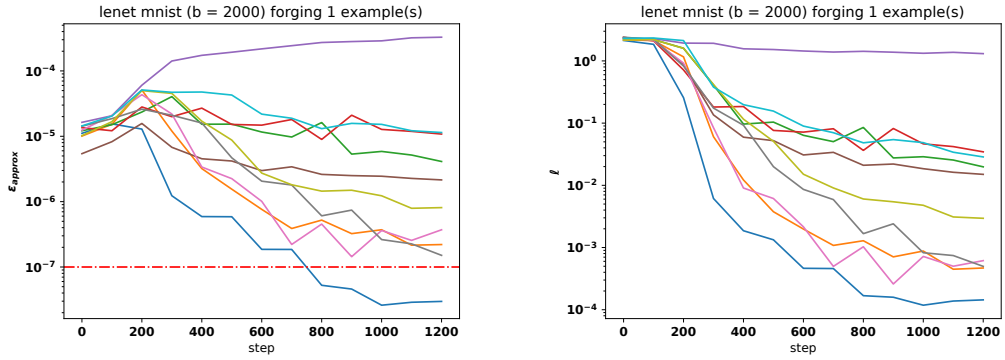
(a) LeNet/MNIST (b) FCN/Adult (c) VGGmini/CIFAR10

Figure 4.9: Data forging approximation errors across stages of training where the forging fraction is 1 for `bfloat16` training.

larger than the reproduction errors which they have observed through experimentation. However, in doing so, they allow for data forging attacks, that would have not been stealthy under a stricter choice of error threshold, to become stealthy, thus exploiting the verifier's pragmatism. A similar situation is seen in the area of Differential Privacy (DP) [22]. A privacy parameter ϵ is chosen to determine how private[6] a given function is. Within DP, ideally $\epsilon < 1$, however, choosing such a small epsilon can hinder the performance of the model. We have a similar setup here, where $\epsilon = 0$ gives the verifier the most confidence in their reproductions, however this may not be practical all the time, necessitating the choice of larger error thresholds. Large error thresholds ϵ, larger batch sizes b, mixed precision training, and smaller forging fractions allow for more *wiggle room*, increasing the likelihood of potential foul play from data forging.

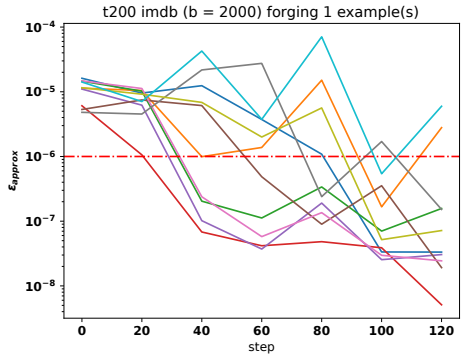
Data Protection, Differential Privacy, and Data Forging. Data protection legislation such as GDPR aims to limit the potential harm that may befall an individual from the use of their personal information. Existing methods to do so are *(i)* to limit the use of their personal data entirely, or *(ii)* to provide guarantees that

[6]Privacy in DP is quantified by how much the output of a function differs for inputs that differ by one datapoint. If they do not differ by much, then inferring whether a datapoint was or was not part of the input is difficult. This difficulty of inference is inversely proportional to the privacy parameter ϵ.

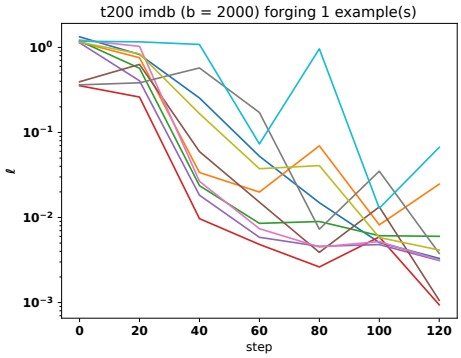


(a) LeNet/MNIST

(b) LeNet/MNIST



(c) Trans/IMDB



(d) Trans/IMDB

Figure 4.10: The correlation between the approximation error when forging a single example across an execution trace and the loss of that example at a given checkpoint.

any undue harm will not result from the use of their data. Regarding the latter, the differential privacy framework has emerged as the state of the art for provable guarantees within machine learning, however it is not without its tradeoffs. Namely, providing meaningful privacy guarantees means poor model performance, resulting in concessions in order to acheive better utility, such as relying on "public" pre-training data [49] to privately fine-tune LLMs, or the use of empirical defences (e.g. DPSGD with large values for epsilon) in order to thwart existing attacks [5]. The former forgoes privacy guarantees for the "public" data LLMs are pre-trained on, data whose use often was not consented to and is liable for recovery from these models [17, 18, 12], and the latter provides no protection against potentially stronger attacks.

The emergence of Proof of Learning (PoL) [39], i.e., the recomputation of execution traces, and further zero-knowledge methods of PoL [4] can be viewed as an alternate approach to limit harm by providing proof of exactly what a model was trained on, and more importantly, what data the model was not trained on. However this is also not without its tradeoffs. In order to thwart data forging, accurately chosen error thresholds are needed to prevent the forged traces from being accepted

and valid traces from being rejected. Our work has shown this requires thorough experimentation to determine accurately. Additionally, closed-source GPU driver code makes this task harder. Further, proving a model's training data requires the recomputation of every step which in some use cases may not be possible e.g., large transformer models on web-scale datasets. Further research into less computationally intensive schemes is necessary [25, 39] for PoL to become more practical, as well as the utilisation of open-source, reproducible floating point algorithms.

4.5 On the Difficulty of Exact Data Forging

Existing data forging attacks attempt to find, for a given mini-batch B, a distinct counterpart B' such that their gradients are *approximately* the same. In previous sections, we have examined how similar the gradients of the original and forged mini-batches produced by existing "greedy search" style attacks [43, 71, 86] are, and found that the magnitude of their approximation errors are too large, allowing us to classify existing data forging attacks as not stealthy. However, little work has been devoted to *exact* data forging *i.e.*, developing data forging attacks that produce forged mini-batches with the *same* gradient. The related subject of gradient inversion [93, 58, 85, 88, 84, 92] asks the reverse question: *given the gradient $\nabla_\theta \mathcal{L}(B)$, can the original mini-batch B be recovered?* This question is pertinent to the area of Federated Learning [51], as the gradients of sensitive information are sent among several parties; and an analysis of the privacy impact of sharing these gradients is required. Within data forging, we are concerned with a related question: *given mini-batch B and its gradient $\nabla_\theta \mathcal{L}(B)$, does there exist another, distinct, mini-batch B' such that $\nabla_\theta \mathcal{L}(B) = \nabla_\theta \mathcal{L}(B')$?* Any error when reproducing the gradients of such a B and B' can only ever be at most, as large as reproduction error, resulting in stealthy data forging. Consequently, this question is of great importance to an adversarial model owner looking to perform stealthy data forging.

A key limitation of prior data forging attacks is the usage of the finite search space $D \backslash U$ when searching for replacements. Baluta et al. [7] analysed the models and datasets considered by the prior work. They find that the execution traces considered by the prior work are *unforgeable* using existing techniques *i.e.*, these attacks cannot produce a forged mini-batch with, analytically, the same gradient. However, there may exist training examples outside $D \backslash U$ that result in zero error. This section is devoted to answering the question of exact data forging *i.e.*, the search for mini-batches outside $D \backslash U$ that result in zero error. Our analysis suggests that while it is possible to construct a distinct mini-batch B' that produces the same gradient, analytically, as B, finding a B' within the allowed input space $\mathcal{X}$ and label space $\mathcal{Y}$ is a non trivial task. In particular, exact data forging involves solving a given

system of linear equations, where each solution corresponds to a forged mini-batch B'. However, of the infinite, distinct, solutions that may exist, there are currently no known efficient techniques for finding solutions that correspond to a mini-batch that falls within the allowed domain $\mathcal{X} \times \mathcal{Y}$.

4.5.1 Exact Data Forging

Within the classification context, we have a model, *i.e.*, parameterised function $f_\theta : \mathcal{X} \to \mathcal{Y}$ and loss function $\ell : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$. Exact data forging may be stated as the problem of finding two distinct mini-batches $B, B' \in \mathcal{Z}^b$, where $\mathcal{Z} := \mathcal{X} \times \mathcal{Y}$, such that $\nabla_\theta \mathcal{L}(B) = \nabla_\theta \mathcal{L}(B')$, where $\mathcal{L}(B) = \frac{1}{b} \sum_{(\mathbf{x},\mathbf{y}) \in B} \ell(f_\theta(\mathbf{x}), \mathbf{y})$. We define distinct mini-batches as follows, ensuring that they differ in at least one training example:

Definition 1 (Distinct mini-batches)**.** *Two mini-batches $B, B' \in \mathcal{Z}^b$ are distinct if $\exists(\boldsymbol{x}, \boldsymbol{y}) \in B$ such that $\forall(\boldsymbol{x}', \boldsymbol{y}') \in B',\ (\boldsymbol{x}, \boldsymbol{y}) \neq (\boldsymbol{x}', \boldsymbol{y}')$.*

In the unrestricted setting, the training examples $(\mathbf{x}, \mathbf{y}) \in B$ may take any real numbered value i.e. $\mathcal{Z} = \mathbb{R}^d \times \mathbb{R}^n$ (d and n are the number of input features and classes respectively). However, depending on the given machine learning application, the domain may be restricted. Let us consider the image classification context, as done in the prior work, where inputs consist of pixel values and the labels are one hot vectors. There exist a finite number of pixel values v and n possible one hot vectors. Modern image formats have $v = 256$, so the domain of possible mini-batches is restricted to the set $\mathcal{Z} = \{\frac{q}{255} : q \in [0..255]\}^d \times \{u \in \{0,1\}^n : \sum_{i=1}^n u_i = 1\}$. We see that this domain is finite, and so one potential exact data forging approach is to enumerate every possible batch and search exhaustively for the one that produces the required gradient. However, this approach is computationally intractable for non-trivial values of b, d and n with no additional guarantee that one may be found[7] (see Appendix C for how exact data forging via brute-force fails for small values of b, d, v and n). Next we consider the task of exact data forging fully connected neural networks first for $b = 1$, and then consider the case for $b > 1$.

4.5.2 Case for $b = 1$

For $b = 1$ we prove that exact data forging is indeed not possible for fully connected neural networks. A L-layer fully connected neural network is made up of parameters that consist of weights and biases $\theta = [\mathbf{W}_1, \mathbf{b}_1 \mathbf{W}_2, \mathbf{b}_2, ..., \mathbf{W}_L, \mathbf{b}_L]$ where each $\mathbf{W}_i \in \mathbb{R}^{d_{i-1} \times d_i}, \mathbf{b} \in \mathbb{R}^{d_i}$. The output of the neural network is given by $f_\theta(\mathbf{x}) = \text{softmax}(\mathbf{z}_L)$, where $\mathbf{z}_L = \mathbf{W}_L^T \mathbf{a}_{L-1} + \mathbf{b}_L$, $\mathbf{a}_i = h(\mathbf{z}_i)$, $\mathbf{z}_i = \mathbf{W}_i^T \mathbf{a}_{i-1} + \mathbf{b}_i$, and h is the activation

[7] There exist a total of $\binom{256^d \times n}{b}$ possible mini-batches of size b.

function. Let $n = d_L$ denote the number of classes, $d = d_0$ the number of input features, and $\mathbf{a}_0 = \mathbf{x} \in \mathbb{R}^d$. The per example crossentropy loss ℓ is given by $\ell(\hat{\mathbf{y}}, \mathbf{y}) = -\sum_{i=1}^{n} y_i \log \hat{y}_i$. Observe in the $L = 1$ case, the network is equivalent to multi-class logistic regression.

Proposition 1 ($b = 1$ Data Forging Fully Connected Neural Networks)**.** *Let* $\mathcal{Z} := \mathbb{R}^d \times \{u \in \{0,1\}^n : \sum_{i=1}^{n} u_i = 1\}$. *For any two training examples* $(\boldsymbol{x}, \boldsymbol{y}), (\boldsymbol{x}', \boldsymbol{y}') \in \mathcal{Z}$ *if* $\nabla_\theta \ell(f_\theta(\boldsymbol{x}), \boldsymbol{y}) = \nabla_\theta \ell(f_\theta(\boldsymbol{x}'), \boldsymbol{y}')$, *then* $\boldsymbol{x} = \boldsymbol{x}'$ *and* $\boldsymbol{y} = \boldsymbol{y}'$.

Proposition 1 states that no two distinct training examples with one hot labels can produce the same gradient, an intuitive result and one that may point towards the difficulty of exact data forging. Further, proposition 1 proves the stronger result of impossibility of exact data forging when $b = 1$ for distinct inputs in the infinite domain $\mathbf{x}, \mathbf{x}' \in \mathbb{R}^d$, not just the finite set of valid discrete-valued images (see appendix A.1 for the full proof). However, models are typically trained with a batch size $b \gg 1$, therefore an investigation of the possibility of exact data forging within this regime is required.

4.5.3 Case of $b > 1$

In contrast to the previous setting, we prove that there can exist distinct mini-batches of size $b > 1$ that produce the same gradient.

Proposition 2 ($b > 1$ Data Forging Fully Connected Neural Networks)**.** *Given a fully connected neural network and mini-batch* $B = \{(\boldsymbol{x}^{(k)}, \boldsymbol{y}^{(k)})\}_{k=1}^{b}$, *if* $db > d_1(d+b)$, *where* d_1 *is the size of the first hidden layer, then there exists an infinite number of perturbation matrices* $\boldsymbol{P} \in \mathbb{R}^{d \times b}$ *such that for the distinct mini-batch* $B' = \{(\boldsymbol{x}^{(k)} + [\boldsymbol{P}]^k, \boldsymbol{y}^{(k)})\}_{k=1}^{b}$, $\nabla_\theta \mathcal{L}(B) = \nabla_\theta \mathcal{L}(B')$.

Proposition 2 proves the existence of perturbations that can be applied to the original training examples $(\mathbf{x}, \mathbf{y}) \in B$ such that the same gradient is still preserved (see Fig. 4.11 for an example). For simplicity, we describe the construction of these perturbations in the case of a $L = 1$ layer network with no bias (see appendix A.3 for a full proof of proposition 2 and description of the construction of these perturbations for arbitrary depth fully connected neural networks with or without biases).

Consider a $L = 1$ network (*i.e.*, equivalent to multi-class logistic regression) with a single parameter matrix $\mathbf{W} \in \mathbb{R}^{d \times n}$. Let $\mathbf{X} \in \mathbb{R}^{d \times b}$ represent the input matrix for mini-batch B *i.e.*, the k-th column of $\mathbf{X}$ is the k-th training example input $\mathbf{x}^{(k)}$ from B, let $\mathbf{Y} \in \mathbb{R}^{n \times b}$ be the corresponding label matrix, and vector $\mathbf{v} \in \mathbb{R}^b$ such that $v_k = \sum_{i=1}^{n} y_i^{(k)}$. Additionally, let $f_{\mathbf{W}}(\mathbf{X}) = \text{softmax}(\mathbf{W}^T \mathbf{X})$ be the batched

softmaxed output[8] of the model. For crossentropy loss, the gradient is given by $\nabla_{\mathbf{W}}\mathcal{L}(B) = \frac{1}{b}\mathbf{X}(f_{\mathbf{W}}(\mathbf{X})\text{diag}(\mathbf{v}) - \mathbf{Y})^T$.

Exact data forging reduces to the problem of finding two matrices $\mathbf{X}' \in \mathbb{R}^{d\times b}, \mathbf{Y}' \in \mathbb{R}^{n\times b}$ where $\mathbf{X}' \neq \mathbf{X}$ such that $\mathbf{X}(f_{\mathbf{W}}(\mathbf{X})\text{diag}(\mathbf{v}) - \mathbf{Y})^T = \mathbf{X}'(f_{\mathbf{W}}(\mathbf{X}')\text{diag}(\mathbf{v}') - \mathbf{Y}')^T$. Having done so, the forged mini-batch B' may be constructed from the columns of $\mathbf{X}'$ and $\mathbf{Y}'$, namely the k-th forged example $\mathbf{x}'$ is the k-th column of $\mathbf{X}'$, and likewise for the labels.

The *mini-batch perturbation* approach to exact data forging produces forged mini-batches with the same labels as the original *i.e.*, $\mathbf{Y}' := \mathbf{Y}$, however we construct a distinct input matrix $\mathbf{X}' \neq \mathbf{X}$. In particular, $\mathbf{X}'$ is to set $\mathbf{X}' := \mathbf{X} + \mathbf{P}$, where $\mathbf{P} \in \mathbb{R}^{d\times b}$ is some non-zero matrix that *perturbs* $\mathbf{X}$ in such a way that maintains the same gradient. For such a matrix to do so, it must satisfy the following equations

$$\begin{cases} \mathbf{W}^T\mathbf{P} = \mathbf{0} \\ \mathbf{P}(f_{\mathbf{W}}(\mathbf{X})\text{diag}(\mathbf{v}) - \mathbf{Y})^T = \mathbf{0}. \end{cases} \tag{4.1}$$

If $\mathbf{P}$ satisfies the first equation $\mathbf{W}^T\mathbf{P} = \mathbf{0}$, then

$$\begin{aligned} f_{\mathbf{W}}(\mathbf{X}') &= \text{softmax}(\mathbf{W}^T(\mathbf{X} + \mathbf{P})) \\ &= \text{softmax}(\mathbf{W}^T\mathbf{X} + \mathbf{W}^T\mathbf{P})) \\ &= \text{softmax}(\mathbf{W}^T\mathbf{X}) \\ &= f_{\mathbf{W}}(\mathbf{X}) \end{aligned} \tag{4.2}$$

. In other words, satisfying the first equation ensures that the original and forged mini-batch produce the same output. If $\mathbf{P}$ satisfies both equations, we then have

$$\begin{aligned} &b\nabla_{\mathbf{W}}\mathcal{L}(B') = \mathbf{X}'(f_{\mathbf{W}}(\mathbf{X}')\text{diag}(\mathbf{v}') - \mathbf{Y}')^T \\ &= (\mathbf{X} + \mathbf{P})(f_{\mathbf{W}}(\mathbf{X}')\text{diag}(\mathbf{v}') - \mathbf{Y}')^T \\ &= (\mathbf{X} + \mathbf{P})(f_{\mathbf{W}}(\mathbf{X})\text{diag}(\mathbf{v}) - \mathbf{Y})^T \\ &= \mathbf{X}(f_{\mathbf{W}}(\mathbf{X})\text{diag}(\mathbf{v}) - \mathbf{Y})^T + \mathbf{P}(f_{\mathbf{W}}(\mathbf{X})\text{diag}(\mathbf{v}) - \mathbf{Y})^T \\ &= \mathbf{X}(f_{\mathbf{W}}(\mathbf{X})\text{diag}(\mathbf{v}) - \mathbf{Y})^T = b\nabla_{\mathbf{W}}\mathcal{L}(B) \end{aligned} \tag{4.3}$$

Thus, satisfying the two equations given in 4.1 ensures that the gradients for the original and forged mini-batch match.

In order to construct $\mathbf{P}$, we first observe that we can write the two equations as $\mathbf{I}_d\mathbf{P}\mathbf{D} = \mathbf{0}$ and $\mathbf{W}^T\mathbf{P}\mathbf{I}_b = \mathbf{0}$, where $\mathbf{I}_k$ is the identity matrix of size k, and $\mathbf{D} = (f_{\mathbf{W}}(X)\text{diag}(\mathbf{v}) - \mathbf{Y})^T$. Using the identity $AXB = C \iff (B^T \otimes A)\text{vec}(X) = \text{vec}(C)$, we can then rewrite our two equations as their corresponding linear systems and combine them into the following single system of linear equations:

[8]Softmax is applied along the columns of $\mathbf{W}^T\mathbf{X}$.

$$\begin{bmatrix} \mathbf{D}^T \otimes \mathbf{I}_d \\ \mathbf{I}_b \otimes \mathbf{W}^T \end{bmatrix} \text{vec}(\mathbf{P}) = \mathbf{0} \tag{4.4}$$

Let $\mathbf{K} \in \mathbb{R}^{n(b+d) \times bd}$ be the above coefficient matrix, then the general solution is given by:

$$\text{vec}(\mathbf{P}) = (\mathbf{I} - \mathbf{K}^+\mathbf{K})\mathbf{F}, \tag{4.5}$$

where $\mathbf{F} \in \mathbb{R}^{bd}$ is an arbitrary column matrix, and $\mathbf{K}^+$ is the psuedo-inverse of $\mathbf{K}$.

For the homogenous linear system $\mathbf{K}\text{vec}(\mathbf{P}) = \mathbf{0}$, there are $bd - \text{rank}(\mathbf{K})$ linearly independent solutions. If $\mathbf{K}$ is full rank and square, there are no non-trivial solutions. In order to ensure at least 1 set of linearly dependent solutions (e.g. one free variable) $bd > n(b + d)$ must hold. This ensures that $bd - \text{rank}(\mathbf{K}) \geq 1$. This is a worst case condition on the existence of non-trivial solutions as $\mathbf{K}$ may not necessarily be full rank.

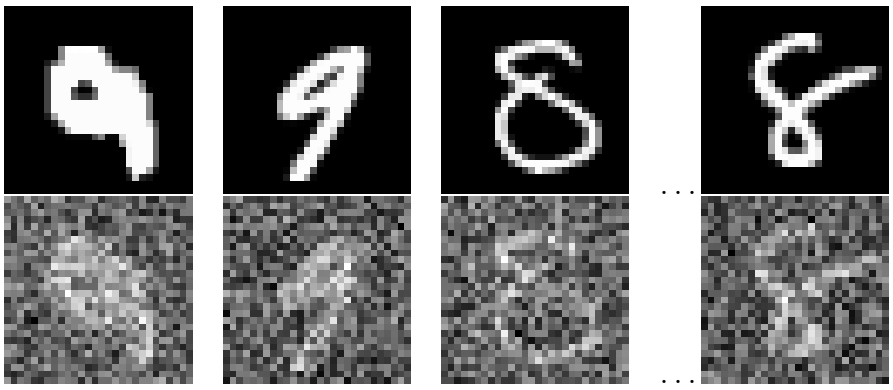


Figure 4.11: **Bottom:** The original mini-batch B consisting of $b = 32$ examples from MNIST. **Top:**. A forged mini-batch B' with an approximation error of $\approx 10^{-7}$. In this setting, the model is an $L = 2$ ReLU neural network with 8 hidden units. Similar mini-batches were constructed by Pan et al. [58] within the context of gradient inversion.

Exact data forging can be performed using this *mini-batch perturbation* approach; see Figure 4.11 for an example. However, while the gradients match, the forged input examples $\mathbf{x}'^{(k)}$ are not guaranteed to be members of the set of valid images $\{\frac{q}{255}, q \in [0..255]\}^d$; the applied perturbation may result in a mini-batch that is outside the allowed input domain. In order to ensure this does not happen, one needs to find one of the infinitely many possible perturbations **i.e.**, solutions to equation (4.5), that preserves image validity. This requires choosing the correct arbitrary column matrix $\mathbf{F}$. Efficiently finding this matrix is a non trivial task, and is currently the main barrier to successful exact data forging. In fact, the question

of whether such matrices may exist is also still unresolved, even for relatively simple linear models.

4.5.4 Exact Data Forging for $L = 1$ Networks

Recall earlier that for a $L = 1$ network with single parameter matrix $\mathbf{W}$, the gradient of crossentropy loss is given by $\nabla_{\mathbf{W}}\mathcal{L}(B) = \frac{1}{b}\mathbf{X}(f_{\mathbf{W}}(\mathbf{X})\text{diag}(\mathbf{v})-\mathbf{Y})^T$, and that exact data forging reduces to the problem of finding two matrices $\mathbf{X}' \in \mathbb{R}^{d\times b}, \mathbf{Y}' \in \mathbb{R}^{n\times b}$ where $\mathbf{X}' \neq \mathbf{X}$ such that $\mathbf{X}(f_{\mathbf{W}}(\mathbf{X})\text{diag}(\mathbf{v}) - \mathbf{Y})^T = \mathbf{X}'(f_{\mathbf{W}}(\mathbf{X}')\text{diag}(\mathbf{v}') - \mathbf{Y}')^T$.

Proposition 3 ($L = 1$ Data Forging)**.** *If $b > rank(\nabla_{\boldsymbol{W}}\mathcal{L}(B))$, then there exists an infinite number of distinct mini-batches $B' \subset \mathbb{R}^d \times \mathbb{R}^n$ of size b such that $\nabla_{\boldsymbol{W}}\mathcal{L}(B') = \nabla_{\boldsymbol{W}}\mathcal{L}(B)$.*

Proposition 3 (see appendix A.4 for the proof) extends the mini-batch perturbation approach for $L = 1$ networks, showing that there can exist two distinct mini-batches with the same gradient that are not necessarily a perturbation of the other, and do not share the same labels (see Fig. 4.12 for an example). We prove this with the following steps:

Let matrix $\mathbf{D} = (f_{\mathbf{W}}(\mathbf{X})\text{diag}(\mathbf{v})-\mathbf{Y})^T$ be the error matrix of the model on input $\mathbf{X}$. Our first method of exact data forging exploits a fundamental property of this matrix (for proof see Appendix A.2).

Lemma 1. *Let $\boldsymbol{D} = (f_{\boldsymbol{W}}(\boldsymbol{X})diag(\boldsymbol{v}) - \boldsymbol{Y})^T$. For any two matrices $\boldsymbol{X} \in \mathbb{R}^{d\times b}$ and $\boldsymbol{Y} \in \mathbb{R}^{n\times b}$, we have that $\sum_{j=1}^{n} \boldsymbol{D}_{ij} = 0$, $\forall i,\ 1 \leq i \leq b$.*

Using this property, we may devise the following *error matrix sampling* approach towards exact data forging for $L = 1$ networks. Given mini-batch B of size b and gradient $\nabla_{\mathbf{W}}\mathcal{L}(B)$ of rank r:

1. Sample any matrix $\mathbf{D} \in \mathbb{R}^{b\times n}$ of such that $\sum_{j=1}^{n} \mathbf{D}_{ij} = 0$ and $\text{rank}(\mathbf{D}) \geq r$.
2. Solve for $\mathbf{X}'$ the matrix equation $\mathbf{X}'\mathbf{D} = b\nabla_{\mathbf{W}}\mathcal{L}(B)$.
3. Set $\mathbf{Y}' := f_{\mathbf{W}}(\mathbf{X}')\text{diag}(\mathbf{v}) - \mathbf{D}^T$.
4. Construct forged mini-batch $B' = \{(\mathbf{x}'^{(k)}, \mathbf{y}'^{(k)})\}_{k=1}^{b}$ where each $\mathbf{x}'^{(k)}, \mathbf{y}'^{(k)}$ are the k-th column of $\mathbf{X}', \mathbf{Y}'$ respectively.

Figure 4.12 gives two such examples of distinct mini-batches that produce the same gradient for both the MNIST and CIFAR10 datasets for an $L = 1$ network. The above error matrix sampling approach is not restricted to constructing forged mini-batches with the same labels as the original, proving the existence of distinct mini-batches with distinct labels can produce the same gradient. The infinite

number of possible error matrices $\mathbf{D}$ as well as the potentially infinite solutions to equation $\mathbf{X}'\mathbf{D} = b\nabla_{\mathbf{W}}\mathcal{L}(B)$ capture all the distinct mini-batches $B' \subset \mathbb{R}^d \times \mathbb{R}^n$ such that $\nabla_{\mathbf{W}}\mathcal{L}(B) = \nabla_{\mathbf{W}}\mathcal{L}(B')$. However, analogous to the mini-batch perturbation approach, choosing the correct error matrix $\mathbf{D}$ such that the mini-batch B' that is constructed from the resulting $\mathbf{X}'$ and $\mathbf{Y}'$ falls within the required domain is a non trivial, computationally infeasible task.

Our analysis takes the first step towards understanding exact data forging. Within the unrestricted mini-batch domain, we prove that exact data forging is indeed possible and provide methods of constructing forged mini-batches with approximation errors that are indistinguishable from reproduction errors. Further research and analysis are required in order to determine whether data forging within these restricted domains is possible. Given the difficulty of this task, we conjecture that it may not be computationally feasible.

Figure 4.12: A mini-batch from MNIST (top) and CIFAR10 (bottom) and distinct forged counterpart produced by our error matrix sampling approach. Both examples produce an approximation error on the order of 10^{-7}, 5 orders of magnitude smaller than existing attacks.

4.6 Related Work

4.6.1 Proof of Learning

Jia et al. [39] observed that the production and verification of an execution trace is tantamount to proving one has trained model θ_T on dataset $D = \bigcup_{i=0}^{T-1} B_i$. They formalise execution traces as *Proof of Learning* (PoL) sequences, and verify the final model θ_T is indeed the result of the optimisation captured in the trace by recomputing the checkpoints using the provided mini-batches and previous checkpoint, ensuring that each recomputed checkpoint is within some pre-defined distance threshold ϵ, e.g. ℓ_2 norm, from what is reported in the trace.

Implicit within the definition of PoL is the uniqueness of execution traces for a given final model θ_T, and that the construction of a distinct trace for θ_T that passes verification is at least as difficult a task as training. Consequently, Jia et al. consider PoL as a viable means of proving a third party has indeed expended the energy to train the model and therefore owns it. Constructing distinct traces that challenge this assumption is known as PoL "spoofing", i.e., producing a trace that differs from the original by employing a different sequence of mini-batches that follow a different training trajectory, but ultimately end at the same final model θ_T. Given such a spoofed execution trace S' for original trace S, let D and D' be the differing datasets used in S and S' respectively. One immediate consequence of successful PoL spoofing regards the training examples $(\mathbf{x}, \mathbf{y}) \notin D \cap D'$; namely the question arises of *"which are the real training examples that trained θ_T"*? PoL spoofing results in the apparent contradiction that a model was both trained and not trained on some set of training examples, in effect *deleting* them from a model's training set.

4.6.2 PoL Adversarial Examples

Zhang et al. [87] challenged the underlying assumption of PoL by constructing "adversarial examples" to PoL *i.e.,* two distinct execution traces for the same final model θ_T. Zhang et al.'s proposed adversarial attack operates by initialising dummy checkpoints such that the distance between consecutive checkpoints is less than the chosen verification distance threshold ϵ, and optimise "random" noise to add to mini-batches consisting of public data such that the gradient is close to zero, ensuring the next model checkpoint lies within the ϵ-ball of the next dummy checkpoint, and thus passing verification. However, Fang et al. [25] find that they are fragile to the hyperparameters of the verification process. Namely, under small checkpointing intervals, the attacks break down. Additionally, Zhang et al. apply their attack to

image classification models, and there is no mention nor guarantee of whether the final synthesized images are still within the problem domain *e.g.,* whether the final pixel values are within $0 - 255$. Any training examples outside the problem domain in the mini-batches of an execution trace immediately point to a detectable spoof. Fang et al. provide an "infinitesimal update" attack against PoL that is conceptually similar to Zhang et al.'s, where a learning rate is chosen to be sufficiently small such that regardless of the training data, the next model checkpoint lies within the ϵ-ball of the dummy checkpoint. The success of these attacks that exploit the static verification threshold and their consequences on the robustness of PoL verification is discussed in [25]. One limitation of these attacks is that they require foreknowledge of the chosen ϵ by the verifier, something that may not be realistic in practice, and the utilisation of infinitesimally small learning rates may also be suspect to a verifier.

Existing PoL adversarial attacks [87, 25] exploit near zero norm gradients to produce distinct execution traces that can pass PoL verification. However it remains an open question whether two distinct execution traces with distinct checkpoints may be constructed for model θ_T with valid, honest gradient updates that do not require near zero norm gradients. Fang et al. evaluate attempts at doing so via data ordering attacks [68] *i.e.,* finding a distinct sequence of valid mini-batches that take random initialisation θ_0 to the desired θ_T; however these were also found to be unlikely to succeed.

4.7 Conclusion

Answers to the question of exact data forging within the restricted problem domain will have wide reaching ramifications for the machine learning and privacy community. Proving its feasibility and the development of methods of performing such a task allows for stealthy data forging; an attack which fully realises the serious data privacy and security implications recognised by the prior work [43, 71, 86]. Membership inference attacks are the bread-and-butter of machine learning privacy. They are used to perform ML auditing and compliance, two tasks that are key in any real world deployment of an ML model. Additionally, machine unlearning is currently the only adequate response to GDPR's *right to be forgotten*, allowing model owners to comply with the law without needing to throw away the entire model. Stealthy data forging has the potential to disrupt these two tools, casting doubt on their methodology and veracity. These are serious implications for the ML privacy community, however, they have not been fully realised by current data forging attacks. Further research is needed on whether stealthy data forging is indeed possible; and whether more powerful attack algorithms exist. We have begun by first conclusively determining that current attacks are not stealthy and are detectable, cancelling any

implications they may have on ML privacy. Additionally, our theoretical analysis takes a first step towards understanding data forging and analysing the conditions where different mini-batches can produce the same update.

Chapter 5

Conclusion

This thesis has been a two part study into methods of privacy preservation in machine learning systems. The first part, consisting of Chapters 2 and 3, provides a critical investigation of the privacy afforded by federated learning. In Chapter 2, we developed an attack against GBoard's FL system, allowing for the reconstruction of typed user words and sentences. This presents a real and pressing threat to user privacy, as participation in Gboard FL system can lead to disclosure of the potentially highly sensitive information contained in what users type on their phones, from web searches, text messages, private notes, etc. We have also seen that existing defences against our attack must choose between the trade-off of training a useful model and maintaining user privacy. Additionally, In Chapter 3, we address the privacy benefit of federated learning and highlight the inordinate amount of trust required by participants in potentially malicious third parties in the FL process for it to be truly private. We identify several key factors such as model architecture and the wider FL system's implementation that can affect the privacy of FL. This research motivates further research into how to reduce the necessary trust required for private learning.

Concluding our study of FL, in Chapter 4, we move on to questions of model provenance. These are important with respect to privacy because model provenance techniques attempt to certify that a model is the result of a given sequence of computations, using a given dataset. We re-affirm the capability of computational proof based methods of model provenance by scrutinising recent forging attacks that have been developed that appear to jeopardise model provenance via computational proofs. We closely analyse their performance and find that they are not strong enough to realise their threat to governance. We also provide strong theoretical evidence that developing such strong forging attacks is computationally infeasible. Future work may look into improving the efficiency of computational proofs as well as furthering our theoretical analysis of data forging more broadly, not only the feasibility of forging techniques but also other methods of adversarial attacks against computational proofs.

Appendix A

Proofs

A.1 Proof of Proposition 1

We first introduce the background, namely the neural network setup, the definition of the loss function, and derive expressions for the gradients of the loss function with respect to the network parameters. Then, we introduce and prove some lemmas that are needed for the full proof. Finally the full proof is given.

Background. A L-layer fully connected neural network consists of weights and biases $\theta = [\mathbf{W}_1, \mathbf{b}_1 \mathbf{W}_2, \mathbf{b}_2, ..., \mathbf{W}_L, \mathbf{b}_L]$ where each $\mathbf{W}_i \in \mathbb{R}^{d_{i-1} \times d_i}, \mathbf{b} \in \mathbb{R}^{d_i}$. The output of the neural network is given by $f_\theta(\mathbf{x}) = \text{softmax}(\mathbf{z}_L)$, where $\mathbf{z}_L = \mathbf{W}_L^T \mathbf{a}_{L-1} + \mathbf{b}_L$, $\mathbf{a}_i = h(\mathbf{z}_i)$, $\mathbf{z}_i = \mathbf{W}_i^T \mathbf{a}_{i-1} + \mathbf{b}_i$, and h is the activation function. Let $n = d_L$ denote the number of classes, $d = d_0$ the number of input features, and $\mathbf{a}_0 = \mathbf{x} \in \mathbb{R}^d$. The per example crossentropy loss ℓ is given by $\ell(\hat{\mathbf{y}}, \mathbf{y}) = -\sum_{i=1}^n y_i \log \hat{y}_i$. Observe in the $L = 1$ case, the network is equivalent to multi-class logistic regression. $\nabla_{\mathbf{W}_i} \ell$ is given by the outer product of the vectors $\mathbf{a}_{i-1}$ and $\boldsymbol{\delta}_i$:

$$\nabla_{\mathbf{W}_i} \ell = \mathbf{a}_{i-1} \boldsymbol{\delta}_i^T \tag{A.1}$$

where $\boldsymbol{\delta}_i = \frac{\partial \ell}{\partial \mathbf{z}_i} \in \mathbb{R}^{d_i}$. Additionally, $\nabla_{\mathbf{b}_i} \ell = \boldsymbol{\delta}_i$. We have that

$$\boldsymbol{\delta}_i = \text{diag}(\mathbf{h}_i) \mathbf{W}_{i+1} \boldsymbol{\delta}_{i+1}, \tag{A.2}$$

where $\mathbf{h}_i = [h'(z_1^{(i)}), ..., h'(z_{d_i}^{(i)})]^T \in \mathbb{R}^{d_i}$. Let $z_j = [\mathbf{z}_L]_j$, $y_i = [\mathbf{y}]_i$ and $\hat{y}_j = [f_\theta(\mathbf{x})]_j$. We can write the j-th element of $\boldsymbol{\delta}_L$ as the following:

$$[\boldsymbol{\delta}_L]_j = \frac{\partial \ell}{\partial z_j} = -\sum_{i=1}^n y_i \cdot \frac{\partial \log \hat{y}_i}{\partial z_j} = -\sum_{i=1}^n \frac{y_i}{\hat{y}_i} \cdot \frac{\partial \hat{y}_i}{\partial z_j} \tag{A.3}$$

We know that the derivative of the softmax function is given by

$$\frac{\partial \hat{y}_i}{\partial z_j} = \begin{cases} \hat{y}_j(1-\hat{y}_j) & \text{if } i = j \\ -\hat{y}_i \cdot \hat{y}_j & \text{otherwise,} \end{cases} \tag{A.4}$$

which allows us to rewrite Equation A.3 as

$$\begin{aligned} [\boldsymbol{\delta}_L]_j = \frac{\partial \ell}{\partial z_j} &= -y_j(1-\hat{y}_j) - \sum_{\substack{i=1 \\ i \neq j}}^{n} y_i \cdot (-\hat{y}_j) \\ [\boldsymbol{\delta}_L]_j &= -y_j + \hat{y}_j \sum_{i=1}^{n} y_i \end{aligned} \tag{A.5}$$

From (A.5), we have that $\boldsymbol{\delta}_L = v\hat{\mathbf{y}} - \mathbf{y}$, where $v = \sum_{i=1}^{n} [\mathbf{y}]_i$.

Associated Lemmas and Corollaries

Lemma 2. *For any two training examples* $(\boldsymbol{x}, \boldsymbol{y})$ *and* $(\boldsymbol{x'}, \boldsymbol{y'})$*, if* $\nabla_{\boldsymbol{W}_1} \ell(\boldsymbol{x'}, \boldsymbol{y'}) = \nabla_{\boldsymbol{W}_1} \ell(\boldsymbol{x}, \boldsymbol{y})$ *and* $\nabla_{\boldsymbol{b}_1} \ell(\boldsymbol{x'}, \boldsymbol{y'}) = \nabla_{\boldsymbol{b}_1} \ell(\boldsymbol{x}, \boldsymbol{y})$*, then* $\boldsymbol{x} = \boldsymbol{x'}$*.*

Proof. If $\nabla_{\mathbf{W}_1} \ell(\mathbf{x}', \mathbf{y}') = \nabla_{\mathbf{W}_1} \ell(\mathbf{x}, \mathbf{y})$, then $\mathbf{x}'\boldsymbol{\delta}_1'^T = \mathbf{x}\boldsymbol{\delta}_1^T$. If $\nabla_{\mathbf{b}_1} \ell(\mathbf{x}', \mathbf{y}') = \nabla_{\mathbf{b}_1} \ell(\mathbf{x}, \mathbf{y})$, then $\boldsymbol{\delta}_1' = \boldsymbol{\delta}_1$, resulting in the equation $\mathbf{x}'\mathbf{u}^T = \mathbf{x}\mathbf{u}^T$ where $\mathbf{u} = \boldsymbol{\delta}_1' = \boldsymbol{\delta}_1$. Since we have an outer product, this is only satsified when $\mathbf{x} = \mathbf{x}'$.

□

Lemma 3. *For any two training examples* $(\boldsymbol{x}, \boldsymbol{y})$ *and* $(\boldsymbol{x'}, \boldsymbol{y'})$*, if* $\nabla_{\boldsymbol{W}_L} \ell(\boldsymbol{x'}, \boldsymbol{y'}) = \nabla_{\boldsymbol{W}_L} \ell(\boldsymbol{x}, \boldsymbol{y})$ *and* $\nabla_{\boldsymbol{b}_L} \ell(\boldsymbol{x'}, \boldsymbol{y'}) = \nabla_{\boldsymbol{b}_L} \ell(\boldsymbol{x}, \boldsymbol{y})$ *then* $\boldsymbol{y} = (v - v')\boldsymbol{u} + \boldsymbol{y'}$*, where* $v = \sum_j [\boldsymbol{y}]_j$*,* $v' = \sum_j [\boldsymbol{y'}]_j$*, and* $\boldsymbol{u} \in \mathbb{R}^n$*.*

Proof. If $\nabla_{\mathbf{b}_L} \ell(\mathbf{x}', \mathbf{y}') = \nabla_{\mathbf{b}_L} \ell(\mathbf{x}, \mathbf{y})$, then $\boldsymbol{\delta}_L' = \boldsymbol{\delta}_L$. If, $\nabla_{\mathbf{W}_L} \ell(\mathbf{x}', \mathbf{y}') = \nabla_{\mathbf{W}_L} \ell(\mathbf{x}, \mathbf{y})$, then $\mathbf{a}_{L-1}'\boldsymbol{\delta}_L'^T = \mathbf{a}_{L-1}\boldsymbol{\delta}_L^T$. Since $\boldsymbol{\delta}_L' = \boldsymbol{\delta}_L$ we have that $\mathbf{a}_{L-1}' = \mathbf{a}_{L-1}$ Therefore, $\mathbf{z}_L' = \mathbf{W}_L^T\mathbf{a}_{L-1} + \mathbf{b}_L = \mathbf{W}_L^T\mathbf{a}_{L-1}' + \mathbf{b}_L = \mathbf{z}_L$. Therefore $\hat{\mathbf{y}}' = \hat{\mathbf{y}}$. Let $\mathbf{u} = \hat{\mathbf{y}}' = \hat{\mathbf{y}}$. Then

$$\begin{aligned} \boldsymbol{\delta}_L' &= \boldsymbol{\delta}_L \\ v'\hat{\mathbf{y}}' - \mathbf{y}' &= v\hat{\mathbf{y}} - \mathbf{y} \\ v'\mathbf{u} - \mathbf{y}' &= v\mathbf{u} - \mathbf{y} \\ \mathbf{y} &= (v - v')\mathbf{u} + \mathbf{y}' \end{aligned} \tag{A.6}$$

□

Corollary 1. *For any two training examples* $(\boldsymbol{x}, \boldsymbol{y})$ *and* $(\boldsymbol{x'}, \boldsymbol{y'})$*, if* $\nabla_{\boldsymbol{W}_L} \ell(\boldsymbol{x'}, \boldsymbol{y'}) = \nabla_{\boldsymbol{W}_L} \ell(\boldsymbol{x}, \boldsymbol{y})$ *and* $\nabla_{\boldsymbol{b}_L} \ell(\boldsymbol{x'}, \boldsymbol{y'}) = \nabla_{\boldsymbol{b}_L} \ell(\boldsymbol{x}, \boldsymbol{y})$ *and* $\sum_j [\boldsymbol{y}]_j = \sum_j [\boldsymbol{y'}]_j$ *then* $\boldsymbol{y} = \boldsymbol{y'}$*.*

Proof. If the first two conditions hold, then, by Lemma 3, $\mathbf{y} = (v - v')\mathbf{u} + \mathbf{y}'$, where $v = \sum_j [\mathbf{y}]_j$, $v' = \sum_j [\mathbf{y}']_j$. If these two sums are equal, then $v - v' = 0$ which, substituting into (A.6) gives $\mathbf{y} = \mathbf{y}'$. □

Full Proof. We now prove Proposition 1.

Proof. The proof follows from Lemma 2 and Lemma 3. From Lemma 2, if $\nabla_{\mathbf{W}_1}\ell(\mathbf{x}', \mathbf{y}') = \nabla_{\mathbf{W}_1}\ell(\mathbf{x}, \mathbf{y})$ and $\nabla_{\mathbf{b}_1}\ell(\mathbf{x}', \mathbf{y}') = \nabla_{\mathbf{b}_1}\ell(\mathbf{x}, \mathbf{y})$, then $\mathbf{x} = \mathbf{x}'$.

From Lemma 3, if $\nabla_{\mathbf{W}_L}\ell(\mathbf{x}', \mathbf{y}') = \nabla_{\mathbf{W}_L}\ell(\mathbf{x}, \mathbf{y})$ and $\nabla_{\mathbf{b}_L}\ell(\mathbf{x}', \mathbf{y}') = \nabla_{\mathbf{b}_L}\ell(\mathbf{x}, \mathbf{y})$, then $\mathbf{y} = (v - v')\mathbf{u} + \mathbf{y}'$. If $\mathbf{y}$ and $\mathbf{y}'$ are one hot labels, then $v = v' = 1$. Consequently, from Corollary 1, $\mathbf{y} = \mathbf{y}'$.

□

A.2 Proof of Lemma 1

Proof. Let $\mathbf{y} = [y_1, y_2, \ldots, y_n]^T$ represent the i-th column of $\mathbf{Y}$, and $\mathbf{s} = [s_1, s_2, \ldots, s_n]^T$ represent the i-th column of $f_{\mathbf{W}}(\mathbf{X})$. We then have that

$$\begin{aligned} \sum_{j=1}^{n} D_{ij} &= -\sum_{j=1}^{n} y_j + \sum_{j=1}^{n} s_j \cdot \sum_{k=1}^{n} y_k \\ &= -\sum_{j=1}^{n} y_j + \sum_{k=1}^{n} y_k = 0. \end{aligned} \tag{A.7}$$

□

A.3 Proof of Proposition 2

We present the proof as follows: first we derive expressions for the gradient of the average loss function $\nabla\mathcal{L}$ for fully connected neural networks, as done similarly in Appendix A.1 for $\nabla\ell$. We then go on to give the full proof.

Background. For the mini-batch setting, we can derive matrix expressions for $\nabla_{\mathbf{W}_i}\mathcal{L}$. Writing everything in matrix notation, we have the input matrix $\mathbf{X} \in \mathbb{R}^{d_0 \times b}$, where the k-th column denotes $\mathbf{x}^{(k)}$. We can also write the batched output of the model $\mathbf{Z}_L \in \mathbb{R}^{d_L \times b}$ as the following matrix multiplication:

$$\mathbf{Z}_L = \mathbf{W}_L^T \mathbf{A}_{L-1} + \mathbf{B}_L, \tag{A.8}$$

where each $\mathbf{A}_i \in \mathbb{R}^{d_i \times b}$ is the batched activation after the i-th layer i.e., the k-th column of $\mathbf{A}_i$ represents the the activation of the network after the i-th layer for the k-th example in the mini-batch $\mathbf{a}_i^{(k)} := [a_1^{(k,i)} a_2^{(k,i)} \ldots a_{d_i}^{(k,i)}]^T$:

$$\mathbf{A}_i = h(\mathbf{Z}_i) = h(\mathbf{W}_i^T \mathbf{A}_{i-1} + \mathbf{B}_i), \tag{A.9}$$

where $\mathbf{A}_0 = \mathbf{X}$, and $\mathbf{B}_i \in \mathbb{R}^{d_i \times b}$, which is a matrix where each of the columns are the same bias vector for the i-th layer repeated. Let the scalar $z_m^{(k,i)}$ represents the

the m-th logit value at the i-th layer of the k-th training example in the mini-batch, and let the scalar $a_m^{(k,i)}$ be the corresponding activation value.

We can write the (mn)-th element of $\nabla_{\mathbf{W}_i}\mathcal{L}(B)$ as the following:

$$\begin{aligned}[\nabla_{\mathbf{W}_i}\mathcal{L}(B)]_{mn} &= \frac{1}{b}\sum_{(\mathbf{x},\mathbf{y})\in B}[\nabla_{\mathbf{W}_i}\ell(\mathbf{x},\mathbf{y})]_{mn} \\ &= \frac{1}{b}\sum_{k=1}^{b} a_m^{(k,i-1)}\cdot\frac{\partial\ell^{(k)}}{\partial z_n^{(k,i)}}\end{aligned} \tag{A.10}$$

To compute the entire gradient $\nabla_{\mathbf{W}_i}\mathcal{L}(B)$, not just a single element, we can view it as the matrix multiplication $b\nabla_{\mathbf{W}_i}\mathcal{L}(B) = \mathbf{A}_{i-1}\mathbf{D}_i$, where $\mathbf{D}_i \in \mathbb{R}^{b\times d_i}$ and the k-th row denotes $\boldsymbol{\delta}_i^{(k)}$ i.e., the error at the i-th layer for the k-th example in the mini-batch. Each $\mathbf{D}_i$ is given by

$$\mathbf{D}_i = (\mathbf{D}_{i+1}\mathbf{W}_{i+1}^T)\odot H_i. \tag{A.11}$$

Equation (A.11) is the matrix version of (A.2). The k-th row of $H_i \in \mathbb{R}^{b\times d_i}$ denotes $\mathbf{h}_i^{(k)}$, and $\odot$ gives the hadamard product. $\nabla_{\mathbf{b}_i}\mathcal{L}$ is given by the average of all the rows of $\mathbf{D}_i$ e.g.

$$[\nabla_{\mathbf{b}_i}\mathcal{L}]_j = \frac{1}{b}\sum_{k=1}^{b}[\mathbf{D}_i]_{kj}$$

Full Proof. Finally we can prove Proposition 2.

Proof. We prove Proposition 2 by proving that there exists a non-zero *perturbation* matrix $\mathbf{P} \in \mathbb{R}^{d_0\times b}$ such that for a given mini-batch $B = \{(\mathbf{x}^{(k)},\mathbf{y}^{(k)})\}_{k=1}^b$, a corresponding forged mini-batch B' can be constructed from B and $\mathbf{P}$, where $B' = \{(\mathbf{x}^{(k)} + [\mathbf{P}]^k, \mathbf{y}^{(k)})\}_{k=1}^b$ and $\nabla_{\mathbf{W}_i}\mathcal{L}(B) = \nabla_{\mathbf{W}_i}\mathcal{L}(B')$ and $\nabla_{\mathbf{b}_i}\mathcal{L}(B) = \nabla_{\mathbf{b}_i}\mathcal{L}(B')$ holds $\forall i \in \{1,2,...,L\}$. The operator $[\cdot]^k$ returns the k-th column of the input matrix. The perturbation matrix $\mathbf{P}$ itself must satisfy the following 2 matrix equations:

$$\begin{cases}\mathbf{P}\mathbf{D}_1 = \mathbf{0} \\ \mathbf{W}_1^T\mathbf{P} = \mathbf{0}\end{cases} \tag{A.12}$$

Let $\mathbf{X}' := \mathbf{X} + \mathbf{P}$, where $\mathbf{X}' \in \mathbb{R}^{d_0\times b}$ represents the input matrix of the forged mini-batch B' and $\mathbf{X}$ is the input matrix of mini-batch B, where the k-th column of $\mathbf{X}$ and $\mathbf{X}'$ denote $\mathbf{x}^{(k)}$ and $\mathbf{x}'^{(k)} := \mathbf{x}^{(k)} + [\mathbf{P}]^k$ respectively. Considering the first equation $\mathbf{P}\mathbf{D}_1 = \mathbf{0}$, if $\mathbf{P}$ satisfies this equation, then $b\nabla_{\mathbf{W}_1}\mathcal{L}(B') = \mathbf{X}'\mathbf{D}_1 = \mathbf{X}\mathbf{D}_1 + \mathbf{P}\mathbf{D}_1 = \mathbf{X}\mathbf{D}_1 = b\nabla_{\mathbf{W}_1}\mathcal{L}(B)$.

Thus satisfying the first equation ensures that the gradient for the first weight matrix $\mathbf{W}_1$ will match.

Let $\mathbf{Z}'_1$ be the batched raw logit output of the first layer, i.e. $\mathbf{Z}'_1 := \mathbf{W}_1^T\mathbf{X}' + \mathbf{B}_1$. If $\mathbf{P}$ satisfies the second equation, then $\mathbf{Z}'_1 = \mathbf{W}_1^T\mathbf{X} + \mathbf{W}_1^T\mathbf{P} + \mathbf{B}_1 = \mathbf{W}_1^T\mathbf{X} + \mathbf{B}_1 = \mathbf{Z}_1$. Therefore, $\mathbf{A}'_1 = \mathbf{A}_1$. Since $\mathbf{A}'_1 = \mathbf{A}_1$, then $\mathbf{A}'_i = \mathbf{A}_i \ \forall i \in \{1, 2, \ldots, L\}$. Satisfying the second equation ensures the activations for the first layer as well as all subsequent layers will be the same as that of the original mini-batch. If we use the same label i.e., each $\mathbf{y}'^{(k)} = \mathbf{y}^{(k)}$, then, for the k-th training example in the forged mini-batch, we have that by $\hat{\mathbf{y}}'^{(k)} = \hat{\mathbf{y}}^{(k)}$, and consequently $\boldsymbol{\delta}_L'^{(k)} = \boldsymbol{\delta}_L^{(k)}$. Therefore $\mathbf{D}'_L = \mathbf{D}_L$, and by (A.11) we can show inductively that $\mathbf{D}'_i = \mathbf{D}_i, \ \forall i \in \{1, 2, ..., L\}$. We have shown the base case $i = L$. To show for $i = L-1$, we first recognise that if $\forall i$, $A'_i = A_i$, then $H'_i = H_i, \ \forall i$. By (A.11) we then have $\mathbf{D}'_i = (\mathbf{D}'_{i+1}\mathbf{W}_{i+1}^T) \odot H'_i = (\mathbf{D}_{i+1}\mathbf{W}_{i+1}^T) \odot H_i = \mathbf{D}_i$. As a result, $\nabla_{\mathbf{b}_i}\mathcal{L}(B') = \nabla_{\mathbf{b}_i}\mathcal{L}(B), \ \forall i \in \{1, 2, \ldots, L\}$. Regarding the gradients for the weight matrices $\mathbf{W}_i$ we then have that $b\nabla_{\mathbf{W}_i}\mathcal{L}(B') = \mathbf{A}'_{i-1}\mathbf{D}'_i = \mathbf{A}_{i-1}\mathbf{D}_i = b\nabla_{\mathbf{W}_i}\mathcal{L}(B)$. As a result $\nabla_{\mathbf{W}_i}\mathcal{L}(B') = \nabla_{\mathbf{W}_i}\mathcal{L}(B), \ \forall i \in \{1, 2, \ldots, L\}$. Therefore, if the perturbation matrix $\mathbf{P}$ satifies the 2 matrix equations given in (A.12), then B and B' will produce the same gradient.

In order to construct $\mathbf{P}$, we first observe that we can write the equations in (A.12) as $\mathbf{IPD}_1 = \mathbf{0}$ and $\mathbf{W}_1^T\mathbf{PI} = \mathbf{0}$, where $\mathbf{I}$ is the identity matrix of appropriate shape. We also have, by the "vec trick", that for any matrices A, X, B and C, $AXB = C \iff (B^T \otimes A)\text{vec}(X) = \text{vec}(C)$, where $\otimes$ is the Kronecker product, and $\text{vec}(\cdot)$ returns the vector after stacking all the columns of the input matrix vertically. We can rewrite our two equations as the following two linear systems:

$$\begin{cases} (\mathbf{D}_1^T \otimes \mathbf{I})\text{vec}(\mathbf{P}) = \text{vec}(\mathbf{0}) \\ (\mathbf{I}^T \otimes \mathbf{W}_1^T)\text{vec}(\mathbf{P}) = \text{vec}(\mathbf{0}) \end{cases}$$

The first coefficient matrix $(\mathbf{D}_1^T \otimes \mathbf{I}) \in \mathbb{R}^{d_1 d_0 \times d_0 b}$, and the second coefficient matrix $(\mathbf{I}^T \otimes \mathbf{W}_1^T) \in \mathbb{R}^{d_1 b \times d_0 b}$, so we can combine them into the single linear system by stacking them vertically:

$$\begin{bmatrix} \mathbf{D}_1^T \otimes \mathbf{I} \\ \mathbf{I}^T \otimes \mathbf{W}_1^T \end{bmatrix} \text{vec}(\mathbf{P}) = \text{vec}(\mathbf{0}) \tag{A.13}$$

Let $\mathbf{K} \in \mathbb{R}^{(d_1 d_0 + d_1 b) \times d_0 b}$ be the above coefficient matrix, then the general solution is given by:

$$\text{vec}(\mathbf{P}) = (\mathbf{I} - \mathbf{K}^+\mathbf{K})\mathbf{F}, \tag{A.14}$$

where $\mathbf{F} \in \mathbb{R}^{d_0 b}$ is an arbitrary matrix, and $\mathbf{K}^+$ is the psuedo-inverse of $\mathbf{K}$.

For the homogenous linear system $\mathbf{K}\text{vec}(\mathbf{P}) = \text{vec}(\mathbf{0})$, there are $d_0 b - \text{rank}(\mathbf{K})$ linearly independent solutions. If $\mathbf{K}$ is full rank and square, there are no non-trivial solutions. In order to ensure at least 1 set of linearly dependent solutions (e.g. one

free variable) $d_0 b > d_1 d_0 + d_1 b$ must hold. This ensures that $d_0 b - \text{rank}(\mathbf{K}) \geq 1$. □

A.4 Proof of Proposition 3

Proof. Let $\mathbf{G} = \nabla_{\mathbf{W}} \mathcal{L}(B; \mathbf{W})$. Finding a batch $B' = \{(\mathbf{x}'^{(1)}, \mathbf{y}'^{(1)}), (\mathbf{x}'^{(2)}, \mathbf{y}'^{(2)}), ..., (\mathbf{x}'^{(b)}, \mathbf{y}'^{(b)})\}$ with $b \geq \text{rank}(\mathbf{G})$, such that $\nabla_{\mathbf{W}} \mathcal{L}(B'; \mathbf{W}) = \mathbf{G}$ requires that for every $(i, j) \in [d] \times [n]$,

$$\frac{\partial \ell'^{(1)}}{\partial z_j'^{(1)}} x_i'^{(1)} + \frac{\partial \ell'^{(2)}}{\partial z_j'^{(2)}} x_i'^{(2)} + ... + \frac{\partial \ell'^{(b)}}{\partial z_j'^{(b)}} x_i'^{(b)} = b\mathbf{G}_{ij}, \tag{A.15}$$

where each $\ell'^{(k)} = \ell(\mathbf{x}'^{(k)}, \mathbf{y}'^{(k)})$, and $\mathbf{z}'^{(k)} = \mathbf{W}^T \mathbf{x}'^{(k)}$. We can restate the problem as finding matrices $\mathbf{X}' \in \mathbb{R}^{d \times b}$ and $\mathbf{D} \in \mathbb{R}^{b \times n}$ such that

$$\mathbf{X}'\mathbf{D} = b\mathbf{G}, \tag{A.16}$$

The k-th column of $\mathbf{X}'$ represents $\mathbf{x}'^{(k)}$, and the k-th row of $\mathbf{D}$ represents $\frac{\partial \ell'^{(k)}}{\partial \mathbf{z}'^{(k)}}$. From Lemma 1, the elements of every row of $\mathbf{D}$ must sum to zero. Construct a matrix $\mathbf{D}$ such that $\text{rank}(\mathbf{D}) = \text{rank}(\mathbf{G})$, and the elements of every row of $\mathbf{D}$ sum to zero. Finding the corresponding $\mathbf{X}'$ amounts to solving the linear system

$$\mathbf{D}^T \mathbf{x}_i' = b\mathbf{g}_i \tag{A.17}$$

where $\mathbf{x}_i'$ and $\mathbf{g}_i$ are the transposed i-th row of $\mathbf{X}'$ and $\mathbf{G}$ respectively. For each i, we know that $\text{rank}(\mathbf{D}^T) = \text{rank}(\mathbf{D}^T | b\mathbf{g}_i)$, therefore we can be certain that at least one solution exists.

Finally, the batch of examples B' whose gradient $\nabla_{\mathbf{W}} \mathcal{L}(B'; \mathbf{W}) = \mathbf{G}$ can constructed from the matrices $\mathbf{X}'$ and $\mathbf{D}$. For every $k \in [b]$, $\mathbf{x}'^{(k)}$ is the given by the k-th column of $\mathbf{X}'$, and as shown in Lemma 1, each $\mathbf{y}'^{(k)} = v^{(k)} \hat{\mathbf{y}}^{(k)} - [\mathbf{D}]_k^T$, for any constant $v^{(k)} \in \mathbb{R}$, and $[\mathbf{D}]_k$ returns the k-th row of $\mathbf{D}$. □

Appendix B

Accuracy Measurements

Table B.1 provides the observed accuracy measurements for the models and datasets considered.

Model/Dataset	$b = 50$	$b = 100$	$b = 500$	$b = 1000$	$b = 2000$
LeNet/MNIST	$0.94 \pm 2.4\text{e}-4$	$0.94 \pm 1.6\text{e}-4$	$0.95 \pm 4.7\text{e}-5$	0.95 ± 0	$0.95 \pm 8.1\text{e}-5$
VGGmini/CIFAR10	$0.63 \pm 3.3\text{e}-4$	$0.62 \pm 2.4\text{e}-4$	$0.64 \pm 1.4\text{e}-4$	$0.64 \pm 1.9\text{e}-4$	$0.55 \pm 4.5\text{e}-4$
FCN/Adult	0.75 ± 0	0.75 ± 0	0.75 ± 0	0.75 ± 0	0.75 ± 0
Transformer/IMDB	$0.87 \pm 1.3\text{e}-3$	$0.85 \pm 1.9\text{e}-3$	$0.83 \pm 9.5\text{e}-3$	$0.84 \pm 2.4\text{e}-03$	$0.84 \pm 1.9\text{e}-3$

Table B.1: Observed variance in accuracy from GPU auto-tuning non determinism

Appendix C

Brute Force Forging

Real images consist of 256 possible pixel values. Given an image consisting of just 2 pixels (with a single channel), 3 possible classes, and a batch size of 2, this would result in $\binom{256^2 \times 3}{2} \approx 1.9 \times 10^{10}$ mini-batches. Computing the gradient of and comparing that many mini-batches is not practical; so an evaluation of the brute force approach is not possible when we allow 256 possible values. However, when we consider much smaller values for v, the number of allowed values, namely just 2 and 3, the task becomes much more feasible. In other words, when $v = 2$, pixels may have only the values 0 or 255 i.e., on or off. For $v = 3$, pixels may take the values, $0, 127$, or 255 i.e., 3 possible values. This drastically reduces the number of mini-batches that we need to search through.

For a given setup i.e., values of d, b, v, and n, we run the following experiment:

1. Sample a true minibatch B from the possible $\binom{v^d \times n}{b}$ number of mini-batches.
2. Calculate it's gradient $\nabla_\theta \mathcal{L}(B)$.
3. Search through all $\binom{v^d \times n}{b} - 1$ other mini-batches for a distinct mini-batch that produces the same gradient.

We run this experiment on a logistic regression model, as well as a $L = 2$ fully connected neural network with 10 hidden units and ReLU activation function We additionally fix $d = 4$ and $n = 3$. Table C.1 summarises our results.

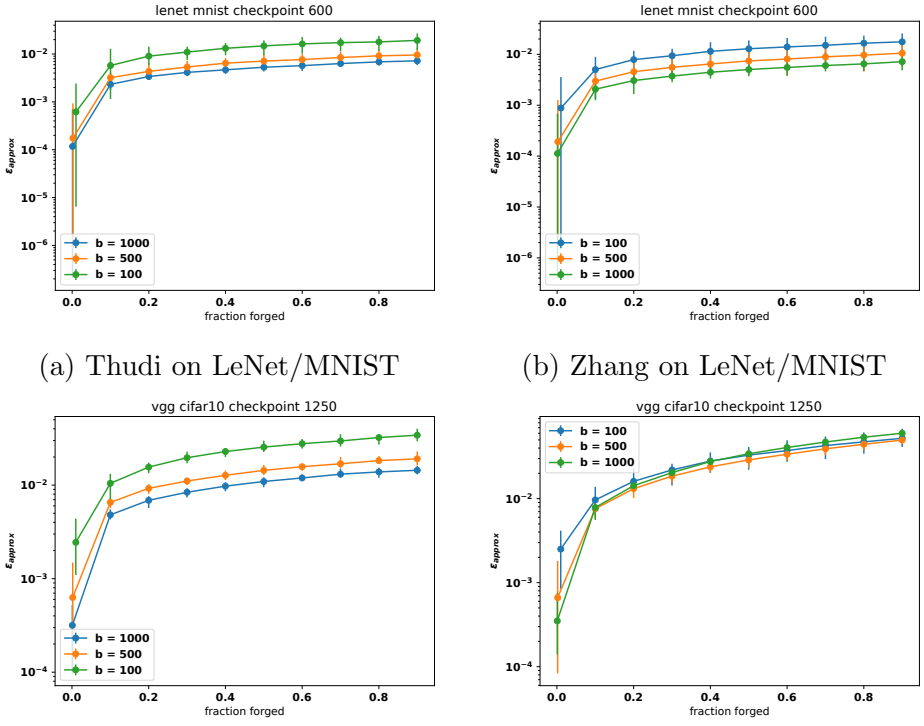
	$b=1$	$b=2$	$b=3$	$b=4$	$b=5$
$v=2$	✗(48)	✗(1128)	✗(17,296)	✗(194,580)	✗(1,712,304)
$v=3$	✗(243)	✗(29,403)	✗(2,362,041)	?(141,722,460)	?(6.77×10^9)
$v=4$	✗(768)	✗(294,528)	?(75,202,816)	?(1.43×10^{10})	?(2.2×10^{12})
$v=5$	✗(1,875)	✗(1,756,875)	?(1.1×10^9)	?(5.13×10^{11})	?(1.92×10^{14})
$v=6$	✗(3,888)	✗(7,556,328)	?(9.79×10^9)	?(9.51×10^{12})	?(7.38×10^{15})

Table C.1: For increasingly large values of v and b (d and n are fixed to $d=4$ and $n=3$), we report, after searching through all the possible mini-batches, whose total number is given in brackets, whether a forged mini-batch that produced the same gradient was found. In every case, we did not find one after an exhaustive search. We give a ? in those setting where we did not conduct a search, due to the large number of possible mini-batches.

Appendix D

Comparison of Data Forging Attacks

In Figure D.1, we plot the performance of both Thudi et al. [71] and Zhang et al. [86]'s data forging attacks, for both LeNet/MNIST and VGGmini/CIFAR10 execution traces.



(a) Thudi on LeNet/MNIST

(b) Zhang on LeNet/MNIST

(c) Thudi onVGGmini/CIFAR10

(d) Zhang on VGGmini/CIFAR10

Figure D.1: We plot the average approximation error of both Thudi et al.'s [71] and Zhang et al.'s [86] attack (over several runs), as well as the min and max observed approximation error for different batch sizes and increasing forging fractions. We see both produce similar performance.

Bibliography

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. {TensorFlow}: a system for {Large-Scale} machine learning. In *12th USENIX symposium on operating systems design and implementation (OSDI 16)*, pages 265–283, 2016.

[2] Martin Abadi, Andy Chu, Ian Goodfellow, H Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, 2016.

[3] Martin Abadi, Andy Chu, Ian Goodfellow, H. Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, page 308–318, New York, NY, USA, 2016. Association for Computing Machinery.

[4] Kasra Abbaszadeh, Christodoulos Pappas, Jonathan Katz, and Dimitrios Papadopoulos. Zero-knowledge proofs of training for deep neural networks. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, pages 4316–4330, 2024.

[5] Michael Aerni, Jie Zhang, and Florian Tramèr. Evaluations of machine learning privacy defenses are misleading. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, pages 1271–1284, 2024.

[6] James Ball. NSA collects millions of text messages daily in 'untargeted' global sweep, 2014.

[7] Teodora Baluta, Ivica Nikolic, Racchit Jain, Divesh Aggarwal, and Prateek Saxena. Unforgeability in stochastic gradient descent. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, pages 1138–1152, 2023.

[8] Franziska Boenisch, Adam Dziedzic, Roei Schuster, Ali Shahin Shamsabadi, Ilia Shumailov, and Nicolas Papernot. When the curious abandon honesty: Federated learning is not private. *arXiv preprint arXiv:2112.02918*, 2021.

[9] Franziska Boenisch, Adam Dziedzic, Roei Schuster, Ali Shahin Shamsabadi, Ilia Shumailov, and Nicolas Papernot. Reconstructing individual data points in federated learning hardened with differential privacy and secure aggregation, 2023.

[10] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. Practical secure aggregation for federated learning on user-held data. *arXiv preprint arXiv:1611.04482*, 2016.

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

[12] Hannah Brown, Katherine Lee, Fatemehsadat Mireshghallah, Reza Shokri, and Florian Tramèr. What does it mean for a language model to preserve privacy? In *Proceedings of the 2022 ACM Conference on Fairness, Accountability, and Transparency*, pages 2280–2292, 2022.

[13] Carole Cadwalladr and Emma Graham-Harrison. Revealed: 50 million facebook profiles harvested for cambridge analytica in major data breach. *The guardian*, 17(1):22, 2018.

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

[15] Nicholas Carlini, Steve Chien, Milad Nasr, Shuang Song, Andreas Terzis, and Florian Tramer. Membership inference attacks from first principles. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 1897–1914. IEEE, 2022.

[16] Nicholas Carlini, Chang Liu, Úlfar Erlingsson, Jernej Kos, and Dawn Song. The secret sharer: Evaluating and testing unintended memorization in neural networks. In *Proceedings of the 28th USENIX Conference on Security Symposium*, SEC'19, page 267–284, USA, 2019. USENIX Association.

[17] Nicholas Carlini, Florian Tramer, Eric Wallace, Matthew Jagielski, Ariel Herbert-Voss, Katherine Lee, Adam Roberts, Tom Brown, Dawn Song, Ulfar Erlingsson, et al. Extracting training data from large language models. In

30th USENIX Security Symposium (USENIX Security 21), pages 2633–2650, 2021.

[18] Nicolas Carlini, Jamie Hayes, Milad Nasr, Matthew Jagielski, Vikash Sehwag, Florian Tramer, Borja Balle, Daphne Ippolito, and Eric Wallace. Extracting training data from diffusion models. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 5253–5270, 2023.

[19] Hong-Min Chu, Jonas Geiping, Liam H Fowl, Micah Goldblum, and Tom Goldstein. Panning for gold in federated learning: Targeted text extraction under arbitrarily large-scale aggregation. In *The Eleventh International Conference on Learning Representations*, 2023.

[20] Aldo Cortesi, Maximilian Hils, Thomas Kriechbaumer, and contributors. mitmproxy: A free and open source interactive HTTPS proxy (v5.01), 2020.

[21] Jieren Deng, Yijue Wang, Ji Li, Chao Shang, Hang Liu, Sanguthevar Rajasekaran, and Caiwen Ding. Tag: Gradient attack on transformer-based language models. *arXiv preprint arXiv:2103.06819*, 2021.

[22] Cynthia Dwork. Differential privacy. In *International colloquium on automata, languages, and programming*, pages 1–12. Springer, 2006.

[23] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating noise to sensitivity in private data analysis. In *Theory Of Cryptography Conference*, 2006.

[24] Stack Exchange. Stack exchange data dump, 2023.

[25] Congyu Fang, Hengrui Jia, Anvith Thudi, Mohammad Yaghini, Christopher A Choquette-Choo, Natalie Dullerud, Varun Chandrasekaran, and Nicolas Papernot. Proof-of-learning is currently more broken than you think. In *2023 IEEE 8th European Symposium on Security and Privacy (EuroS&P)*, pages 797–816. IEEE, 2023.

[26] Liam H Fowl, Jonas Geiping, Wojciech Czaja, Micah Goldblum, and Tom Goldstein. Robbing the fed: Directly obtaining private data in federated learning with modified models. In *International Conference on Learning Representations*, 2022.

[27] 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.

[28] Jonas Geiping, Hartmut Bauermeister, Hannah Dröge, and Michael Moeller. Inverting gradients - how easy is it to break privacy in federated learning? In *Advances in Neural Information Processing Systems*, 2020.

[29] Google. Gboard – the Google Keyboard. `https://play.google.com/store/apps/details?id=com.google.android.inputmethod.latin`, 2022. Accessed: 2022-10-24.

[30] Scott Grauer-Gray, Lifan Xu, Robert Searles, Sudhee Ayalasomayajula, and John Cavazos. Auto-tuning a high-level language targeted to gpu codes. In *2012 innovative parallel computing (InPar)*, pages 1–10. Ieee, 2012.

[31] Klaus Greff, Rupesh K. Srivastava, Jan Koutnik, Bas R. Steunebrink, and Jurgen Schmidhuber. Lstm: A search space odyssey. *IEEE Transactions on Neural Networks and Learning Systems*, 28(10):2222–2232, Oct 2017.

[32] Yuhao Gu and Yuebin Bai. Ldia: Label distribution inference attack against federated learning in edge computing. *Journal of Information Security and Applications*, 74:103475, 2023.

[33] Samyak Gupta, Yangsibo Huang, Zexuan Zhong, Tianyu Gao, Kai Li, and Danqi Chen. Recovering private text in federated learning of language models. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022.

[34] Niv Haim, Gal Vardi, Gilad Yehudai, Ohad Shamir, and Michal Irani. Reconstructing training data from trained neural networks. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 22911–22924. Curran Associates, Inc., 2022.

[35] Katie Harbath and Collier Fernekes. History of the cambridge analytica controversy. *Bipartisan Policy Center (blog)*, 2023.

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

[37] Andrew Hard, Kanishka Rao, Rajiv Mathews, Swaroop Ramaswamy, Françoise Beaufays, Sean Augenstein, Hubert Eichner, Chloé Kiddon, and Daniel Ramage. Federated learning for mobile keyboard prediction, 2019.

[38] Hongsheng Hu, Zoran Salcic, Lichao Sun, Gillian Dobbie, Philip S. Yu, and Xuyun Zhang. Membership inference attacks on machine learning: A survey. *ACM Comput. Surv.*, 54(11s), sep 2022.

[39] Hengrui Jia, Mohammad Yaghini, Christopher A Choquette-Choo, Natalie Dullerud, Anvith Thudi, Varun Chandrasekaran, and Nicolas Papernot. Proof-of-learning: Definitions and practice. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 1039–1056. IEEE, 2021.

[40] Xiao Jin, Pin-Yu Chen, Chia-Yi Hsu, Chia-Mu Yu, and Tianyi Chen. Cafe: Catastrophic data leakage in vertical federated learning. *Advances in Neural Information Processing Systems*, 34:994–1006, 2021.

[41] Xiao Jin, Pin-Yu Chen, Chia-Yi Hsu, Chia-Mu Yu, and Tianyi Chen. Catastrophic data leakage in vertical federated learning. *Advances in Neural Information Processing Systems*, 34, 2021.

[42] Peter Kairouz, H Brendan McMahan, Shuang Song, Om Thakkar, Abhradeep Thakurta, and Zheng Xu. Practical and private (deep) learning without sampling or shuffling. *arXiv preprint arXiv:2103.00039*, 2021.

[43] Zhifeng Kong, Amrita Roy Chowdhury, and Kamalika Chaudhuri. Can membership inferencing be refuted? *arXiv preprint arXiv:2303.03648*, 2023.

[44] Nicolas Kourtellis, Kleomenis Katevas, and Diego Perino. Flaas: Federated learning as a service. In *Proceedings of the 1st Workshop on Distributed Machine Learning*, DistributedML'20, page 7–13, New York, NY, USA, 2020. Association for Computing Machinery.

[45] California Legislature. California consumer privacy act (ccpa), 2018. California Civil Code §§ 1798.100 - 1798.199.

[46] Douglas Leith. What Data Do The Google Dialer and Messages Apps On Android Send to Google? In *Proc Securecomm*, 2022.

[47] Douglas J. Leith. Mobile Handset Privacy: Measuring The Data iOS and Android Send to Apple And Google. In *Proc Securecomm*, 2021.

[48] Douglas J. Leith and Stephen Farrell. Contact Tracing App Privacy: What Data Is Shared By Europe's GAEN Contact Tracing Apps. In *Proc IEEE INFOCOM*, 2021.

[49] Xuechen Li, Florian Tramer, Percy Liang, and Tatsunori Hashimoto. Large language models can be strong differentially private learners. *arXiv preprint arXiv:2110.05679*, 2021.

[50] A. Marzal and E. Vidal. Computation of normalized edit distance and applications. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(9):926–932, 1993.

[51] 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*, 2017.

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

[53] Luca Melis, Congzheng Song, Emiliano De Cristofaro, and Vitaly Shmatikov. Exploiting unintended feature leakage in collaborative learning. In *2019 IEEE symposium on security and privacy (SP)*, pages 691–706. IEEE, 2019.

[54] Dinh C Nguyen, Quoc-Viet Pham, Pubudu N Pathirana, Ming Ding, Aruna Seneviratne, Zihuai Lin, Octavia Dobre, and Won-Joo Hwang. Federated learning for smart healthcare: A survey. *ACM Computing Surveys (CSUR)*, 55(3):1–37, 2022.

[55] Helen Nissenbaum. Privacy as contextual integrity. *Wash. L. Rev.*, 79:119, 2004.

[56] Daniel R. O'Day and Ricardo A. Calix. Text message corpus: Applying natural language processing to mobile device forensics. In *2013 IEEE International Conference on Multimedia and Expo Workshops (ICMEW)*, 2013.

[57] Xudong Pan, Mi Zhang, Yifan Yan, Jiaming Zhu, and Min Yang. Theory-oriented deep leakage from gradients via linear equation solver. *arXiv preprint arXiv:2010.13356*, 2020.

[58] Xudong Pan, Mi Zhang, Yifan Yan, Jiaming Zhu, and Zhemin Yang. Exploring the security boundary of data reconstruction via neuron exclusivity analysis. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 3989–4006, 2022.

[59] Dario Pasquini, Danilo Francati, and Giuseppe Ateniese. Eluding secure aggregation in federated learning via model inconsistency. *arXiv preprint arXiv:2111.07380*, 2021.

[60] Dario Pasquini, Danilo Francati, and Giuseppe Ateniese. Eluding secure aggregation in federated learning via model inconsistency. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, CCS '22, page 2429–2443, New York, NY, USA, 2022. Association for Computing Machinery.

[61] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

[62] Hung Viet Pham, Shangshu Qian, Jiannan Wang, Thibaud Lutellier, Jonathan Rosenthal, Lin Tan, Yaoliang Yu, and Nachiappan Nagappan. Problems and opportunities in training deep learning software systems: An analysis of variance. In *Proceedings of the 35th IEEE/ACM international conference on automated software engineering*, pages 771–783, 2020.

[63] Ofir Press and Lior Wolf. Using the output embedding to improve language models. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, pages 157–163, Valencia, Spain, April 2017. Association for Computational Linguistics.

[64] Suraj Rajendran, Jihad S Obeid, Hamidullah Binol, Kristie Foley, Wei Zhang, Philip Austin, Joey Brakefield, Metin N Gurcan, and Umit Topaloglu. Cloud-based federated learning implementation across medical centers. *JCO clinical cancer informatics*, 5:1–11, 2021.

[65] Hanchi Ren, Jingjing Deng, and Xianghua Xie. Grnn: Generative regression neural network—a data leakage attack for federated learning. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 13(4):1–24, 2022.

[66] Alexander Schlögl, Nora Hofer, and Rainer Böhme. Causes and effects of unanticipated numerical deviations in neural network inference frameworks. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

[67] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. Membership inference attacks against machine learning models. In *2017 IEEE symposium on security and privacy (SP)*, pages 3–18. IEEE, 2017.

[68] Ilia Shumailov, Zakhar Shumaylov, Dmitry Kazhdan, Yiren Zhao, Nicolas Papernot, Murat A Erdogdu, and Ross J Anderson. Manipulating sgd with data ordering attacks. *Advances in Neural Information Processing Systems*, 34:18021–18032, 2021.

[69] Robin Staab, Mark Vero, Mislav Balunovic, and Martin Vechev. Beyond memorization: Violating privacy via inference with large language models. In B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun, editors, *International Conference on Representation Learning*, volume 2024, pages 33832–33878, 2024.

[70] Romain Thomas. DroidGuard: A Deep Dive into SafetyNet. `https://www.sstic.org/media/SSTIC2022/SSTIC-actes/droidguard_a_deep_dive_into_safetynet/SSTIC`, 2022. [Online; accessed 04-May-2023].

[71] Anvith Thudi, Hengrui Jia, Ilia Shumailov, and Nicolas Papernot. On the necessity of auditable algorithmic definitions for machine unlearning. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 4007–4022, 2022.

[72] Stacey Truex, Nathalie Baracaldo, Ali Anwar, Thomas Steinke, Heiko Ludwig, Rui Zhang, and Yi Zhou. A hybrid approach to privacy-preserving federated learning. In *Proceedings of the 12th ACM Workshop on Artificial Intelligence and Security*, AISec'19, page 1–11, New York, NY, USA, 2019. Association for Computing Machinery.

[73] Stacey Truex, Ling Liu, Ka-Ho Chow, Mehmet Emre Gursoy, and Wenqi Wei. Ldp-fed: Federated learning with local differential privacy. In *Proceedings of the Third ACM International Workshop on Edge Systems, Analytics and Networking*, EdgeSys '20, page 61–66, New York, NY, USA, 2020. Association for Computing Machinery.

[74] European Union. General data protection regulation (gdpr), 2016. Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016.

[75] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, 2017.

[76] Paul Voigt and Axel Von dem Bussche. The eu general data protection regulation (gdpr). *A Practical Guide, 1st Ed., Cham: Springer International Publishing*, 10(3152676):10–5555, 2017.

[77] Yijue Wang, Jieren Deng, Dan Guo, Chenghong Wang, Xianrui Meng, Hang Liu, Caiwen Ding, and Sanguthevar Rajasekaran. Sapag: A self-adaptive privacy attack from gradients. *arXiv preprint arXiv:2009.06228*, 2020.

[78] Yijue Wang, Jieren Deng, Dan Guo, Chenghong Wang, Xianrui Meng, Hang Liu, Chao Shang, Binghui Wang, Qin Cao, Caiwen Ding, and Sanguthevar

Rajasekaran. Variance of the gradient also matters: Privacy leakage from gradients. In *2022 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2022.

[79] 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*, page 2512–2520. IEEE Press, 2019.

[80] Kang Wei, Jun Li, Ming Ding, Chuan Ma, Howard H. Yang, Farhad Farokhi, Shi Jin, Tony Q. S. Quek, and H. Vincent Poor. Federated learning with differential privacy: Algorithms and performance analysis. *IEEE Transactions on Information Forensics and Security*, 15:3454–3469, 2020.

[81] Kang Wei, Jun Li, Ming Ding, Chuan Ma, Howard H Yang, Farhad Farokhi, Shi Jin, Tony QS Quek, and H Vincent Poor. Federated learning with differential privacy: Algorithms and performance analysis. *IEEE Transactions on Information Forensics and Security*, 15:3454–3469, 2020.

[82] Yuxin Wen, Jonas A. Geiping, Liam Fowl, Micah Goldblum, and Tom Goldstein. Fishing for user data in large-batch federated learning via gradient magnification. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato, editors, *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 23668–23684. PMLR, 17–23 Jul 2022.

[83] Heng Xu, Tianqing Zhu, Lefeng Zhang, Wanlei Zhou, and Philip S Yu. Machine unlearning: A survey. *ACM Computing Surveys*, 56(1):1–36, 2023.

[84] Haomiao Yang, Mengyu Ge, Dongyun Xue, Kunlan Xiang, Hongwei Li, and Rongxing Lu. Gradient leakage attacks in federated learning: Research frontiers, taxonomy and future directions. *IEEE Network*, pages 1–8, 2023.

[85] Hongxu Yin, Arun Mallya, Arash Vahdat, Jose M Alvarez, Jan Kautz, and Pavlo Molchanov. See through gradients: Image batch recovery via gradinversion. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16337–16346, 2021.

[86] Binchi Zhang, Zihan Chen, Cong Shen, and Jundong Li. Verification of machine unlearning is fragile. In *Forty-first International Conference on Machine Learning*, 2024.

[87] Rui Zhang, Jian Liu, Yuan Ding, Zhibo Wang, Qingbiao Wu, and Kui Ren. "adversarial examples" for proof-of-learning. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 1408–1422. IEEE, 2022.

[88] Bo Zhao, Konda Reddy Mopuri, and Hakan Bilen. idlg: Improved deep leakage from gradients. *arXiv preprint arXiv:2001.02610*, 2020.

[89] Joshua C Zhao, Ahmed Roushdy Elkordy, Atul Sharma, Yahya H Ezzeldin, Salman Avestimehr, and Saurabh Bagchi. The resource problem of using linear layer leakage attack in federated learning. *arXiv preprint arXiv:2303.14868*, 2023.

[90] Joshua C. Zhao, Atul Sharma, Ahmed Roushdy Elkordy, Yahya H. Ezzeldin, Salman Avestimehr, and Saurabh Bagchi. Secure aggregation in federated learning is not private: Leaking user data at large scale through model modification, 2023.

[91] Junyi Zhu and Matthew Blaschko. R-gap: Recursive gradient attack on privacy. *arXiv preprint arXiv:2010.07733*, 2020.

[92] Junyi Zhu and Matthew Blaschko. R-gap: Recursive gradient attack on privacy. *Proceedings ICLR 2021*, 2021.

[93] Ligeng Zhu, Zhijian Liu, , and Song Han. Deep leakage from gradients. In *Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2019.