Towards a Re-evaluation of Data Forging Attacks in Practice

Mohamed Suliman[1,2], Anisa Halimi[1], Swanand Ravindra Kadhe[1], Nathalie Baracaldo[1], Douglas Leith[2]
[1] *IBM Research* [2] *Trinity College Dublin, The University of Dublin*

Abstract

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.

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 [28, 29] and the CCPA [17] 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.

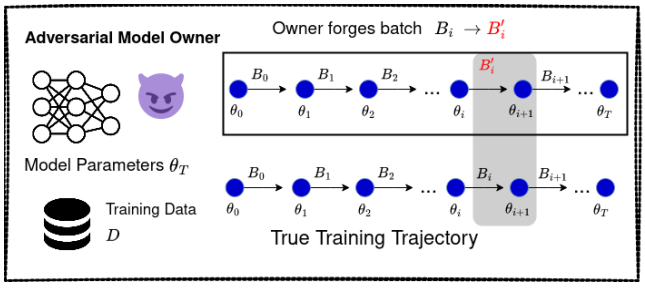


Figure 1: The adversarial model owner has access to the true execution trace denoting the sequence of model parameters and associated mini-batches. When performing a data forging attack, the model owner *forges* (*i.e.*, replaces) a batch B_i such that $(\mathbf{x}, \mathbf{y}) \in B_i$ with a different batch B_i' not containing $(\mathbf{x}, \mathbf{y})$ such that the resulting models are nearly identical.

Data forging [4, 16, 27, 33] 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 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 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 [15]. 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 [16, 27, 33] 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) [8, 14, 24] 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. 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* [5], 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 [5, 7, 30] has emerged as a response to GDPR's [29] *right to be forgotten* framework, and prior work [27, 33] 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 [16, 27, 33] argue that their attacks produce approximation errors that are indistinguishable from benign reproduction errors. On the other hand, Baluta et al. [4] 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. [4] 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 [16, 27, 33] 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. [4] 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.

2 Background

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 $B_T = \emptyset$).

2.2 Reproduction Errors

Recent work by Schlögl et al. [23] 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 [13] 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.

2.3 Data Forging

Thudi et al.'s [27] 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 [16, 27] 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. [16] improve upon its efficiency, with similar performance. Recently, Zhang et al. [33] 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.

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 1, 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 [4, 16, 27, 33]. We allow for a non zero error between recomputed and original checkpoints in contrast to the data

forging game defined by [4], who required $\varepsilon = 0$, 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 2.2 for a more detailed discussion).

Algorithm 1 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 $\varepsilon_{t+1} \leftarrow \|\theta_{t+1} - \theta^{\mathcal{V}}_{t+1}\|_2$
if $(\varepsilon_{t+1} > \varepsilon)$ **then**
    return REJECT
else
    return ACCEPT
end if

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-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 $\varepsilon_{t+1} := \|\theta_{t+1} - \theta^{\mathcal{V}}_{t+1}\|_2$ between their recomputed checkpoint, and the one present in the trace. The game ends with a REJECT result if $\varepsilon_{t+1} > \varepsilon$, 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 Are Current Data Forging Attacks Stealthy?

Previous data forging works [16, 27, 33] 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. [16] 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, [27, 33] 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. [27] have proven the existence of an adversary that can win the forging game defined in the previous section for any $\varepsilon > 0$, 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 [27] 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 $\varepsilon \ll 1$ (see Sec. 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 deter-

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

mine 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 [16, 27, 33]. **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.

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 $\varepsilon_{repr} := \arg\max_i \varepsilon_{i+1}$ *i.e.*, the reproduction error is the largest observed ℓ_2 distance between recomputations. In Table 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 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 [4, 16, 27] (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 [22] 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.

4.1.1 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 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

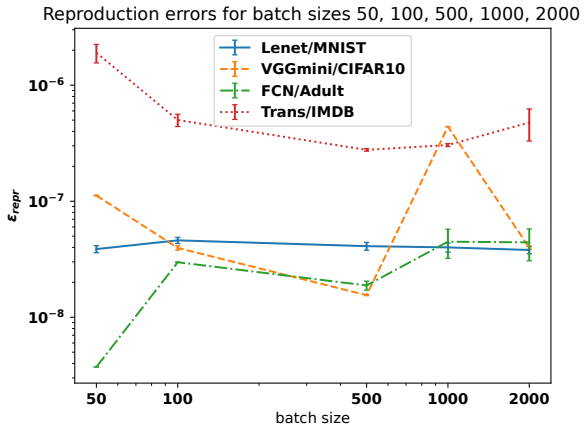


Figure 2: The observed largest reproduction error across multiple recomputations across batch size and model type and size.

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 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).

4.1.2 Mixed precision training can result in larger reproduction errors

Mixed 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 4.1.1.

To evaluate this claim, we re-train our models using both `float16` and `bfloat16` specifications and provide the largest observed reproduction errors in Table 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.

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

4.1.3 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 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 [1] and PyTorch [21] 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 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.

[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 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. 3b, the diagonal entries are zero because deterministic training results in zero error when recomputing on the same hardware.

4.1.4 Discussion

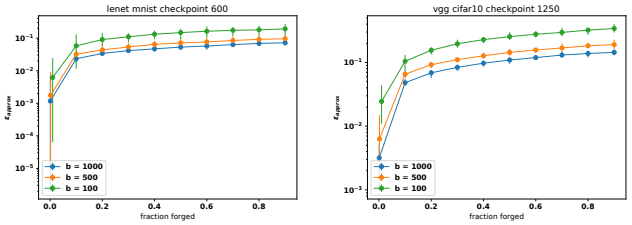
In this section, we have provided our results on observed reproduction errors as we vary models (large and small), datasets, batch sizes, floating point precision, and recomputation hardware, however we refrain from making any statement about what directly results in the magnitude of the reproduction 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.

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.

4.2.1 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 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. [27] and Zhang et al. [33]'s attacks have similar performance (see Appendix D for a comparison).



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

Figure 3: The average approximation error of Thudi et al.'s attack [27] (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.

4.2.2 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.

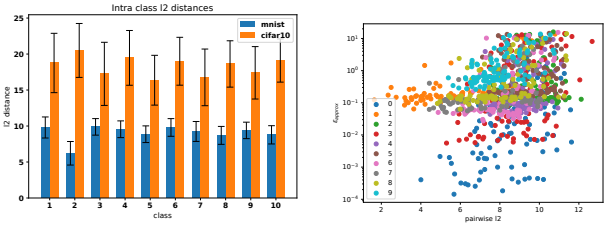
2. We then forge mini-batch B into B' across training at each checkpoint and measure the approximation error.

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 at least 4 orders of magnitude larger than our largest observed reproduction errors for every model and dataset setup we consider for both `float32` and `float16` formats. Approximation errors were closer to reproduction errors only for the `bfloat16` mixed precision regime, however they were still 2 orders of magnitude larger. In Figures 5, 8, and 10, 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 6, 7, 9 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. 9). However, the performance of the attack is not consistent across training, as seen in Figures 6, and 7, where approximation errors can vary from being only a single order of magnitude larger to nearly 5 orders of magnitude larger (see Fig. 6a).

One possible explanation regarding the difference in forging performance is the similarity of examples within the MNIST dataset, however, Figure 4 highlights that there is no observed correlation between pairwise ℓ_2 distance and approximation error. Instead, 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. Figures 11a,11b 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



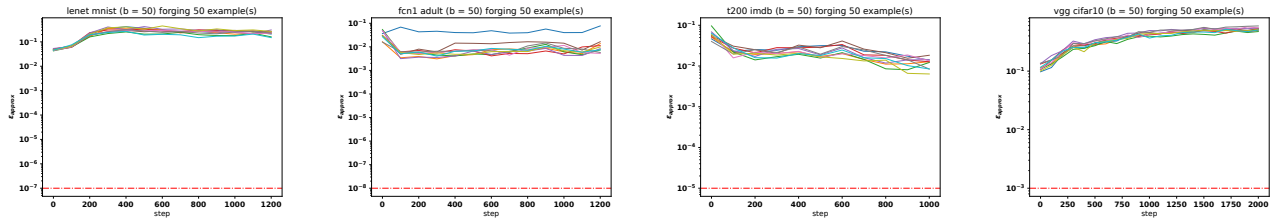
(a) (b)

Figure 4: While MNIST has lower intra-class pairwise ℓ_2 variance than CIFAR10 (Fig 4a), we do not find a correlation between pairwise distance and approximation error (Fig 4b).

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 11c, and 11d for similar behaviour for a transfomer.

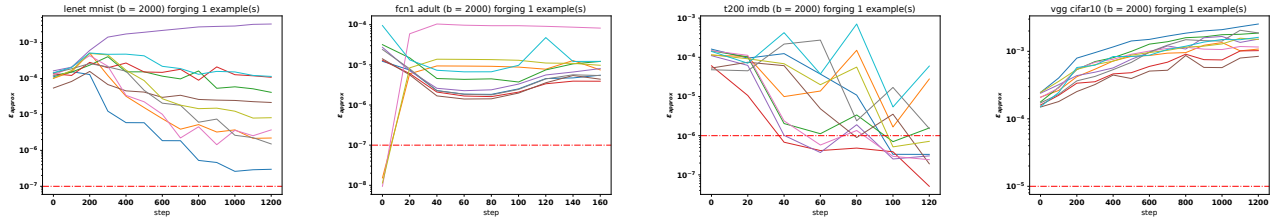
4.3 Is Data Forging Stealthy?

From our experimentation in Section 4.1, we found that the model type, batch size, floating point precision, and GPU hardware can affect the magnitude of benign reproduction errors, and in Section 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 5: 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.



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

Figure 6: Data forging approximation errors across stages of training where the forging fraction is $1/2000$ for `float32` training.

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.

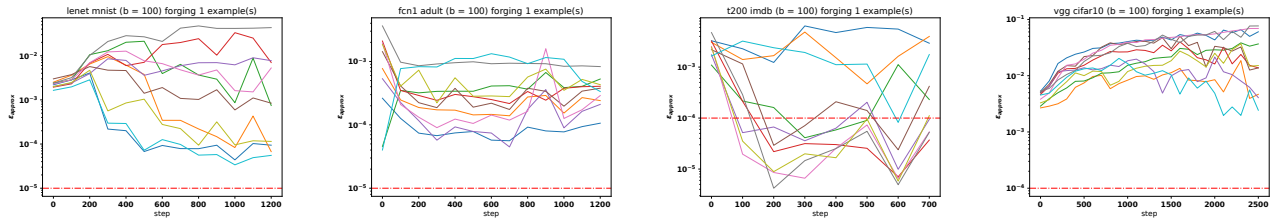
However the approach is not perfect. Determining accurate error thresholds value can be tricky when it is not possible to perform cross hardware recomputation. Consider the following scenario that highlights the difficulty of correctly setting ε in practice: a verifier intends to verify a LeNet/MNIST ($b = 100$) execution trace on a RTX4090 GPU. They set $\varepsilon = 10^{-8}$ for their error threshold as this is their observed level of reproduction error on their hardware[4]. Now suppose that the 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 3b). As a result, even the honest trace would not pass verification if $\varepsilon = 10^{-8}$. 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. [4] consider only the scenario of a strict verifier that always selects $\varepsilon = 0$. 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 these conditions[5] and may be necessary in high stakes applications such as medical machine learning models, we argue that always demanding $\varepsilon = 0$ is unrealistic in practice, particularly when the original hardware is not available.

Due to the existence of reproduction error, prior data forging works [16, 27, 33] have recognised the practical necessity of a non-zero threshold during verification, and consider the $\varepsilon > 0$ 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 pro-

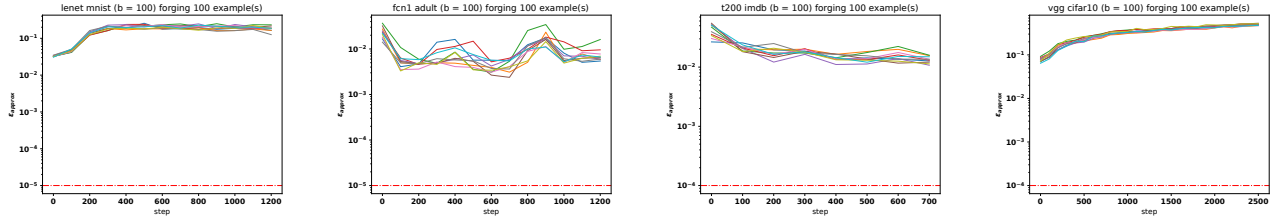
[4] See the corresponding plot in Figure 2 for LeNet/MNIST ($b = 100$).

[5] See Table 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) Trans/IMDB (d) VGGmini/CIFAR10

Figure 7: Data forging approximation errors across stages of training where the forging fraction is $1/100$ for `float16` training.



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

Figure 8: Data forging approximation errors across stages of training where the forging fraction is 1 for `float16` training.

vide 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.

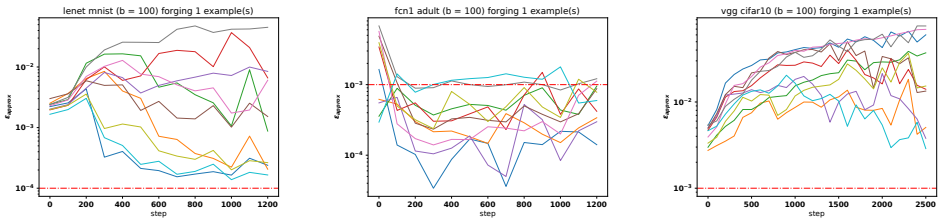
Verifiers are therefore met with a tradeoff when choosing their error threshold. Seeking to not reject honest traces, and being cognisant of the fact that there may exist other factors that affect reproduction errors; they set their threshold to be larger than the reproduction errors which they have 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) [11]. A privacy parameter ε is chosen to determine how private[6] a given function is. Within DP, ideally $\varepsilon < 1$, however, choosing such a small epsilon can hinder the performance of the model. We have a similar setup here, where $\varepsilon = 0$ 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.

[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.

4.4.1 Data Protection, Differential Privacy, and Data Forging

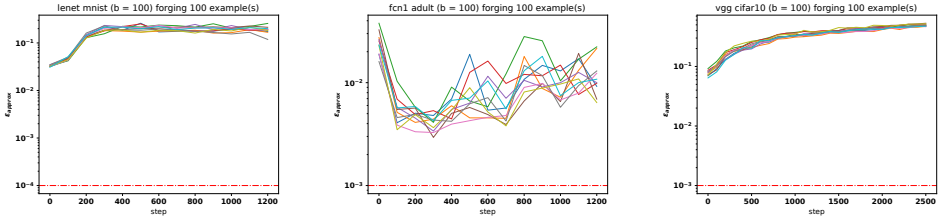
Data protection legislation such as GDPR aims to limit the potential harm that may befall an individual from the use of their personal information. Existing methods to do so are *(i)* to limit the use of their personal data entirely, or *(ii)* to provide guarantees that any undue harm will not result 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 [18] 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 [3]. 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 [6, 9, 10], and the latter provides no protection against potentially stronger attacks.

The emergence of Proof of Learning (PoL) [15], i.e., the recomputation of execution traces, and further zero-knowledge methods of PoL [2] 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



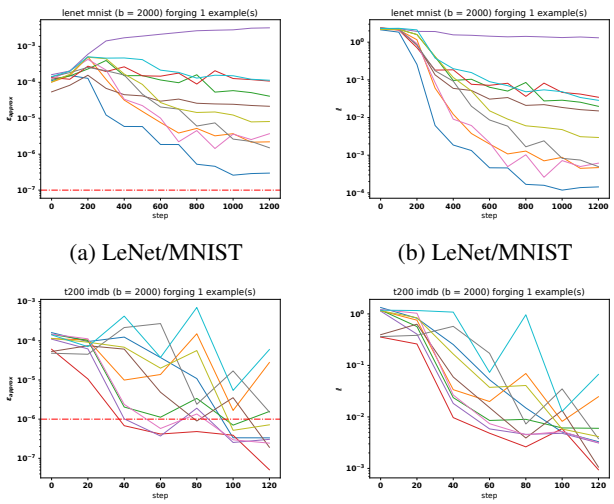
(a) LeNet/MNIST (b) FCN/Adult (c) VGGmini/CIFAR10

Figure 9: 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 10: Data forging approximation errors across stages of training where the forging fraction is 1 for `bfloat16` training.



(a) LeNet/MNIST (b) LeNet/MNIST

(c) Trans/IMDB (d) Trans/IMDB

Figure 11: 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.

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 [12, 15] for PoL to become more practical, as well as the utilisation of open-source, reproducible floating point algorithms.

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 [16, 27, 33] 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 [20, 31, 32, 35–37] 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 [19], 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. [4] 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. Previous work by [26] has shown that the use of the gradient matching optimisation algorithm pioneered by [37] does not converge to solutions with sufficiently identical gradients, thus an analytical approach to the problem of exact data forging is required. 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 $\mathcal{X} \times \mathcal{Y}$.

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 $b > 1$.

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 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.

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

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

that the same gradient is still preserved (see Fig. 12 for an example).

Figure 12: **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. [20] within the context of gradient inversion.

Exact data forging can be performed using this *mini-batch perturbation* approach. 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. While we prove the existence of an infinite number of possible perturbations that may applied to the original mini-batch that preserve the gradient, we have no guarantee that a perturbation that preserves image validity exists, even for relatively simple linear models. Efficiently finding such a perturbation is a non trivial task, and is currently the main barrier to successful exact data forging.

5.4 Exact Data Forging for $L = 1$ Networks

Recall that for a $L = 1$ network with single parameter matrix $\mathbf{W}$, the gradient of crossentropy loss is given by $\nabla_{\mathbf{W}}\mathcal{L}(B) = \frac{1}{b}\mathbf{X}(f_{\mathbf{W}}(\mathbf{X})\text{diag}(\mathbf{v}) - \mathbf{Y})^T$, where $\mathbf{v}_k = \sum_{j=1}^{n} \mathbf{y}_j^{(k)}$, 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 $b > rank(\nabla_{\mathbf{W}}\mathcal{L}(B))$, then there exists an infinite number of distinct mini-batches $B' \subset \mathbb{R}^d \times \mathbb{R}^n$ of size b such that $\nabla_{\mathbf{W}}\mathcal{L}(B') = \nabla_{\mathbf{W}}\mathcal{L}(B)$.*

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. 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: *(i)* sample any matrix $\mathbf{D} \in \mathbb{R}^{b\times n}$ of such that $\sum_{j=1}^{n} \mathbf{D}_{ij} = 0$ and $\text{rank}(\mathbf{D}) \geq r$, *(ii)* solve for $\mathbf{X}'$ the matrix equation $\mathbf{X}'\mathbf{D} = b\nabla_{\mathbf{W}}\mathcal{L}(B)$, *(iii)* set $\mathbf{Y}' := f_{\mathbf{W}}(\mathbf{X}')\text{diag}(\mathbf{v}) - \mathbf{D}^T$, and *(iv)* 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 13 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.

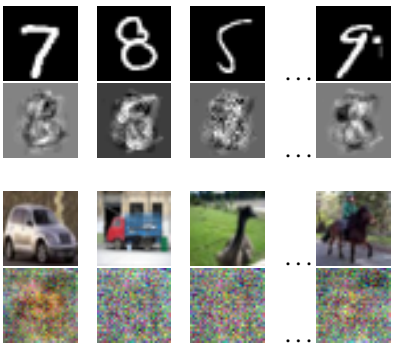


Figure 13: A mini-batch from MNIST (top) and CIFAR10 (bottom) and distinct forged counterpart produced by our error matrix sampling approach. Both examples produce an approximation error on the order of 10^{-7}, 5 orders of magnitude smaller than existing attacks.

6 Related Work

6.1 Proof of Learning

Jia et al. [15] 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.

6.2 PoL Adversarial Examples

Zhang et al. [34] 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. [12] 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 [12]. 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 [12, 34] 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 [25] *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.

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 [16, 27, 33]. 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.

8 Ethics Considerations

The danger of data forging attacks is that they appear to be the perfect tool to evade data auditing and compliance regulations. Malicious model owners who have trained their models on copyrighted, and or sensitive personal data can *forge* these datasets into legally compliant ones, and publicly claim that the forgery is in fact the true dataset. As a result, these problematic models can remain operational, furthering the risk of potential harm to the data owners. This can come in the form of copyright infringement, and or the unconsented disclosure of sensitive data to 3rd parties. Research that furthers the development of data forging must contend with potential harm these attacks can facilitate.

However, in our work, we have shown that attempts at data forging are easily detectable by an informed adversary, and that, in their current instantiation, they are not a real threat against data governance. From an ethical point of view, our work can be seen as a furthering an optimistic view of the future risk of evading data regulation via data forging. We have seen that current attacks are not able to fully realise their serious implications on data governance, and our theoretical results suggests that developing attacks that can is a non trivial, computationally infeasible task.

In carrying out our research, we believe there are no ethical issues. Our results are the result of training on public datasets (MNIST and CIFAR10), and we replicate data forging attacks in a closed environment on execution traces resulting from these public datasets.

9 Open Science

We are committed to the open science policy by providing all our code at the following link.

References

[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] 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.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] Nicholas Carlini, Florian Tramèr, Eric Wallace, Matthew Jagielski, Ariel Herbert-Voss, Katherine Lee, Adam Roberts, Tom Brown, Dawn Song, Úlfar Erlingsson, Alina Oprea, and Colin Raffel. Extracting training data from large language models. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 2633–2650, August 2021.

[10] 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.

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

[12] 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.

[13] 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.

[14] 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.

[15] 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.

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

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

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

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

[20] 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.

[21] 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.

[22] 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.

[23] 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.

[24] 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.

[25] 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.

[26] Mohamed Suliman, Swanand Kadhe, Anisa Halimi, Douglas Leith, Nathalie Baracaldo, and Ambrish Rawat. Data forging is harder than you think. In *Privacy Regulation and Protection in Machine Learning*, 2024.

[27] 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.

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

[29] 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.

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

[31] 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.

[32] H. Yin, A. Mallya, A. Vahdat, J. M. Alvarez, J. Kautz, and P. Molchanov. See through gradients: Image batch recovery via gradinversion. In *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 16332–16341, Los Alamitos, CA, USA, jun 2021. IEEE Computer Society.

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

[34] 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.

[35] Bo Zhao, Konda Reddy Mopuri, and Hakan Bilen. idlg: Improved deep leakage from gradients, 2020.

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

[37] Ligeng Zhu, Zhijian Liu, and Song Han. Deep leakage from gradients. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

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 multiclass logistic regression. $\nabla_{\mathbf{W}_i} \ell$ is given by the outer product of the vectors $\mathbf{a}_{i-1}$ and δ_i:

$$\nabla_{\mathbf{W}_i} \ell = \mathbf{a}_{i-1} \delta_i^T \tag{1}$$

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

$$\delta_i = \text{diag}(\mathbf{h}_i) \mathbf{W}_{i+1} \delta_{i+1}, \tag{2}$$

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 δ_L as the following:

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

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

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

which allows us to rewrite Equation 3 as

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

From (5), we have that $\delta_L = v\hat{\mathbf{y}} - \mathbf{y}$, 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 $\mathbf{x}' \delta_1'^T = \mathbf{x} \delta_1^T$. If $\nabla_{\mathbf{b}_1} \ell(\mathbf{x}', \mathbf{y}') = \nabla_{\mathbf{b}_1} \ell(\mathbf{x}, \mathbf{y})$, then $\delta_1' = \delta_1$, resulting in the equation $\mathbf{x}' \mathbf{u}^T = \mathbf{x} \mathbf{u}^T$ where $\mathbf{u} = \delta_1' = \delta_1$. 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 $\delta_L' = \delta_L$. If, $\nabla_{\mathbf{W}_L} \ell(\mathbf{x}', \mathbf{y}') = \nabla_{\mathbf{W}_L} \ell(\mathbf{x}, \mathbf{y})$, then $\mathbf{a}_{L-1}' \delta_L'^T = \mathbf{a}_{L-1} \delta_L^T$. Since $\delta_L' = \delta_L$ 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

$$\begin{aligned} \delta_L' &= \delta_L \\ v'\hat{\mathbf{y}}' - \mathbf{y}' &= v\hat{\mathbf{y}} - \mathbf{y} \\ v'\mathbf{u} - \mathbf{y}' &= v\mathbf{u} - \mathbf{y} \\ \mathbf{y} &= (v - v')\mathbf{u} + \mathbf{y}' \end{aligned} \tag{6}$$

□

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 (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

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

□

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:

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

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 $\mathbf{a}_i^{(k)} := [a_1^{(k,i)} a_2^{(k,i)} \ldots a_{d_i}^{(k,i)}]^T$:

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

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:

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

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 $\delta_i^{(k)}$ 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

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

Equation (11) is the matrix version of (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:

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

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 $\delta_L'^{(k)} = \delta_L^{(k)}$. Therefore $\mathbf{D}_L' = \mathbf{D}_L$, and by (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 (11) we then have $\mathbf{D}'_i = (\mathbf{D}'_{i+1}\mathbf{W}^T_{i+1}) \odot H'_i = (\mathbf{D}_{i+1}\mathbf{W}^T_{i+1}) \odot H_i = \mathbf{D}_i$. 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 (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 (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:

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

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:

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

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

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

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

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

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

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

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 $\mathbf{D}$. □

B Accuracy Measurements

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.

	$b=1$	$b=2$	$b=3$	$b=4$	$b=5$
$v=2$	✗(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 4: 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.

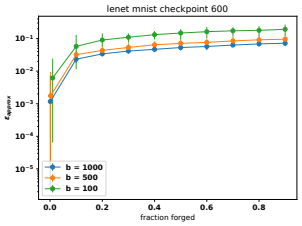
Model/Dataset	$b=50$	$b=100$	$b=500$	$b=1000$	$b=2000$
LeNet/MNIST	$0.94\pm2.4e-4$	$0.94\pm1.6e-4$	$0.95\pm4.7e-5$	0.95 ± 0	$0.95\pm8.1e-5$
VGGmini/CIFAR10	$0.63\pm3.3e-4$	$0.62\pm2.4e-4$	$0.64\pm1.4e-4$	$0.64\pm1.9e-4$	$0.55\pm4.5e-4$
FCN/Adult	0.75 ± 0	0.75 ± 0	0.75 ± 0	0.75 ± 0	0.75 ± 0
Transformer/IMDB	$0.87\pm1.3e-3$	$0.85\pm1.9e-3$	$0.83\pm9.5e-3$	$0.84\pm2.4e-03$	$0.84\pm1.9e-3$

Table 5: Observed variance in accuracy from GPU auto-tuning non determinism

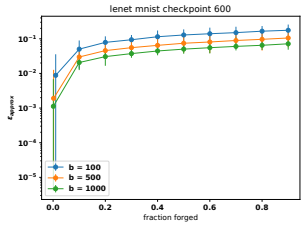
We run this experiment on a logistic regression model, as well as a $L=2$ fully connected neural network with 10 hidden units and ReLU activation function We additionally fix $d=4$ and $n=3$. Table 4 summarises our results.

D Comparison of Data Forging Attacks

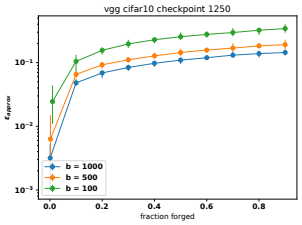
In Figure 14, we plot the performance of both Thudi et al. [27] and Zhang et al. [33]'s data forging attacks, for both LeNet/MNIST and VGGmini/CIFAR10 execution traces.



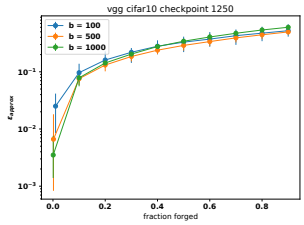
(a) Thudi on LeNet/MNIST



(b) Zhang on LeNet/MNIST



(c) Thudi onVGGmini/CIFAR10



(d) Zhang on VGGmini/CIFAR10

Figure 14: We plot the average approximation error of both Thudi et al.'s [27] and Zhang et al.'s [33] 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.