# Privacy Backdoors: Stealing Data with Corrupted Pretrained Models

**Shanglun Feng** [1]   **Florian Tramèr** [1]

## Abstract

Practitioners commonly download pretrained machine learning models from open repositories and finetune them to fit specific applications. We show that this practice introduces a new risk of *privacy backdoors*. By tampering with a pretrained model's weights, an attacker can fully compromise the privacy of the finetuning data. We show how to build privacy backdoors for a variety of models, including transformers, which enable an attacker to reconstruct individual finetuning samples, with a guaranteed success! We further show that backdoored models allow for *tight* privacy attacks on models trained with differential privacy (DP). The common optimistic practice of training DP models with loose privacy guarantees is thus insecure if the model is not trusted. Overall, our work highlights a crucial and overlooked supply chain attack on machine learning privacy.

## 1. Introduction

Sharing and finetuning of large pretrained models has become a common practice, enabling rapid prototyping and development of new applications across various domains. Platforms like Huggingface currently host nearly 500,000 models, shared by various companies, researchers, and other users. This trend brings to light new security concerns, particularly in the form of supply chain attacks. Prior work has recognized the impact of such attacks on model *integrity*, where a *backdoor* is planted into a model to hijack its downstream behavior (Gu et al., 2017; Liu et al., 2018).

In this work, we shift the focus from integrity to *privacy* vulnerabilities, and introduce **privacy backdoors**, where a malicious model provider tampers with model weights to compromise the privacy of future finetuning data. Our backdoor attacks create "data traps" that directly write some data points to the model weights during finetuning. The trapped data can then be extracted by reading from the finetuned model's weights. Compared to prior related attacks in Federated Learning (Boenisch et al., 2023; Fowl et al., 2021) that steal data points in a single training step, our data traps have to survive an entire finetuning run, with multiple training epochs and thousands of update steps.

We thus propose a new backdoor design that is single-use: once our backdoor activates and a data point is written to the model's weights, the backdoor becomes inactive, thereby preventing further alteration of those weights during training. Our backdoor thus acts a bit like a *latch*, the logic circuit underlying digital memory: once the backdoor is set and the data is written to memory (i.e., to the model weights), it "latches" on until the end of training.

By design, our attacks capture individual training examples with high probability, with minimal impact on the pretrained model's utility. We apply these attacks to MLPs and pretrained transformers (ViT (Dosovitskiy et al., 2020) and BERT (Devlin et al., 2019)) and reconstruct dozens of finetuning examples across various downstream tasks.

We then consider a stronger black-box threat model, where the attacker only has query access to the finetuned model. By adapting techniques from the model extraction literature (Tramèr et al., 2016; Carlini et al., 2020), we show that even a black-box attacker can recover entire training inputs. We further show that our backdoors enable simpler, *perfect membership inference attacks*, which infer with 100% accuracy whether a data point was used for training. We use these backdoors to build the first *tight* end-to-end attack on the seminal differentially private SGD algorithm of Abadi et al. (2016). That is, the privacy leakage observable by our adversary nearly matches the provable upper-bound from the algorithm's privacy analysis. Our result thus challenges the common assumption that the privacy guarantees of DP-SGD are overly conservative in practice. This has led to the adoption of loose privacy budgets (e.g., $\varepsilon \geq 9$ in (Ramaswamy et al., 2020)), which we show to be insecure in the presence of backdoored models.

Overall, our results bring to light a new attack vector in the modern machine learning supply chain, and emphasizes the need for more stringent privacy protections when operating with untrusted shared models.

---

[1]Department of Computer Science, ETH Zurich, Zurich, Switzerland. Correspondence to: Florian Tramèr <florian.tramer@inf.ethz.ch>.

Code to reproduce our experiments is at .

## 2. Related Work

**Privacy attacks.** Neural networks memorize training data. Attackers can exploit this to launch membership inference attacks (Shokri et al., 2017) or data extraction attacks (Carlini et al., 2021; 2023). These attacks can be strengthened by an attacker who *poisons* the model to increase memorization (Tramèr et al., 2022; Panda et al., 2023). Song et al. (2017) show that an attacker who tampers with the training *code* can cause the model to exfiltrate training data.

**Backdoor attacks.** Gu et al. (2017); Liu et al. (2018) show how to tamper with a pretrained model to inject malicious targeted behaviors that compromise the model's *integrity*. Hong et al. (2022) show that such backdoors can be *hand-crafted* by directly editing the weights of a pretrained model.

**Data stealing in Federated Learning.** A malicious server in Federated Learning can tamper with the model so that clients' gradients leak training data (Zhao et al., 2023; Fowl et al., 2021; 2022; Boenisch et al., 2023; Wen et al., 2022; Phong et al., 2018). We extend the attack of Fowl et al. (2021), which creates sparse activations in a linear layer, so that the gradient encodes training inputs.

**Privacy backdoors in pretrained models.** Concurrent work by Wen et al. (2024) and Liu et al. (2024) also introduces privacy backdoors for pretrained models, but with a different focus than ours. They consider weaker attacker goals (respectively membership inference or extraction without success guarantees), but also a weaker attacker (with only black-box access to the finetuned model). Their attacks are thus incomparable to ours: they are easier to mount, but also less devastating than ours.

**Tightness of DP-SGD.** The DP-SGD algorithm (Abadi et al., 2016) trains models with differential privacy (Dwork et al., 2006). The algorithm's worst-case analysis assumes that one model weight is updated if and only if a target input is present. This analysis is tight for a strong attacker who can observe all training steps (Nasr et al., 2021), but it is presumed to be loose for a more realistic "end-to-end" attacker who only observes the final model. Privacy parameters are thus often set very optimistically in practice.

## 3. Threat Model

In contrast to prior attacks on Federated Learning, we assume a weaker attacker who cannot observe individual gradient updates. Instead, our attacker tampers with a pretrained model *once*, before sending it to the victim.

We assume the victim finetunes the backdoored model on a classification task using SGD for multiple epochs. The victim adds a new linear layer to the backdoored model and then finetunes the entire model ("full finetuning"). We leave an extension to other types of finetuning (e.g., LoRA (Hu et al., 2021)) for future work.

We consider two types of attackers, with **white-box** or **black-box** access to the finetuned model. A white-box attacker can inspect the finetuned model's weights, while the black-box attacker can query the model on arbitrary inputs. White-box access is a stronger assumption, which enables our most powerful data extraction attacks. We show that the weaker black-box setting allows for perfect membership inference attacks, as well as data extraction attacks for simple models.

## 4. Warmup: White-box Data Stealing in MLPs

### 4.1. Attack Description

In order to illustrate our approach, we start with the simplest case of backdooring a linear unit (i.e., one element of a linear layer), as in (Fowl et al., 2021):

$$h = \text{ReLU}\left(\mathbf{w}^\top \mathbf{x} + b\right), \tag{1}$$

where $\mathbf{x} \in \mathbb{R}^m$ is the unit's input, and $\mathbf{w} \in \mathbb{R}^m, b \in \mathbb{R}$ are the unit's weight and bias.

As described in Fowl et al. (2021), we can set the parameters $\mathbf{w}, b$ to ensure that, with high probability, a single input $\mathbf{x}$ in a training batch activates the neuron (i.e., $h > 0$). If we backpropagate the training loss $\mathcal{L}$, we then get:

$$\nabla_{\mathbf{w}}\mathcal{L} = \frac{\partial \mathcal{L}}{\partial h} \cdot \mathbf{x}, \quad \nabla_b \mathcal{L} = \frac{\partial \mathcal{L}}{\partial h}. \tag{2}$$

Dividing $\nabla_{\mathbf{w}}\mathcal{L}$ by the scalar $\nabla_b \mathcal{L}$ recovers the input $\mathbf{x}$.

In a Federated Learning setting, we are done since the attacker gets to see these gradient updates directly. But in our case, the gradient updates are simply added to the parameters and training continues. The attacker only sees the final model. This introduces a new challenge: we need the captured inputs to "survive" the entire training run, without being mixed with other inputs captured in subsequent training steps. We thus propose to build a backdoored unit with a stronger "single-use" property: *Once the backdoor has been activated, it will never be active again afterwards.*

We call such a backdoor a **data trap**. To design a data trap, we need the gradient update to "shut down" the backdoor. Assume the backdoor activates on input $\hat{\mathbf{x}}$, and we update the weights using SGD with learning rate $\eta$ (with the gradients in (2)). The updated backdoor's output on a new input $\mathbf{x}$ is:

$$h' = \mathbf{w}'^\top \mathbf{x} + b' = (\mathbf{w}^\top \mathbf{x} + b) - \eta \cdot \frac{\partial \mathcal{L}}{\partial h} \cdot \left(\hat{\mathbf{x}}^\top \mathbf{x} + 1\right). \tag{3}$$
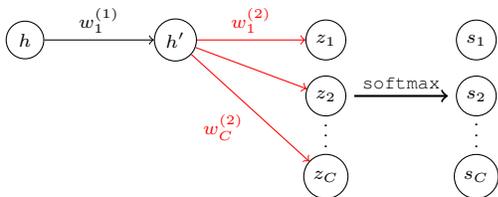
*Figure 1.* Illustration of the output of a data trap ($h = \text{ReLU}\left(\mathbf{w}^\top \mathbf{x} + b\right)$) connected to the model's output. The weights of the classification head (in red) are typically not under the control of the attacker, and randomly initialized before finetuning.

A sufficient condition for the backdoor to shut down (i.e., $h' \leq 0$ for all $\mathbf{x}$) is that:

1. $\hat{\mathbf{x}}^\top \mathbf{x} + 1 > 0$.

2. $\frac{\partial \mathcal{L}}{\partial h}$ is positive and large enough.

The first condition, $\hat{\mathbf{x}}^\top \mathbf{x} + 1 > 0$, depends only on the input of the backdoor unit, and the attacker can ensure the inputs satisfy this (e.g., by mapping inputs to $[0, 1]^m$).

The second condition, that $\frac{\partial \mathcal{L}}{\partial h}$ is positive and large, requires more work. Our construction is illustrated in Figure 1. We connect the backdoor's output $h$ to a hidden unit $h'$, with weight $w_1^{(1)}$, i.e., $h' = \text{ReLU}\left(w_1^{(1)} h\right)$. The unit $h'$ then connects to the model's logit layer $\mathbf{z}$ with weights $\mathbf{w}^{(2)}$, and the logits are passed through a softmax before computing the cross-entropy loss. A standard gradient calculation gives:

$$\frac{\partial \mathcal{L}}{\partial h} = \frac{\partial \mathcal{L}}{\partial h'} \cdot \frac{\partial h'}{\partial h} = \Big( \sum_{i=1}^{C} (s_i - y_i) \cdot w_i^{(2)} \Big) \cdot w_1^{(1)} , \quad (4)$$

where $C$ is the number of classes, $s_i$ is the softmax score for the $i$-th class, and $y_i$ is one if the input belongs to the $i$-th class, and zero otherwise.

Recall that we want this derivative to be positive and large. The "large" part is easy: we just set $w_1^{(1)}$ to be large. The sign of the derivative depends on the ground-truth class of the captured input (i.e, $s_i - y_i$ is negative for the index of the true class). It turns out that setting $w_1^{(1)}$ to be large guarantees that the derivative is positive with probability $1 - 1/C$ (e.g., 99.9% for ImageNet classification). Indeed, if $w_1^{(1)}$ is large enough the logits $\mathbf{z}$ are essentially random (the benign part of the model has negligible influence). The softmax scores then concentrate on the class $j$ with the highest weight $w_j^{(2)}$, and the sum in (4) is positive if $j$ is a wrong class. So we get a large positive gradient as desired!

In conclusion, if our backdoor fires on a misclassified input, it shuts down and is inactive for the rest of training. If we are unlucky and we get a negative gradient into the backdoor, i.e., $\frac{\partial \mathcal{L}}{\partial h} < 0$, the backdoor does not shut and activates again on future inputs. This is the main challenge we have to tackle when scaling our backdoor construction to large

transformers, which we discuss in Section 5.

### 4.2. Multiple Backdoor Units

So far, we described a single backdoor unit that captures one training input. We can simply replicate this construction above for multiple linear units, and initialize the backdoor weights so that they activate on different inputs.

The finetuning data induces a distribution over the values $\mathbf{w}^\top \mathbf{x}$, and we set the bias $b$ so that a small fraction of inputs give positive activations. To capture diverse inputs in different backdoors, we select weights $\mathbf{w}$ that align with different subsets of the training data (in practice, we simply sample the weights from a uniform distribution over the sphere).

The biases can be set independently for each backdoor. If the bias is too large, multiple inputs in a batch might trigger the same backdoor which makes reconstruction difficult. If the bias it too small, some backdoors may never fire on any training input. For a dataset of size $D$ and batch size of $B$, we thus want the fraction of training inputs $p$ that activate a backdoor to satisfy $p > 1/D$ and $p \ll 1/B$. For typical finetuning setups (where $B \ll D$), the range of allowable biases is large, so the attacker only needs to know a loose approximation of the distribution $\mathbf{w}^\top \mathbf{x}$.

Note that each backdoor can fail independently with probability $1 - 1/C$, if the captured input is correctly classified. With $k$ perfectly calibrated backdoors, we thus expect to capture $k \cdot (1 - 1/C)$ inputs.

### 4.3. Experiments

We evaluate our backdoor construction on a MLP model trained on CIFAR-10. We implement a 3-layer backdoored perceptron as shown in Figure 1. The model's layers have 256 hidden units, 64 of which are used for backdoors. Input images are scaled and flattened to the range $[0, 1]^{3072}$.

Figure 2 shows reconstructed images from this toy MLP model, most of which are indistinguishable from their corresponding ground truth images. The attack can fail in two orthogonal cases. If the backdoor gets a negative gradient signal (because the captured input is correctly classified), the backdoor fails to shut and is re-activated many times in a subsequent batch. This typically results in an unrecoverable backdoor (e.g., top row, fourth from the left). A more benign failure case is when multiple inputs in a batch activate the backdoor, in which case our attack recovers a mixture of the captured inputs (e.g., top row, second from the right).

## 5. Privacy Backdoors in Transformers

In the previous section, we designed data traps for simple MLPs. To illustrate the general applicability of privacy backdoors, we now extend our study to real transformer models
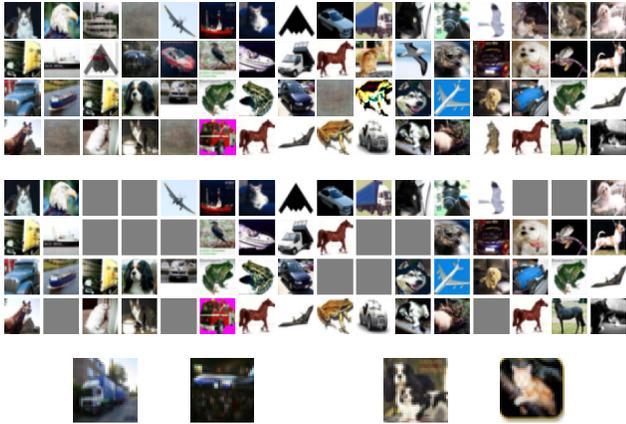
*Figure 2.* Reconstructed images from a backdoored MLP model. **Top**: Reconstruction; **Middle**: Ground truth, or a gray image if the backdoor failed to capture a unique input (see Appendix D.1 for details); **Bottom**: Ground truths for failed backdoors that captured a mix of two inputs (left corresponds to row 2, column 4; right corresponds to row 1, column 15).

for both vision and text. The transformer architecture underlies most foundation models in use today, and is thus a natural candidate for a backdoored pretrained model.

We assume victims finetune the entire backdoored model for classification (i.e., *full finetuning*). We leave the study of parameter-efficient methods like LoRA (which are popular in computationally constrained settings) to future work.

**Transformers.** A transformer (Vaswani et al., 2017) takes as input a sequence of *tokens*, each mapped to an *embedding vector*. For vision transformers (Dosovitskiy et al., 2020), the tokens are patches from the input image that are mapped to embeddings using a convolutional layer. For text transformers, tokens are (sub-)words in the input sentence that are mapped to embedding vectors using a dictionary. Each token embedding is augmented with a *position embedding*, which encodes the *index* of the token in the input sequence.

We focus on ViT (Dosovitskiy et al., 2020) and BERT (Devlin et al., 2019), whose embeddings have 768 dimensions. The transformer's input is thus a $k \times 768$ matrix, where $k$ is the number of tokens in an input (padded to a fixed length).

The transformer encoder then consists of a number of *blocks*, which apply a dimension-preserving function $f : \mathbb{R}^{k \times 768} \to \mathbb{R}^{k \times 768}$. These blocks combine the following operations:

- MLPs: One or two linear layers with a non-linear activation (typically a GELU (Hendrycks & Gimpel, 2016)) are applied to each of the $k$ feature vectors.
- Attention layers: A layer that computes dependencies between the $k$ token vectors, and reweighs them.
- Shortcut connections: A block's input is typically added to its output, to facilitate learning (He et al., 2016).

- Layer normalizations: Each feature vector is normalized to zero mean and unit variance, followed by a learned affine transform.

The model outputs features of dimension $k \times 768$. To finetune a classifier, we add a linear layer to the features of the first token (a special [CLS] token prepended to the input).

### 5.1. Challenges

**Maintaining model utility.** If the attacker wants the backdoored model to be used, it should be useful for finetuning. So the attacker needs to ensure the backdoored model retains a "benign" part that acts as a useful feature extractor.

For simplicity, we do not retrain existing transformers to accommodate backdoors. We start from benign pretrained transformers (ViT (Dosovitskiy et al., 2020) and BERT (Devlin et al., 2019)), and remove a fraction of the model's feature representations and encoder blocks for our backdoor construction (see Appendix D.2 for details about model preprocessing). Our design then aims to minimize interference between the benign and backdoor portions of the model.

**Moving beyond linear layers.** Transformers are more complex than MLPs! In addition to linear layers (which are useful for building data traps), they have attention layers, layer normalizations, and more complex activations than ReLUs. We will need additional tricks to ensure that all these layers propagate the backdoor activations and gradients properly without destroying the model's utility.

**Capturing all tokens in an input.** If we apply the construction from Section 4 to capture multiple input vectors of a transformer, we would capture individual *tokens*. These tokens would likely belong to different input sequences (i.e., we might capture just one word from many different sentences). Instead, we would like to capture a few *full* inputs. This will require designing a family of backdoors that fire in tandem for all tokens in an input sequence, or not at all.

**Building robust backdoors.** Our backdoors for MLPs in Section 4 only required setting adversarial weights in two layers. These weights are only updated when the backdoor first activates, and are unused after that. In contrast, our backdoors for transformers are much more complex and combine a number of modules (as described in Section 5.2).

One challenge is that the initial weights in these modules shift during training, as gradients will flow into them. This will disrupt the adversarial behavior that these modules were designed to implement and risks destroying the backdoor. Another challenge is maintaining the model's utility in the presence of "failing" backdoors, where a correctly classified input activates the backdoor and causes negative gradients to further open the backdoor. The backdoor then fires many
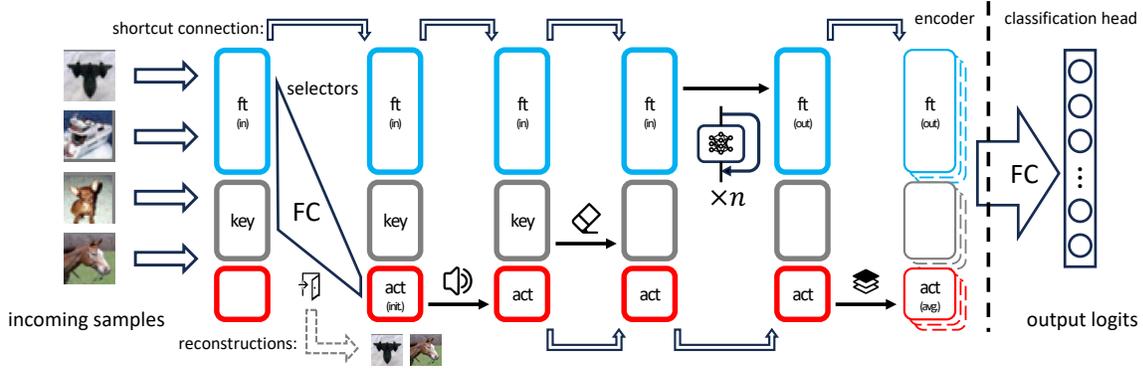
*Figure 3.* The logical architecture of a backdoored transformer. **ft**: *benign features* inheriting the utility of the pretrained weights. **key**: the features for selectively activating a backdoor, and the data to be captured. **act**: *activation signals*, i.e., outputs from backdoor units. From left to right, our backdoor construction consists of: an *input module* that creates the logical feature separation $\mathbf{x} = [\mathbf{ft}, \mathbf{key}, \mathbf{0}]$; the main *backdoor module* (🚪) that captures inputs in a linear layer; an *amplifier module* (🔊) that increases backdoor activation signals; an *erasure module* (◇) that wipes out unneeded features; $n$ *signal propagation modules* (🕸) that compute standard benign features **ft** while carrying the backdoor outputs **act**; and an *output module* (🍃) that averages all features onto the class token for finetuning.

times in subsequent steps, which leads to large gradients. In a MLP, this is inconsequential since a backdoor's gradients cannot impact other benign weights in the model. But with transformers, we cannot avoid small interferences between backdoor features and benign features. Ensuring the robustness of our backdoor construction, and of the model's utility thus requires more careful tuning (see Appendix C.4).

### 5.2. Overview of our Approach

In this section we give an overview of our techniques for inserting data traps into transformers. For ease of exposition, we leave many technical details to Appendix C. We also release code to reproduce all our experiments.

The architecture of our attack is illustrated in Figure 3. We maintain an invariant that the transformer's internal features (a vector of size $d = 768$) are split into three components

$$[\mathbf{ft}, \mathbf{key}, \mathbf{act}] \ .$$

The first part, **ft**, is used for the transformer's benign features for downstream applications. The second part, **key**, stores information to be captured by the backdoor. The third part, **act**, propagates the output activations of the backdoors all the way to the model's last layer, and amplifies them to ensure that gradient signals will shut down the backdoor. Details about how we divide feature vectors into these three components are in Table 2 in Appendix C.

The backdoor construction combines different *modules*, described below. We further employ a number of numerical tricks to deal with the complexity of the model and training process, which we discuss at the end of this section.

**Input module.** This module maps the input tokens to embeddings of the form $[\mathbf{ft}, \mathbf{key}, \mathbf{0}]$. The benign features sim-

ply contain a truncated part of the original embeddings. For the ViT, the backdoor input to be captured is a downscaled and grayscaled version of the original $32 \times 32$ patch. For BERT, we capture a sub-vector of the token embedding, which is sufficient to recover the underlying token.

As described above, one of the key challenges with transformers is to capture entire inputs (i.e., a sequence of tokens), rather than simply capturing individual tokens from different inputs. We achieve this by enhancing our backdoors with *sequence and position selectors*.

We further split the backdoor input **key** into three parts $\mathbf{key} = [\mathbf{tok}, \mathbf{pos}, \mathbf{seq}]$. The first part, **tok**, is the token information to be captured. The second part, **pos**, encodes the token's position in the input. The third part, **seq**, contains features that identify the sequence of input tokens (i.e., a sentence or an image). We obtain these features by making the model's first attention layer compute an average across all input tokens (Fowl et al. (2022) use a similar idea in their Federated Learning attack). As we describe below, by setting the backdoor weights appropriately, we can ensure that with high probability all captured inputs correspond to tokens at different positions in the same input sequence.

**Backdoor module.** In the first encoder block, we use the MLP to implement multiple backdoors that capture the input features **key**, with the same design as in Section 4. That is, we map the full input vector $[\mathbf{ft}, \mathbf{key}, \mathbf{0}]$ to the output $[\mathbf{0}, \mathbf{0}, \{(\mathbf{w}^{(i)})^\top \cdot \mathbf{key} + b^{(i)}\}_{i=1}^k]$, where $k$ is the number of backdoors. The weights and biases are chosen so that the backdoor output is positive for a small fraction of inputs. After the block's shortcut connection, we get the block's final output of the form $[\mathbf{ft}, \mathbf{key}, \{(\mathbf{w}^{(i)})^\top \cdot \mathbf{key} + b^{(i)}\}_{i=1}^k]$.

To capture all tokens in an input, we design "keyed" back-

5

doors that activate only for tokens at a specific position in one input sequence. In more detail, the backdoor weights are of the form $\mathbf{w}^{(i)} = [\mathbf{0}, \mathbf{pos}^{(i)}, \mathbf{w}_{\text{seq}}]$. The position features $\mathbf{pos}^{(i)}$ are designed to be close to orthogonal, i.e., $(\mathbf{pos}^{(i)})^\top \cdot \mathbf{pos}^{(j)}$ is small if $i \neq j$. The bias $b^{(i)}$ is set to a high quantile of the expected distribution of values $\|\mathbf{pos}^{(i)}\|^2 + \mathbf{w}_{\text{seq}}^\top \cdot \mathbf{seq}$ over the finetuning set. This ensures that the backdoor will only activate on a token in the $i$-th position of a sequence with a key $\mathbf{seq}$ similar to $\mathbf{w}_{\text{seq}}$.

**Amplifier module.** Recall from Section 4 that we need the activations from the backdoor to become large so that they contribute a large enough (positive) gradient signal to shut down the backdoor. As in the case of a simple MLP, we do this by multiplying the backdoor output, $\mathbf{act}$, by a large constant in the first MLP module that follows the backdoor.

**Erasure module.** After the backdoor, we do not need the **key** features anymore. To make our lives easier when dealing with the rest of the model, it is easiest to remove these features. But because of the layer normalization, we have to be careful how we do this. For BERT, the layer normalization is at the end of the encoder block (after the linear layers) and so we can just use this layer to set all outputs corresponding to the **key** features to zero. In a ViT, the layer normalization (LN) comes before the linear layers so things are more complicated. Here, we use a combination of two MLP layers to erase the unused features (see Appendix C.2 for details).

**Signal propagation modules.** We now just have to ensure that the amplified backdoor outputs propagate all the way to the end of the network, without interacting (too much) with the benign features $\mathbf{ft}$. For the ViT (where each module's layer-normalization occurs before the shortcut connection), we can simply set all weights of the backdoor components $\mathbf{act}$ and $\mathbf{key}$ to zero, so that these activations are propagated solely via the shortcut connection. For BERT, the layer-normalization occurs at the end of each module, and so we need some extra care (see Appendix C.3 for details). For both ViT and BERT, we maintain the pretrained transformer's weights for the benign features $\mathbf{ft}$, so that the model extracts meaningful features for downstream tasks.

**Output module.** At the last layer of the transformer, we aggregate the activation outputs $\mathbf{act}$ from all tokens onto the special CLS token that is used for classification during finetuning. We do this by rewiring the attention module in the last layer, to simply average all $\mathbf{act}$ features.

The features $[\mathbf{ft}, \mathbf{0}, \mathbf{act}]$ from the CLS token are passed through a final linear layer (randomly initialized prior to finetuning) that maps to $C$ classes. Following the same argument as in Section 4, an input that activates a backdoor

will be misclassified with probability $1 - 1/C$ (since the large activations from the backdoor will boost an arbitrary class), and this will result in a large positive gradient that flows all the way back to the backdoor module and shuts it.

## 5.3. Numerical Tricks

Our backdoor construction above ignores some technical challenges that arise due to architectural specificities of transformer models, in particular the use of GELU activations and layer normalizations. These require some additional numerical tricks to ensure that backdoor signals do not vanish or blow-up during training.

We note that these issues could be sidestepped by making small *architectural* changes to the pretrained model (specifically, using ReLU activations and applying layer normalizations only to the benign features $\mathbf{ft}$). But this might make the backdoor attack too obvious.

**Stabilizing layer normalization.** Layer normalization is an important architectural component in modern transformers, but it serves no functional purpose for our backdoor construction (in fact, its presence complicates things drastically). A layer normalization applies an almost linear transformation: $\text{LN}(\mathbf{x}) \approx W\mathbf{x} + \mathbf{b}$, except that the magnitude of the weight matrix $W$ scales inversely with that of the input $\mathbf{x}$. These layers can be problematic for two reasons: they apply an input-dependent rescaling, and they introduce strong dependencies between different components of the features vector. The data-dependent rescaling of inputs introduces a number of challenges: during the forward pass, rescaling of the backdoor inputs **key** can mess up subsequent modules (e.g., the erasure module of the ViT, see Appendix C.2.3). In the backward pass, the rescaling can lead to very small gradients for large backdoor outputs.

We deal with both issues simultaneously by "virtually" splitting the normalization function. Specifically, given a feature vector with two components $[\mathbf{x}_L, \mathbf{x}_R]$, we add a large constant $C$ to one component, $[\mathbf{x}_L, \mathbf{x}_R + C]$, and then rescale the layer's output accordingly. In Appendix A.1, we show that for a large enough $C$, the dependencies between the two feature components are removed, and the layer no longer applies an input-dependent scaling.

**Dealing with GELUs.** In contrast to ReLUs, GELU activations (Hendrycks & Gimpel, 2016) can not be fully "shut down", as their output is non-zero for all inputs. But for large enough inputs (in absolute value), a GELU is near-identical to a ReLU. We could thus simply re-scale all GELU inputs by a large constant. This requires some care though, because we will always get some "unintended" gradients flowing through the backdoor (i.e., non-zero gradients when the activation output is negative). To deal with this
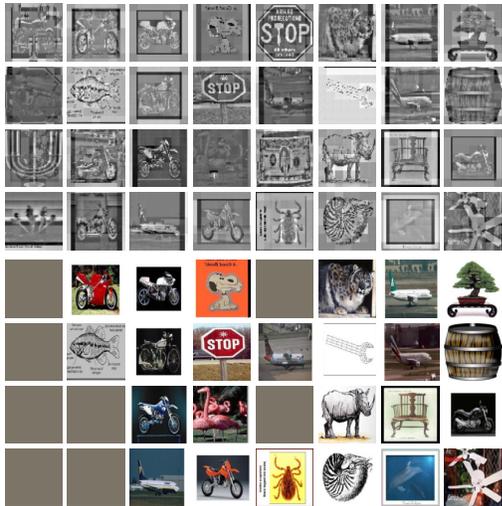
*Figure 4.* Reconstructed complete images from a backdoored ViT-B/32 with GELU activations (finetuned on Caltech 101). **Top**: Reconstruction; **Bottom**: Ground truth, when it is unambiguous.

issue, we use the MLP that follows each GELU to dampen small activation outputs, thereby reducing the impact of the unintended gradients (see Appendix A.2 for details).

### 5.4. Experiments

We carry out our data-stealing attack on popular pretrained transformers and verify that the backdoored models still perform well on the downstream tasks. We use a pretrained ViT-B/32 from Torchvision and BERT-base from Hugging-Face, and finetune on small downstream datasets for illustration: Oxford-IIIT Pet, Caltech 101, and TREC. More details about the finetuning setup are in Appendix D.

Figure 4 shows reconstructed images from the ViT on Caltech 101. The reconstructions are grayscale as we applied this transformation to make the backdoors more space-efficient. Individual patches also have slight mismatches in brightness—an artifact of layer normalizations—but our attack clearly succeeds in recovering recognizable images.

Some selected reconstructed examples from BERT finetuned on TREC-50 are in Table 1. These reconstructions are near-perfect, except for some rare spurious suffix tokens (this occurs when a short sentence fails to activate all backdoors, and these backdoors later capture other tokens). The full lists of reconstructed inputs for ViT and BERT are in Appendix F. We also provide additional results using transformers with ReLU activations, where backdoors are more robust and therefore more successful at capturing inputs.

A notable challenge with transformers is that backdoors that fail to shut down risk destroying the model's benign utility (due to overly large gradient flows in subsequent batches). To mitigate this risk, our experiments focus on settings where the number of backdoors is smaller than the number

*Table 1.* Selected reconstructed sentences from a backdoored BERT model finetuned on the TREC-50 dataset. We highlight tokens which match a ground truth sentence in the dataset. The complete list of reconstructions is in Table 11.

> what must a las vegas blackjack dealer do when he reaches 16?
> which two states enclose chesapeake bay?ruplets??
> how is easter sunday's date determined?
> what do the letters d. c. stand for in washington, d. c.?np.
> what schools in the washington, dc nn nn vbp nn nn nn nn.
> why do heavier objects travel downhill faster? go to college?

of output classes (so that all backdoors shut down with high probability). The results we report here are for finetuning runs where the model's benign utility did not break down (in practice, a victim would presumably notice if the finetuning failed and retry with other hyperparameters).

Our ViT model reaches 83% test accuracy on Caltech 101, and our BERT model reaches 78% test accuracy on TREC-50. While the backdoor does reduce test accuracy (see Appendix D.2), this could be compensated by releasing a larger, more capable backdoored model.

## 6. Black-box Attacks

So far we considered an attacker with *white-box* access to the finetuned model, who can read-off captured inputs from model weights. In this section, we consider a weaker *black-box* attacker with query-only access to the final model.

We first show in Section 6.1 that in this highly practical setting, an attacker can mount *perfect membership inference attacks*, to determine with certainty if a data point was used for finetuning. This result has surprising implications for the tightness of the DP-SGD algorithm of Abadi et al. (2016).

In Section 6.2, we explore black-box data extraction attacks. We show that techniques from the model stealing literature can be used to reconstruct backdoor weights (and thus captured inputs) using a few thousand model queries.

### 6.1. Black-box Membership Inference and Applications to Differential Privacy

Implementing a backdoor that yields perfect membership inference is simple. We set the model's weights so that the activations $\mathbf{h} = [h_1, \ldots, h_n]$ in the 2nd-to-last layer satisfy:

- $\mathbf{h} = [C, 0, \ldots, 0]$ for the target $\mathbf{x}$, for a large $C$.
- $h_1 = 0$ for all other inputs.

The targeted input $\mathbf{x}$ thus only activates the first unit in the layer (with a large value), and that unit is not activated by any other input. During training, the weights connected to the backdoor unit $h_1$ are only updated if the target $\mathbf{x}$ is in the training set. After training, the attacker can query the model on the target $\mathbf{x}$ and observe whether there is a significant change in the logits compared to before training.
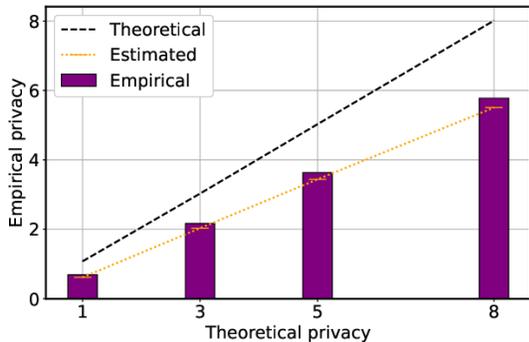
*Figure 5.* Empirical privacy for backdoored models for an end-to-end attacker, compared to DP-SGD's provable upper bound. The estimation in orange is a conservative lower bound of the empirical privacy budget, to account for the target gradients not fully concentrating on the backdoor weights.



(a) Our black-box attack extracts captured inputs pixel-by-pixel, by recovering the critical point where each backdoored unit activates.

(b) A CIFAR-10 image recovered from a backdoored MLP using our black-box attack.

*Figure 6.* Black-box data reconstruction using partial model stealing. By finding the critical points of the backdoored ReLU units (left), we can reconstruct a captured input (right).
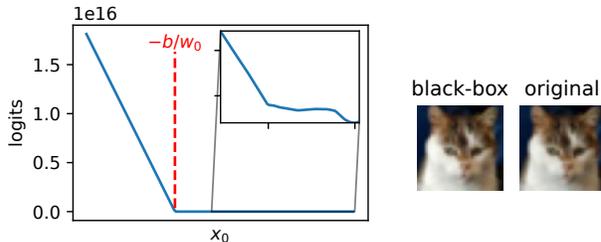
Such backdoors have important implications for the tightness of the DP-SGD algorithm of Abadi et al. (2016). DP-SGD ensures privacy by bounding each training data's influence on the model weights. The algorithm's analysis assumes a worst-case where: (1) one weight has a large gradient if and only if some input is in the training set; and (2) the attacker sees all noisy gradient updates. This analysis is presumed to be pessimistic, which leads to the adoption of loose privacy guarantees.[1] Yet, we show that our backdoor construction leads to a near-tight privacy leakage for an end-to-end attacker who only observes the final model.

In more detail, in DP-SGD each input's gradient is clipped to norm $C$ (we can assume $C = 1$), and Gaussian noise $\mathcal{N}(0, \sigma I)$ is added to the batch gradient. The privacy analysis of Abadi et al. (2016) assumes a worst-case where the gradient at some position $i$ is 1 on input $\mathbf{x}$ and 0 otherwise, and the adversary sees all noisy gradient updates.

Our backdoor above ensures this worst-case scenario occurs in every training step: since the unit $h_1$ is large, the clipped gradient concentrates on the weights of $h_1$. Our black-box attack then compares the logits before and after finetuning, to infer whether the target was present or not. As we show in Appendix E, an end-to-end black-box attacker who can only interact with the final model gives a privacy lower-bound that is close to the provable guarantee.

To illustrate, we consider backdoored CNN models with a frozen backbone (keeping some weights frozen simplifies the backdoor design and analysis). Figure 5 shows that the empirical privacy budget for our attack is close to the theoretical bound from the DP-SGD analysis. Our experiment has two important consequences: (1) loose privacy budgets considered in the literature (e.g., $\varepsilon > 8$ in the production

Federated Learning system of Ramaswamy et al. (2020)) are unsafe if the model provider is untrusted; (2) attempts at a tighter end-to-end privacy bound for DP-SGD (e.g., as in (Chourasia et al., 2021; Feldman et al., 2018)) must make assumptions about the model to rule out backdoors.

## 6.2. Towards Black-box Data Reconstruction

Finally, we consider the challenging problem of black-box data extraction. Our insight is that this problem can be reduced to *model stealing* (Tramèr et al., 2016; Carlini et al., 2020; Rolnick & Kording, 2020; Shamir et al., 2023): if the attacker can recover the backdoored weights using only black-box queries, then they can recover the captured inputs.

Existing (exact) model stealing attacks exploit the piecewise linearity of ReLU networks to extract weights layer-by-layer. But they only work for small models, and require high-precision outputs (typically double precision). Luckily, our problem is simpler for three reasons: (1) we only need to recover the weights of the first layer; (2) we do not require high precision recovery; (3) the backdoor units are connected to a very large weight.

These properties enable a simple and efficient attack. Consider a backdoor unit $h = \mathbf{w}^\top \mathbf{x} + b$ as in Section 4. By querying the model on inputs of the form $\mathbf{x} = [c, 0, \ldots, 0]$, the attacker can find values of $c$ that activate the unit $h$ (we assume the attacker can make arbitrary queries in $\mathbb{R}^m$). Since $h$ is connected to a large weight, the model's output behaves approximately like a ReLU (see Figure 6a). A black-box attacker can find the critical point $c = -b/w_0$ of this function (with just two queries as in (Carlini et al., 2020)). Repeating this procedure for each input coordinate recovers the full weight $\mathbf{w}$ up to a constant. From this, the attacker can read off the captured input. An example is in Figure 6b, for a backdoored MLP trained on CIFAR-10. The full attack requires under 10,000 queries.

---

[1] Nasr et al. (2021) show that when the attacker can observe all intermediate steps, the analysis is empirically tight. But for more realistic end-to-end attackers, their approach yields loose bounds.

# 7. Conclusion

Thousands of developers implicitly trust that foundation models shared online (e.g., on hubs like HuggingFace) have not been tampered with to embed malicious functionality. Our work shows that model backdoors can not only compromise model *integrity*, but also *privacy*. By manipulating a pretrained model's weights, an attacker can recover entire training inputs from finetuned models. Our work expands the scope of supply chain attacks on the machine learning pipeline, and shows we must account for worst-case weights when reasoning about the privacy of a deployed model.

# Impact Statement

Our paper introduces a new attack vector in the machine learning supply chain. There is of course a risk that publishing this attack technique could lead to it being exploited. But as is standard in computer security research, we believe the benefit from raising awareness of these attack vectors outweighs these risks. Our work reinforces the necessity for more robust trust mechanisms for shared machine learning resources.

# References

Abadi, M., Chu, A., Goodfellow, I., McMahan, H. B., Mironov, I., Talwar, K., and Zhang, L. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, pp. 308–318, 2016.

Boenisch, F., Dziedzic, A., Schuster, R., Shamsabadi, A. S., Shumailov, I., and Papernot, N. When the curious abandon honesty: Federated learning is not private. In *2023 IEEE 8th European Symposium on Security and Privacy (EuroS&P)*, pp. 175–199. IEEE, 2023.

Carlini, N., Jagielski, M., and Mironov, I. Cryptanalytic extraction of neural network models. In *Annual International Cryptology Conference*, pp. 189–218. Springer, 2020.

Carlini, N., Tramer, F., Wallace, E., Jagielski, M., Herbert-Voss, A., Lee, K., Roberts, A., Brown, T., Song, D., Erlingsson, U., et al. Extracting training data from large language models. In *30th USENIX Security Symposium (USENIX Security 21)*, pp. 2633–2650, 2021.

Carlini, N., Hayes, J., Nasr, M., Jagielski, M., Sehwag, V., Tramer, F., Balle, B., Ippolito, D., and Wallace, E. Extracting training data from diffusion models. In *32nd USENIX Security Symposium (USENIX Security 23)*, pp. 5253–5270, 2023.

Chourasia, R., Ye, J., and Shokri, R. Differential privacy dynamics of langevin diffusion and noisy gradient descent. *Advances in Neural Information Processing Systems*, 34: 14771–14781, 2021.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. BERT: Pre-training of deep bidirectional transformers for language understanding, 2019.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

Dwork, C., McSherry, F., Nissim, K., and Smith, A. Calibrating noise to sensitivity in private data analysis. In *Theory of Cryptography: Third Theory of Cryptography Conference, TCC 2006, New York, NY, USA, March 4-7, 2006. Proceedings 3*, pp. 265–284. Springer, 2006.

Dwork, C., Roth, A., et al. The algorithmic foundations of differential privacy. *Foundations and Trends® in Theoretical Computer Science*, 9(3–4):211–407, 2014.

Feldman, V., Mironov, I., Talwar, K., and Thakurta, A. Privacy amplification by iteration. In *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*, pp. 521–532. IEEE, 2018.

Fowl, L., Geiping, J., Czaja, W., Goldblum, M., and Goldstein, T. Robbing the fed: Directly obtaining private data in federated learning with modified models. *arXiv preprint arXiv:2110.13057*, 2021.

Fowl, L., Geiping, J., Reich, S., Wen, Y., Czaja, W., Goldblum, M., and Goldstein, T. Decepticons: Corrupted transformers breach privacy in federated learning for language models. *arXiv preprint arXiv:2201.12675*, 2022.

Glorot, X. and Bengio, Y. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pp. 249–256. JMLR Workshop and Conference Proceedings, 2010.

Gu, T., Dolan-Gavitt, B., and Garg, S. BadNets: Identifying vulnerabilities in the machine learning model supply chain. *arXiv preprint arXiv:1708.06733*, 2017.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Hendrycks, D. and Gimpel, K. Gaussian error linear units (GELUs). *arXiv preprint arXiv:1606.08415*, 2016.

Hong, S., Carlini, N., and Kurakin, A. Handcrafted backdoors in deep neural networks. *Advances in Neural Information Processing Systems*, 35:8068–8080, 2022.

Hovy, E., Gerber, L., Hermjakob, U., Lin, C.-Y., and Ravichandran, D. Toward semantics-based answer pinpointing. In *Proceedings of the First International Conference on Human Language Technology Research*, 2001. URL https://www.aclweb.org/anthology/H01-1069.

Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., and Chen, W. LoRA: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.

Jagielski, M., Ullman, J., and Oprea, A. Auditing differentially private machine learning: How private is private SGD? *Advances in Neural Information Processing Systems*, 33:22205–22216, 2020.

Kairouz, P., Oh, S., and Viswanath, P. The composition theorem for differential privacy. In *International conference on machine learning*, pp. 1376–1385. PMLR, 2015.

Krizhevsky, A. Learning multiple layers of features from tiny images. Technical report, 2009.

Li, X. and Roth, D. Learning question classifiers. In *COLING 2002: The 19th International Conference on Computational Linguistics*, 2002. URL https://www.aclweb.org/anthology/C02-1150.

Liu, R., Wang, T., Cao, Y., and Xiong, L. Precurious: How innocent pre-trained language models turn into privacy traps. *arXiv preprint arXiv:2403.09562*, 2024.

Liu, Y., Ma, S., Aafer, Y., Lee, W.-C., Zhai, J., Wang, W., and Zhang, X. Trojaning attack on neural networks. In *25th Annual Network And Distributed System Security Symposium (NDSS 2018)*. Internet Soc, 2018.

Mironov, I., Talwar, K., and Zhang, L. Renyi differential privacy of the sampled gaussian mechanism. *arXiv preprint arXiv:1908.10530*, 2019.

Nasr, M., Songi, S., Thakurta, A., Papernot, N., and Carlin, N. Adversary instantiation: Lower bounds for differentially private machine learning. In *2021 IEEE Symposium on security and privacy (SP)*, pp. 866–882. IEEE, 2021.

Panda, A., Zhang, Z., Yang, Y., and Mittal, P. Teach GPT to phish. In *The Second Workshop on New Frontiers in Adversarial Machine Learning*, 2023.

Parkhi, O. M., Vedaldi, A., Zisserman, A., and Jawahar, C. V. Cats and dogs. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2012.

Phong, L. T., Aono, Y., Hayashi, T., Wang, L., and Moriai, S. Privacy-preserving deep learning via additively homomorphic encryption. *IEEE Transactions on Information Forensics and Security*, 13(5):1333–1345, 2018.

Ramaswamy, S., Thakkar, O., Mathews, R., Andrew, G., McMahan, H. B., and Beaufays, F. Training production language models without memorizing user data. *arXiv preprint arXiv:2009.10031*, 2020.

Rolnick, D. and Kording, K. Reverse-engineering deep relu networks. In *International Conference on Machine Learning*, pp. 8178–8187. PMLR, 2020.

Shamir, A., Canales-Martinez, I., Hambitzer, A., Chavez-Saab, J., Rodriguez-Henriquez, F., and Satpute, N. Polynomial time cryptanalytic extraction of neural network models. *arXiv preprint arXiv:2310.08708*, 2023.

Shokri, R., Stronati, M., Song, C., and Shmatikov, V. Membership inference attacks against machine learning models. In *2017 IEEE symposium on security and privacy (SP)*, pp. 3–18. IEEE, 2017.

Song, C., Ristenpart, T., and Shmatikov, V. Machine learning models that remember too much. In *Proceedings of the 2017 ACM SIGSAC Conference on computer and communications security*, pp. 587–601, 2017.

Steinke, T., Nasr, M., and Jagielski, M. Privacy auditing with one (1) training run. *arXiv preprint arXiv:2305.08846*, 2023.

Tramer, F. and Boneh, D. Differentially private learning needs better features (or much more data). *arXiv preprint arXiv:2011.11660*, 2020.

Tramèr, F., Zhang, F., Juels, A., Reiter, M. K., and Ristenpart, T. Stealing machine learning models via prediction {APIs}. In *25th USENIX security symposium (USENIX Security 16)*, pp. 601–618, 2016.

Tramèr, F., Shokri, R., San Joaquin, A., Le, H., Jagielski, M., Hong, S., and Carlini, N. Truth serum: Poisoning machine learning models to reveal their secrets. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pp. 2779–2792, 2022.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Wen, Y., Geiping, J., Fowl, L., Goldblum, M., and Goldstein, T. Fishing for user data in large-batch federated learning via gradient magnification. *arXiv preprint arXiv:2202.00580*, 2022.

Wen, Y., Marchyok, L., Hong, S., Geiping, J., Goldstein, T., and Carlini, N. Privacy backdoors: Enhancing membership inference through poisoning pre-trained models, 2024.

Xu, J., Sun, X., Zhang, Z., Zhao, G., and Lin, J. Understanding and improving layer normalization. *Advances in Neural Information Processing Systems*, 32, 2019.

Yousefpour, A., Shilov, I., Sablayrolles, A., Testuggine, D., Prasad, K., Malek, M., Nguyen, J., Ghosh, S., Bharadwaj, A., Zhao, J., Cormode, G., and Mironov, I. Opacus: User-friendly differential privacy library in PyTorch. *arXiv preprint arXiv:2109.12298*, 2021.

Zhao, J. C., Sharma, A., Elkordy, A. R., Ezzeldin, Y. H., Avestimehr, S., and Bagchi, S. Loki: Large-scale data reconstruction attack against federated learning through model manipulation. In *2024 IEEE Symposium on Security and Privacy (SP)*, pp. 30–30. IEEE Computer Society, 2023.

# A. Common Modules and Tricks For Transformer Backdoors

In this appendix, we provide more details on some common modules and numerical tricks we use to implement privacy backdoors in transformer models. In Appendix C, we then describe how we combine these modules for ViT and BERT models.

## A.1. Stabilizing Layer Normalization

Layer normalization is a critical component of the transformer architecture, which enables smoother gradients, faster training, and better accuracy (Xu et al., 2019). However, it brings no direct benefit to our backdoor construction.

In fact these layers are quite problematic because they cause a strong coupling between different features, and they rescale inputs according to their magnitude. The coupling of features is challenging because we want different portions of the feature vector to be used for different purposes (e.g., for benign features, or for backdoor inputs and outputs). The rescaling effect of layer normalization can attenuate gradients, which risks recovering illegible mixtures.

To mitigate these issues, we show how to "decouple" two parts of a feature vector, so that the layer normalization acts (nearly) independently on both parts. As a by-product, our approach also gets rid of the layer's input-dependent rescaling effect!

A layer normalization module with learned parameters $\gamma, \beta$ implements the following operation:

$$\text{LN}\left[\mathbf{x}\right] = \mathbf{z} * \boldsymbol{\gamma} + \boldsymbol{\beta}, \quad \text{where} \quad \mathbf{z} = \frac{\mathbf{x} - \mu}{\sigma} .$$

Here $\mu$ and $\sigma$ are the sample average and standard deviation of $\mathbf{x}$, $*$ is a coordinate-wise product, and for simplicity we omit the numerical stability correction that ensures that the standard deviation is non-zero.

Let $m$ be the dimension of $\mathbf{x}$. The gradient of the normalization operations satisfies:

$$\frac{\partial z_j}{\partial x_j} = \frac{\left(1 - \frac{1}{m}\right) - \frac{(x_j - \mu)^2}{m\sigma^2}}{\sigma} \qquad \forall j \qquad \frac{\partial z_i}{\partial x_j} = \frac{-\frac{1}{m} - \frac{(x_j - \mu)(x_i - \mu)}{m\sigma^2}}{\sigma} \qquad \forall i \neq j . \tag{5}$$

If the input is amplified as $\mathbf{x} \to \mathbf{x}' = c \cdot \mathbf{x}$ for $c \gg 1$, the gradient becomes smaller. Unlike in a MLP layer, the gradient flowing back to the input is thus input-dependent. Moreover, the gradient for coordinate $x_j$ depends on the value of all other input coordinates.

To stabilize a layer normalization, we consider a special case where the input is partitioned as $\mathbf{x} \to (\mathbf{x}_L, \mathbf{x}_R)$. We then modify the input as $(\mathbf{x}_L, \mathbf{x}_R + \mathbf{C})$, where $\mathbf{C}$ is a vector of *stabilizer constants* and $C \gg \max_{\mathbf{x}}\|\mathbf{x}\|_\infty$. We denote the respective dimensions of $\mathbf{x}_L$ and $\mathbf{x}_R$ as $m_R$ and $m_L$ (with $m_R + m_L = m$), and their respective means and standard deviations as $\mu_R, \mu_L, \sigma_L, \sigma_R$.

Asymptotically in $C$, we can then write the forward pass as:

$$(z_L)_j = \frac{(x_L)_j - \mu_L - \frac{m_R}{m}C}{\sqrt{\frac{m_L m_R}{m^2}}C} \qquad (z_R)_j = \frac{(x_R)_j - \mu_R + \frac{m_L}{m}C}{\sqrt{\frac{m_L m_R}{m^2}}C} .$$

Similarly, the asymptotic behavior of the gradients is:

$$\frac{\partial (z_L)_i}{\partial (x_L)_i} = \frac{1 - \frac{1}{m_L}}{C\sqrt{\frac{m_L m_R}{m^2}}} , \qquad \frac{\partial (z_L)_i}{\partial (x_L)_j} = \frac{-\frac{1}{m_L}}{C\sqrt{\frac{m_L m_R}{m^2}}}$$

$$\frac{\partial (z_L)_i}{\partial (x_R)_j} = -\frac{1}{m} \frac{\left[\frac{m}{m_R}((x_L)_i - \mu) - \frac{m}{m_L}((x_R)_j - \mu) - 2(\mu_R - \mu_L)\right]}{C^2\sqrt{\frac{m_L m_R}{m^2}}}$$

$$\frac{\partial (z_R)_i}{\partial (x_R)_i} = \frac{1 - \frac{1}{m_R}}{C\sqrt{\frac{m_L m_R}{m^2}}} , \qquad \frac{\partial (z_R)_i}{\partial (x_R)_j} = \frac{-\frac{1}{m_R}}{C\sqrt{\frac{m_L m_R}{m^2}}}$$

$$\frac{\partial (z_R)_i}{\partial (x_L)_j} = -\frac{1}{m} \frac{\left[\frac{m}{m_R}((x_L)_j - \mu) - \frac{m}{m_L}((x_R)_i - \mu) - 2(\mu_R - \mu_L)\right]}{C^2\sqrt{\frac{m_L m_R}{m^2}}} .$$

We have thus achieved two things: (1) the gradients of the left and right outputs with respect to the corresponding input no longer depend on the input; (2) the "cross-term" gradients between the left output and right input (and vice-versa) decay quadratically faster with $C$.

Then we just rescale the outputs properly. To this end, we define a special module

$$\text{StabLN}(\mathbf{x}; \mathcal{J}; \boldsymbol{\gamma}_L, \boldsymbol{\beta}_L, \boldsymbol{\gamma}_R, \boldsymbol{\beta}_R; C) \,,$$

implemented by the layer normalization module. Here $\mathcal{J} \subseteq \{1, \ldots, d\}$ is a subset of the feature coordinates in $\mathbf{x}$, and $\bar{\mathcal{J}}$ is its set complement. The forward pass of this module satisfies:

$$\mathbf{x}_L \leftarrow \mathbf{x}_{\mathcal{J}} \qquad \mathbf{x}_R \leftarrow \mathbf{x}_{\bar{\mathcal{J}}}$$

$$(\mathbf{z}_L, \mathbf{z}_R) = \text{norm}(\mathbf{x}_L, \mathbf{x}_R + \mathbf{C})$$

$$\mathbf{x}'_L = \boldsymbol{\gamma}_L * \sqrt{\frac{m_L m_R}{m^2}} C \mathbf{z}_L + \boldsymbol{\gamma}_L * \frac{m_R}{m} \mathbf{C} + \boldsymbol{\beta}_L = \boldsymbol{\gamma}_L * (\mathbf{x}_L - \mu_L) + \boldsymbol{\beta}_L$$

$$\mathbf{x}'_R = \boldsymbol{\gamma}_R * \sqrt{\frac{m_L m_R}{m^2}} C \mathbf{z}_R - \boldsymbol{\gamma}_R * \frac{m_L}{m} \mathbf{C} + \boldsymbol{\beta}_R = \boldsymbol{\gamma}_R * (\mathbf{x}_R - \mu_R) + \boldsymbol{\beta}_R \,.$$

This module almost degenerates into a simple one-to-one affine layer. For the entire module, we get the following gradients:

$$\frac{\partial (x'_L)_i}{\partial (x_L)_i} = (\gamma_L)_i \left(1 - \frac{1}{m_L}\right) \qquad \frac{\partial (x'_L)_i}{\partial (x_L)_j} = -(\gamma_L)_i \frac{1}{m_L} \qquad \frac{\partial (x'_L)_i}{\partial (x_R)_j} = \mathcal{O}(1/C)$$

$$\frac{\partial (x'_R)_i}{\partial (x_R)_i} = (\gamma_R)_i \left(1 - \frac{1}{m_R}\right) \qquad \frac{\partial (x'_R)_i}{\partial (x_R)_j} = -(\gamma_R)_i \frac{1}{m_R} \qquad \frac{\partial (x'_R)_i}{\partial (x_L)_j} = \mathcal{O}(1/C) \,.$$

We have thus successfully achieved two effects. First, we have decoupled the two parts of an input vector corresponding to the indices $\{\mathcal{J}, \bar{\mathcal{J}}\}$. Second, we got rid of the input-dependent scaling.

## A.2. Dealing with GELUs

Modern transformers typically use GELU activations rather than ReLUs. The GELU function (Gaussian Error Linear Unit) is defined as $\text{GELU}(x) = x\Phi(x)$, where $\Phi(x)$ is the cumulative distribution function for a standard Gaussian distribution. A straightforward calculation of its derivative shows that

$$\frac{d}{dx}\text{GELU}(x) = \Phi(x) + x\phi(x) \qquad \min_x \left\{ \frac{d}{dx}\text{GELU}(x) \right\} \approx -0.13 \,,$$

where $\phi(x)$ is the probability density function of a standard Gaussian distribution. Unlike a ReLU, a GELU can thus have negative outputs, and negative derivatives. For inputs of large enough magnitude (e.g., $|x| > 10$), the GELU is essentially equivalent to the ReLU for our purposes.

GELUs are problematic for two reasons: (1) a backdoor can never be fully shut down, and so a backdoor will continuously capture small parts of inputs even after its first activation. This reduces the quality of reconstructed inputs; (2) if the backdoor activates, we get a large gradient flow into the backdoor. If the GELU has a negative gradient, then we get a large negative gradient into the backdoor, which opens the backdoor further and risks breaking down the model's behavior in subsequent batches.

A naive solution is to increase the magnitude of backdoor inputs $\mathbf{x} \to \mathbf{x}' = c \cdot \mathbf{x}$ for some $c \gg 1$. Then, we expect that all backdoor outputs will fall into the region where the GELU behaves like a ReLU. This scaling also requires to increase the magnitude of the shutdown term (defined in Appendix A.3) when the backdoor is first activated, thereby helping to keep the backdoor shut. Unfortunately, this approach is still prone to some problems. In practice, some negative gradient signal into a backdoor unit is inevitable. By simply increasing the magnitude of inputs, we also amplify the impact of these unintended negative gradients, which risks breaking the backdoor fully open.

To attenuate these issues we make use of two strategies. First, we set backdoor thresholds somewhat loosely, so that some backdoor units may effectively be activated multiple times. This makes those backdoors harder to reconstruct, but lowers the probability of a full model breakdown. Second, we use the MLP directly following each backdoor to ensure that we only

get strong gradient signals into the backdoor if the activation output is large (and confirmative) enough (i.e., in the regime where the GELU acts like a ReLU). Otherwise, the gradient signal will not be amplified by the MLP and thus only have a negligible impact on the backdoor weights (and the subsequent selective activation process).

### A.3. Boosting Shutdown Terms

As we saw in Section 4, when a backdoor has captured an input $\hat{\mathbf{x}}$, the output of the updated backdoor on a new input $\mathbf{x}$ is equal to

$$\mathbf{w}'^\top \mathbf{x} + b' = (\mathbf{w}^\top \mathbf{x} + b) - c \cdot (\hat{\mathbf{x}}^\top \mathbf{x} + 1),$$

where $c$ is a positive term.

The term $\hat{\mathbf{x}}^\top \mathbf{x}$ determines how easy the backdoor is to shut down, if we keep $c$ fixed. We call this term the "shutdown term". Ideally, we would want this term to be as positive as possible. However, simultaneously, we expect to decouple the selective activation process with the positivity of the shutdown term so that we can have the strongest distinguishability for different backdoor weights.

We can achieve this by logically splitting the backdoor. Assume that the weight of the backdoor is of the form $\mathbf{w} = (\mathbf{0}, \mathbf{w}_R)$, and an input is similarly split as $\mathbf{x} = (\mathbf{x}_L, \mathbf{x}_R)$. Then, the activation value only depends on the second half of the input:

$$h = \texttt{ReLU}\left(\mathbf{w}^\top \mathbf{x} + b\right)$$
$$= \texttt{ReLU}\left(\mathbf{w}_R^\top \mathbf{x}_R + b\right) \ .$$

The shutdown term, however, depends on the entire input:

$$\hat{\mathbf{x}}^\top \mathbf{x} = \left[\hat{\mathbf{x}}_L^\top, \hat{\mathbf{x}}_R^\top\right] \begin{bmatrix} \mathbf{x}_L \\ \mathbf{x}_R \end{bmatrix} = \hat{\mathbf{x}}_L^\top \mathbf{x}_L + \hat{\mathbf{x}}_R^\top \mathbf{x}_R \ .$$

Thus, by boosting the left-hand part $\hat{\mathbf{x}}_L^\top \mathbf{x}_L$ of the input of the backdoor, we can create larger shutdown terms without changing the backdoor's selectivity (i.e., which inputs can activate a given backdoor).

**Partial input boosting using a layer normalization.** We can achieve this boosting of part of the backdoor input using a (stabilized) layer normalization operation, which directly precedes the MLP modules that implements the backdoor. We combine these operations into a reusable structure we call SpecLinear:

$$\text{SpecLinear}\left([\mathbf{x}_L, \mathbf{x}_R], [\boldsymbol{\gamma}_L, \boldsymbol{\gamma}_R], [\boldsymbol{\beta}_L, \boldsymbol{\beta}_R], [\mathbf{w}_L, \mathbf{w}_R], b\right):$$
$$\textbf{return } [\mathbf{w}_L, \mathbf{w}_R]^\top \cdot \left([\boldsymbol{\gamma}_L, \boldsymbol{\gamma}_R] * \texttt{norm}([\mathbf{x}_L, \mathbf{x}_R]) + [\boldsymbol{\beta}_L, \boldsymbol{\beta}_R]\right) + b$$

Here, $\boldsymbol{\gamma}$ and $\boldsymbol{\beta}$ are the affine transforms implemented by the layer normalization, $\texttt{norm}$ normalizes the input to zero mean and unit variance, and $*$ is the coordinate-wise product of two vectors.

A special instantiation of interest is when $\boldsymbol{\gamma}_L = \mathbf{w}_L = \mathbf{0}_L, \boldsymbol{\beta}_R = \mathbf{0}_R$. Assuming for simplicity that $\mathbf{x}$ is already normalized[2], the module output then simplifies to:

$$\mathbf{w}_R^\top(\boldsymbol{\gamma}_R * \mathbf{x}_R) + b \ ,$$

and the shutdown term for the backdoor is

$$\|\boldsymbol{\beta}_L\|^2 + (\boldsymbol{\gamma}_R * \hat{\mathbf{x}}_R)^\top (\boldsymbol{\gamma}_R * \mathbf{x}_R) \ .$$

Then, we can arbitrarily increase the magnitude of $\boldsymbol{\beta}_L$ to increase the shutdown term, without any effect on the backdoor output.

### A.4. Aggregating Information across Tokens

As we explain in Section 5.2, a challenge we face with transformers is to capture all tokens in an input sequence, rather than individual tokens belonging to different inputs.

---

[2]In practice, the layer normalization here is always stabilized. The final effect is just like the input has already been normalized.

We solve this problem by computing a *sequence identifier* that is common to all tokens in a sequence, and which we can later use as a feature to selectively activate backdoors. We implement a sequence identifier using an attention module that averages features across all tokens in a sequence.

Let $X = \begin{bmatrix} \mathbf{x}^{(1)} & \mathbf{x}^{(2)} & \cdots & \mathbf{x}^{(k)} \end{bmatrix}^\top$ be an input sequence consisting of $k$ token embeddings. The output of self-attention module $Z = \mathrm{SelfAttention}\,[X]$ is then:

$$
\begin{aligned}
Q &= \begin{bmatrix} \mathrm{W}^{(Q)}\mathbf{x}^{(1)} + \mathbf{b}^{(Q)} & \mathrm{W}^{(Q)}\mathbf{x}^{(2)} + \mathbf{b}^{(Q)} & \cdots & \mathrm{W}^{(Q)}\mathbf{x}^{(k)} + \mathbf{b}^{(Q)} \end{bmatrix}^\top \\
K &= \begin{bmatrix} \mathrm{W}^{(K)}\mathbf{x}^{(1)} + \mathbf{b}^{(K)} & \mathrm{W}^{(K)}\mathbf{x}^{(2)} + \mathbf{b}^{(K)} & \cdots & \mathrm{W}^{(K)}\mathbf{x}^{(k)} + \mathbf{b}^{(K)} \end{bmatrix}^\top \\
V &= \begin{bmatrix} \mathrm{W}^{(V)}\mathbf{x}^{(1)} + \mathbf{b}^{(V)} & \mathrm{W}^{(V)}\mathbf{x}^{(2)} + \mathbf{b}^{(V)} & \cdots & \mathrm{W}^{(V)}\mathbf{x}^{(k)} + \mathbf{b}^{(V)} \end{bmatrix}^\top \\
Z &= \begin{bmatrix} \mathbf{z}^{(1)} & \mathbf{z}^{(2)} & \cdots & \mathbf{z}^{(k)} \end{bmatrix}^\top = \mathrm{Softmax}\left(QK^\top\right) V \,.
\end{aligned}
\tag{6}
$$

Let $\mathcal{J} \subseteq \{1, \ldots, 768\}$ be a set of coordinates in a token embedding, and $\rho > 0$ some constant. We then define a module $\mathrm{Syn}(X; \mathcal{J}; \rho)$, which sets the attention weights as follows:

$$
\begin{aligned}
\mathrm{W}^{(Q)} &= \mathbf{0} & \mathbf{b}^{(Q)} &= \mathbf{0} \\
\mathrm{W}^{(K)} &= \mathbf{0} & \mathbf{b}^{(K)} &= \mathbf{0} \\
\mathrm{W}^{(V)}_{ij} &= \begin{cases} \rho & \text{if } i = j \in \mathcal{J} \\ 0 & \text{else} \end{cases} & \mathbf{b}^{(V)} &= \mathbf{0} \,.
\end{aligned}
\tag{7}
$$

The query $Q$, key $K$ and value $V$ matrices are then:

$$
\begin{aligned}
Q &= K = \mathbf{0} \\
V_{ij} &= \begin{cases} \rho x_j^{(i)} & \text{if } j \in \mathcal{J} \\ 0 & \text{else} \end{cases} \,.
\end{aligned}
$$

The softmax value is then just a uniform matrix with value $1/k$ in each entry. Finally, the output $Z$ satisfies:

$$
z_j^{(i)} = \begin{cases} \rho \frac{1}{k} \sum_{1 \le l \le k} x_j^{(l)} & \text{if } j \in \mathcal{J} \\ 0 & \text{else} \end{cases} \quad \forall 1 \le i \le k.
$$

In words, we sum up all the token vectors (scaled by some factor $\rho/k$), and zero-out coordinates that do not belong to the set $\mathcal{J}$. This averaged input is our sequence key which we will use to selectively activate backdoors.

The same design works for multi-headed attention. For a masked attention module, we can simply take the average only over the unmasked tokens.

## B. Creating Keyed Backdoors

In this section, we describe our main backdoor design for transformers. To capture all individual tokens from a sequence, we build a family of backdoors so that all backdoors fire only for specific *sequence identifiers*, and so that each backdoor fires only for tokens in a specific *position*.

**Creating position identifiers.** We use a portion of each token's feature vector to represent the token's position in a sequence (as is done with position embeddings in a benign transformer).

The only difference is that we craft these position identifiers to be near-orthogonal to each other, so that we can use them to selectively activate different backdoors. Suppose that tokens can be in one of $n$ positions, and that we reserve $m$ coordinates in the feature vector for a position identifier. We then design these position identifiers $\mathbf{u}^{(1)}, \ldots, \mathbf{u}^{(n)} \in \mathbb{R}^m$ to satisfy the following properties:

$$
\begin{aligned}
\|\mathbf{u}^{(j)}\|^2 &= u_0 \quad \forall j \\
\left(\mathbf{u}^{(j)}\right)^\top \mathbf{u}^{(k)} &< u_+ \ll u_0 \quad \forall j \ne k \\
\sum_k u_k^{(j)} &= 0 \quad \forall j \,,
\end{aligned}
\tag{8}
$$

where $u_+$ is a positive threshold determined by the attacker. The first two conditions ensure that the position keys are not strongly aligned with each other, so that they can selectively activate individual backdoors. The last condition that keys have zero mean is merely for convenience, to facilitate interactions with a subsequent layer normalization.

**Creating sequence identifiers.**  To create our sequence identifiers, we use the module $\mathrm{Syn}(X, \mathcal{J}, \rho)$ we defined in Appendix A.4, where $\mathcal{J}$ represents the indices of the token features to sum up to create a sequence key. This sequence key is then stored in $|\mathcal{J}_{\mathrm{seq}}|$ coordinates of each token's feature representation. We migrate features from the source features to the target features $\mathbf{seq}$ using the projection linear layer of an attention module.

**The backdoor design.**  We can finally present the full keyed backdoor design. This backdoor is implemented in the transformer's first MLP module, which maps token features of dimension 768 to hidden units of dimension 3072. For ViT models, an input sequence consists of 49 tokens, each of which represents a patch of $32 \times 32$ pixels. For BERT, a sequence consists of 48 token embeddings. We thus want to build a family of backdoors that can capture all these individual tokens.

Denote an input sequence as $X = \begin{bmatrix} \mathbf{x}^{(1)} & \mathbf{x}^{(2)} & \cdots & \mathbf{x}^{(k)} \end{bmatrix}$. Each token vector $\mathbf{x}^{(i)}$ is split into multiple components:

$$\mathbf{x}^{(i)} = [\mathbf{x}_{\mathrm{ft}}^{(i)}, \mathbf{x}_{\mathrm{act}}^{(i)}, \underbrace{\mathbf{x}_{\mathrm{pos}}^{(i)}, \mathbf{x}_{\mathrm{seq}}^{(i)}, \mathbf{x}_{\mathrm{tok}}^{(i)}}_{\mathbf{x}_{\mathrm{key}}^{(i)}}] \,, \tag{9}$$

where $\mathbf{x}_{\mathrm{ft}}^{(i)}$ contains benign features, $\mathbf{x}_{\mathrm{act}}^{(i)}$ is used to store backdoor activations, and $\mathbf{x}_{\mathrm{key}}^{(i)}$ contains the features to activate the backdoor, which are further split into three components: $\mathbf{x}_{\mathrm{pos}}^{(i)} = \mathbf{u}^{(i)}$ is the token's position identifier, $\mathbf{x}_{\mathrm{seq}}^{(i)} = \mathbf{seq}$ is the sequence identifier which is shared by all tokens in the sequence, and $\mathbf{x}_{\mathrm{tok}}^{(i)}$ contains token features to be captured.

We then build families of backdoors where the $j$-th backdoor has parameters

$$\mathbf{w} = [\mathbf{0}, \mathbf{0}, \theta_{\mathrm{pos}}\mathbf{u}^{(j)}, \mathbf{w}_{\mathrm{seq}}, \mathbf{0}], \qquad b = b_{\mathrm{pos}}^{(j)} + b_{\mathrm{seq}} \tag{10}$$

Here $\theta_{\mathrm{pos}}\mathbf{u}^{(j)}$ is the weight that aligns with tokens in the $j$-th position, $\mathbf{w}_{\mathrm{seq}}$ is the backdoor weight that aligns with the sequence identifier, and $b_{\mathrm{pos}}^{(j)}$ and $b_{\mathrm{seq}}$ are corresponding thresholds to be set. On an input token $\mathbf{x}^{(i)}$, the output of the backdoor becomes:

$$h = \left( \theta_{\mathrm{pos}}(\mathbf{u}^{(j)})^{\top}\mathbf{x}_{\mathrm{pos}}^{(i)} + b_{\mathrm{pos}}^{(j)} \right) + \left( \mathbf{w}_{\mathrm{seq}}^{\top}\mathbf{x}_{\mathrm{seq}}^{(i)} + b_{\mathrm{seq}} \right)$$
$$= \left( \theta_{\mathrm{pos}}(\mathbf{u}^{(j)})^{\top}\mathbf{u}^{(i)} + b_{\mathrm{pos}}^{(j)} \right) + \left( \mathbf{w}_{\mathrm{seq}}^{\top}\mathbf{seq} + b_{\mathrm{seq}} \right) \,.$$

By adjusting the thresholds $b_{\mathrm{pos}}^{(j)}$ and $b_{\mathrm{seq}}$, we can ensure that the backdoor only activates when both $(\mathbf{u}^{(j)})^{\top}\mathbf{u}^{(i)}$ and $\mathbf{w}_{\mathrm{seq}}^{\top}\mathbf{seq}$ are large. The first term is large for tokens in the $j$-the position, and the second term follows some distribution over all training inputs. We set these thresholds so that a token in an incorrect position $i \neq j$ never activates the backdoor, i.e.,

$$\max_{\mathbf{x}}(\mathbf{w}_{\mathrm{seq}}^{\top}\mathbf{x}_{\mathrm{seq}} + b_{\mathrm{seq}}) \ll b_{\mathrm{pos}}^{(j)} - \max_{k \neq j}(\theta_{\mathrm{pos}}(\mathbf{u}^{(j)})^{\top}\mathbf{u}^{(k)}) \,.$$

Finally, we apply the trick described in Appendix A.3 to boost the backdoor's "shutdown" term, to ensure that an activated backdoor gets a large gradient modifcation that shuts it down.

Specifically, we use a combination of a layer normalization and MLP to implement the module:

$$\mathrm{SpecLinear}\left( \left( \underbrace{\begin{bmatrix} \mathbf{x}_{\mathrm{ft}} \\ \mathbf{x}_{\mathrm{act}} \end{bmatrix}}_{\mathbf{x}_L}, \underbrace{\begin{bmatrix} \mathbf{x}_{\mathrm{pos}} \\ \mathbf{x}_{\mathrm{seq}} \\ \mathbf{x}_{\mathrm{tok}} \end{bmatrix}}_{\mathbf{x}_R} \right), (\mathbf{0}, \boldsymbol{\gamma}_R), (\boldsymbol{\beta}_L, \mathbf{0}), \left( \underbrace{\begin{bmatrix} \mathbf{0} \\ \mathbf{0} \end{bmatrix}}_{\mathbf{w}_L}, \underbrace{\begin{bmatrix} \theta_{\mathrm{pos}}\mathbf{u}^{(j)} \\ \mathbf{w}_{\mathrm{seq}} \\ \mathbf{0} \end{bmatrix}}_{\mathbf{w}_R} \right), b_{\mathrm{pos}}^{(j)} + b_{\mathrm{seq}} \right) ,$$

**Multiple families of backdoors.**  To capture multiple full inputs, we replicate the above design of a backdoor family, each time using a different sequence weight $\mathbf{w}_{\mathrm{seq}}$ and corresponding bias $b_{\mathrm{seq}}$. Since there are 3072 hidden units in the MLP, we

could implement many backdoor families. But at the end of the module, we need to fit all the backdoor outputs into a feature vector of dimension 768. We do this by aggregating the activation signals of an entire backdoor family into a single feature. That is, assuming that the $i$-th backdoor family uses the units in positions $I_i \subset \{1, \ldots, 3072\}$, we use the final linear layer in the MLP module to project all these units onto a single coordinate $j$ of the feature vector:

$$W_{j,k}^{(O)} = \begin{cases} \theta & \text{if } k \in \mathcal{I}_i \\ 0 & \text{else} \end{cases}, \tag{11}$$

where $\theta$ is an amplifier coefficient. Using these weights, we ensure that the gradient signal is shared between all backdoors in a family, so that they all shut down simultaneously when activated.

## C. Detailed Description of Transformer Backdoors for ViT and BERT

In this appendix, we provide details about how to craft backdoors for ViT and BERT models, by making use of some of the common tricks and modules described in Appendix A.

### C.1. Structure of a Transformer

In the main body of the paper, we give some high-level overview of the architecture of a transformer, and how we implement the backdoor into it. Here we provide additional details necessary for describing the backdoor modules.

We begin by looking at the structure of the transformer's encoder blocks, which combine three operations: a layer normalization, a MLP, and an attention module. The MLP module $\mathbf{x}'' = \mathrm{MLP}\left[\mathbf{x}\right], \mathbf{x} \in \mathbb{R}^{768}$ is composed of two linear layers:

$$\begin{aligned} \mathbf{x}' &= \mathrm{GELU}(\mathrm{W}\mathbf{x} + \mathbf{b}) \quad \mathbf{x}' \in \mathbb{R}^{3072} \\ \mathbf{x}'' &= \mathrm{W}'\mathbf{x}' + \mathbf{b}' \quad \mathbf{x}'' \in \mathbb{R}^{768} \ . \end{aligned} \tag{12}$$

Note that there are 3072 intermediate units inside a MLP module. A complete attention module $Z' = \mathrm{CompleteAttention}\left[X\right]$ contains one more projection linear module than the simplified description in Equation (6), i.e.,

$$\begin{aligned} Z &= \begin{bmatrix} \mathbf{z}^{(1)} & \mathbf{z}^{(2)} & \cdots & \mathbf{z}^{(k)} \end{bmatrix}^T = \mathrm{SelfAttention}\left[X\right] \\ Z' &= \begin{bmatrix} \mathrm{W}^{(O)}\mathbf{z}^{(1)} + \mathbf{b}^{(O)} & \mathrm{W}^{(O)}\mathbf{z}^{(2)} + \mathbf{b}^{(O)} & \cdots & \mathrm{W}^{(O)}\mathbf{z}^{(k)} + \mathbf{b}^{(O)} \end{bmatrix}^T \ . \end{aligned} \tag{13}$$

The three modules are arranged in a different order for ViT and BERT models. For convenience, we set all dropout probabilities to zero—a reasonable setup for finetuning tasks.

**Partitioning features and blocks.** We only have limited resources to build a backdoor structure in a transformer. There are 12 encoder blocks and 768 features. Here we describe how we partition these resources for our backdoor construction. This partition trades between the utility of the model (i.e., the more features we leave unmodified, the better accuracy we can achieve on downstream tasks), and the quality of reconstructed images and sentences. Recall that we split a token's feature vector as described in Equation (9). When referring to the indices of these different parts of the feature vector we use the notation $\mathcal{J}_{\mathrm{ft}}$, $\mathcal{J}_{\mathrm{key}}$, etc. We split the encoder blocks into three sets, "prefix", "propagation" and "suffix". The prefix is used to prepare inputs for the backdoor, and run the backdoor. The propagation blocks propagate features to the end of the transformer. The suffix block prepares the output features. For the ViT model, we consider two designs: one that captures entire images, and one that simply captures individual patches (i.e., without using any sequence or position keys). We summarize our partition in Table 2:

*Table 2.* Partition of encoder blocks and features. **prefix**: backdoor module, amplifier module, erasure module; **propagation**: signal propagation module; **suffix**: output module. **ft**: benign features for down-stream tasks; **key**: features for selectively activating a backdoor; **act**: backdoor activation outputs. **key** is further partitioned into **pos**, **seq**, **tok**, where **pos** are the features used for position keys, **seq** are features used for sequence keys, and **tok** are the features captured by the backdoor.

| | | | | | key | | | | |
|---|---|---|---|---|---|---|---|---|---|
| **Model** | Prefix | Propagation | Suffix | ft | (total) | pos | seq | tok | act |
| ViT (image) | 3 | 8 | 1 | 448 | 256 | 128 | 64 | 64 | 64 |
| ViT (patch) | 3 | 8 | 1 | 448 | 256 | 0 | 0 | 256 | 64 |
| BERT | 2 | 9 | 1 | 512 | 192 | 64 | 128 | 0 | 64 |

## C.2. ViT

The encoder block of a ViT model, $Z_{\text{out}} = \text{ViTEncoderBlock}\left[X_{\text{in}}\right]$ is written as:

$$\mathbf{x}^{(i)} = \text{LN}\left[\mathbf{x}_{\text{in}}^{(i)}\right], \text{ for } 1 \leq i \leq k$$

$$\left[\tilde{\mathbf{x}}^{(1)} \quad \tilde{\mathbf{x}}^{(2)} \quad \cdots \quad \tilde{\mathbf{x}}^{(k)}\right]^T = \text{CompleteAttention}\left[\left[\mathbf{x}^{(1)} \quad \mathbf{x}^{(2)} \quad \cdots \quad \mathbf{x}^{(k)}\right]^T\right]$$

$$\mathbf{z}_{\text{in}}^{(i)} = \tilde{\mathbf{x}}^{(i)} + \mathbf{x}_{\text{in}}^{(i)}$$

$$\mathbf{z}^{(i)} = \text{LN}\left[\mathbf{z}_{\text{in}}^{(i)}\right]$$

$$\tilde{\mathbf{z}}^{(i)} = \text{MLP}\left[\mathbf{z}^{(i)}\right]$$

$$\mathbf{z}_{\text{out}}^{(i)} = \tilde{\mathbf{z}}^{(i)} + \mathbf{z}_{\text{in}}^{(i)} .$$

Observe that the features are normalized before being fed into the MLP or attention layer. Besides, a series of two shortcut connections directly connects the encoder's input and output.

### C.2.1. INPUT MODULE

The first layer of a vision transformer is a convolutional module that extracts embeddings for each input patch. For ViT-B/32 used in our experiments, this layer has 768 kernels with size and stride of 32. The output is thus a feature representation of a $32 \times 32$ patch within the input image. During training, a $3 \times 224 \times 224$ image is fed into the convolutional module and produces a $768 \times 7 \times 7$ vector. This intermediate state is permuted and reshaped to a $49 \times 768$ hidden state, regarded as 49 token vectors with 768 features each. The hidden state is then concatenated with an additional $1 \times 768$ class-token vector, and added to a $(49 + 1) \times 768$ position embedding. This is the input to the model's first encoder block.

Since we only get (part of) a 768 feature vector to represent a $32 \times 32$ patch, we need to compress the patch somehow. We do this by using some convolutional filters to convert each patch to a downscaled and grayscaled version (by selecting and grayscaling one pixel per patch). [3] The output is of size either $16 \times 16 = 256$ (if we only reconstruct individual patches) or $8 \times 8 = 64$ (if we reconstruct entire images). These features are stored in the **tok** portion of the input feature vector for each patch. As an additional trick, we ensure that these features have zero mean, which helps for the subsequent layer normalization.

The kernels that correspond to the benign input positions **ft** are not modified, so as to keep valuable features from the pretrained model. If we want to reconstruct entire images, we use the position embeddings to implement a position key as described in Appendix B. The kernels that map to other parts of the feature vector (i.e., **act**, **seq**) are set to zero.

### C.2.2. SEQUENCE KEY CREATOR & BACKDOOR MODULE

The first encoder block of the ViT is used to create sequence keys and to implement the backdoor module (as described in Appendix B).

---

[3]We can use a kernel to extract a target grayscaled pixel from each patch. By combining different extractor kernels, we can extract a downscaled sub-image.

To create sequence keys, we first stabilize the layer normalization before the attention module with the module $\mathrm{StabLN}(\mathbf{x}, \mathcal{J}_{\mathrm{key}}, \mathbf{1}, \mathbf{0}, \mathbf{0}, \mathbf{0})$ from Appendix A.1. We then use the attention module to create sequence keys, as described in Appendix A.4. We use the features in $\mathcal{J}_{\mathrm{tok}}$ to create sequence keys, and store the keys in the **seq** positions.

We then implement the backdoor module as described in Appendix B, using the $\mathrm{SpecLinear}$ module that combines a layer normalization with the subsequent MLP. If the target is simply to reconstruct individual patches, rather than complete images, the backdoor construction is simplified as we just apply the backdoor weight to the whole backdoor key **key**, without any regard for position or sequence information.

### C.2.3. AMPLIFIER & ERASURE MODULE

The amplifier module is instantiated using the MLP of the second encoder block. The attention part of this encoder block is skipped using the shortcut connection. The layer normalization before the MLP module is stabilized. Only features belonging to **act** are fed into the MLP module. If an activation signal is above a pre-assigned noise threshold, it gets amplified significantly (we can achieve this conditional amplification using a simple 1-layer MLP). Features that do not belong to $\mathcal{J}_{\mathrm{act}}$ skip the whole encoder block through the shortcut connections and remain unchanged.

Note that we could have also used the second layer of the MLP in the first encoder block to amplify activation signals, but this approach is problematic for GELU transformers because we cannot avoid unexpected gradients (i.e., when the GELU inputs are close to zero) being amplified. By pushing the amplifier into the next encoder block, we can use additional layers to ensure that small backdoor outputs gets dampened out, before the amplification.

The erasure module is implemented using the MLP part of the third encoder block. Since the **key** features have already played their role, they are wiped out to facilitate signal propagation. The reason this is necessary is because some of the backdoor inputs (e.g., the sequence keys) have large magnitude and this would distort the benign features and cause an accuracy drop on downstream tasks.

Hidden states from the second encoder block are directly fed into the MLP module through the shortcut connection. The layer normalization is stabilized using the module $\mathrm{StabLN}\,(\mathbf{x}, \mathcal{J}_{\mathrm{key}}; \boldsymbol{\gamma}, \mathbf{0}, \mathbf{0}, \mathbf{0})$, which wipes out any features not belonging to **key**. The weights and biases of the MLP module are manipulated to obtain the following forward pass:

$$\mathbf{x}_L \leftarrow \mathbf{0} * (\mathbf{key} - \mu_{\mathbf{key}}) = \mathbf{0} \qquad\qquad \mathbf{x}_R \leftarrow \boldsymbol{\gamma} * (\mathbf{key} - \mu_{\mathbf{key}}) = \boldsymbol{\gamma} * \mathbf{key} \qquad \text{stable layer normalization}$$

$$\mathbf{x}'_L \leftarrow \mathbf{0} \qquad\qquad \mathbf{x}'_R \leftarrow \mathrm{GELU}(\mathrm{W}^{(1)}\mathbf{x}_R + \mathbf{b}^{(1)}) \qquad \text{linear layer 1}$$

$$\mathbf{x}''_L \leftarrow \mathbf{0} \qquad\qquad \mathbf{x}''_R \leftarrow \mathrm{W}^{(2)}\mathbf{x}'_R + \mathbf{b}^{(2)} \qquad \text{linear layer 2}$$

$$\mathbf{z}_L \leftarrow \mathbf{0} + \mathbf{x}_L = \mathbf{0} \qquad\qquad \mathbf{z}_R \leftarrow \mathbf{x}''_R + \mathbf{x}_R = \mathbf{0} \qquad \text{shortcut connection}$$

$$\text{s.t.} \quad \mathrm{W}^{(2)}\mathrm{W}^{(1)}\boldsymbol{\gamma} = -\mathbf{1} \qquad \mathbf{b}^{(2)} = -\mathrm{W}^{(2)}\mathbf{b}^{(1)} \ .$$

$$(14)$$

In words, we ensure that the MLP implements a negative identity mapping, so that the shortcut connection wipes out the features. The above assumes that the input to the GELU is large enough so that it behaves like an identity. We achieve this by setting $\mathbf{b}^{(1)}$ to a large positive value. The choice of weights $\boldsymbol{\gamma}, \mathrm{W}^{(1)}, \mathrm{W}^{(2)}$ is important to guarantee the robustness of this module. If they are not chosen carefully, this module will drift from its original design after a few gradient updates.

### C.2.4. SIGNAL-PROPAGATION & OUTPUT MODULE

The Signal propagation module is implemented on the eight remaining encoder blocks before the final output block. We want the classification-irrelevant information belonging to **key** and **act** to have as little impact on these encoder blocks as possible. Therefore, all weights and biases connected to these features are set to zero in the layer normalizations, attention module, and MLP. Then, the backdoor activation signals simply propagate through the entire block by the shortcut connections.

The output module's goal is to aggregate activation signals onto the class token CLS. This is implemented in the last encoder block. The first layer normalization is stabilized in order to avoid any signal attenuation, and we then re-use the same trick as for sequence keys by using the attention module to aggregate information across all tokens. Specifically, we use the structure $\mathrm{Syn}(X, \mathcal{J}_{\mathrm{act}}; 1)$ to aggregate all activation signals. The remaining modules in this block are used to implement identity mappings. At the end of the encoder, the class-token vector is passed to a stabilized layer normalization, which ensures that the **act** features are not downscaled or mixed with the benign features.

## C.3. BERT

The encoder block of a BERT model, $Z_{\text{out}} = \text{BERTEncoderBlock}\left[X_{\text{in}}\right]$ is written as:

$$\left[\tilde{\mathbf{x}}^{(1)} \quad \tilde{\mathbf{x}}^{(2)} \quad \cdots \quad \tilde{\mathbf{x}}^{(k)}\right]^T = \text{CompleteAttention}\left[\left[\mathbf{x}_{\text{in}}^{(1)} \quad \mathbf{x}_{\text{in}}^{(2)} \quad \cdots \quad \mathbf{x}_{\text{in}}^{(k)}\right]^T\right]$$

$$\mathbf{x}^{(i)} = \tilde{\mathbf{x}}^{(i)} + \mathbf{x}_{\text{in}}^{(i)}, \text{ for } 1 \le i \le k$$

$$\mathbf{z}_{\text{in}}^{(i)} = \text{LN}\left[\mathbf{x}^{(i)}\right]$$

$$\tilde{\mathbf{z}}^{(i)} = \text{MLP}\left[\mathbf{z}_{\text{in}}^{(i)}\right]$$

$$\mathbf{z}^{(i)} = \tilde{\mathbf{z}}^{(i)} + \mathbf{z}_{\text{in}}^{(i)}$$

$$\mathbf{z}_{\text{out}}^{(i)} = \text{LN}\left[\mathbf{z}^{(i)}\right] .$$

Thus, in contrast to the ViT, the inputs are not directly connected to the outputs with a shortcut connection. Inputs are thus necessarily distorted twice by layer normalizations before being output.

### C.3.1. INPUT MODULE

The first layer of a BERT model is an embedding layer. Discrete words and positions are converted to 768-entry word and position embeddings. For a word $d_j$ at position $j$, its word embedding vector $\tilde{\mathbf{e}}^{(d)}$ and position embedding vector $\mathbf{e}^{(j)}$ are added together for a complete representation $\mathbf{x}^{(d,j)} = \tilde{\mathbf{e}}^{(d)} + \mathbf{e}^{(j)}$.

We use the approach described in Appendix B to create position identifiers for our backdoors. We add an additional requirement to Equation (8), to ensure that special tokens (such as the padding token [PAD]) do not activate backdoors. For this, we ensure that the shared embedding of these special tokens (in the coordinates corresponding to the position key $\mathbf{pos}$) is some vector $\mathbf{u}^{(-)}$ that is negatively aligned with all other position keys $\mathbf{u}^{(1)}, \ldots, \mathbf{u}^{(n)}$.

More precisely, the following operations are done to produce embeddings:

$$\text{word embedding}: \quad \tilde{\mathbf{e}}_{\text{pos}}^{(d)} \leftarrow \begin{cases} \mathbf{0} & \text{if } d \text{ is not a special token} \\ \mathbf{u}^{(-)} & \text{if } d \text{ is a special token} \end{cases} ; \quad \tilde{\mathbf{e}}_{\text{seq}}^{(d)} \leftarrow \mathbf{0}, \quad \tilde{\mathbf{e}}_{\text{act}}^{(d)} \leftarrow \mathbf{0}, \quad \tilde{\mathbf{e}}_{\text{ft}}^{(d)} \leftarrow \tilde{\mathbf{e}}_{\text{ft}}^{(d)} \quad \forall d;$$

$$\text{position embedding}: \quad \mathbf{e}_{\text{pos}}^{(j)} \leftarrow \mathbf{u}^{(j)}; \quad \mathbf{e}_{\text{seq}}^{(j)} \leftarrow \mathbf{0}, \quad \mathbf{e}_{\text{act}}^{(j)} \leftarrow \mathbf{0}, \quad \mathbf{e}_{\text{ft}}^{(j)} \leftarrow \mathbf{e}_{\text{ft}}^{(j)} \quad \forall j .$$

In summary, the entries for benign features $\mathbf{ft}$ are not changed to inherit the utility of the pretrained weights. There is a layer normalization module next to these embedding modules, which is stabilized as $\text{StabLN}\left(\mathbf{x}, \mathcal{J}_{\text{ft}}; \mathbf{1}, \mathbf{0}, \mathbf{1}, \mathbf{0}\right)$. ($\mathbf{ft}$ are not filtered out in order to recover words.)

### C.3.2. SEQUENCE KEY CREATOR & BACKDOOR & AMPLIFIER & ERASURE MODULE

The backdoor module follows the design outlined in Appendix B. We use the first encoder's attention module to create sequence keys, and the subsequent MLP to implement all our backdoor units. Since the benign features contain both the word and the position information, the benign features are selected as the source from which sequence keys are generated. The sequence keys are shared by all unmasked tokens, i.e., non-pad tokens. To prepare for backdoor units, the layer normalization after the first attention module is stabilized using the structure $\text{StabLN}(\mathbf{x}, \mathcal{J}_{\text{ft}}; \mathbf{1}, \boldsymbol{\beta}_L, \mathbf{1}, \mathbf{0})$.

The amplifier module is directly realized by the second linear layer of the first MLP using the coefficient $\theta$ in Equation (11). The erasure module is implemented by the last layer normalization module of the first encoder block. After generating activation signals, position information and sequence keys are useless for the following logical modules. They are wiped out by assigning weights and biases belonging to **key** of this LN to zero. This LN is stabilized for convenience and serves as an almost identity mapping for benign features belonging to $\mathbf{ft}$ and activation signals belonging to **act**. The second encoder block is used to fuse-in weights from the first encoder's layer normalization, which were dropped to stabilize this LN. In this way, we guarantee that the benign features are still computed correctly.

C.3.3. SIGNAL-PROPAGATION & OUTPUT MODULE

The signal propagation module uses the nine remaining encoder blocks before the final output block. A total of 512 features belonging to **ft** are used for downstream tasks. These encoder blocks strengthen the utility of benign features and propagate the backdoor activation signals. Yet, it is difficult to propagate the activation signals through dozens of layer normalizations (here, in contrast to the ViT model, we cannot simply propagate through the shortcut connection). If the activation signals are too strong, we get some dampening in each LN, which causes too small gradients. In contrast, if the activation signals are too weak, it is difficult to distinguish them from noise. The easiest solution would be to stabilize each layer normalization using the trick in Appendix A.1. But this would require changing these layers' weights, which would break all the benign features too (unless we retrain the backdoored BERT from scratch). Our simpler solution is to amplify the activation signals in every encoder's MLP in order to approximately offset the signal attenuation we will natively get from the layer normalization (when only using shortcut connections to propagate backdoor outputs). In the MLP, activation signals are amplified if they are more significant than some noise threshold, i.e,

$$\mathbf{x}'_{\text{amp}} = \text{GELU}(\mathbf{x}_{\text{act}} - \delta)$$
$$\mathbf{x}''_{\text{act}} = \theta_{\text{propag}} \cdot \mathbf{x}'_{\text{amp}} .$$

Here $\delta$ is a noise threshold and $\theta_{\text{propag}}$ an amplifier that are heuristically chosen for each encoder block, based on various finetuning experiments. We use amplifiers between $0.2$ and $0.3$ and a noise threshold of $0.2$. The difficulty of overcoming the signal attenuation effect increases with the number of layer normalizations. Fortunately, it is affordable for the number of LNs in the models we experimented with. The unavoidable signal attenuation also brings some benefits for GELU activations, since only significant enough activation signals and their corresponding gradient signals survive through the model.

The last encoder block is used to implement the output module, by averaging activation signals across tokens using the structure $\text{Syn}(X, \mathcal{J}_{\text{act}}; 1)$ from Appendix A.4. On top of the BERT's encoder is an additional pooling module, which is essentially a MLP using a tanh activation. In our experiments, we replace this tanh function by a ReLU for convenience, as this has no impact on downstream performance. The first layer of the pooling module implements an identity mapping for benign features and an amplifier for signal features. Generally, we find that activation signals are not of much greater magnitude than the benign features at this point, to the signal attenuation in LNs. By amplifying the activations, we ensure they receive large enough gradient signals to shut down backdoors.

## C.4. Backdoor Robustness

As mentioned in the main body, the finetuned backdoored transformer should be helpful on downstream tasks, with enough robustness to afford unexpected weight disturbances. In this appendix, we give a more detailed description of robustness, which is the critical challenge in this work.

Gradients that flow back to shut down a backdoor are always large, to ensure the backdoor gets shut properly. These large gradients can negatively impact the model's benign features. To make matters worse, our backdoor construction relies on a number of *reusable* modules (e.g., for preparing inputs, or computing sequence keys). The weights in these modules are necessarily updated when a backdoor fires. A too large deviation could cause these modules to break down, and generate large random features that negatively impact the downstream performance. Fortunately, these large backdoor gradients occur rarely, and become acceptable after batch averaging. However, if we are unlucky and such large gradients occur many times in an optimization step (e.g., because of a negative gradient signal into a backdoor unit that did not shut the backdoor down), they can cause unrecoverable damage to the pretrained weights.

In practice, the robustness of a transformer is usually determined by a specific bottleneck module. The robustness of different modules is correlated: we sometimes need a module to have larger weights than the module preceeding it, but this then causes large gradients to flow into a module with small weights. We thus have to carefully tune each module's parameters (sometimes empirically across multiple finetuning runs) to ensure that the transformer is robust.

**Remark.** A trivial approach to increase the robustness of a model would be to use a very small learning rate or to freeze some weights. This would inevitably raise suspicion. In this work, we thus adopt a conventional finetuning setup in which all parameters are updatable, and the learning rate is reasonable to obtain good downstream performance.

### C.4.1. Robustness Analysis Examples

The robustness of a backdoored transformer is determined by a few bottleneck modules. In this section we analyze some typical bottleneck modules as examples. In our experiments, we found that when a module is not robust, a breakdown usually occur at the first few updates. For simplicity, we thus analyze the robustness issue by looking at the very first update.

**Erasure module of ViT.** The erasure module generally serves as the bottleneck of a backdoored ViT. Consider the structure defined in Equation (14), which is supposed to zero out the inputs, and focus on the second linear layer. Given the backward gradient signal $\lambda_i := \frac{\partial \mathcal{L}}{\partial h_i}, i \in \mathcal{J}_{\text{act}}$ of a backdoor unit, the update of this layer is calculated as:

$$\Delta W_{ij}^{(2)} = \begin{cases} -\frac{\eta}{B}\lambda_i x'_j & \text{if } j \in \mathcal{J}_{\text{key}} \\ 0 & \text{else} \end{cases},$$

where $B$ is the batch size, $\eta$ is the learning rate. In the next step, the output of the second linear layer is calculated as:

$$x_i'' = \sum_{j \in \mathcal{J}_{\text{key}}} -\left(\frac{\eta}{B}\lambda_i x'_j\right) x'_j \approx -|\mathcal{J}_{\text{key}}|\frac{\eta}{B}\lambda_i \delta_1^2 \, .$$

In our experiments, $\eta = 10^{-4}, B = 128, |\mathcal{J}_{\text{key}}| = 256$, and $\delta_1 \sim 10$ for GELU. So we have $|x_i''| \approx 0.02\lambda_i$ in all token vectors. The module thus no longer zeros out inputs exactly. If there is an amplified module after this erasure module, we thus risk that these non-zero outputs get magnified to the point where they override any benign features (especially for the token vector).

**Sequence key creator.** The hyper-parameter $\rho$ of the sequence key creator in Equation (7) is an essential hyper-parameter for robustness. The sequence key creator module essentially consists of two consecutive linear layers:

$$\begin{aligned}
&\begin{bmatrix} \mathbf{x}^{(1)} & \mathbf{x}^{(2)} & \cdots & \mathbf{x}^{(k)} \end{bmatrix} \longrightarrow \\
&\begin{bmatrix} W^{(V)}\frac{1}{k}\sum_j \mathbf{x}^{(j)} + \mathbf{b}^{(V)} & \cdots \end{bmatrix}^\top \longrightarrow \begin{bmatrix} W^{(O)}\left(W^{(V)}\frac{1}{k}\sum_j \mathbf{x}^{(j)} + \mathbf{b}^{(V)}\right) + \mathbf{b}^{(O)} & \cdots \end{bmatrix} \, .
\end{aligned} \tag{15}$$

These weights are assigned using sparse matrices according to Equation (7) and Equation (11), and the later backdoor thresholds are chosen based on the initial distribution of sequence keys. Unfortunately, this distribution shifts during training: the initial sparse matrices become fully connected after the very first update due to the non-zero inputs and gradients. Even though the noise of one single weight is tiny, the accumulated effect of so many weights is significant. If the sequence keys shift too significantly, they cannot activate their original backdoor families and decrease the number of reconstructed sequences. Sometimes, worse may happen due to the unpredictability of distribution shifts. This defect can be attenuated by utilizing a greater hyper-parameter $\rho$ of the attention-based module.

## D. Experimental Setup

We implement our backdoor attack using Python 3.10, Pytorch 2.0, and Opacus 1.4. For attackers, weight manipulations and reconstruction can be executed on a laptop's CPU within a few seconds. All our models are trained or finetuned on a GPU within a few minutes.

**Datasets.** CIFAR-10 (CIFAR-100) (Krizhevsky, 2009) is a 10-class (100-class) object recognition dataset of 50,000 training samples with a size of $32 \times 32$ pixels. Oxford-IIIT Pet (Parkhi et al., 2012) is a 37-class animal recognition dataset containing 3,680 training samples. Caltech 101 is a 101-class object recognition dataset containing 9,146 images, of varying sizes of roughly $300 \times 200$ pixels. Since Caltech 101 does not distinguish between the training set and the test set, we divide two-thirds of the samples into the training set and one-third into the test set. TREC (Hovy et al., 2001; Li & Roth, 2002) is a question classification dataset containing 5,452 labeled questions in the training set. There are 6 coarse class labels in TREC, which can be further divided into 50 fine class labels. In the training set, TREC contains short sentences that have at most 41 tokens.

**Data Stealing in MLP.** Our toy MLP model in Section 4 is trained on a randomly selected subset of 10,000 samples from the CIFAR-10 training set. We use a standard SGD optimizer with a batch size $64$ and a learning rate of $0.05$, and train

for 20 epochs. We use quantile thresholds $\{Q(0.001)\}$ (see Equation (16)) so that ten samples activate each backdoor unit in our training set. For the second layer of the toy MLP model, we use amplifier coefficients between 500 and 1000. We replicate the same setup for CIFAR-100.

**Data Stealing in ViT.**   Since Caltech 101 contains more classes than Oxford-IIIT Pet, we use the Oxford-IIIT Pet dataset when the classification head is handcrafted by the adversary and the Caltech 101 dataset for the harder setting where the classification head is randomly initialized by the victim. Raw images are normalized before being fed into a model. We apply a typical finetuning recipe, with a large learning rate for the encoder and a small learning rate for the classification head. Specially, we use a standard SGD optimizer with a learning rate of $(0.2, 10^{-4})$ and a batch size of 128. The pretrained model is finetuned for 12 epochs. We set quantile thresholds $\{Q(0.001)\}$ (see Equation (16)) when the target is to reconstruct complete images. When the target is to reconstruct individual patches only, the definition of the quantile threshold is subtle because an image is cut into 49 patches, which are the actual inputs into a backdoor unit. In this situation, ten patches are allowed by an artificially assigned threshold to activate each backdoor unit. This threshold causes more backdoors to capture a mixture of inputs.

**Data Stealing in BERT.**   We use TREC as the downstream task for BERT. We use the coarse class labels of TREC when the classification head is handcrafted by the adversary, and the fine class labels for the harder setting where the classification head is randomly initialized by the victim. Since TREC only contains short sentences, the maximum length of the tokenizer is assigned to 48. We use an SGD optimizer with a learning rate of $(0.05, 10^{-4})$ and a batch size of 32. The pretrained model is finetuned for 12 epochs. We set quantile thresholds $\{Q(0.001)\}$ (see Equation (16)) for the sequence keys.

### D.1. Reconstructing and Matching Captured Inputs

Since some backdoors may be activated more than once during training (and GELU transformers always leak small gradients into the backdoor for all inputs), the reconstructed input may not always match a ground truth exactly.

When presenting the ground truth for our reconstructions, we thus need some criteria to determine that there is a unique match. One solution could be to check if the reconstructed input is highly similar (under some metric) to one training input. But similarity metrics for images and text can be error prone, as they often allow for some pathological matches (e.g., two images with very similar backgrounds). Instead, we measure the strength of the backdoor activations during training, and mark a training sample as captured if the activation signal exceeds some threshold (a hyper-parameter that we set manually through experimentation)

For experiments with images, we show the ground truth when a backdoor unit was only strongly activated by a single training sample during training. Otherwise, we consider the ground truth to be ambiguous and do not report it. For text models, we report all training sentences that strongly activated the backdoor corresponding to the first word in that sentence. In some cases, we thus list multiple possible ground truth sentences for a reconstructed sentences. However, in many cases we find that the signal of one of the captured sentences "dominates" others, and so the backdoor reconstruction is very close to one of the ground truths.

### D.2. Large Model Preprocessing

**Initializing Backdoored Transformers**   Our backdoor constructions require carving out parts of an original transformer's weights and features. Specifically, we need to reduce the transformer's benign capacity both "horizontally" (by reserving part of the internal feature vectors for propagating backdoor inputs and outputs) and "vertically" (by reserving the first encoder blocks for the backdoor modules). As a result, we thus embed a shorter and narrower benign transformer inside a larger backdoored transformer.

Specifically, we start from a pretrained transformer with 12 encoder blocks, and internal features of dimension $768 = 12 \cdot 64$. We reserve the first $k$ blocks for the backdoor construction, and 256–320 features for backdoor signals (see Table 2 for details). To this end, we *shift* all transformer blocks upward, keeping only the $12 - k$ first encoder blocks and stacking them on top of the blocks used for the backdoor. We keep the weights connected to the benign features unchanged, and modify the rest according to our backdoor construction.

This construction reduces the original transformer's utility in two ways. First, we now only have $12 - k$ blocks used for feature extraction. This is thus equivalent to using the original transformer and extracting classification features from the $k^{\text{th}}$ to last layer, rather than the last layer. Second, we have removed part of the transformer's internal features, which thus leads

to sparser and less expressive representations. In Figure 7, we analyze the downstream effect of this approach for a ViT model finetuned on Oxford-IIIT Pet, as a function of the number of benign blocks and features that are retained.
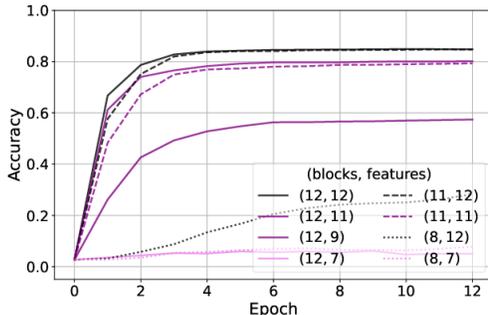


*Figure 7.* Accuracy of ViTs on Oxford-IIIT Pet as a function of the number of retained blocks (out of 12) and features ($\times 64$, out of $12 \times 64 = 768$).

The ViT we use in our experiments retains 8 blocks and $448 = 7 \cdot 64$ features, which results in low downstream performance out-of-the-box. We thus first finetune this smaller retained transformer on ImageNet before we conduct our backdooring experiments. For experiments with BERT, we find that the small retained benign transformer still performs well on the TREC dataset without any need for initial finetuning. Once the small benign transformer has been finalized, we add our handcrafted backdoor blocks. This yields the final backdoored model that is handed to the victim for finetuning on private data.

**ReLU Transformers.** In some of our additional experiments in Appendix F we use transformers with ReLU activations, as these are easier to backdoor than the original GELU transformers. However, there are no pretrained transformers of this type. For BERT, we find that directly replacing GELUs with ReLUs still leads to an acceptable performance on the TREC downstream task. For the ViT, a ReLU-version has poor performance on Caltech 101 and Oxford-IIIT Pet. Therefore, we first finetune the ReLU-version ViT on ImageNet to improve its utility, starting from the pretrained weights for GELU.

**Utility of backdoored transformers.** Our backdoored transformer obviously has somewhat worse performance on the downstream tasks than the original pretrained transformer. Part of this is due to the fact that we only retain a portion of the original benign weights, as noted above. In addition, gradients from the backdoors can interact with, and harm some of the model's benign weights and features during finetuning.

The compound effect of both issues is investigated in Table 3, for a backdoored ViT finetuned on Caltech 101. By comparing the complete and small model, we observe that the retained benign sub-transformer performs closely to the complete model for this task (after initial finetuning on ImageNet, as described above).

*Table 3.* Compare the performance of backdoored ViTs with typical ViTs on the Caltech 101 dataset. **G**: using GELU activation; **R**: using ReLU activation. **Complete**: a typical pretrained vision transformer downloaded from Torchvision (before preprocessing); **Small**: the small benign part of the backdoored transformer (after preprocessing); **Malicious**: a backdoored transformer for reconstructing complete images (after manipulating).

| Accuracy | Complete | | Small | | Malicious | |
|---|---|---|---|---|---|---|
| | GELU | ReLU | GELU | ReLU | GELU | ReLU |
| Train | 99.7% | 99.9% | 99.9% | 99.9% | 88.3% | 69.3% |
| Test | 95.0% | 95.1% | 91.3% | 91.9% | 82.6% | 67.8% |

The addition of the full backdoor functionality ("malicious") does show that the backdoor negatively influences the model's utility. With GELU activations, there is an accuracy drop of around ten percent, which could be compensated by backdooring a larger model, or by more carefully tuning the backdoored model for downstream utility before publishing it.

Figure 8 shows the accuracy of our backdoored BERT model (with GELUs or ReLUs) when finetuned on TREC. The backdoored model performs well on this fairly easy classification task.
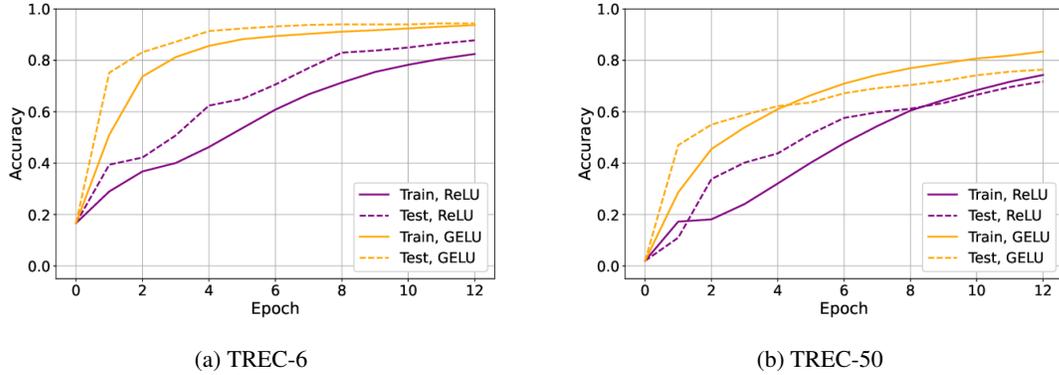
(a) TREC-6

(b) TREC-50

*Figure 8.* Accuracy of backdoored BERTs on the TREC dataset during finetuning.

### D.3. Setting Backdoor Parameters

**Setting backdoor thresholds.** We want to set thresholds for our backdoors so that a small fraction of inputs can trigger the backdoor. For simplicity, in our experiments we artificially set *quantile thresholds* $Q(p)$ for each backdoor weight $\mathbf{w}$:

$$\Pr\left[\mathbf{w}^\top \mathbf{x} > Q(p) \,\middle|\, \forall \mathbf{x} \in \mathcal{D}_0\right] = p \,, \tag{16}$$

where $\mathcal{D}_0$ is the set of backdoor inputs computed by passing the finetuning dataset to the backdoor at initialization. Of course, attackers do not have access to the exact private finetuning dataset. However, they can use a similar dataset (e.g., a public test dataset) to estimate these quantile thresholds (see (Fowl et al., 2021)). We emulate such a setting in our experiments by selecting loose quantiles that allow for a small number $k > 1$ of inputs to trigger the backdoor (typically $k \approx 10$).

**Targeted backdoors.** The backdoor attacks we describe in the main body, and which we use for most of our experiments, make use of randomly generated backdoor weights. The attacker thus has no control over the type of inputs that might be captured. However, we know that a backdoor is likely to capture inputs that are similar (i.e., have large inner product with) to the backdoor weight. An attacker could thus also aim to target specific types of examples by selecting appropriately similar backdoor weights.

For images, a simple approach for creating targeted backdoor weights is to directly set weights as a targeted image (e.g., chosen from some dataset similar to the finetuning set). In Figure 9, we instantiate such an attack and compare the reconstructed images to the corresponding "bait" images used as weights. As expected, the captured images share close similarities (e.g., very similar backgrounds) to the backdoor weights.



(a) Reconstruction

(b) Weight

*Figure 9.* Real images can be used as targeted backdoor weights. We finetune a MLP with a random classification head on the CIFAR-10 training set. The backdoor weights are randomly selected from the CIFAR-10 test dataset (with quantile thresholds $Q(0.001)$).

# E. Backdoored Models are Nearly Tight for DP-SGD

This appendix contains further details and analysis for our experiments with DP-SGD in Section 6.1.

We recall our overall idea: we aim to build a backdoored model so that some unit in the model activates if and only if some target data point is in the finetuning set. In turn, this will cause the weights of that unit to be updated only if the target is present. This is an instantiation of the worst-case scenario considered in the privacy analysis of DP-SGD, except that the analysis assumes a strong attacker who sees all intermediate noisy gradients. Instead, we will see how tight this worst-case scenario is for a more realistic "end-to-end" adversary who only sees the final model.

**Differential privacy.**   We start with the standard definition of $(\varepsilon, \delta)$-differential privacy (Dwork et al., 2006):

**Definition E.1.**  A randomized mechanism $\mathcal{M} : \mathcal{D} \to \mathcal{R}$ is $(\varepsilon, \delta)$-differentially private if for all neighboring datasets $D, D' \in \mathcal{D}$ (i.e., datasets that differ in a single element), and any set of outputs $S \subseteq \mathcal{R}$, we have that:

$$\Pr\left[\mathcal{M}(D) \in S\right] \leq e^{\varepsilon} \cdot \Pr\left[\mathcal{M}(D') \in S\right] + \delta \,.$$

When training a machine learning model with differential privacy, the mechanism $\mathcal{M}$ is a randomized training algorithm, and the definition implies that any final model would have been obtained with approximately the same probability if one training point had been removed.

**Differentially private stochastic gradient descent.**   The DP-SGD algorithm of Abadi et al. (2016) trains machine learning models that are provably differentially private. The algorithm works as follows:

1. In each training step, we sample a batch by picking each training input uniformly at random with probability $q$.

2. We compute a gradient **g** for each element in the batch, and *clip* each gradient to a maximal $\ell_2$ norm of $C$.

3. We sum all clipped gradients, add Gaussian noise $\mathcal{N}(\mathbf{0}, \sigma^2 C^2 \cdot I)$, and divide by the expected batch size $q \cdot |D|$.

4. We take an update step with learning rate $\eta$, and repeat.

The privacy analysis of DP-SGD operates roughly as follows:

1. Each noisy gradient step is differentially private (with respect to neighboring *batches*) by virtue of the Gaussian mechanism (Dwork et al., 2014).

2. The random sampling of the batch amplifies privacy (Abadi et al., 2016).

3. The combination of all noisy gradient steps is differentially private by applying an advanced composition theorem (Kairouz et al., 2015; Mironov et al., 2019; Abadi et al., 2016).

The first step is tight in the worst-case. An example is when there is one data point with a gradient of the form $[C, 0, \ldots, 0]$ and all other data points have gradients of the form $[0, 0, \ldots, 0]$. Then, observing the change in the model's first weight leaks the maximum amount of privacy. The full privacy analysis, however, is only known to be tight if this worst-case materializes in *every single training step*, and *the attacker observes every intermediate noisy gradient*. This analysis is thus presumed to be quite loose in practice. But we show that a backdoored model can result in a fairly close lower bound.

**Lower bounding privacy leakage.**   Differential privacy can be seen as a bound on the error of any adversary that tries to distinguish two neighboring databases $D$ and $D'$, that differ in some example **x**. By instantiating such an adversary and recording their empirical success rates, we can get an empirical lower bound on the privacy level $\varepsilon$ (see e.g., (Nasr et al., 2021; Jagielski et al., 2020; Steinke et al., 2023)). Yet, this approach is computationally expensive, especially if we want to refute large values of $\varepsilon$. Instead, we will analyze our backdoor constructions analytically, and then directly get a DP lower bound by numerically computing the attacker's distinguishing power.

Concretely, our (ideal) attacker will backdoor the model so that the gradient for some example $\mathbf{x}$ concentrates uniformly on some weight indices $I$, with $|I| = n$. That is, the gradient of $\mathbf{x}$ is of the form:

$$g_i = \begin{cases} \pm C/\sqrt{n} & \text{if } i \in I \\ 0 & \text{otherwise .} \end{cases} \tag{17}$$

Moreover, the gradients of all other training examples are zero at the indices $I$. Now suppose we have a noisy gradient $\tilde{\mathbf{g}}$ where noise $\mathcal{N}(0, \sigma^2 C^2)$ is added to every coordinate, and let

$$\Delta(\tilde{\mathbf{g}}) = \frac{1}{\sqrt{n}} \cdot \sum_{i \in I} \texttt{sign}(g_i) \cdot \tilde{g}_i . \tag{18}$$

We then have:

$$\Delta(\tilde{\mathbf{g}}) = \begin{cases} C + \mathcal{N}(0, \sigma^2 C^2) & \text{if } \mathbf{g} \text{ is the gradient of } \mathbf{x} \\ \mathcal{N}(0, \sigma^2 C^2) & \text{otherwise .} \end{cases} \tag{19}$$

So we still get a tight instantiation of the Gaussian mechanism in every batch. Suppose that the attacker knows the original value of the weights $\mathbf{w}_I$ before finetuning, and can observe (or infer) the final values $\mathbf{w}'_I$ after finetuning for $T$ steps. The attacker can then compute $\Delta(\mathbf{w}'_I - \mathbf{w}_I)$ and use this as a statistic for their hypothesis test. The value of this statistic will be:

$$\Delta(\mathbf{w}'_I - \mathbf{w}_I) = \begin{cases} \sum_{j=1}^T \Pr[B(T,q) = j] \cdot \mathcal{N}(jC, T\sigma^2 C^2) & \text{if } \mathbf{x} \text{ is in the dataset} \\ \mathcal{N}(0, T\sigma^2 C^2) & \text{otherwise ,} \end{cases} \tag{20}$$

where $B(T, q)$ is the binomial distribution with $T$ trials and probability $q$ (which represents the number of times that $\mathbf{x}$ is sampled in a batch across the entire training run).

Let us call this quantity $\Delta_{\mathbf{w}}$. Differential privacy provides a direct bound on an adversary's ability to distinguish between $D$ and $D'$ by looking at the value of $\Delta_{\mathbf{w}}$ (see e.g., (Kairouz et al., 2015)). Suppose the adversary uses the test statistic $\{\Delta_{\mathbf{w}} \geq t\}$ for some threshold $t$ to distinguish $D$ and $D'$. We can then *lower-bound* the privacy leakage $\varepsilon$, for a fixed parameter $\delta$, by the quantity:

$$\tilde{\varepsilon} = \max_t \log \left[ \frac{\Pr\left[\Delta_{\mathbf{w}} \geq t \mid \mathbf{x} \text{ in dataset}\right] - \delta}{\Pr\left[\Delta_{\mathbf{w}} \geq t \mid \mathbf{x} \text{ not in dataset}\right]} \right] . \tag{21}$$

Note that we can compute this quantity numerically for any value of $t$: the denominator is simply the complementary CDF of a Gaussian, while the numerator is the complementary CDF of a mixture of $T$ Gaussians. We search for the maximum by a grid search over $t$. Note that compared to prior DP lower bounds such as (Nasr et al., 2021), we do not need to compute any confidence intervals or guard against multiple comparisons: we are not performing any random sampling here, and instead directly numerically computing the ratio of two distributions.

We start by showing that such an "end-to-end" attack on DP-SGD can be close to tight, for some choices of parameters. In Figure 10, we plot the provable privacy guarantee $\varepsilon$ (Theoretical) and our lower-bound from Equation (21) (Estimated) for various choices of DP-SGD's noise multiplier $\sigma$. As the noise multiplier increases, the lower bound becomes increasingly tight.

### E.1. Backdoor Constructions

We now describe our backdoor constructions, that aim to instantiate the worst-case setting described above, where a target's gradient solely concentrates on a few weights in every training set. We consider a setting where the victim downloads a pretrained model and finetunes the top $k$ layers while keeping the bottom layers frozen. This is a common setup for differentially private transfer learning (Abadi et al., 2016; Tramer & Boneh, 2020).

We consider two cases:

- $k = 1$: Only the final classification head is trained.
- $k > 1$: The last $k$ layers of the model form a MLP which is finetuned, with the prior feature extraction layers frozen.
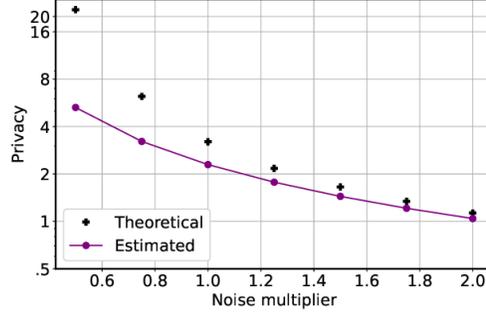
*Figure 10.* The effect of the noise multiplier ($\sigma$) on the tightness of our end-to-end lower bound for DP-SGD (privacy budgets are computed for 30 epochs of DP-SGD, with a sampling rate of $q = 0.01$).

### E.1.1. FINETUNING THE CLASSIFICATION HEAD

Consider the case that a single-layer classification head is finetuned on top of frozen features. Let $\mathbf{h}$ denote the model's final feature layer, $\mathbf{z}$ denote the logits, and $\mathbf{s}$ denote the softmax scores. Let $h_1$ be a backdoored unit, with the model's frozen weights set so that $h_1 = 0$, except for the target $\mathbf{x}$ where the unit is active, $h_1 \gg 0$.

The logits are then given by:

$$z_j = w_{j,1} \cdot h_1 + \sum_{k>1} w_{j,k} \cdot h_k + b_j \tag{22}$$

The gradients for the finetuned weights are then:

$$\frac{\partial \mathcal{L}}{\partial w_{j,k}} = \begin{cases} h_k(s_j - 1) & \text{if } j \text{ is the correct label} \\ h_k \cdot s_j & \text{else} \end{cases}$$

Since $h_1 \gg \sqrt{\sum_{j>1} h_j^2}$, the weights $\{\mathbf{w}_{j,1}\}$ absorb most of the gradient norm after clipping. Moreover, due to the high value of $h_1$, we get a sparse softmax vector of the form $\mathbf{s} \approx [0, \ldots, 0, 1, 0, \ldots, 0]$. Assume the target input $\mathbf{x}$ is of true class $y$, and is misclassified as $y'$. Then, the clipped gradient (assuming $C = 1$) satisfies:

$$\text{clip}\left(\frac{\partial \mathcal{L}}{\partial w_{j,1}}; 1\right) = \begin{cases} +\frac{1}{\sqrt{2}} & \text{if } j = y' \\ -\frac{1}{\sqrt{2}} & \text{if } j = y \\ 0 & \text{else} \end{cases} . \tag{23}$$

If the target $\mathbf{x}$ does not appear, the clipped gradient of $\{w_{j,1}\}$ is obviously zero. We are thus in the setting of Equation (17).

**Stability.** The above analysis implicitly assumes that when the target $\mathbf{x}$ appears, it will always be misclassified into the same class $y' \neq y$. But the classification weights will change during finetuning. If the target's classification were to change during training, we get a different gradient than in Equation (23), and our empirical privacy estimate is wrong. If the attacker chooses the initial weights in the classification head, we can prevent this by ensuring that one weight dominates all others.

**Black-box inference.** So far, we have shown how to get a privacy lower-bound for an attacker who can observe the final value of the finetuned weights. We can turn this into a fully black-box attack by showing that the attacker can infer these values from query-only access to the finetuned model.

Rearranging Equation (22), we have:

$$w_{j,1} = \frac{z_j - b_j - \sum_{k>1} w_{j,k} h_k}{h_1} .$$

We can manipulate the frozen features to ensure that $h_2 = \cdots = h_n = 0$ for the target $\mathbf{x}$. If we query the finetuned model on the target $\mathbf{x}$ and get logits $\mathbf{z}$, we then have:

$$w_{j,1} = \frac{z_j - b_j}{h_1} .$$

The value of $h_1$ is known to the attacker, since it is computed by frozen features. The only unknown is thus $b_j$. But since $h_1$ is large (which leads to a large logit $z_j$), the value of $b_j$ is negligible here. The attacker can thus query the model before and after finetuning on the target $\mathbf{x}$ to obtain logits $\mathbf{z}^{\text{init}}$ and $\mathbf{z}^{\text{final}}$ respectively, and then compute the test statistic

$$\frac{(\mathbf{z}_{y'}^{\text{final}} - \mathbf{z}_{y'}^{\text{init}}) - (\mathbf{z}_y^{\text{final}} - \mathbf{z}_y^{\text{init}})}{\sqrt{2}h_1} \,,$$

which is distributed as in Equation (20) (up to small approximation errors, omitting hyper-parameters of optimization).

### E.1.2. FINETUNING MULTIPLE LINEAR LAYERS

We now consider a more complex setting, where multiple final linear layers of a backdoored model are finetuned with differential privacy.
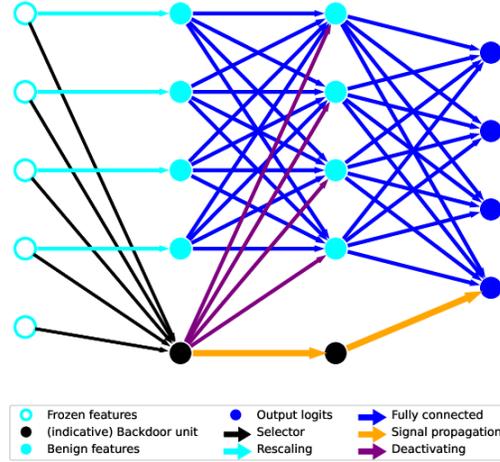


*Figure 11.* Backdoor structure based on the MLP part of a CNN. If two units are not shown as connected, the initial weight between them is assigned to zero.

The backdoor structure is illustrated in Figure 11. The frozen layers compute some features $\mathbf{x}^{(\text{in})}$. The MLP to be finetuned consists of two layers of hidden activations $\mathbf{h}^{(1)}$ and $\mathbf{h}^{(2)}$, which finally connect to the logits $\mathbf{z}$. We have

$$\mathbf{h}^{(1)} = \text{ReLU}(\mathbf{W}^{(0)}\mathbf{x}^{(\text{in})} + \mathbf{b}^{(0)})$$
$$\mathbf{h}^{(2)} = \text{ReLU}(\mathbf{W}^{(1)}\mathbf{h}^{(1)} + \mathbf{b}^{(1)})$$
$$\mathbf{z} = \mathbf{W}^{(2)}\mathbf{h}^{(2)} + \mathbf{b}^{(2)}$$

The backdoor units are $h_1^{(1)}$ and $h_1^{(2)}$. We further ensure that at initialization, the backdoor unit connects to an incorrect class $y'$ for the targeted input $\mathbf{x}$.

We design the backdoor so that the following conditions are satisfied, throughout the entire training process.

1. $h_1^{(1)} > 0$ if and only if the target $\mathbf{x}$ appears
2. $h_1^{(2)} > 0$ if and only if $h_1^{(1)} > 0$
3. $[h_2^{(2)}, \dots, h_n^{(2)}] = \mathbf{0}$ if $h_1^{(1)} > 0$.

In words, the two backdoor units $h_1^{(1)}$ and $h_1^{(2)}$ activate only on the target $\mathbf{x}$. Moreover, on the target $\mathbf{x}$ the rest of the penultimate features in $\mathbf{h}^{(2)}$ are all zero. To ensure that the above conditions hold throughout training, we can set the manipulated backdoor weights to some large values $\gg C$, so that their clipped gradient updates have a negligible influence.

We set the values of the initial weights (both the frozen weights and finetuning weights) to satisfy the following relations:

$$\|\mathbf{x}^{(\text{in})}\| \ll 1, \quad \|\mathbf{h}^{(1)}\| \ll W_{1,1}^{(1)}, \quad |h_1^{(1)}| \ll W_{1,1}^{(2)}, \quad \sqrt{\sum_{j>1} \left(W_{1,j}^{(1)}\right)^2} \ll W_{1,1}^{(1)} \tag{24}$$

**Analysis.** We hope that $\frac{\partial \mathcal{L}}{\partial b_1^{(0)}}$ can be much greater than the norm of all other active parameters' gradients. We have active parameters: $W^{(0)}, \mathbf{b}^{(0)}, W^{(1)}, \mathbf{b}^{(1)}, W^{(2)}, \mathbf{b}^{(2)}$. For an arbitrary weight of a linear unit, its gradient is determined by its backward gradient and the layer's input. For an arbitrary bias of a linear unit, its gradient is only determined by the backward gradient. Specifically, we want $W_{1,1}^{(1)} \cdot W_{1,1}^{(2)} \approx \|\nabla \mathcal{L}\|$. We can analyze each group of parameters separately:

- $W^{(0)}$: If $\|\mathbf{x}\|$ is small, the gradient with respect to $W^{(0)}$ is small.
- $\mathbf{b}^{(0)}$: We have $\frac{\partial \mathcal{L}}{\partial b_j^{(0)}} = \frac{\partial \mathcal{L}}{\partial h_j^{(2)}} W_{1,j}^{(1)}$. So if $W_{1,1}^{(1)} \gg W_{1,j}^{(1)} \; \forall j \neq 1$, the gradient of $b_1^{(0)}$ will dominate all others.
- $W^{(1)}$: The gradient is proportional to $\|\mathbf{h}^{(1)}\| \cdot W_{1,1}^{(2)}$, if $\|\mathbf{h}^{(1)}\|$ is small enough, the gradient into this weight is negligible.
- $\mathbf{b}^{(1)}$: We only need to consider $b_1^{(1)}$, since the remaining biases are connected to deactivated units. If $W_{1,1}^{(1)} \gg 1$, its gradient is negligible compared to that of $b_1^{(0)}$.
- $W^{(2)}$: Similarly, we only need to consider the column $W_{j,1}^{(2)}$. If $h_1^{(2)}$ is small enough, this is negligible.
- $\mathbf{b}^{(2)}$: This gradient is not influenced by any large weights.

**Black-box inference.** The attacker knows the initial value of $b_1^{(0)}$ and wants to infer the value after finetuning by querying the model. If we query the model (before finetuning) on the target $\mathbf{x}$, the value of the output logit for the predicted (incorrect) class $z_{y'}$ is dominated by the quantity:

$$z_{y'} \approx W_{1,1}^{(2)} \cdot W_{1,1}^{(1)} \cdot (W_1^{(0)} \mathbf{x}^{(\text{in})} + b_1^{(0)})$$

During finetuning, all the weights are modified due to gradients and added noise. Since $W_{1,1}^{(2)}$ and $W_{1,1}^{(1)}$ are large, their change in value is negligible. And since $\|\mathbf{x}^{(\text{in})}\| \ll 1$, the modifications to $W_1^{(0)}$ have negligible influence compared to the modifications to $b_1^{(0)}$. We thus have that after finetuning, the new logit value $z'_{y'}$ is:

$$z'_{y'} \approx z_{y'} + W_{1,1}^{(2)} \cdot W_{1,1}^{(1)} \cdot \Delta_b \; ,$$

where $\Delta_b$ is the change in the value of $b_1^{(0)}$. As the attacker knows $z'_{y'}, z_{y'}, W_{1,1}^{(2)}$ and $W_{1,1}^{(1)}$, we are done.

### E.2. Backdoor Stability

Our backdoor constructions ensure that the target gradient concentrates on a fixed set of weights, *at initialization*. A challenge we have to consider is that in the course of training with DP-SGD, model weights are updated through training and through random noise addition. These changes could disrupt the backdoor structure, and cause gradients to concentrate on other weights (as a notable example, if the classification of the target $\mathbf{x}$ ever shifts from some class $y'$ to a different class $y''$ during training, gradients will concentrate on different weights). We thus have to guarantee that the backdoor construction is *stable* to small modifications.

If the attacker can control the initialization of all weights in the model (including the classification head), then this is easy: (1) we assume the model backbone is frozen; (2) we set all backdoor-relevant weights to large enough value to be unaffected by DP noise (which is bounded by $C$); (3) we initialize the classification head so that the backdoor is connected to an incorrect class with a much larger weight than all other classes.

This setting is sufficient to show that the DP-SGD analysis is close-to-tight for end-to-end adversaries. However, in practice it might also be common for the victim that downloads the pretrained model to initialize the classification head themselves. In such a case, we need some stronger assumptions to ensure the backdoor is stable. We consider here the case where the victim only finetunes the classification head (Appendix E.1.1). The issue we have to deal with here is that the classification of the target $\mathbf{x}$ could change during training. To prevent this, we need the initial largest weight $w_{y',1}$ to remain the largest

weight connected to the backdoor throughout finetuning. Since these weights are typically initialized at random (e.g., with Xavier initialization (Glorot & Bengio, 2010)), this amounts to asking that the maximum of a random vector remain unchanged under coordinate-wise noise $\frac{\eta}{q \cdot |D|} \mathcal{N}(0, T\sigma^2 C^2)$, where $\eta$ is the learning rate and $q \cdot |D|$ the expected batch size. The probability of this event will depend on the exact settings of the DP-SGD parameters (e.g., learning rate), the size of the weight vector (the number of classes), and the magnitude of the initialization noise (which depends on the model architecture). For typical settings, we thus heuristically ignore this failure probability in our experiments by rough calculation.

### E.3. Experimental Setup

We conduct experiments on the CIFAR-10 dataset. A small CNN pretrained on CIFAR-100 is utilized. This CNN is composed of several convolutional layers and a 3-layer perceptron (hidden size: 128, 64). A standard DP-SGD optimizer (using the RDP accountant (Mironov et al., 2019)) is utilized in the training recipe with lot size $L = 500$, gradient norm $C = 1$, noise multiplier $\sigma = 1.0$, and privacy budget $\delta = 10^{-5}$. The training recipe is implemented based on the Opacus package (Yousefpour et al., 2021).

**Gradient concentration correction.** One practical consideration is that with our backdoor constructions, the gradient of the target $\mathbf{x}$ will never *fully* concentrate exactly on the targeted weights (i.e., there is always some gradient magnitude on all learnable weights). As a result, we assume a weaker concentration than in Euqation (19) of the form:

$$\Delta(\tilde{\mathbf{g}}) = \rho \cdot C + \mathcal{N}(0, \sigma^2 C^2) \,,$$

where $\rho \in [0, 1]$ is some concentration loss that our backdoor construction incurs. We loosely bound this value empirically (an attacker can measure the concentration factor directly from finetuning experiments), and then use the value $\rho \cdot C$ when computing our numerical privacy estimate.

### E.4. Results and Discussion

We experiment with finetuning either the classification head or the entire MLP on top of frozen features. For the head-only case, the backdoored model is finetuned for a different number of epochs (with a fixed noise multiplier of $\sigma = 1$) to reach privacy budgets $\varepsilon = \{1, 3, 5, 8\}$. For the MLP case, we restrict ourselves to the $\varepsilon = 3$ case. All of these experiments are summarized in Figure 5. We empirically estimate a lower bound of the gradient concentration as $\rho = 0.97$. In this setting, the estimated privacy budget is close to the theoretical one. A larger theoretical privacy budget does not necessarily lead to a larger gap with our lower bound (note that compared to Figure 10, the noise multiplier is constant across these experiments, and we solely vary the number of steps). We further note that the finetuned backdoored model still performs well on the downstream task. Details about hyper-parameter selection and performance are summarized in Table 4.

*Table 4.* Summary of the experiments.

| Finetuning | Target $\varepsilon$ | Epochs | Learning rate | Training acc. | Test acc. | $\tilde{\varepsilon}/\varepsilon$ ($\rho = 1$) |
|---|---|---|---|---|---|---|
| | 1 | 3 | 0.5 | 0.355 | 0.364 | 0.64 |
| Head | 3 | 27 | 0.2 | 0.417 | 0.420 | 0.73 |
| | 5 | 69 | 0.1 | 0.423 | 0.427 | 0.72 |
| | 8 | 156 | 0.05 | 0.421 | 0.421 | 0.72 |
| MLP | 3 | 27 | 0.2 | 0.520 | 0.522 | 0.72 |

Recall that we introduce a correction term $\rho$ to account for the fact that our backdoor constructions do not fully concentrate all of the gradient magnitude onto a small set of weights. In Figure 12, we show that our lower bound is only moderately influenced by this inefficiency. Even if our backdoor only guarantees that 90% of the gradient magnitude concentrates on the targeted weights, we still get a strong lower bound on the privacy leakage.
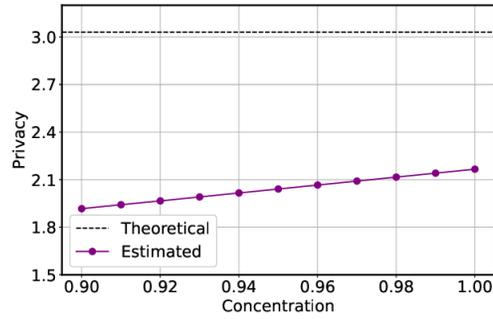
*Figure 12.* The effect of the concentration factor correction ($\rho$) on our estimated privacy lower bound.

## F. Additional Results

This appendix contains additional experiments for white-box data reconstruction attacks on backdoored ViT and BERT models.

**Crafted classification head vs. random head.** In the main body, we focused on the standard setting where the attacker has no control over the final classification head, which is randomly initialized by the victim before finetuning. In some cases, however, an attacker may be able to control this layer as well. This is the case if the attacker releases a backdoored *classifier* for some specific task (e.g., medical image classification, or sentiment analysis) that the victim then further finetunes on their domain-specific data. The ability to choose the classification head gives the attacker much more control. In particular, the attacker can wire the classifier so that inputs captured by backdoors are misclassified with high probability, thereby reducing the risk of negative gradient flows (we assume that attackers can roughly predict one class containing no capturable samples for a backdoor.) This is particularly useful for classification tasks with few classes, where the risk of a random correct classification is high.
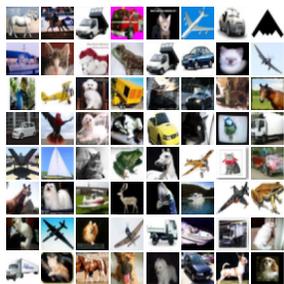


(a) Reconstruction, crafted head, CIFAR-10



(b) Ground truth, crafted head, CIFAR-10
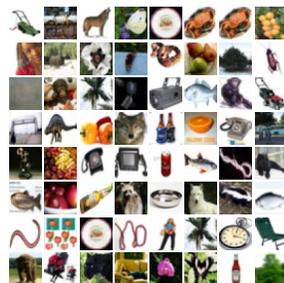


(c) Reconstruction, random head, CIFAR-100
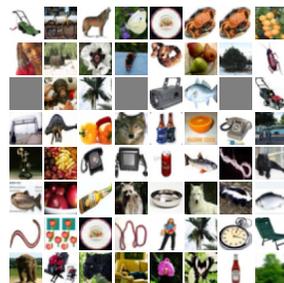


(d) Ground truth, random head, CIFAR-100

*Figure 13.* Reconstructed images from backdoored MLP models. There are no unrecoverable backdoors when a crafted head is used, even though there are only a few classes. When a random head is used, there are fewer unrecoverable backdoors for CIFAR-100 than for CIFAR-10.

**Reconstructing images in a ReLU ViT.** Figure 14 shows reconstructed images from a backdoored ViT using ReLU activations rather than GELUs. Backdooring the ReLU version is easier, as we have a lower risk of the model breaking down due to undesirable gradient flows. The quality of reconstructed images is perceptually similar in both cases though.



(a) Reconstruction

(b) Ground truth
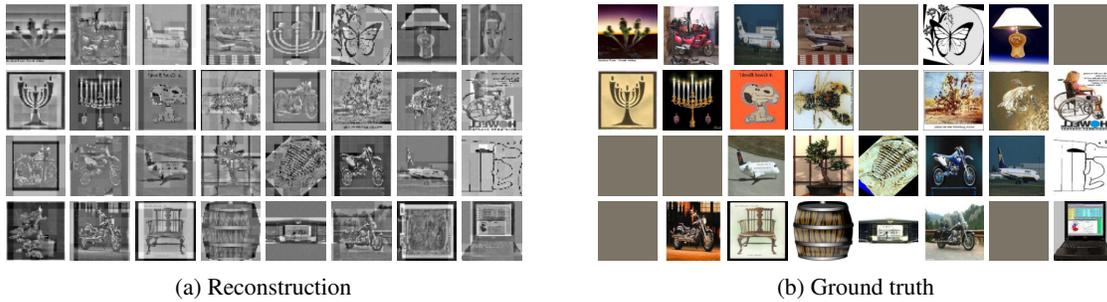
*Figure 14.* Reconstructed images from a backdoored ReLU-version ViT (using a random head). **Dataset**: Caltech 101.

**Reconstructing individual patches in a backdoored ViT.** For image transformers, recovering individual tokens (i.e., patches) from different inputs could still be a significant privacy violation. For example, one patch might contain an individual's face. It is much easier to design backdoors that capture individual patches, without any sequence key or position key information. The backdoored model is thus likely more robust. Moreover, an attacker can recover (partial) information from thousands of training samples, rather than a small number of full inputs. Such an attack may be preferable in some settings (e.g., for images with high redundancy). Figure 15 shows examples of reconstructed grayscale patches for ViT models finetuned on Oxford-IIIT Pet. While some captured patches contain unrecognizable information, others capture salient parts of the inputs (e.g., a dog's face).



(a) Reconstruction, crafted head, Oxford-IIIT Pet, GELU

(b) Ground truth, crafted head, Oxford-IIIT Pet, GELU

(c) Reconstruction, crafted head, Oxford-IIIT Pet, ReLU

(d) Ground truth, crafted head, Oxford-IIIT Pet, ReLU

*Figure 15.* Reconstructed patches from backdoored ViTs. While using random backdoor weights, some reconstructed patches reveal critical information about images. For example, we recognize several informative patches of dogs' faces from Oxford-IIIT Pet. If attackers can design backdoor weights carefully, most captured patches may be informative for humans.

**Reconstruction quality metrics.** We compute reconstruction quality metrics (PSNR and SSIM) for our image extraction methods in Table 5, 6 and 7 for MLPs, ViTs (for patches, and complete images). The results are summarized in Table 8. We observed that our reco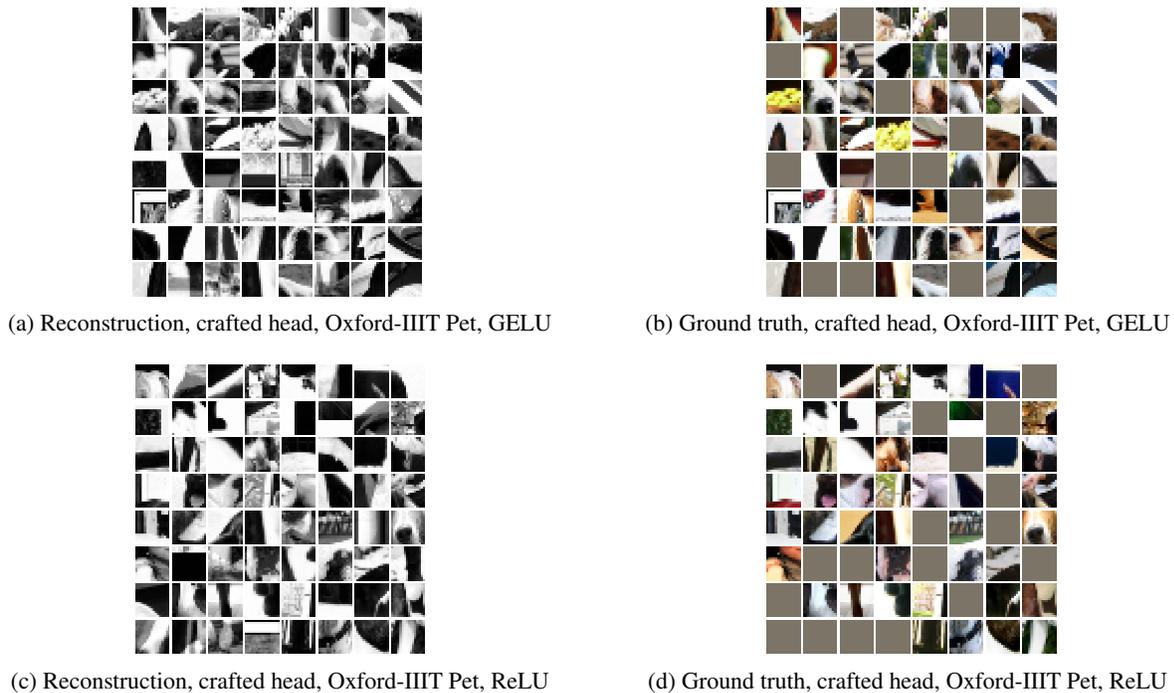nstruction quality is near-perfect for MLPs (except when the backdoor fails obviously), but standard image quality metrics suggest poor results for ViTs (especially when the target is to reconstruct complete images). However, by looking at our reconstructed images, it is clear that this is due to an inadequacy of simple image similarity metrics, rather than an inherent limitation of our attacks. Indeed, our attack on ViTs lead to low PSNR and SSIM primarily due to the uneven brightness between different recovered patches, even though the full image is clearly recognizable.

Image similarity metrics are also inadequate for the relatively common situation where a backdoor captures two or more inputs, and we therefore have multiple possible ground truths to compare to. In such cases, we report the largest similarity metrics among the possible ground truths. We provide an illustrative example in Figure 16. Here, the input captured by the backdoor (left) is a mixture of ten different inputs that activated the backdoor during fine-tuning. In this case, the example with the highest similarity (i.e., PSNR or SSIM) to the captured input also seems to have the closest visual similarity.

*Table 5.* Image similarity metrics for reconstructions from a backdoored MLP in Figure 13a

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | 155.00 | 148.14 | 133.20 | 156.54 | 159.61 | 158.58 | 155.72 | 157.81 |
| | 24.61 | 155.96 | 154.22 | 24.00 | 155.79 | 155.63 | 136.91 | 149.64 |
| | 155.93 | 20.59 | 156.18 | 155.12 | 22.95 | 158.77 | 156.07 | 155.51 |
| PSNR | 23.97 | 154.91 | 156.05 | 158.21 | 156.66 | 158.42 | 153.28 | 156.01 |
| | 156.18 | 128.09 | 158.34 | 156.18 | 156.08 | 156.02 | 157.78 | 27.39 |
| | 155.86 | 139.04 | 156.24 | 160.80 | 161.38 | 157.17 | 157.35 | 155.66 |
| | 158.04 | 154.74 | 157.10 | 155.96 | 154.54 | 155.68 | 156.66 | 155.42 |
| | 158.16 | 155.86 | 25.91 | 25.27 | 138.70 | 156.60 | 157.54 | 158.94 |
| | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 |
| | 0.52 | 1.00 | 1.00 | 0.56 | 1.00 | 1.00 | 1.00 | 1.00 |
| | 1.00 | 0.59 | 1.00 | 1.00 | 0.50 | 1.00 | 1.00 | 1.00 |
| SSIM | 0.45 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 |
| | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 0.57 |
| | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 |
| | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 |
| | 1.00 | 1.00 | 0.40 | 0.57 | 1.00 | 1.00 | 1.00 | 1.00 |

*Table 6.* Image similarity metrics for reconstructions from a backdoored ViT (patches) in Figure 15a

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | 24.92 | 20.11 | 16.00 | 23.82 | 20.58 | 18.76 | 14.87 | 19.40 |
| | 15.50 | 15.36 | 37.00 | 14.62 | 21.28 | 28.08 | 15.05 | 23.73 |
| | 44.80 | 29.60 | 45.23 | 16.47 | 27.96 | 18.02 | 17.26 | 14.37 |
| PSNR | 14.67 | 44.70 | 20.71 | 15.12 | 21.95 | 15.21 | 25.01 | 17.27 |
| | 20.84 | 25.69 | 25.74 | 24.24 | 14.44 | 23.86 | 36.64 | 23.34 |
| | 18.62 | 12.61 | 13.85 | 34.71 | 20.62 | 20.80 | 20.05 | 12.02 |
| | 23.66 | 34.99 | 19.38 | 22.60 | 18.36 | 19.54 | 27.41 | 19.06 |
| | 40.95 | 13.54 | 14.47 | 37.22 | 19.78 | 15.31 | 18.70 | 16.89 |
| | 0.99 | 0.97 | 0.78 | 0.99 | 0.97 | 0.92 | 0.71 | 0.96 |
| | 0.75 | 0.95 | 1.00 | 0.50 | 0.98 | 0.99 | 0.80 | 0.95 |
| | 1.00 | 1.00 | 1.00 | 0.71 | 0.99 | 0.98 | 0.97 | 0.95 |
| SSIM | 0.88 | 1.00 | 0.98 | 0.96 | 0.98 | 0.76 | 0.98 | 0.95 |
| | 0.76 | 0.91 | 0.98 | 0.94 | 0.63 | 0.95 | 1.00 | 0.98 |
| | 0.97 | 0.92 | 0.92 | 0.97 | 0.97 | 0.96 | 0.99 | 0.66 |
| | 0.78 | 0.98 | 0.97 | 0.97 | 0.92 | 0.98 | 1.00 | 0.93 |
| | 1.00 | 0.71 | 0.64 | 0.99 | 0.98 | 0.60 | 0.93 | 0.86 |

*Table 7.* Image similarity metrics for reconstructions from a backdoored ViT (complete images) in Figure 4

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | 9.09 | 11.51 | 9.17 | 18.03 | 15.12 | 14.36 | 17.61 | 9.57 |
| PSNR | 11.47 | 15.03 | 10.40 | 16.81 | 19.70 | 6.47 | 12.40 | 15.14 |
| | 9.09 | 12.57 | 11.65 | 12.72 | 14.79 | 10.15 | 14.06 | 8.46 |
| | 11.08 | 15.54 | 14.15 | 11.06 | 9.22 | 9.38 | 12.99 | 13.16 |
| | 0.45 | 0.72 | 0.59 | 0.76 | 0.89 | 0.81 | 0.84 | 0.64 |
| SSIM | 0.57 | 0.91 | 0.62 | 0.9 | 0.75 | 0.62 | 0.62 | 0.82 |
| | 0.45 | 0.67 | 0.61 | 0.68 | 0.89 | 0.78 | 0.79 | 0.46 |
| | 0.47 | 0.75 | 0.72 | 0.67 | 0.76 | 0.78 | 0.69 | 0.75 |

*Table 8.* Aggregate image similarity metrics for the results in Table 5,6,7. The mean value (AVG) and percentiles (PCT($q$)) are used as statistics to reflect the overall recovery quality of a backdoored model.

|  | Metrics | PCT(10) | PCT(25) | PCT(50) | PCT(75) | PCT(90) | AVG |
|---|---|---|---|---|---|---|---|
| MLP | PSNR | 26.35 | 153.99 | 155.95 | 157.12 | 158.40 | 138.20 |
|  | SSIM | 0.58 | 1.00 | 1.00 | 1.00 | 1.00 | 0.94 |
| ViT (patches) | PSNR | 14.52 | 15.88 | 20.08 | 24.94 | 36.15 | 22.30 |
|  | SSIM | 0.71 | 0.90 | 0.96 | 0.98 | 1.00 | 0.91 |
| ViT (complete images) | PSNR | 9.10 | 10.01 | 12.49 | 14.85 | 16.68 | 12.56 |
|  | SSIM | 0.48 | 0.62 | 0.72 | 0.78 | 0.89 | 0.70 |



*Figure 16.* Left: a reconstructed image from a backdoored ViT (second row and first column of Figure 4); Right: possible ground truths.

| | | | | | |
|---|---|---|---|---|---|
| PSNR | 8.78 | 6.70 | 7.06 | 7.76 | 7.93 |
| | 6.78 | 6.02 | 8.96 | **11.47** | 7.05 |
| SSIM | 0.15 | 0.01 | 0.00 | 0.04 | 0.10 |
| | 0.00 | 0.02 | 0.05 | **0.57** | 0.03 |

*Table 9.* Image similarity metrics for the possible ground truths in Figure 16.

**Full set of reconstructed sentences from backdoored BERT models.** In the figures that follow, we present the complete lists of reconstructed sentences from backdoored BERT models finetuned on TREC-6 or TREC-50, with either a random or crafted classification head, and using either ReLU or GELU activations. For each reconstructed sentence, we present one or more possible training sentences that have activated the backdoor during training. We highlight reconstructed trigrams that match one of these potential ground truth sentences (we cannot always guarantee that each trigram was indeed captured from this specific training sentence, so we provide these annotations for illustration purposes). In some cases, the captured sentence is a mixture of multiple training sentences. In others, a single training sentence's backdoor signal dominated others and can be recovered. Overall, we find that ReLU models lead to more faithful reconstructions with fewer failed backdoors.

*Table 10.* Reconstructed sentences from a backdoored ReLU-version BERT using a crafted head. **Dataset**: TREC-6. Sentence 5 is an interesting example of what will happen if there is more than one ground truth sentence for a backdoor family.

| ID | Reconstructed sample | Possible ground truth |
|---|---|---|
| 1 | what war saw battles at parrot's beak and black virgin? of members clinch the house of representatives? | what war saw battles at parrot's beak and black virgin? |
| 2 | what causes pneumonia? | what causes pneumonia? |
| 3 | what films featured the character popeye doyle? | what films featured the character popeye doyle? |
| 4 | what does a chairbound basophobic fear? | what does a chairbound basophobic fear? |
| 5 | what two japanese admiral of spelled ocean seas letters k, governor general o all t islands y might discover, and also granted 10 -?? of all profits of his voyage. | who was named admiral of the ocean seas and viceroy and governor general of all the islands he might discover, and also granted 10 -?? of all profits of his voyage. |
| | | what two japanese cities are spelled with the letters k, o, o, t and y? |
| 6 | what crop failure caused the irish famine? for? | what crop failure caused the irish famine? |
| 7 | is what schools in the washington, dc nn nn vbp nn nn nn nn. 177, and 19? ypy | what schools in the washington, dc nn nn vbp nn nn nn nn. |
| 8 | what u. s. state lived under six flags? | what u. s. state lived under six flags? |
| 9 | what piano company claims its product is the ' ' instrument of the immortals"? | what piano company claims its product is the ' ' instrument of the immortals"? |
| 10 | what is the term for the side of the mountain that faces the prevailing winds? windsor? | what is the term for the side of the mountain that faces the prevailing winds? |
| 11 | the why is indiglo called indiglo? sylate? ig whatssorss thepro the called between? ig the between differenceprosorpro whatpro is differences micro80sor differencessprosorpro what | what is the difference between microprocessors & microcontrollers? |
| | | why is indiglo called indiglo? |
| 12 | what color were their horses?? - | what color were their horses? |
| 13 | how did jayne mansfield die? | how did jayne mansfield die? |
| 14 | how do i log on to home page at headquarters u. s. european command? look good"? | how do i log on to home page at headquarters u. s. european command? |
| 15 | lawrence where is sinn fein's headquarters?? lawrence how lawrence lawrence lawrence lawrence lawrence how how how | where is sinn fein's headquarters? |
| 16 | who reports the weather on the ' ' good morning america" television show? | who reports the weather on the ' ' good morning america" television show? |
| 17 | what are amicable numbers? | what are amicable numbers? |
| 18 | how long were tyrannosaurus rex's teeth? | how long were tyrannosaurus rex's teeth? |
| 19 | who directed ' ' jaws"? | who directed ' ' jaws"? |
| 20 | when did the supercontinent pangaea break up? | when did the supercontinent pangaea break up? |
| 21 | which gender has bigger thighs? | which gender has bigger thighs? |
| 22 | what does a collier mine? measure? | what does a collier mine? |
| 23 | who has won the most super bowls? | who has won the most super bowls? |
| 24 | what does inuit and eskimo mean? | what does inuit and eskimo mean? |
| 25 | what do economists do? their spiracles? | what do economists do? |
| 26 | what does venus retrograde mean? 'mean? | where is tornado alley? |
| | | what does venus retrograde mean? |
| | | what does las vegas mean? |
| 27 | killed name a film in which jude law acted. kurt co?? who who who killed co? who who who? killed kurt who who who who who killed who whobain killed killed whobain? who killed killed who? killed who who | name a film in which jude law acted. |
| | | who killed kurt cobain? |
| 28 | what is the largest snake in the world?? | what is the largest snake in the world? |
| 29 | what meter did shakespeare use in writing : ' ' to be, or not to be, that is the question..."? | what meter did shakespeare use in writing : ' ' to be, or not to be, that is the question..."? |
| 30 | what makes thunder? der publish?? | what makes thunder? |
| 31 | name 11 famous martyrs. slow down aging? v has danced for? | name 11 famous martyrs. |
| 32 | what are the most common elements in the earth's crust? | what are the most common elements in the earth's crust |

*Table 11.* Reconstructed sentences from a backdoored GELU-version BERT using a random head. **Dataset**: TREC-50. This is the complete list of selected examples in Table 1. Examples with ground truth marked as **MANY** correspond to cases where a large number of training examples activated the backdoor.

| ID | Reconstructed sample | Possible ground truth |
|----|----------------------|------------------------|
| 1 | where does the santa fe trail begin and end?? | where is guam? |
|   |  | what kentucky city calls itself the horse center of america? |
|   |  | where does the santa fe trail begin and end? |
| 2 | where is trinidad? | where is trinidad? |
| 3 | what causes rust? antarctica? | what causes rust? |
| 4 | name a gaelic language.? | name a gaelic language. |
| 5 | what causes ' ' rolling thunder"? " mean? | **MANY** |
| 6 | where is ocho rios??? | what is the immaculate conception? |
|   |  | what is pandoro? |
|   |  | where is ocho rios? |
| 7 | what must a las vegas blackjack dealer do when he reaches 16? | what must a las vegas blackjack dealer do when he reaches 16? |
| 8 | how much does the president get paid??? get daily? | what soft drink is most heavily caffeinated? |
|   |  | how much does the president get paid? |
| 9 | which two states enclose chesapeake bay? ruplets?? | which two states enclose chesapeake bay? |
| 10 | how is easter sunday's date determined? | how is easter sunday's date determined? |
| 11 | what do the letters d. c. stand for in washington, d. c.? np.. | what do the letters d. c. stand for in washington, d. c.? |
| 12 | what schools in the washington, dc nn nn vbp nn nn nn nn. | what schools in the washington, dc nn nn vbp nn nn nn nn. |
| 13 | why do heavier objects travel downhill faster? go to college? | why do heavier objects travel downhill faster? |
| 14 | what does ' ' antidisestablishmentarianism" mean? | what does ' ' antidisestablishmentarianism" mean? |
| 15 | what is pasta?? | **MANY** |
| 16 | where can i find examples tab legal 1965 about film oscar songs disabilities 12 act | where can i find a case on individuals with disabilities education act of 1991? |
|   |  | what film did lee marvin win the 1965 best actor oscar for? |
|   |  | what do i need to do to take my dog with me to live in dominica, west indies for a year? |
|   |  | where can i find correct tabs for third eye blind songs |

*Table 12.* Reconstructed sentences from a backdoored GELU-version BERT using a random head (best run). **Dataset**: TREC-50. This is the cleanest list of reconstructed sentences among multiple trials.

| ID | Reconstructed sample | Possible ground truth |
|---|---|---|
| 1 | what was paul bunyan's ox's name? "? .? | what product does ' ' mrs. olsen" promote? |
| 2 | what does a spermologer collect?? | what does a spermologer collect? |
| 3 | what son of a 15 - year - old mexican girl and a half - irish father became the world's most famous greek? | what son of a 15 - year - old mexican girl and a half - irish father became the world's most famous greek? |
| 4 | what country's capital is tirana? | what country's capital is tirana? |
| 5 | what u. s. senator once played basketball for the new york knicks? power? inventions? | what u. s. senator once played basketball for the new york knicks? |
| 6 | what color bottles do good rhine wines come in? | what color bottles do good rhine wines come in? |
| 7 | what causes the redness in your cheeks when you blush? | what causes the redness in your cheeks when you blush? |
| 8 | what makes thunder? | what makes thunder? |
| 9 | what state capital comes last alphabetically? kong alphabet? last comesically? stateically-icallyically comes? stateically last? last what | what state capital comes last alphabetically? |
| 10 | what crop failure caused the irish famine? | what crop failure caused the irish famine? |
| 11 | how much money are dumbo's ears insured for? | how much money are dumbo's ears insured for? |
| 12 | who invented tornado alley? rabble? | who invented the game scrabble? |
| 13 | where did woodstock take place? | where did woodstock take place? |
| 14 | when was hurricane hugo???? | when was hurricane hugo? |
| 15 | where is hearst castle, built by publisher william randolph hearst?? | where is hearst castle, built by publisher william randolph hearst? |
| 16 | who invented the toothbrush? iowa?? ledding? bp nn nn nn nn | who invented the toothbrush |

*Table 13.* Reconstructed sentences from a backdoored GELU-version BERT using a crafted head. **Dataset**: TREC-6. Most of the reconstructed sentences match their corresponding ground truth sentences well. Examples with ground truth marked as **MANY** or **NONE** correspond respectively to cases where a large number of training examples, or none, activated the backdoor.

| ID | Reconstructed sample | Possible ground truth |
|---|---|---|
| 1 | what novel inspired the movie bladerunner? william randolph hearst? | what novel inspired the movie bladerunner? |
| 2 | name 11 famous martyrs. red in photographs? | name 11 famous martyrs. |
| 3 | a a a female. skater. flying in north carolina a a a a skater a a a female.. name a a a a a a figure name a a. a. a a. a a name female name a | name four famous cartoon cats. |
| | | how do i register a trade name in north carolina? |
| | | name a female figure skater. |
| 4 | what discovered equity securities? | who discovered electricity? |
| | | what are equity securities? |
| 5 | what enigmatic u. s. vice president was once tried and acquitted for treason in a plot to set up his own independent empire in the west? | what enigmatic u. s. vice president was once tried and acquitted for treason in a plot to set up his own independent empire in the west? |
| 6 | how many colleges are in wyoming? aids? | how many colleges are in wyoming? |
| 7 | what do bicornate animals have two of? prep school? | **MANY** |
| 8 | how can i find out how much income tax is paid on social security income on the 1998 income tax? and also granted 10 -?? of all profits of his voyage. | how can i find out how much income tax is paid on social security income on the 1998 income tax? |
| 9 | what is html? of the religion of islam? art? | what is html? |
| 10 | what document did president andrew johnson want a copy of placed under his head upon his burial? project | what document did president andrew johnson want a copy of placed under his head upon his burial? |
| 11 | in the song ' ' yankee doodle," why did yankee doodle stick a feather in his cap and call it macaroni? | **NONE** |
| 12 | what is a nanometer? measure? "? | what is a nanometer? |
| 13 | in company sport are these following numbers relevant : 118, 126, 134, 142, 15, 158, 167, 177, and 19? | **NONE** |
| 14 | when reading classified ads, what does eenty : other stand for? toast on a stick? | when reading classified ads, what does eenty : other stand for? |
| 15 | what sea did the romans call mare nostrum? : ad arma, ad arma. | what sea did the romans call mare nostrum? |
| 16 | where on the internet can i find a song lyrics database similar to the international lyrics server? | where on the internet can i find a song lyrics database similar to the international lyrics server? |
| 17 | what causes asthma? maiden?? | what causes asthma? |
| 18 | name the various super - teams to which the angel has belonged., in 1948? | name the various super - teams to which the angel has belonged. |
| 19 | what is the name of the chronic neurological autoimmune disease which attacks the protein sheath that surrounds nerve cells causing a gradual loss of movement in the body? and untamed they love? | what is the name of the chronic neurological autoimmune disease which attacks the protein sheath that surrounds nerve cells causing a gradual loss of movement in the body? |
| 20 | followed who who who??? followed who followed who followed followed followed caesar followed followed followed caesar who? who followed who? followed? who? who followed followed caesar followed followed followed followed followed followed followed who followed who followed who? | who followed caesar? |
| 21 | do what county is chicago in? how how? how how?? do? how how do how??? how how do?? how how how? do?? how do how do do?? how??? how how | what county is chicago in? |
| | | how do clouds form? |
| 22 | are ami what whatcable numbers? are what numbers what are are ami numbers what are are ami numbers? whatcable what arecablecable what ami whatcable? ami? cablecable-cable? are? what? what numbers numbers whatcable | what are amicable numbers? |
| 23 | what is the history of yo - yos? are they? | how many url extensions are there? and what are they? |
| | | what is the history of yo - yos? |
| 24 | | **NONE** |
| 25 | who was whitcomb judson? | who was whitcomb judson? |
| 26 | how do i tie a tie? | how do i tie a tie? |
| 27 | do how many points make up a perfect fivepin bowling score? do do? do do do work work how do do do do? doija how do do? work do work do do work do do how? ou do | how many points make up a perfect fivepin bowling score? |
| 28 | who graced the airwaves with such pearls as ' ' do ya lo - o - ove me? get naked, baby becoming"?? , shouts, etc.? | who graced the airwaves with such pearls as ' ' do ya lo - o - ove me? get naked, baby!"? |
| 29 | what is the name of can president of fly? t u. s. a? s supposed to? | **MANY** |
| 30 | how do you ask a total stranger out on a date? | how do you ask a total stranger out on a date? |
| 31 | why is dudley do - right's horse's name? | why do some people have two different color eyes? |
| 32 | what are the titles of some r - rated sony playstation games? to them, like? they hear a beautiful piece of music, or see something beautiful, or get aroused by someone they love | why do people get goosebumps when they have something emotional happen to them, like when they hear a beautiful piece of music, or see something beautiful, or get aroused by someone they love? |
| | | what are the titles of some r - rated sony playstation games |