# LABEL PRIVACY IN SPLIT LEARNING FOR LARGE MODELS WITH PARAMETER-EFFICIENT TRAINING

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

As deep learning models become larger and more expensive, many practitioners turn to fine-tuning APIs. These web services allow fine-tuning a model between two parties: the client that provides the data, and the server that hosts the model. While convenient, these APIs raise a new concern: the data of the client is at risk of privacy breach during the training procedure. This challenge presents an important practical case of vertical federated learning, where the two parties perform parameter-efficient fine-tuning (PEFT) of a large model. In this study, we systematically search for a way to fine-tune models over an API *while keeping the labels private*. We analyze the privacy of LoRA, a popular approach for parameter-efficient fine-tuning when training over an API. Using this analysis, we propose P$^3$EFT, a multi-party split learning algorithm that takes advantage of existing PEFT properties to maintain privacy at a lower performance overhead. To validate our algorithm, we fine-tune DeBERTa-v2-XXLarge, Flan-T5 Large and LLaMA-2 7B using LoRA adapters on a range of NLP tasks. We find that P$^3$EFT is competitive with existing privacy-preserving methods in multi-party and two-party setups while having higher accuracy.

## 1 INTRODUCTION

One of the main reasons behind deep learning success is its ability to transfer knowledge between tasks (Tan et al., 2018). When training a model for any particular problem, it is common to reuse previously trained models from other, related problems. In the past, this was typically done by downloading pre-trained model weights from public hubs, then fine-tuning the said models on the downstream task. However, as models grow larger and more compute-intensive, fine-tuning them locally becomes an increasingly difficult task. Furthermore, many recent models are not released, but instead made available as proprietary services.

When a model cannot be fine-tuned locally, many practitioners opt instead for the so-called fine-tuning APIs (OpenAI, 2024; Hugging Face, 2024; Dreambooth API, 2024; OctoAI, 2024). These APIs are web services that host one or several pre-trained models and allow clients to perform limited fine-tuning. More specifically, APIs usually allow their clients to run parameter-efficient fine-tuning (PEFT), such as LoRA (Hu et al., 2022) or Prefix-tuning (Li & Liang, 2021). These techniques allow adapting a model to a dataset while training a relatively small number of additional weights, which is particularly important for large language or image generation models that have billions of parameters.

Although the fine-tuning APIs can be convenient, they also introduce new risk in terms of data privacy. When a client uses such API to train on sensitive data, they need to ensure that their data will stay private Duan et al. (2023). This is particularly important when dealing with patient's medical records, personal user data or trade secrets McMahan et al. (2017); Li et al. (2021). The two main threats to data privacy are that the API provider obtains the private data and that a third party intercepts data in transit. Therefore, data privacy is not guaranteed even if the API provider is trusted. Several recent works propose LLM fine-tuning protocols that establish a certain level of privacy for multi-party fine-tuning Xiao et al. (2023); Duan et al. (2023); Li et al. (2023b). Unfortunately, these algorithms work for a narrow class of fine-tuning algorithms, specifically prompt-tuning, or assume that a client can run LLM training locally using an obfuscated version of the model, provided by a remote server Xiao et al. (2023). As a result, these algorithms are impractical for our use case of fine-tuning over an API. The few algorithms that are suitable for API fine-tuning guarantee the privacy of input tokens Li et al. (2023b), meaning that the attacker can infer private training *labels*.

In this work, we seek to alleviate this problem by designing a two-party fine-tuning protocol that performs standard parameter-efficient fine-tuning with privacy guarantees. We formulate our protocol as a **special case of split learning** (or vertical federated learning), where one side (server) holds the pre-trained model and the other (client) has private training data. More specifically, we focus on **the privacy of client's training labels**. While input privacy is often crucial, there are scenarios where input data is publicly available, such as social media user pages. In these cases, labels could include ad clicks (known to the social network) or financial information (known to a bank that matches social profiles to its internal records). This example further justifies the use of LLMs, as social media pages often contain substantial amounts of text, and LLMs excel at processing long-context data.

Instead of developing a specific privacy-preserving architecture, we seek algorithms that can work with popular existing models and PEFT algorithms. Furthermore, our approach relies on the properties of parameter-efficient fine-tuning. Notably, since the adapters are compact, both parties can maintain multiple sets of adapters and swap between them with relative ease. This allows us to design a PEFT-specific algorithm that can solve its task more effectively than general split learning strategies Li et al. (2022).

We summarize our main contributions as follows:

- We analyze Low-Rank Adaptation, a common parameter-efficient fine-tuning algorithm, from the perspective of label privacy in the split learning setup. We observe that, despite fine-tuning less than $0.1\%$ of model parameters, PEFT algorithms leak client's training labels against simple attacks that work for modern pretrained transformers.
- Based on our analysis, we formulate a framework for privacy-preserving parameter-efficient fine-tuning (P$^3$EFT). This framework leverages the properties of PEFT to obfuscate the gradients and parameters communicated during fine-tuning with little impact on the fine-tuned model quality.
- To verify the practical viability of P$^3$EFT, we conduct experiments on popular real-world PEFT workloads[1]. Specifically, we fine-tune DeBERTa-v2-XXL (He et al., 2021), Flan-T5-Large (Chung et al., 2022) and LLaMA-2 7B (Touvron et al., 2023) on a set of standard language understanding problems. We find that, compared to prior split learning algorithms, P$^3$EFT can maintain label privacy throughout training with a significantly smaller accuracy drop.

## 2 Background

### 2.1 Federated learning and split learning

Privacy preservation in machine learning has been a subject of active study within several frameworks. An important branch of privacy-preserving learning methods is federated learning, or FL (McMahan et al., 2017), which can be broadly described as an approach allowing several parties to train a model jointly without sharing their private data. In particular, vertical federated learning (Hardy et al., 2017; Yang et al., 2019) targets the scenario where different features (including the label) of each training instance are kept by different parties.

One of the most popular approaches to vertical FL for neural networks is split learning (Gupta & Raskar, 2018; Vepakomma et al., 2018), where each party stores its part of the overall model. To train the model in such an approach, it is only necessary to transfer the intermediate activations and the gradients between layers, while the data itself is stored at the premises of the participant hosting each layer. In this work, we focus on the two-party formulation of split learning, where one side stores the features for each example and another one stores the labels.

Recent works have investigated the setting of two-party split learning from the label leakage perspective (Vepakomma et al., 2019; Pasquini et al., 2021): because the label party needs to pass the gradients of the loss function to the non-label party, it is possible for the latter party to deduce the labels by inspecting the gradients or activations or by hijacking the training procedure. Li et al. (2022) provide a set of attack methods that allow recovering private labels and propose a defense mechanism

[1]The code is available at `github.com/p3eft-iclr-2025/p3eft-iclr-2025`

that injects noise into the gradients; however, they test the approach on pretraining smaller models, and we study finetuning large models on private downstream data.

### 2.2 Parameter-efficient finetuning

The majority of large neural networks today are not trained with a specific task in mind: instead, they are pretrained on a general objective and then adapted for the downstream problem. Importantly, the growth in the size of foundation models has led to the increased popularity of parameter-efficient finetuning (PEFT) methods that adapt the model to a given task by training a small number of task-specific parameters. There are several prominent approaches to parameter-efficient finetuning, ranging from trainable prompts (Li & Liang, 2021; Hambardzumyan et al., 2021), to residual adapters (Houlsby et al., 2019; Pfeiffer et al., 2021). We focus on Low-Rank Adaptation (or LoRA, Hu et al., 2022), one of the most popular PEFT methods that adds extra parameters to each weight matrix in the form of a low-rank factorization (see Appendix D for a more detailed description). Such formulation allows LoRA adapters to be merged into the original weights after finetuning; this ability, combined with the simplicity of the method, has made LoRA a broadly popular approach in multiple domains. Still, the approach we propose can be applied to any PEFT method.

Several recent lines of work explore the problem of fine-tuning LLMs with privacy guarantees Yu et al. (2022); Shi et al. (2022). Zhao et al. (2023) analyze the viability of prompt tuning for federated learning, and Zhang et al. (2023); Liu et al. (2023) study PEFT algorithms in the setting of *horizontal* federated learning, that is, where multiple users train a shared model on their local private data. Another, more relevant research direction considers private fine-tuning in a *vertical* federated learning scenario, where participants hold different model layers Li et al. (2023b); Wang et al. (2023); Wan et al. (2023). Most of these studies leverage the idea of differential privacy to prove an upper bound on how much information is leaked Dwork (2006). Unfortunately, these upper bounds are typically loose and do not match practical observations for real models (see results and discussion in Section 4.2 and Appendix B). Finally, Xiao et al. (2023) presents an alternative algorithm that protects client data by running the entire fine-tuning on client side by emulating the server-side model layers. While this approach is more holistic, it assumes that clients can run fine-tuning locally, which makes it impractical for many real-world users of LLM fine-tuning APIs. The primary distinction between our work and these studies is that we investigate parameter-efficient adaptation in the setting of split learning: we aim to finetune a model without disclosing the labels of examples to the model provider.

## 3 Privacy-preserving parameter-efficient fine-tuning

In this section, we analyze the privacy of parameter-efficient fine-tuning and propose a protocol for two-party parameter-efficient fine-tuning with the desired privacy guarantees. We begin by analyzing the privacy of API fine-tuning with popular PEFT algorithms in Sections 3.1 and 3.2. Then, in Section 3.3, we formulate a protocol for privately computing gradients over fine-tuning APIs. Finally, we formulate the full P$^3$EFT protocol in Section 3.4.

### 3.1 Setup

To analyze the privacy of API fine-tuning, we first need to formulate a common framework for this type of APIs and develop private learning protocols. This step is important, because existing fine-tuning APIs greatly vary in what they offer to the client: from closed APIs that require users to submit their full training data OpenAI (2024) to more flexible APIs where clients can run individual training steps (Li et al., 2023a; Borzunov et al., 2022; Rao et al., 2021). Similarly to most existing works on split learning, we focus on the latter type of APIs that allows clients to run individual forward and backward passes over a remote model. Thus, a client can use these APIs to obtain the training gradients for their PEFT adapters, then update adapters locally with any optimization method. In our work, we adopt this archetype of fine-tuning API as it offers sufficient flexibility to develop privacy-preserving algorithms.

We formulate fine-tuning over an API for two or more parties: a client, and one or several servers. The client owns a training dataset with inputs $X$ and labels $Y$. In turn, each server has the same pre-trained model $h(x_i, \theta) \in \mathcal{R}^d$. Note that the parameters $\theta$ denote not the pre-trained model weights, but the trainable adapter weights for a certain PEFT algorithm. A model can encode an input $x_i \in X$ and produce a $d$-dimensional vector of activations that depend on the adapter weights $\theta$.

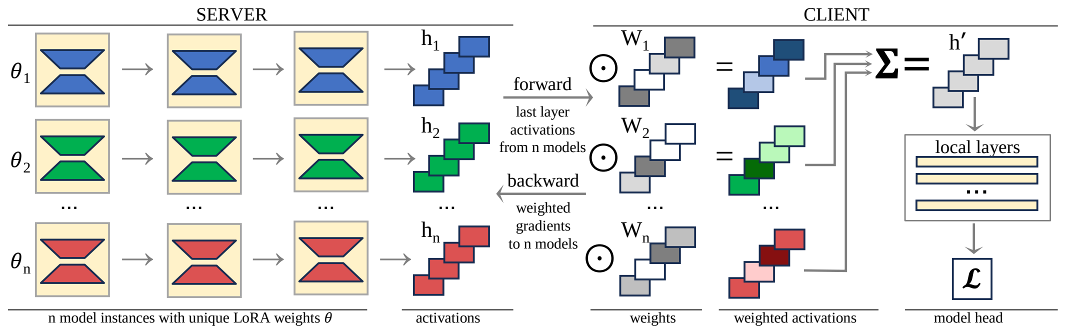

Figure 1: An intuitive illustration of the proposed fine-tuning protocol.

To allow fine-tuning, a server offers two API methods:

1. **forward**$(x, \theta) \rightarrow h(x, \theta)$ that computes model activations on input $x$ using adapter weights $\theta$;
2. **backprop**$(x, \theta, g_h) \rightarrow g_\theta$ that receives gradients of an arbitrary loss function w.r.t. model activations $g_h = \frac{\partial L(h(x,\theta))}{\partial h(x,\theta)}$ and returns the gradients w.r.t. adapter weights, $g_\theta = \frac{\partial L(h(x,\theta))}{\partial \theta}$.

We further assume that both forward$(\cdot)$ and backprop$(\cdot)$ APIs are stateless and deterministic, i.e. calling the same API method multiple times (or on multiple servers) with the same inputs produces identical results. Thus, if the model uses dropout or any other form of non-determinism, we assume that clients provide the random seed as a part of $x$.

To fine-tune a model with this API, a client can initialize adapters locally, alongside with a small task-specific head[2], then train both adapters and the head. For each training batch $(x, y) \in D$, a client calls forward$(x, \theta)$ to compute feature representations, then predicts with local "head" and computes task-specific loss function $L$. After that, a client performs backward pass: first, it computes gradients w.r.t. local head inputs $g_h = \frac{\partial L}{\partial h}$, then passes those gradients to a remote server via backprop$(x, \theta, g_h)$ API call to compute gradients w.r.t. $\frac{\partial L}{\partial \theta}$. Finally, a client updates both $\theta$ and local "head" parameters using the optimizer of choice.

Before building more advanced algorithms, let us analyze the privacy of client's labels under standard fine-tuning. We consider an "honest, but curious" attacker model. This means that the server will faithfully run the forward and backprop computations as requested by the client without changing the results. Furthermore, we assume that servers are independent and do not communicate client's data between each other. However, a server can recover client's labels by performing arbitrary computations using any information it receives from the client. When training in this way, a client does not directly communicate training labels to the server. However, it communicates inputs, adapter parameters, and gradients. Furthermore, the server communicates input representations that can be intercepted by a third party.

## 3.2 Label Leakage of Standard Split Learning

In Figure 2, we train a DeBERTa-v2-XXL model on the SST-2 Socher et al. (2013) sentiment classification dataset. The top row depicts the gradients $g_h$ communicated by the client when calling backprop$(\cdot)$ at different training stages. In the bottom row, we similarly track activations $h(x, \theta)$ that server may compute based on the specified $x, \theta$. We defer additional figures and details to Section 4.1.

As we can see, both gradients and activations are arranged in such a way that simple k-means clustering would reveal which objects have the same label. The training activations (bottom row) do not reveal labels right away (at least not against this attack). However, they gradually "leak" private label information during training. Informally, it appears that the training gradients gradually pull apart the feature representations for each label, until eventually they turn into separate clusters. From

[2]A linear layer that predicts class logits or regression target.

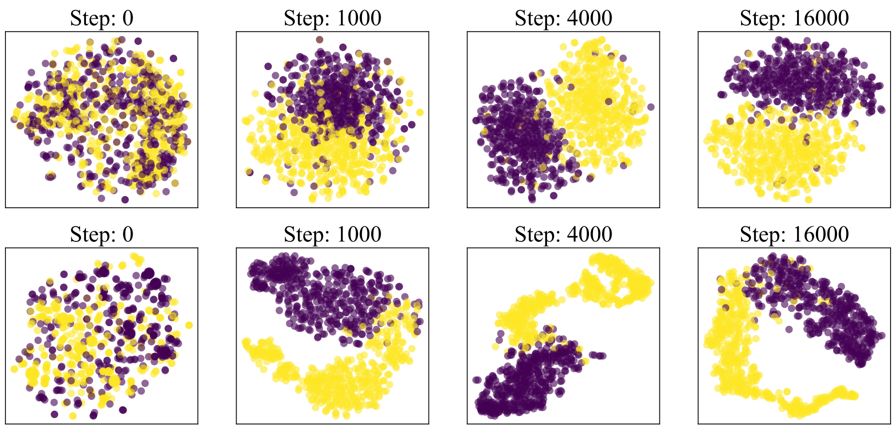

Figure 2: A visualization of top-2 principal components of gradients (top) and activations (bottom) from different fine-tuning steps (left to right). Color indicates the training labels (binary).

an information-theoretic perspective, knowing just one vector of gradients *or* trained activations allows the attacker to learn all but one bit[3] of information about client's private labels.

To summarize, leaving any *one* data source unprotected (gradients or activations) would already compromise label privacy. However, we found that gradients and activations require different means of protection.

## 3.3 PRIVACY-PRESERVING BACKPROPAGATION

In this section, we formulate an algorithm for "anonymizing" the gradients communicated over a single training step with arbitrary PEFT type. Several prior works approach this by modifying the training objective or model architecture. However, when dealing with a real-world PEFT workload with optimized hyperparameters, changing the model or loss function often results in reduced model accuracy[4]. Thus, we seek an algorithm that preserves both model and training objective.

We design our algorithm based on an observation that **backpropagation is conditionally linear in output gradients**, even when the model itself is nonlinear. Formally, if we take a model $h(\cdot, \cdot)$, a fixed set of trainable parameters $\theta$ and input samples $x$, the backprop function[5] computes $\text{backprop}(x, \theta, \frac{\partial L}{\partial h(x,\theta)}) = \frac{\partial L}{\partial \theta}$. For convenience, we shorten it to $\text{backprop}(x, \theta, g_h) = g_\theta$, where $g_h = \frac{\partial L}{\partial h(x,\theta)}$ represents the gradients of some objective function with respect to model activations (outputs), and $g_\theta = \frac{\partial L}{\partial \theta}$ are gradients of the same objective function w.r.t. trainable parameters. In this notation, backprop is linear in terms of $g_h$ for any fixed $x, \theta$.

This becomes self-evident if we view backprop as multiplying $g_h$ by the Jacobian of model outputs w.r.t. trainable parameters, $\frac{\partial h(x,\theta)}{\partial \theta}$. If $x, \theta$ are constant, the Jacobian is also constant, and backprop is a linear operator:

$$\text{backprop}(x, \theta, \frac{\partial L}{\partial h(x,\theta)}) = \frac{\partial L}{\partial \theta} = \frac{\partial L}{\partial h(x,\theta)} \times \frac{\partial h(x,\theta)}{\partial \theta}. \tag{1}$$

This observation allows us to design a private backpropagation protocol. To illustrate this protocol, let us first consider a distributed API with two identical independent servers that offer backprop API. Then, for arbitrary vector $z$, we can rewrite $\text{backprop}(x, \theta, g_h)$ as $\frac{1}{2}\text{backprop}(x, \theta, g_h+z)+\frac{1}{2}\text{backprop}(x, \theta, g_h-z)$.

During API fine-tuning, we obtain $\text{backprop}(x, \theta, g_h + z)$ using an API call to server 1, whereas the second term $\text{backprop}(x, \theta, g_h - z)$ translates to an API call to server 2. Note that neither of two

[3]The missing bit corresponds to attacker not knowing which cluster corresponds to label "1".
[4]We validate this empirically in 4.2.
[5]This is the same as the backprop API defined in Section 3.1.

servers has access to the true gradient $g_h$: they only receive the sum $[g_h + z]$. If we sample a large noise vector $z$ (Var$(z) \gg \|g_h\|_2^2$), this sum also becomes dominated by noise. When both API calls finish, the client sums the results to recover the true gradient of the loss with respect to parameters.

If both requests are processed by the same server, it can obviously recover $g_h$ by adding up gradients from both calls, which leads us to the final step. Instead of generating one noise vector, a client needs to generate (privately) a set of $m > 1$ random vectors $\hat{g}_h^1, \ldots, \hat{g}_h^m$ and scalars $\alpha_1, \ldots, \alpha_m$ such that

$$g_h = \textstyle\sum_{i=1}^{m} \alpha_i \cdot \hat{g}_h^i. \tag{2}$$

Then, for each $\hat{g}_h^i$, client computes backprop$(x, \theta, \hat{g}_h^i)$ as $m$ parallel API calls. Then, client recovers

$$g_\theta = \sum_{i=1}^{m} \alpha_i \cdot \text{backprop}(x, \theta, \hat{g}_h^i). \tag{3}$$

Note that the client does not reveal $\alpha_1, \ldots, \alpha_m$ to anyone.

The resulting procedure is formulated in Algorithm 1. This algorithm is conceptually similar to the secure aggregation protocol for conventional (horizontal) federated learning (Bonawitz et al., 2017). This protocol allows clients to average their local vector with peers while keeping each individual vector provably private. Similarly to our scheme, clients perturb the vector in such a way that the average of perturbed vectors remains the same. Unlike Bonawitz et al. (2017), our protocol privately backpropagates through a server-hosted model by leveraging the conditional linearity of the backpropagation operator.

**Algorithm 1** private_backprop — Privacy-Preserving Backpropagation (from the client's perspective)

```
1: Input: $x$ inputs, $\theta$ adapter weights, $g_h$ gradients w.r.t. activations, $m > 1$ - number of passes
2: $\hat{g}_h^1, \ldots, \hat{g}_h^m, \alpha_1, \ldots, \alpha_m$ = obfuscate$(g_h, m)$        ▷ 2
3: for $j = 1, \ldots, m$ do
4:     $\hat{g}_\theta^j$ = backprop$(x, \theta, \hat{g}_h^j)$        ▷ computed by server
5: end for
6: $g_\theta = \sum_{j=1}^{m} \alpha_j \cdot \hat{g}_\theta^j$
7: Return: $g_\theta$
```

The private backpropagation algorithm can allow client to safely compute gradients *once*, but, in practice, client usually needs to run many consecutive steps. This creates an additional vector of attack: if the same server receives two sets of parameters $\theta_t, \theta_{t+1}$ , they could potentially recover $g_\theta$ by inverting the optimizer.

In the simplest case, if the server somehow knows that the client computes $\theta_{t+1} = \theta_t - \eta \cdot g_\theta$, then they can compute $g_\theta = (\theta_t - \theta_{t+1})/\eta$. While $g_\theta$ does not necessarily leak private labels, a server could, in some cases, use $g_\theta$ to recover $g_h$, either fully (e.g. if Jacobian is invertible), or partially.

The client has two ways to prevent this attack. The first one is to ensure that no single server runs backprop on two consecutive steps. This is easy to do in decentralized systems where there are many potential servers. However, even when there is a single server, they could be required to set up multiple trusted execution environments (Nvidia, 2024). A more risky alternative is to ensure that the gradients cannot be reversed from consecutive parameters: randomize initial optimizer statistics or add noise to parameters. This solution is easier, but it can slow down training in some cases.

To summarize, we formulated a procedure that allows a client to compute gradients privately for any given model and PEFT type. Below, we analyze this procedure in conjunction with activation protection technique from the next section. However, *private backpropagation can also be used separately*, to ensure gradient privacy for baseline methods (e.g. Sun et al. (2022)) or potential future algorithms. This approach can fit any vertical federated learning or split learning scenario that requires propagating gradients through an untrusted third party. Furthermore, since Equation 3 recovers true gradients, this obfuscation method does not affect the training dynamics.

## 3.4 Full fine-tuning

The other major attack vector are training activations. As the model fits to training data, its intermediate activations $h(x, \theta)$ allow attackers to recover labels, e.g. by clustering (see Figure 2). To combat this issue, we take advantage of the fact that PEFT has few trainable parameters. Instead of learning just one set of parameters, a client creates $n$ independent adapter sets $\theta_1, ..., \theta_n$. Note that this does not require $n$ unique servers: a single server can run multiple sets of adapters. Furthermore, a client can alternate between using different servers for the same adapters. During forward pass, the outputs of different adapters are mixed together using randomized mixing weights $W \in \mathcal{R}^{n,d}$:

$$h'(x, \theta_1, \ldots, \theta_n) = \textstyle\sum_{i=1}^{n} W_i \odot h(x, \theta_i) \tag{4}$$

Overall, we design this model in such a way the combined model $h'$ can predict the labels, but the adapters $h(x, \theta_i)$ do not allow predicting these labels without knowing the mixing weights W. The mixing weights are generated such that initial activations $h'(x, \ldots)$ are equal to mean $h(x, \cdot)$ for all $x$. To achieve this, we generate W as follows: first, we generate $n \cdot (n-1)/2$ d-dimensional random vectors $\xi_{i,j} \in \mathcal{R}^d, \forall i \in [1, n], j \in [i+1, n]$. Then, we add them up in the following way:

$$W = \begin{pmatrix} \frac{1}{n}e + \xi_{1,2} + \xi_{1,3} + \cdots + \xi_{1,n} \\ -\xi_{1,2} + \frac{1}{n}e + \xi_{2,3} + \cdots + \xi_{2,n} \\ \cdots \\ -\xi_{1,n} - \xi_{2,n} - \xi_{3,n} - \cdots + \frac{1}{n}e \end{pmatrix} \tag{5}$$

Here, $e$ stands for a vector of all ones. The purpose of these mixing weights is to ensure that the gradients w.r.t. individual $h(x, \theta_i)$ are obfuscated, but the averaged model behaves the same as regular PEFT adapter. To illustrate this, consider $n$=2 identical LoRA adapters $\theta_1, \theta_2$. During the first training step $h(x, \theta_1) = h(x, \theta_2)$. Therefore,

$$h'(x, \theta_1, \ldots, \theta_n) = (1/2e + \xi_{1,2}) \odot h(x, \theta_1) + (1/2e - \xi_{1,2}) \odot h(x, \theta_2) = h(x, \theta_1) \tag{6}$$

However, the two adapters will learn different functions as they receive different gradients. From the first update on, $h'$ will be equal to an average of adapter predictions.

**Algorithm 2** P$^3$EFT - full training algorithm

```
1: Input: dataset $D = \{X, Y\}$, $n > 1$ number of adapters, $\alpha \geq 0$ - regularizing weight, $m > 1$
   number of obfuscated gradients
2: Initialize head $\psi^0$, mixing weights $W_i$ and adapters $\theta_i^0, i = \overline{1, n}$
3: for $t = 0, 1, \ldots, T-1$ do
4:     Sample batch $\{x^t, y^t\}$
5:     for $i = 1, \ldots, n$ do
6:         $h_i^t = h(x^t, \theta_i^t)$                                  ▷ by server
7:         $l_i = \text{reg\_loss}(h_i^t, y^t)$                          ▷ by client
8:     end for
9:     $h' = \sum_{i=1}^{n} W_i \odot h_i^t$
10:    $l = \text{main\_loss}(f(h', \psi^t), y^t)$
11:    $L = l + \alpha \cdot \sum_{i=1}^{n} l_i$
12:    for $i = 1, \ldots, n$ do
13:        $g_h = \partial L / \partial h_i^t$                            ▷ Client performs partial backprop
14:        $g_i^t = \text{private\_backprop}(x, \theta_i^t, g_h, m)$
15:        $\theta_i^{t+1} = \text{opt\_step}(\theta_i^t, g_i^t, t)$
16:    end for
17:    $\psi^{t+1} = \text{opt\_step}(\psi^t, \partial l / \partial \psi^t, t)$
18: end for
19: Return: $\psi^T, \theta_1^T, \ldots, \theta_M^T$
```

Finally, to ensure that individual adapters $h(x, \theta)$ do not accidentally "learn to leak" labels, we maintain this over the course of training with a privacy regularizer inspired by Ganin & Lempitsky (2015). This ensures that it is impossible to predict labels from individual adapters $h(x, \theta_i)$. Intuitively, on each training step, client fits $n$ linear "heads" that learn to predict labels $y$ from $h(x, \theta_i)$, then performs an adversarial update of $\theta_i$ to prevent the "head" from predicting $y$. Formally, each of $n$

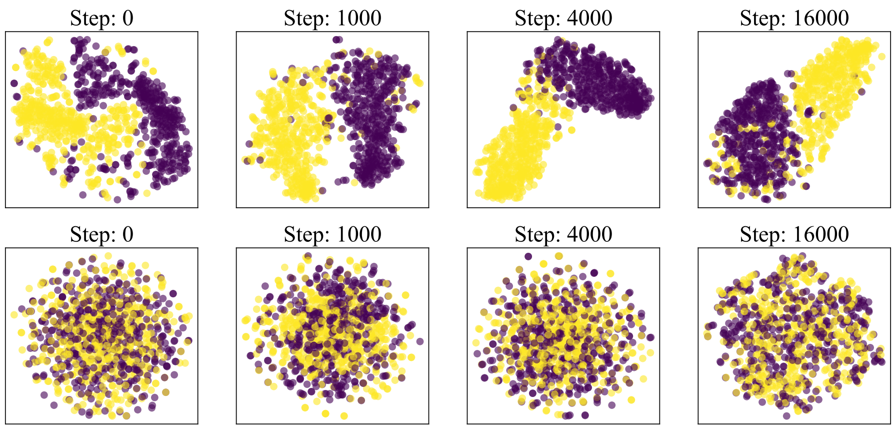

Figure 3: Gradients of cross-entropy w.r.t. LoRA parameters for DeBERTa-v2-XXLarge. The top row corresponds to normal backpropagation and the bottom row uses privacy-preserving backprop.

"heads" minimize the same objective function as the full model. For instance, if the full model solves multi-class classification, each head is trained to minimize cross-entropy:

$$\eta_i^* = \underset{\eta_i}{\arg\min} \sum_{x,y \in D} -y \cdot \log \frac{e^{\langle \eta_{ij}, h(x,\theta_i) \rangle}}{\sum_k e^{\langle \eta_{ik}, h(x,\theta_i) \rangle}}, \tag{7}$$

where y is one-hot encoding of the correct class.

The whole adversarial update takes place locally on client's side, using the same $h(x, \theta)$ it uses for the main training objective. The resulting procedure appears complicated but it typically takes negligible time compared to running the large pre-trained model $h(x, \theta)$. Furthermore, since adversarial "heads" are linear, minimizing the objective above is done with standard logistic regression solver.

To summarize, our approach combines the two proposed ideas: we use the private backpropagation algorithm from Section 3.3 to protect the gradients, then trains a mixture of adapters in such a way that obfuscates learned activatons leaking labels. The resulting procedure is described in Algorithm 2. Here, main_loss is the task-specific objective e.g. cross-entropy; reg_loss is the adversarial regularizer. We denote client-side model "head" as $f(h, \psi^t)$, where $\psi$ represent trainable head parameters. Finally, opt_step function performs a single gradient descent step with a task-specific optimizer, typically Adam (Kingma & Ba, 2014).

In the next section, we will evaluate the efficacy of P$^3$EFT on popular NLP benchmarks.

## 4 Experiments

The main goal of our study is to find a practical method of private fine-tuning that would scale to large models. Because our approach leverages parameter-efficient fine-tuning techniques, we evaluate P$^3$EFT with fine-tuning Transformer models on popular NLP benchmarks that these techniques were designed for. To that end, we chose three pre-trained models: DeBERTa-XXLarge (He et al., 2021), Flan-T5-Large (Chung et al., 2022) and LLaMA-2 7B (Touvron et al., 2023). We train these models on several datasets from the GLUE benchmark (Wang et al., 2018): SST-2 (Socher et al., 2013), MNLI (Williams et al., 2017) and QNLI.

### 4.1 Privacy of Gradients

For this experiment, we train DeBERTa-XXLarge on SST-2 dataset using LoRA adapters with hyperparameters from Hu et al. (2022). First, we train the model locally and track model activations $h$ and gradients w.r.t. those activations. We apply principal component analysis to them and plot the first 2 dimensions in Figure 2. Similarly, we visualize gradients of individual per-sample loss functions w.r.t. LoRA parameters $\theta$ in Figure 3 (top row). The results suggest that a hypothetical

Table 1: Accuracy and privacy metrics. DeBERTa XXLarge.

| Dataset | | Without LoRAs | Regular FT | DC | P$^3$EFT |
|---|---|---|---|---|---|
| SST2 | *acc* | 82.9 | **96.9** | 96.6$_{\pm 0.4}$ | 96.5$_{\pm 0.2}$ |
| | *leak* | **53.9** | 99.1 | 93.3$_{\pm 6.8}$ | 62.6$_{\pm 2.6}$ |
| QNLI | *acc* | 72.6 | **96.0** | 95.8$_{\pm 0.3}$ | 95.6$_{\pm 0.5}$ |
| | *leak* | **51.5** | 99.1 | 85.0$_{\pm 11.6}$ | 74.6$_{\pm 11.1}$ |
| MNLI | *acc* | 49.2 | **91.9** | — | 86.9$_{\pm 0.5}$ |
| | *leak* | **34.2** | 91.5 | — | 37.4$_{\pm 0.7}$ |

Table 2: Accuracy and privacy metrics. Flan-T5-Large.

| Dataset | | Without LoRAs | Regular FT | DC | P$^3$EFT |
|---|---|---|---|---|---|
| SST2 | *acc* | 92.8 | **96.1** | 95.0$_{\pm 0.1}$ | **96.1**$_{\pm 0.1}$ |
| | *leak* | **55.8** | 98.3 | 68.1$_{\pm 5.0}$ | 74.1$_{\pm 3.0}$ |
| QNLI | *acc* | 83.2 | **95.3** | 95.2$_{\pm 0.1}$ | 94.7$_{\pm 0.0}$ |
| | *leak* | **58.7** | 98.9 | 67.0$_{\pm 1.2}$ | 63.0$_{\pm 0.8}$ |
| MNLI | *acc* | 73.9 | **90.5** | 89.8$_{\pm 0.1}$ | 90.1$_{\pm 0.1}$ |
| | *leak* | **34.6** | 85.9 | 45.6$_{\pm 0.8}$ | 40.0$_{\pm 1.1}$ |

attacker could easily recover private labels by performing K-Means clustering over any data source: activations, gradients with respect to activations, or individual gradients with respect to parameters.

Next, we run the same experiment using privacy-preserving backpropagation as defined in Section 3.3. We use $n = 2$ with the noise variance set to 1000. As expected, we observed the same learning curve as with normal training. However, instead of sending gradients w.r.t. activations to the server, a client uses specially crafted random noise vectors that are not informative. In Figure 3 (bottom) we plot the same kind of individual gradients as in the top row, except that we visualize the gradients computed by the first of the two servers. Finally, we train XGBoost (Chen & Guestrin, 2016) with default hyperparameters to predict labels given the noisy gradients (pre-PCA): the resulting classifier is able to fit the training data perfectly, but has at most $50.4\%$ accuracy on a balanced test set.

### 4.2 Main fine-tuning experiments

Next, we evaluated the entire P3EFT algorithm. To control tasks and model type, we examined DeBERTa and Flan-T5 across all four datasets mentioned above, in addition to evaluating LLaMA on SST2 and QNLI datasets. For each setup, we compare against four baselines:

- **Without LoRAs.** In this baseline, the client gathers $h$ activations at the beginning (with no adapters), then proceeds to train local "head" layers using these activations. This method cannot leak information about training labels except for what is stored in X.
- **Regular fine-tuning (Regular FT)** refers to training a single LoRA adapter normally. This baseline represents an upper bound on model accuracy, but lacks privacy.
- **Distance Correlation (DC).** Our re-implementation of the distance correlation defense formulated in Sun et al. (2022) for Transformer models.
- **Private Split Learning Framework (PSLF)** represents another reimplementation of the framework from Wan et al. (2023). The protection mechanism in this method is based on the utilization of Randomized Response (Warner, 1965) and satistifies the conditions of Label Differential Privacy.

For each algorithm, we evaluated an accuracy, as well as the privacy leakage value for the 3 following measures:

- **Spectral attack AUC** — a measure of vulnerability to an attack proposed in Sun et al. (2022), measured as classifier ROC AUC: lower value corresponds to better privacy.
- **Norm attack AUC** — vulnerability to a variant of attack proposed in Li et al. (2022), measured as classifier ROC AUC (lower is better). Despite the initial proposal of this approach for attacking gradients, we observed that it is also well-suited for attacking activations.
- **K-means accuracy** — vulnerability to clusterization attack, measured in the percentage of correctly clustered activations, lower is better.

For all setups, we report the worst (least private) value among these metrics throughout the entire training period as a measure of privacy leakage, because it is the worst possible scenario that matters from the client's perspective. In Tables 1 to 3, we label accuracy as *acc* and privacy leakage as *leak*. For DC and P$^3$EFT, we specify the values for the best configuration in terms of the utility-privacy

Table 3: Accuracy and privacy metrics for LLaMA-2 7B.

| Dataset | | Without LoRAs | Regular FT | DC | P$^3$EFT |
|---|---|---|---|---|---|
| SST2 | *acc* | 94.6 | **97.4** | 97.1$_{\pm 0.1}$ | 95.8$_{\pm 0.1}$ |
| | *leak* | **59.1** | 99.3 | 83.6$_{\pm 10.6}$ | 68.9$_{\pm 2.6}$ |
| QNLI | *acc* | 77.0 | 95.0 | **95.2**$_{\pm 0.1}$ | 94.7$_{\pm 0.2}$ |
| | *leak* | **53.3** | 85.5 | 66.6$_{\pm 4.1}$ | 62.9$_{\pm 0.8}$ |

Table 4: PSLF baseline on SST2 with DeBERTa and Flan-T5.

| DeBERTa | $\mathcal{K}=9$ | | | | P$^3$EFT |
|---|---|---|---|---|---|
| $\varepsilon$ | 0.0 | 0.1 | 0.2 | 0.3 | — |
| *acc* | 75.7 | 85.0 | 92.3 | 93.7 | 96.5 |
| *leak* | 55.3 | 64.9 | 97.2 | 97.3 | 62.6 |
| **Flan-T5** | $\mathcal{K}=9$ | | | | P$^3$EFT |
| $\varepsilon$ | 0.0 | 0.1 | 0.2 | 0.3 | — |
| *acc* | 91.3 | 92.9 | 94.8 | 95.4 | 96.1 |
| *leak* | 64.0 | 70.1 | 95.9 | 97.6 | 74.1 |

trade-off. See details in Appendix A. We also report adjusted standard deviations for the P$^3$EFT and DC. To do so, we run the full training procedure from scratch with 3 random seeds.

Despite a thorough grid search over the privacy budget $\varepsilon$ and the number of pseudo-classes $\mathcal{K}$, we were unable to find a stable configuration for training using PSLF (Wan et al., 2023). An example of DeBERTa and Flan-T5 training on SST2 with $\mathcal{K}=9$ is presented in Table 4. Full results can be found in Appendix B and Table 7.

The results for DeBERTa are presented in Table 1. To improve reproducibility, we reuse the hyperparameters (including rank $r=8$) from Hu et al. (2022), with the exception of the dropout value. We disable dropout because it interferes with the mixing weights Equation 5. In preliminary experiments, we observed that with dropout enabled, both P$^3$EFT and DC begin to perform significantly worse.

We use $n=2$ adapter sets for P$^3$EFT for all datasets and adhered to the same approach for the other models as well. Overall, P$^3$EFT achieves nearly the same accuracy as traditional (non-private) fine-tuning, outperforming the DC-based algorithm in terms of privacy given the same accuracy level. On the MNLI dataset, we could not find the hyperparameters for DC that ensure stable training while maintaining privacy. Meanwhile, P$^3$EFT maintains consistent performance on this task.

Table 2 a reports evaluation for the Flan-T5 base model Chung et al. (2022). For this model, we adapt the exact same hyperparameters as in the previous evaluation with DeBERTa-XXLarge. Compared to DeBERTa, these results are more closely matched. Both our algorithm and DC consistently solve all three tasks, but P$^3$EFT slightly outperforms DC in terms of privacy.

To evaluate how our algorithm scales to larger models, we also fine-tune Llama-2 7B Touvron et al. (2023) on SST2 Socher et al. (2013) and QNLI Wang et al. (2018) datasets. For these evaluations, we use LoRA hyperparameters that Hu et al. (2022) used when fine-tuning GPT-3, with several changes inspired by Dettmers et al. (2023). Namely, we use the NF4 weight format, apply LoRA to both attention and MLP layers with rank 16. We fine-tune both tasks with maximum context length of 512 and weight decay 0.01. Table 3 summarizes our results: for QNLI, both DC and P$^3$EFT achieve reasonable accuracy on the target task, while P$^3$EFT surpasses DC in terms of privacy. On SST2, P$^3$EFT shows similarly favorable trade-offs while DC struggles to preserve privacy.

## 5 Conclusion and Discussion

In this work, we analyze privacy-preserving fine-tuning of large neural networks in the context of parameter-efficient fine-tuning and the two-party split learning setting. We show that while standard fine-tuning suffers from label leakage even in the parameter-efficient case, it is possible to leverage the efficiency of PEFT to alter the procedure without any significant performance drawbacks. We test the resulting method, named P$^3$EFT, on a range of pretrained language models and multiple datasets, showing that it is competitive with a strong baseline in terms of label privacy while having higher task performance.

In future work, it is natural to explore how this approach can be extended to establish holistic privacy in both labels and inputs. This problem can be approached from two directions: either adapt the ideas of P$^3$EFT for input privacy, or combine it with an existing work like Li et al. (2023b). Another important direction for future research is exploring the privacy of the long-term client-provider interaction. In a typical real-world use case of API fine-tuning, a client performs multiple training runs on overlapping data and hyperparameters. This could open additional attacks vectors that combine information from multiple training runs.

## REFERENCES

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

Alexander Borzunov, Dmitry Baranchuk, Tim Dettmers, Max Ryabinin, Younes Belkada, Artem Chumachenko, Pavel Samygin, and Colin Raffel. Petals: Collaborative inference and fine-tuning of large models. *arXiv preprint arXiv:2209.01188*, 2022. URL `https://arxiv.org/abs/2209.01188`.

Tianqi Chen and Carlos Guestrin. XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pp. 785–794, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4232-2. doi: 10.1145/2939672.2939785. URL `http://doi.acm.org/10.1145/2939672.2939785`.

Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Eric Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, Albert Webson, Shixiang Shane Gu, Zhuyun Dai, Mirac Suzgun, Xinyun Chen, Aakanksha Chowdhery, Sharan Narang, Gaurav Mishra, Adams Yu, Vincent Zhao, Yanping Huang, Andrew Dai, Hongkun Yu, Slav Petrov, Ed H. Chi, Jeff Dean, Jacob Devlin, Adam Roberts, Denny Zhou, Quoc V. Le, and Jason Wei. Scaling instruction-finetuned language models, 2022. URL `https://arxiv.org/abs/2210.11416`.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. Qlora: Efficient finetuning of quantized llms. *arXiv preprint arXiv:2305.14314*, 2023.

Dreambooth API. Dreambooth API – Easily finetune Stable Diffusion and generate customised AI images — dreamboothapi.ai. `https://dreamboothapi.ai/`, 2024. [Accessed 28-09-2024].

Haonan Duan, Adam Dziedzic, Nicolas Papernot, and Franziska Boenisch. Flocks of stochastic parrots: Differentially private prompt learning for large language models. *arXiv preprint arXiv:2305.15594*, 2023.

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

Yaroslav Ganin and Victor Lempitsky. Unsupervised domain adaptation by backpropagation. In Francis Bach and David Blei (eds.), *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pp. 1180–1189, Lille, France, 07–09 Jul 2015. PMLR. URL `https://proceedings.mlr.press/v37/ganin15.html`.

Otkrist Gupta and Ramesh Raskar. Distributed learning of deep neural network over multiple agents. *Journal of Network and Computer Applications*, 116:1–8, 2018. ISSN 1084-8045. doi: https://doi.org/10.1016/j.jnca.2018.05.003. URL `https://www.sciencedirect.com/science/article/pii/S1084804518301590`.

Karen Hambardzumyan, Hrant Khachatrian, and Jonathan May. WARP: Word-level Adversarial ReProgramming. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pp. 4921–4933, Online, August 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.acl-long.381. URL `https://aclanthology.org/2021.acl-long.381`.

Stephen Hardy, Wilko Henecka, Hamish Ivey-Law, Richard Nock, Giorgio Patrini, Guillaume Smith, and Brian Thorne. Private federated learning on vertically partitioned data via entity resolution and additively homomorphic encryption, 2017.

Pengcheng He, Xiaodong Liu, Jianfeng Gao, and Weizhu Chen. Deberta: Decoding-enhanced bert with disentangled attention. In *International Conference on Learning Representations*, 2021. URL `https://openreview.net/forum?id=XPZIaotutsD`.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for NLP. In Kamalika Chaudhuri and Ruslan Salakhutdinov (eds.), *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 2790–2799. PMLR, 09–15 Jun 2019. URL `https://proceedings.mlr.press/v97/houlsby19a.html`.

Edward J Hu, yelong shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022. URL `https://openreview.net/forum?id=nZeVKeeFYf9`.

Hugging Face. AutoTrain — huggingface.co. `https://huggingface.co/autotrain`, 2024. [Accessed 28-09-2024].

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Oscar Li, Jiankai Sun, Xin Yang, Weihao Gao, Hongyi Zhang, Junyuan Xie, Virginia Smith, and Chong Wang. Label leakage and protection in two-party split learning. In *International Conference on Learning Representations*, 2022. URL `https://openreview.net/forum?id=cOtBRgsf2fO`.

Qinbin Li, Zeyi Wen, Zhaomin Wu, Sixu Hu, Naibo Wang, Yuan Li, Xu Liu, and Bingsheng He. A survey on federated learning systems: Vision, hype and reality for data privacy and protection. *IEEE Transactions on Knowledge and Data Engineering*, 2021.

Shen Li, Pritam Damania, Luca Wehrstedt, and Rohan Varma. PyTorch RPC: Distributed Deep Learning Built on Tensor-Optimized Remote Procedure Calls. In *Proceedings of Machine Learning and Systems 5 (MLSys)*, 2023a.

Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pp. 4582–4597, Online, August 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.acl-long.353. URL `https://aclanthology.org/2021.acl-long.353`.

Yansong Li, Zhixing Tan, and Yang Liu. Privacy-preserving prompt tuning for large language model services. *ArXiv*, abs/2305.06212, 2023b. URL `https://api.semanticscholar.org/CorpusID:258588141`.

Xiao-Yang Liu, Rongyi Zhu, Daochen Zha, Jiechao Gao, Shan Zhong, and Meikang Qiu. Differentially private low-rank adaptation of large language model using federated learning. *arXiv preprint arXiv:2312.17493*, 2023.

Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-Efficient Learning of Deep Networks from Decentralized Data. In Aarti Singh and Jerry Zhu (eds.), *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, volume 54 of *Proceedings of Machine Learning Research*, pp. 1273–1282. PMLR, 20–22 Apr 2017. URL `https://proceedings.mlr.press/v54/mcmahan17a.html`.

Nvidia. Nvidia confidential computing. `https://www.nvidia.com/en-us/data-center/solutions/confidential-computing`, 2024. [Accessed 28-09-2024].

OctoAI. Fine-tuning Stable Diffusion — docs.octoai.cloud. `https://octo.ai/docs/media-gen-solution/fine-tuning-stable-diffusion/fine-tuning-stable-diffusion`, 2024. [Accessed 28-09-2024].

OpenAI. OpenAI Platform — platform.openai.com. `https://platform.openai.com/docs/guides/fine-tuning`, 2024. [Accessed 28-09-2024].

Dario Pasquini, Giuseppe Ateniese, and Massimo Bernaschi. Unleashing the tiger: Inference attacks on split learning. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, CCS '21, pp. 2113–2129, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450384544. doi: 10.1145/3460120.3485259. URL `https://doi.org/10.1145/3460120.3485259`.

Jonas Pfeiffer, Aishwarya Kamath, Andreas Rücklé, Kyunghyun Cho, and Iryna Gurevych. Adapterfusion: Non-destructive task composition for transfer learning, 2021.

Yuma Rao, Jacob Steeves, Ala Shaabana, Daniel Attevelt, and Matthew McAteer. Bittensor: A peer-to-peer intelligence market, 2021.

Weiyan Shi, Ryan Shea, Si Chen, Chiyuan Zhang, Ruoxi Jia, and Zhou Yu. Just fine-tune twice: Selective differential privacy for large language models. *arXiv preprint arXiv:2204.07667*, 2022.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pp. 1631–1642, Seattle, Washington, USA, October 2013. Association for Computational Linguistics. URL `https://www.aclweb.org/anthology/D13-1170`.

Jiankai Sun, Xin Yang, Yuanshun Yao, and Chong Wang. Label leakage and protection from forward embedding in vertical federated learning. *arXiv preprint arXiv:2203.01451*, 2022.

Chuanqi Tan, Fuchun Sun, Tao Kong, Wenchang Zhang, Chao Yang, and Chunfang Liu. A survey on deep transfer learning. In Věra Kůrková, Yannis Manolopoulos, Barbara Hammer, Lazaros Iliadis, and Ilias Maglogiannis (eds.), *Artificial Neural Networks and Machine Learning – ICANN 2018*, pp. 270–279, Cham, 2018. Springer International Publishing. ISBN 978-3-030-01424-7.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL `https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf`.

Praneeth Vepakomma, Otkrist Gupta, Tristan Swedish, and Ramesh Raskar. Split learning for health: Distributed deep learning without sharing raw patient data, 2018.

Praneeth Vepakomma, Otkrist Gupta, Abhimanyu Dubey, and Ramesh Raskar. Reducing leakage in distributed deep learning for sensitive health data. 05 2019.

Xinwei Wan, Jiankai Sun, Shengjie Wang, Lei Chen, Zhenzhe Zheng, Fan Wu, and Guihai Chen. Pslf: Defending against label leakage in split learning. In *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management*, pp. 2492–2501, 2023.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*, 2018.

Yiming Wang, Yu Lin, Xiaodong Zeng, and Guannan Zhang. Privatelora for efficient privacy preserving llm. *arXiv preprint arXiv:2311.14030*, 2023.

Stanley L Warner. Randomized response: A survey technique for eliminating evasive answer bias. *Journal of the American statistical association*, 60(309):63–69, 1965.

Adina Williams, Nikita Nangia, and Samuel R Bowman. A broad-coverage challenge corpus for sentence understanding through inference. *arXiv preprint arXiv:1704.05426*, 2017.

Guangxuan Xiao, Ji Lin, and Song Han. Offsite-tuning: Transfer learning without full model. *arXiv preprint arXiv:2302.04870*, 2023.

Qiang Yang, Yang Liu, Tianjian Chen, and Yongxin Tong. Federated machine learning: Concept and applications. *ACM Trans. Intell. Syst. Technol.*, 10(2), jan 2019. ISSN 2157-6904. doi: 10.1145/3298981. URL https://doi.org/10.1145/3298981.

Da Yu, Saurabh Naik, Arturs Backurs, Sivakanth Gopi, Huseyin A Inan, Gautam Kamath, Janardhan Kulkarni, Yin Tat Lee, Andre Manoel, Lukas Wutschitz, Sergey Yekhanin, and Huishuai Zhang. Differentially private fine-tuning of language models. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=Q42f0dfjECO.

Zhuo Zhang, Yuanhang Yang, Yong Dai, Qifan Wang, Yue Yu, Lizhen Qu, and Zenglin Xu. FedPETuning: When federated learning meets the parameter-efficient tuning methods of pre-trained language models. In *Findings of the Association for Computational Linguistics: ACL 2023*, pp. 9963–9977, Toronto, Canada, July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.findings-acl.632. URL https://aclanthology.org/2023.findings-acl.632.

Haodong Zhao, Wei Du, Fangqi Li, Peixuan Li, and Gongshen Liu. Fedprompt: Communication-efficient and privacy preserving prompt tuning in federated learning, 2023.

## A Hyperparameters search

In P$^3$EFT and Distance Correlation methods resulting loss $L$ function can be viewed in the form

$$L = L_m + \alpha \cdot L_r,$$

where $L_m$ - main task loss, $L_r$ - regularizer and $\alpha$ is a coefficient that controls the tradeoff between these two losses. The selection of this coefficient affects the final performance of the model. Therefore, to find the best configurations for both methods, we iterated through this hyperparameter using a grid search.

We started with $\alpha = 1$ and then altered it with a multiplicative step of $10^{\frac{1}{2}}$. Values were discarded if the quality did not exceed that achieved by solely training the classifier without LoRA. This criterion was adopted because such outcomes would suggest the method's inability to outperform training scenarios in which the server does not engage with the labels whatsoever. Additionally, we excluded values that led to unstable training. By this, we mean instances where, although the model initially trained on the primary task, at some point, the regularizer began contributing significantly more, and the utility value dropped to the starting value. We observed this issue for the DC method with DeBERTa on the MNLI. From the remaining values, we aimed to choose the one that offered the lowest privacy leakage. The final hyperparameter values for P$^3$EFT can be found in the Table 5 and for DC in the Table 6.

Table 5: Regularization parameter $\alpha$ for the P$^3$EFT method. The values in the table represent powers of the $10^{\frac{1}{2}}$.

| | SST2 | QNLI | MNLI |
|---|---|---|---|
| DeBERTa XXLarge | 1 | 1 | 1 |
| Flan-T5-Large | -1 | 1 | 1 |
| LLaMA-2 7B | 0 | 0 | — |

## B Additional experiments

**PSLF Baseline.** In this section we present results of training DeBERTa and Flan-T5 on SST2 and QNLI using PSLF (Wan et al., 2023). As it can be observed from Table 7, despite thorough grid

Table 6: Regularization parameter $\alpha$ for the DC method. The values in the table represent powers of the $10^{\frac{1}{2}}$.

| | SST2 | QNLI | MNLI |
|---|---|---|---|
| DeBERTa XXLarge | 0 | -1 | — |
| Flan-T5-Large | 2 | -1 | 0 |
| LLaMA-2 7B | -1 | -1 | — |

Table 7: PSLF baseline on SST2 and QNLI with DeBERTa and Flan-T5. The asterisk indicates those runs that were stopped earlier because the leak value was too high (above 90).

| DeBERTa SST2 | $\mathcal{K} = 9$ | | | | $\mathcal{K} = 4$ | | $\mathcal{K} = 18$ | | P$^3$EFT (ours) |
|---|---|---|---|---|---|---|---|---|---|
| $\varepsilon$ | 0.0 | 0.1 | 0.2 | 0.3 | 0.1 | 0.2 | 0.3 | 0.4 | — |
| *acc* | 75.7 | 85.0 | 92.3 | 93.7 | 76.4 | 95.1 | 90.8 | 93.9 | 96.5 |
| *leak* | 55.3 | 64.9 | 97.2 | 97.3 | 55.2 | 97.1 | 71.1 | 95.5 | 62.6 |
| **Flan-T5 SST2** | $\mathcal{K} = 9$ | | | | $\mathcal{K} = 4$ | | $\mathcal{K} = 18$ | | P$^3$EFT (ours) |
| $\varepsilon$ | 0.0 | 0.1 | 0.2 | 0.3 | 0.0 | 0.1 | 0.1 | 0.2 | — |
| *acc* | 91.3 | 92.9 | 94.8 | 95.4 | 92.0 | 94.6 | 92.4 | 94.8 | 96.1 |
| *leak* | 64.0 | 70.1 | 95.9 | 97.6 | 59.0 | 91.0 | 60.9 | 93.6 | 74.1 |
| **DeBERTa QNLI** | $\mathcal{K} = 2$ | | | | $\mathcal{K} = 4$ | | | | P$^3$EFT (ours) |
| $\varepsilon$ | 0.0 | 0.3 | 0.6 | 0.9 | 0.0 | 0.3 | 0.6 | 0.9 | — |
| *acc* | 66.3 | 91.4 | 65.3 | 95.0* | 67.2 | 82.4 | 78.4 | 94.4* | 95.6 |
| *leak* | 56.9 | 96.5 | 88.4 | 98.7* | 55.6 | 85.1 | 82.6 | 98.5* | 74.6 |
| **Flan-T5 QNLI** | $\mathcal{K} = 2$ | | | | $\mathcal{K} = 4$ | | | | P$^3$EFT (ours) |
| $\varepsilon$ | 0.0 | 0.1 | 0.2 | 0.3 | 0.0 | 0.1 | 0.2 | 0.3 | — |
| *acc* | 81.2 | 90.0 | 91.9* | 93.2* | 81.1 | 92.2 | 92.3* | 92.6* | 94.7 |
| *leak* | 61.6 | 76.0 | 94.8* | 97.2* | 63.2 | 86.7 | 95.9* | 97.0* | 63.0 |

search across privacy budget $\varepsilon$ and the number of the pseudo-classes $\mathcal{K}$ we didn't find stable training configuration.

We note, that in Tables 4 and 7 we have entries with $\varepsilon = 0$. While we acknowledge that from the formal definition of differential privacy, setting $\varepsilon = 0$ might not be entirely rigorous (as some definitions require the privacy budget $\varepsilon$ to be a positive real number), it carries meaningful practical implications. Specifically, if we consider Randomized Response (equation (3) in Wan et al. (2023)), which forms the foundation of the PSLF framework, setting $\varepsilon = 0$ means that for each training sample, the flipped label $\tilde{y}$ has an equal probability of belonging to any class. Consequently, training with $\varepsilon = 0$ is equivalent to training with random labels, corresponding to the setup with theoretical lower bound for privacy. We included these entries in the table to better illustrate the trend of results across the hyperparameter grid.

**Ablation study on the number of adapters.** We also present the experiments using the DeBERTa model on SST2 and QNLI datasets with $n = 1, n = 3$ and $n = 4$ adapter sets. The results in the Table 8 generally demonstrate, that increasing the number of adapters has minimal influence on the resulting privacy and accuracy. We have also evaluated the efficacy of our setup when utilizing a single set of adapters. Despite slightly reduced stability concerning the $\alpha$ (reguralization weight hyperparameter), this setup proved highly competitive, which opens promising direction for further research.

## C THEORETICAL ANALYSIS FOR THE PRIVATE_BACKPROP ALGORITHM

In this section we provide theoretical analysis for privacy guarantees of the private_backprop algorithm (Algorithm 1). We use notations $B$ for batch_size and $d$ for hidden_size; $h_b, g_b, l_b$ correspond to activations, gradient and loss of the $b$-th batch element; $g_h$ represents a vector of concatenated gradients $(g_1, \ldots, g_B)$ of all batch elements. We consider binary classification as a task with the

Table 8: Ablation study on the number of adapter sets ($n$) with DeBERTa. Values in the $\alpha$ row represent the power of $10^{1/2}$. The asterisk indicates those runs that were stopped earlier because the leak value was too high (above 90).

| DeBERTa SST2 | $n=1$ | | $n=2$ (original paper) | $n=3$ | n=4 |
|---|---|---|---|---|---|
| $\alpha, \sqrt{10}^{(\cdot)}$ | $-2$ | $-1$ | $1$ | $0$ | $1$ |
| *acc* | $96.9_{\pm 0.2}$ | $95.5_{\pm 0.1}$ | $96.5_{\pm 0.2}$ | $95.8_{\pm 1.5}$ | $96.5_{\pm 0.3}$ |
| *leak* | $72.1_{\pm 6.9}$ | $62.9_{\pm 0.8}$ | $62.6_{\pm 2.6}$ | $65.7_{\pm 2.8}$ | $72.4_{\pm 8.6}$ |
| DeBERTa QNLI | $n=1$ | | $n=2$ (original paper) | $n=3$ | n=4 |
| $\alpha, \sqrt{10}^{(\cdot)}$ | $0$ | $1$ | $1$ | $1$ | $1$ |
| *acc* | $95.9^{*}_{\pm 0.1}$ | $94.0_{\pm 1.9}$ | $95.6_{\pm 0.5}$ | $95.6_{\pm 0.1}$ | $95.9_{\pm 0.0}$ |
| *leak* | $94.9^{*}_{\pm 0.0}$ | $71.3_{\pm 9.5}$ | $74.6_{\pm 11.1}$ | $76.5_{\pm 4.7}$ | $71.8_{\pm 3.8}$ |

minimum number of possible label combinations - $2^B$ (however, similar reasoning extends to many other loss functions, e.g., MSE).

We consider significantly stronger assumptions regarding the attacker's capabilities - namely, a white-box scenario. We assume the server knows the client-side model and, consequently, all possible $2^B$ vectors $g_h$ for different label sets. Thus, the server's task is to determine which of the $2^B$ label sets corresponds to the current batch (or at least determine which sets are more or less probable) based on the transmitted vectors.

We examine several possible setups and investigate the minimum $m$ required to ensure that all $2^B$ sets remain equally probable from the attacking server's perspective:

1. The case with two non-interacting servers. In this scenario, it suffices to set $m = 2$ and send one vector $\xi_i$ to each server. From each server's perspective, all $2^B$ variants have equal probability because for any $\xi_i$ and a given $g_h$, there exists a vector $\eta$ such that $g_h$ belongs to the span of $\xi_i$ and $\eta$.
2. Single server case. We demonstrate that in this scenario, it is sufficient to set $m = B$.

   To show this, we note that for the $b$-th element of the batch holds $\partial l_b / \partial h_b = \partial l_b / \partial p_b \times \partial p_b / \partial h_b$, where $p_b \in \mathbb{R}$ is the head's prediction for activations $h_b$ - the probability of class 1. $\partial p_b / \partial h_b$ is a constant Jacobian of rank 1 and does not depend on the label value. Thus, both possible vectors $\partial l_b(y) / \partial h_b$ lie in the Jacobian's image and belong to the same one-dimensional subspace.

   Therefore, it is sufficient for the client to send to the server a basis vector of the corresponding one-dimensional subspace $\partial p_b / \partial h_b$ for each batch element $b$ (and zero vectors for the remaining batch elements). Knowing $\alpha_b$, the client can reconstruct the corresponding contribution of $b$-th element to the weight gradient $g_\theta$. The server, however, cannot determine which label generated the given gradient for each example, as both lie on the same line.
3. Single server case, $m < B$. In this setup, the client is not able to protect all gradients of the batch. Indeed, for $B = 3$ and $m = 2$, the set of $2^3$ possible gradient combinations cannot be embedded in any 2-dimensional plane that the client can construct from 2 vectors (the linear span of these $2^3$ is 3-dimensional). At most, the client can fully protect $m - 1$ labels, while the server will know the remaining $b - m + 1$ labels up to a flip of all labels.

We want to emphasize again that the above results obtained under a white-box assumption, which is significantly stronger than our practical setup. The general case is considerably more challenging for the attacking side, and developing a possible attack or determining the theoretical limits of the attacker's capabilities is a complex task. However, we believe that the theoretical analysis presented above may be a good first step in this direction.

## D Informal description of LoRA fine-tuning

For convenience, we provide a brief summary of fine-tuning with LoRA (Hu et al., 2022). This PEFT method was originally designed for fine-tuning large pre-trained language models on downstream NLP tasks. These language models are typically based on the Transformer architecture (Vaswani et al., 2017), where most trainable parameters are allocated to linear layers in multi-head self-attention and feedforward blocks.

In the first stage of LoRA fine-tuning, user augments the model with adapters. To do so, a user goes over linear layers in transformer blocks and adds two trainable matrices, $A$ and $B$ that affect this layer's forward pass. Let $W_i \times x + b_i$ be the original layer with $n$ inputs and $m$ hidden units. Here, $W_i \in \mathcal{R}^{m \times n}$ is a pre-trained weight matrix, $b_i \in \mathcal{R}^m$ is a pre-trained intercept vector and $x \in \mathcal{R}^n$ represents a vector of inputs to this particular layer. During the forward pass, a layer with LoRA adapter computes $W_i \times x + b_i + B_i \times A_i \times x$, or equivalently, $(W_i + B \times A) \times x + b_i$. Here, $A_i$ and $B_i$ are two newly added matrices that constitute a LoRA adapter.

The adapter matrices $A \in \mathcal{R}^{r \times n}$ and $B \in \mathcal{R}^{m \times r}$ have a very small intermediate dimension $r$. For instance, when training GPT-3 with LoRA adapters, authors use $1 \leq r \leq 64$, whereas the main weight dimensions are $m = n = 12288$. The first matrix $A$ is initialized with small random normal values, and the second matrix $B$ is initialized at zeros. That way, initial $A$ and $B$ do not affect the model predictions.

Once all adapters are initilized, the user trains all $A_i$ and $B_i$ matrices of the model, while keeping the rest of the weights frozen. This way, only a small faction (less than $1\%$) of model weights are updated. Once the training is over, the learned adapters $A_i$ and $B_i$ can be merged into the main weights ($W_i := W_i + A_i \times B_i$) or used separately.

LoRA adapters are designed with two objectives in mind: i) to allow fine-tuning models in limited GPU memory and ii) to allow inferencing many fine-tuned models using one inference server. When fine-tuning, LoRA achieves small memory footprint due to the fact that user does not need to compute gradients (or optimizer statistics) for billions of main model parameters. During inference, a server can keep a library of several adapters for different tasks and swap between them on demand.