Is Split Learning Privacy-Preserving for Fine-Tuning Large Language Models?

Dixi Yao, *Member, IEEE*, Baochun Li, *Fellow, IEEE*

Abstract—With the success of pre-trained large language models in various tasks, users, individuals and enterprises alike, may need to fine-tune these models with their own datasets. Split learning was proposed to divide the model and place a portion on each user's own device, and intermediate results in each iteration of training will be sent to the server to complete the forward pass. There were concerns in the literature about whether private data can be leaked by sending such intermediate results from the training process. In this paper, we conduct empirical studies on typical large language models, such as GPT-2, OPT, Llama, and Qwen, to show that in most situations, an honest-but-curious server is not able to reconstruct private data using such intermediate results. To find out the reason why large language models preserve data privacy better in these situations, we present our theoretical analyses on these empirical observations. In one special case, where a state-of-the-art existing attack can reconstruct data in the first iteration, we show that it can be easily defended with a simple but effective solution leveraging publicly accessible data.

Index Terms—Privacy, Large Language Model, Split Learning

1 Introduction

Large Language Models (LLMs) demonstrate exceptional proficiency in a spectrum of natural language tasks, encompassing text generation and question answering. Users can directly interact with these models by feeding their questions, paragraphs or even entire books as inputs. There have been public releases of pre-trained LLMs such as OpenChat [1], Llama [2], Falcon [3], and Qwen [4] offering alternatives to production LLMs like ChatGPT and Google BARD. These models empower users to tailor their own language models with their local data through fine-tuning.

To fine-tune a customized language model, however, users may not be willing to upload their local data to a public server because of privacy concerns. On the other hand, users cannot fine-tune such large models completely on the client device, for the sake of preserving data privacy. For example, fine-tuning a Llama 2 model with 7 billion parameters requires 28 GB of GPU memory, which exceeds the memory capacity of most client devices.

Split learning [5] (SL) is a feasible distributed training paradigm for fine-tuning such large models. The clients only need to train the first few portion of layers with their local data, and transmit the *intermediate results* to the public server. The server then sequentially sends the gradients back to the clients after the forward pass and back-propagation. However, it is pointed out in the recent literature that SL can be vulnerable to adversarial attacks and the intermediate results have the potential to leak private data [6]–[9].

To leverage split learning for fine-tuning LLMs, we will first understand whether such a risk of privacy leakage exists. We develop a system for simulation based on Plato [10], wherein clients and the server operate in distinct processes or devices. In such a way, the server only has access to the intermediate features during training and can reconstruct the private data only on the data it receives. With such a design, we have a thorough analysis to validate the assumptions inherent in prior research and verify the effectiveness of the attacks with proper assumptions.

With our empirical study, though attacking methods, such as UnSplit [9], can achieve good attacking results in the task of image classification and ECG classification [11], they exhibited limited effectiveness on the task of text generation using large language models. We begin with GPT-2 models and do a series of experiments over different models including GPT, OPT, Qwen, and Llama. We find UnSplit works only in a special case where we do split learning with GPT-2 and place one transformer layer on the clients, the server can reconstruct private data in the first iteration. After the first iteration, no matter what language model is used or how many layers are placed on the clients, the UnSplit attacks are all ineffective. In the first iteration, placing more layers on the clients can also make the attacks ineffective.

In summary, our experimental findings yield three key takeaways. First, the effectiveness of UnSplit is inversely proportional to the number of model parameters and layers placed on the clients; specifically, a client model with more parameters and layers makes UnSplit less likely to succeed. Second, after iteration one, if the client model has more trainable parameters, the UnSplit attack is less likely to be effective. Third, notably, after the first iteration, UnSplit proves ineffective when applied to language models, no matter we fine-tune the whole model or fine-tune fewer parameters with LoRA [12] method.

To understand the reasons behind the ineffectiveness of previous successful attacks on large language models, we present our theoretical analyses over empirical studies, across various aspects including models, data, and features. We explain that the presence of some layers, such as dropout layers and layer normalization layers, which are widely used in language models constitutes a critical factor. The

• *Dixi Yao, and Baochun Li are with the University of Toronto, Toronto, ON M5S 1A1, Canada. E-mail: dixi.yao@mail.utoronto.ca, bli@ece.toronto.edu.*

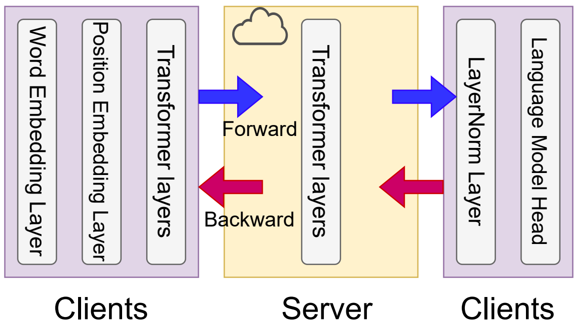


Fig. 1: The deployment of language model across the clients and the server in split learning.

model scale of language models is also larger, contributing to another factor. In terms of data, text data has a larger dimension and is harder to reconstruct. To substantiate these findings, we visualize the similarity between input and output features at each layer in language models, which is much smaller compared to convolutional neural networks.

In addressing the special cases where the UnSplit attack is effective, we propose a simple yet helpful solution. We just need to let the clients fine-tune with public data in the first iteration during split learning. With the abundant empirical study and analysis of the properties of language models and text data, we conclude that split learning can preserve data privacy when fine-tuning large language models.

2 Background and Relate Work

2.1 Large Language Model

Large language models (LLMs) represent a category of models primarily built on transformers, and are trained on massive amounts of text data to learn the patterns of natural language. We have seen notable advancements in production level LLMs, such as GPT-4 and public releases of these pre-trained LLMs [1]–[3], [13]. Open Pre-trained Transformers (OPT) [13] is a collection of models trying to match the performance of GPT-3. Llama 2 [2] is a collection of pre-trained and fine-tuned large language models ranging from 7 billion to 70 billion parameters. Llama 3 is a more advanced version of Llama2, ranging from 8 billion to 70 billion parameters. OpenChat [1] is a collection of open-source language models trying to match the performance of GPT-4. Qwen [4] is is a language model series including decoder language models of different model sizes from 0.5B to 72B. Large language models usually have a lot of parameters and are trained on large amounts of data with supervised fine-tuning and reinforcement learning fine-tuning. As a result, users can fine-tune the large-scale, pre-trained language model to diverse downstream applications [12].

Fine-tuning large language models involves various methodologies. One is fine-tuning the whole pre-trained model. The entire model, including its pre-trained weights, is subjected to further training. Another is to use the low-rank adaption [12], [14] (LoRA). Conversely, it maintains the frozen state of the original model weights. Low-rank decomposition matrices are introduced into the model and trained alongside the model updates. The trainable parameters will be greatly reduced and the GPU memory can be saved.

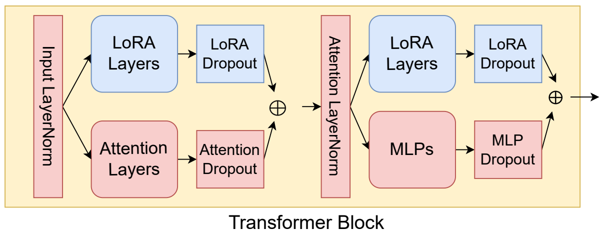


Fig. 2: The structure of a typical transformer block (with LoRA fine-tuning method).

The typical structure of a large language model is depicted in Fig. 1. We input token indicies into the model, where a token id corresponds to the index of a token in a specific lookup table. The first layer is an embedding layer to convert token id into embedding features. The second layer is a position embedding layer. The final layers will be a final layer normalization layer and a language model head layer, such as a linear layer in text generation and text classification. The remaining part is composed of several transformer blocks.

The structure of a transformer block, as illustrated in Fig. 2, includes attention layers and Multilayer Perceptron (MLP) layers. To optimize the efficiency of these layers, the Low-Rank Adaption (LoRA) method can be applied. This involves freezing the weights of attention layers and MLPs, updating only the weights of LoRA layers. The intermediate features are then computed as the sum of the outputs from LoRA layers and the original attention layers or MLPs.

2.2 Split Learning

To fine-tune LLMs on local data, there are different decentralized training paradigms. In federated learning [15], each client trains a model locally and a server will collect the weights of these models and aggregate them to a global model. LoRA [12], freezes the pre-trained model weights and injects trainable rank decomposition matrices into each layer of a transformer, greatly reducing the number of trainable parameters of a model. However, given the substantial parameter count in large models, clients often face challenges locally fine-tuning the entire model or utilizing LoRA. For instance, fine-tuning a Llama 2 model [2] with 7 billion parameters necessitates over 8 NVIDIA A100 80GB GPUs. Even with LoRA, the process still demands over 28GB of GPU memory or 10GB with QLoRA.

Split learning [5] (SL) offers a decentralized training paradigm where only the initial layers of the model are placed on the client side, while the remaining layers reside on the server. The client trains the network up to the partition layer and sends the intermediate features to the server. Upon receiving the features, the server takes over training the remaining layers and completes forward propagation. During the backward propagation, the server will first conduct backward propagation till to the partition point and send back the gradients of the partition layers to the client. The client will update the local parameters by conducting backward propagation with the received gradients.

To deploy a language model in split learning, as shown in Fig. 1, we place the first several layers including the token embedding layer, the position embedding layer and the first a few transformer layers on the clients. In order to preserve the privacy of local data, we also need to keep the labels on the clients. The reason is that for the task of text generation, the labels are generated by shifting one word of the inputs. As a result, we place the last several layers on the clients.

2.3 Privacy Leakage in Split Learning

Though split learning can bring benefits to decentralized training, existing literature points out that there can be potential privacy leakage through the intermediate features. He et.al. [16] proposed a query-based attack to let the server send some specific designed inputs to the clients and observe the patterns of corresponding intermediate features. Zhang et.al. [17] assumes that the server knows the model weights of the client models and conducts a white-box attack. In PatchShuffling [7], DataMix [8], and Shuffled Transformer [18], [19], they assume the server have a public dataset, which has the similar distribution as the private datasets. And then they train an inverse-network to reconstruct private data from intermediate features on the public dataset. Language Model Inversion [20] requires to train the inverse-network on the same instruction dataset on text data.UnSplit [9] constructed a randomized guessed client model on the server and a randomized guessed inputs first. They then iteratively updated the guessed inputs and the weights of the guessed model through gradient descent. The gradient descent is based on the mean square error between intermediate features and the outputs generated on the server. Finally the converged guessed inputs will be the private data trying to reconstruct.

Label leakage [21] assumed that the labels contain the private information and inference the private label through observing the distribution of the backwarding gradients. However, such an attack is only applicable for binary classification tasks in split learning. Inference attack [22] steals private data by sending attacker-designed gradients to fool client models into sending features that the attacker uses to reconstruct private data. While this attack strategy may succeed in extracting information, it comes at the cost of degrading the performance of the trained model. Consequently, the impact on model performance makes this type of attack easier to detect, thus limiting its effectiveness.

To the best of our knowledge, the majority of existing research has primarily focused on reconstructing private data in scenarios related to image classification, convolutional neural networks, or simple language models. PatchShuffling [7] also studied the privacy in terms of tabular data in recommendation system. Sharif et.al. [11] studied the privacy of split learning in terms of ECG classification with 1D CNN. We are the first to try to study whether the risk of privacy leakage exists in terms of split learning over text generation with large language models.

3 Attacks Towards LLMs in Split Learning

3.1 Threat Modeling

We first give a definition of the threat model in practical cases. We assume the server is honest but curious. The server will send the correct back-warding gradients to the clients. But at the same time, it will try to reconstruct private data using the received intermediate features. The server can run the reconstruction process in the background so that clients will not notice the attacks. The server does not know the updated weights of the client models during training. On the other hand, it does know the pre-trained weights of the client models before SL gets started because pre-trained weights are usually publicly available on the Internet. Under such a threat model, the server only knows two contents: the transmitted intermediate features and the pre-trained weights of the client models before the training gets started.

3.2 Possible Attacks

3.2.1 Query-Based Attacks

In previous literature, one kind of attack is query-based attacks. Such an attack needs the server to send specific designed data or manipulated gradients to clients so that the server guides them to leak private information. However, as the gradients or intermediate features deviate greatly from their correct values, the performance of the fine-tuned model will be affected, which is easy to be noticed by the clients. Hence, we can easily detect such attacks and do not focus on this kind of attack in this paper.

3.2.2 Inverse Network Based Attacks

Another kind of attack first trains an inverse-network and then uses it to take the intermediate features as the input and outputs the private data. However, since private data are kept locally, the server does not know any prerequisite information about local data and thus will not be able to train an inverse network on a dataset which is similar to a private dataset. On text data [20], the attacker needs to know the private instruction dataset. However, if it already knows the contents similar to the private dataset, the privacy has been leaked. There is no need to conduct the attacks.

3.2.3 UnSplit Attack

As a result, due to these reasons, in this paper, we do not consider query-based attacks and the attacks that need to train an inverse network. The assumptions of these attacks are not valid in the actual scenarios. We will focus on whether UnSplit-like attacks are effective for LLMs.

The initial UnSplit is designed to recover the private images on local clients in image classification tasks. They randomly initialized a copy of the client model on the server, which we call it a guessed client model, notating it as M, and a training sample, notating it as x. The weights of the guessed client model is θ. After the UnSplit attack finishes, it took the converged training samples as the wanted reconstruction of private data. In each split learning iteration, after receiving the intermediate features, notating it as $\hat{h}$ here, the server inputs the training sample into the guessed client model and gets the output. So, the servers first does several inner iterations to update the x with $\nabla_x L_{\text{MSE}}(M_\theta(x), \hat{h})$. The server next does several inner iterations to update the θ with $\nabla_\theta L_{\text{MSE}}(M_\theta(x), \hat{h})$. These two steps are repeated for several outer iterations until convergence.

The private data are tokens or token ids for the task of text generation. In the original UnSplit, we need to calculate the gradients over x and θ. However, first, token ids are

Algorithm 1 UnSplit attack over LLMs

```
1:  Output: reconstructed token ids.
2:  Server receives intermediate features $\hat{h}$.
3:  Server sets $\theta$ the same as pre-trained weights and initializes $x'$ randomly. The guessed client model $M$ has the same structure as the client models.
4:  for outer loop do
5:      for inner loop do
6:          optimize $x'$ with $\nabla_{x'} L_{\text{MSE}}(M_\theta(x'), \hat{h})$.
7:      end for
8:      for inner loop do
9:          optimize $\theta$ with $\nabla_\theta L_{\text{MSE}}(M_\theta(x'), \hat{h})$.
10:     end for
11:     $\theta^e \leftarrow$ extract weights of the embedding layers.
12:     Token ids $\leftarrow \{j|\text{argmin}\|\theta^{\text{e}}_{\text{j}} - \text{x}'_{\text{i}}\|_2, \forall x'_i \in x'\}$.
13: end for
```

discrete variables in the space of integers, not continuous variables in the image space. We are not able to calculate their gradients. Second, for the language model, the embedding layers will not propagate gradients over x. Hence, the original UnSplit does not work on the language model as we are not able to calculate ∇x and update x. Another difference for fine-tuning language model is that the server does not need to randomize the θ but use the weights from the pre-trained models.

After investigating the structure of language models, we propose an improved UnSplit attack specific to text generation tasks with language models. Since the word embedding layer cannot propagate gradients over x', we first reconstruct the output of the word embedding layers. We do the UnSplit attack with available $\hat{h}$ until convergence and get the reconstructed embedding features x'. Next, we take out the weights of the word embedding layers from the pre-trained model on the server. For each embedding feature x', we calculate the Euclidean distance between it and each embedding vector in the weights and find the one with the minimal distance. The corresponding index of that embedding vector will be the token id we want to get. The whole process is shown in Algorithm 1 and in such a way, we can still do UnSplit attack over language models.

3.3 Attacking Results

3.3.1 *Experimental Settings*

For fine-tuning the whole model and with LoRA, we used the AdamW optimizer with a learning rate of 5×10^{-6}, and 2×10^{-4} respectively. For the Unsplit attack, the number of outer and inner iterations is 100. We update x' using Adam optimizer with a learning rate of 0.1 and θ using AdamW optimizer with a learning rate of 0.001. The batch size is set as 4. Before forwarding the inputs into a model, we use the corresponding tokenizer of each model to convert the tokens into token indices.

We use three datasets for fine-tuning. The first one is WikiText103 [23], which contains over 100 million tokens extracted from the set of verified good and featured articles on Wikipedia. Given that WikiText103 has been extensively used in the pretraining of some LLMs such as OPT and OpenChat, we have chosen two additional private datasets

Original data	millimeter (0 @.@ 8 in) Oerlikon light AA guns on single mounts. In addition Caradoc was fitted with a Type 271 and Type 290
Accuracy(%)	Reconstructed data
2.95	Pillopolis medi wrestling Am Overt Jeff num par sl Med Inf Hey ev Fre Kem M Jehovah Solomon Gleyx SpaceEngineers minim Ke
11.69	device idea h 0 @cm@ 8 onappend Oerlikon light AA guns There single what Lee I addition Caradoc proved fittedak HiddenType 271
43.55	by some 2 0 that that 8 that< |endoftext| >< |endoftext| > that 33 that light AA guns that single mount< |endoftext| >< |
82.57	byiece / 0 @ It@ 8 that, Oerlikon light AA guns on single mounts. In addition Caradoc was fitted at a Type 271 and Type290 surf

TABLE 1: The example of the reconstructed data and the original data under different attacking accuracy.

not utilized in existing LLM training processes. TextGenerator mini 4 dataset [24] contains over 37,000 rows of data not included in major text generation datasets. Another dataset is the Enron Aslec Emails dataset [25]. It is a comprehensive collection of email communications within the Enron Corporation, comprising approximately 500,000+ emails from around 150 employees. Typically, as LLMs are not used to generate email addresses, we use such datasets solely to verify the effectiveness of attacks.

To evaluate the effectiveness to the attacks, we use the accuracy between origin token ids and reconstructed ids and the ROUGE [26]. ROUGE score ranges from 0 to 1, with higher values indicating better summary quality. ROUGE-1 measures the overlap of single words and ROUGE-2 measures the overlap of two-words between two text samples. ROUGE-L measures the longest common sequence between two texts. It can measure semantic similarity as it finds the longest common sequence regardless of word order. ROUGE-Lsum splits the text into multiple sentences and sums the ROUGE-L of each sentence. In our evaluation, we did not differentiate between upper and lower cases.

To clearly illustrate these metrics, we present examples of both reconstructed and original data from WikiText-103 at various levels of attack accuracy in Table 1. Each sample contains over 5000 characters; thus, we display only the first 128 characters for demonstration purposes.

3.3.2 *Implementation*

We implement the process of fine-tuning large language models with split learning in PLATO [10] and PYTORCH. We use the implementation the LoRA method provided in PEFT. We download pre-trained models and datasets from HUGGINGFACE. We use the tokenizer corresponding to each model. The maximum length of each tokenizer embedding block is set to 1024. Our experiments are done on the NVIDIA A4500 20GB GPUs, NVIDIA A100 40GB GPUs, and Apple Mac M1 Pro 16GB.

3.3.3 *Iteration 1*

We first consider the situation that the server tries to reconstruct private data at iteration one. At iteration one, the

TABLE 2: The effectiveness of improved UnSplit attack in the first and second iteration during split learning, on WikiText-103. Number of parameters are counted in millions. Acc, R1 F1, R2 F1, RL F1, and RL sum F1 represent Accuracy, ROUGE-1 F1, ROUGE-2 F1, ROUGE-L F1, and ROUGE-L sum F1 respectively.

		First iteration					Second iteration				
	# Param	Acc%	R1 F1%	R2 F1%	RL F1%	RL sum F1%	Acc%	R1 F1%	R2 F1%	RL F1%	RL sum F1%
GPT-2	46.5	82.57	86.38	61.96	78.90	86.18	0.415	0.941	0.761	0.602	0.836
OPT	157.6	43.55	41.54	16.61	38.49	40.35	2.95	4.99	3.02	3.60	4.88
OpenChat 3.5	349.2	29.13	28.82	8.73	26.69	28.14	0.0	0.09	0.11	0.077	0.121
Qwen 1.5	361.8	11.72	16.92	9.04	13.54	16.92	1.17	0.17	0.0	0.14	0.17
Llama 2	333.6	36.08	45.79	31.98	45.79	45.73	0.879	1.01	0.0	0.864	1.01
Llama 3	743.6	19.82	25.57	10.46	23.23	25.25	0.12	0.348	0.0	0.349	0.349

TABLE 3: The effectiveness of improved UnSplit attack at the second iteration during split learning, on WikiText-103. The models are fine-tuned with the LoRA method. Number of parameters are counted in thousands.

	# Parameters	Accuracy %	ROUGE-1 F1%	ROUGE-2 F1%	ROUGE-L F1%	ROUGE-L sum F1%
GPT-2	24.6	11.69	10.09	0.98	7.04	6.88
OPT	65.5	1.95	7.42	1.02	5.92	7.15
Qwen 1.5	312.3	0.85	0.21	0.11	0.17	0.21
Llama 2	131.1	6.74	6.29	4.87	11.32	6.26
Llama 3	106.50	0.20	3.25	3.73	2.o1	3.25

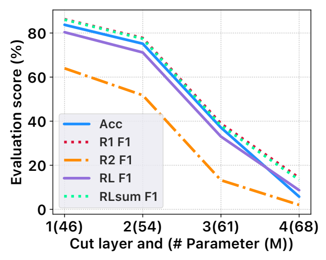
model weights of the client model have not been updated, which are known to the server. With the beginning of the experiments, we first place only the first transformer layer on the clients and the remaining layers on the server. With the improved UnSplit attack, we begin an experiment using the GPT-2 model. As shown in Table 2, the server can successfully reconstruct private data. Hence, we further test over other larger language models with the same setting. The OPT [13] and OpenChat 3.5 [1] have similar performance to GPT-3 and GPT-4 respectively. Llama [2] and Qwen [4] are two latest widely used larger language models. In our experiments, we choose the Llama –7B, Llama 3–8B, and Qwen 1.5–1.8B. It shows that with the number of parameters increasing, the effectiveness of the attack decreases. With such an observation, we would like to figure out in which situations the attack is effective in the first iteration.

Attack effectiveness along number of layers on clients. First, we investigate whether the total number of parameters in the model is a determining factor. We repeat the experiments but place additional layers on the clients, thereby increasing the number of parameters there. As shown in Fig. 3, UnSplit no longer functions effectively as more layers are added. Interestingly, when we place four transformer layers from GPT-2 on the clients, resulting in 68 million parameters, the UnSplit attack already becomes ineffective. For the larger OPT model, as the number of parameters on the clients increases, the UnSplit attack also fails to retrieve the correct private data.

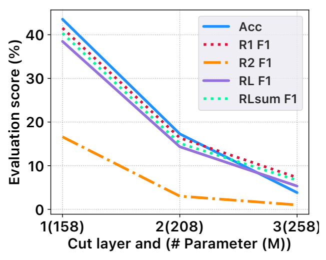
Takeaway 3.1. *We can draw the conclusion that at iteration 1, if the client model has more model parameters, and more layers placed on the clients, the UnSplit is less likely to be effective.*

3.3.4 Iteration > 1

At iteration 1, the client models have not been fine-tuned. We study the effectiveness of attacks after the client model is fine-tuned for one iteration. The attacking results over the GPT-2 model in the second iteration are shown in Table 2, where we see that with just one iteration of updating the client model, it is surprising that the UnSplit attack no longer works. Besides the GPT-2 model, for the other language model, UnSplit does not work either. To investigate further, we conduct an experiment with the GPT-2 model. We use the UnSplit attack at different iterations and from Fig. 4, except for the first iteration, which we have already discussed, UnSplit is not effective from the second iteration



(a) GPT-2



(b) OPT

Fig. 3: The attacking results of UnSplit over GPT-2 and OPT with placing different number of transformer layers on the clients, in the first iteration.

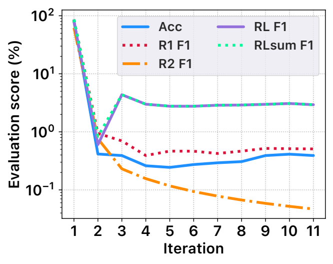


Fig. 4: The attacking results of UnSplit over GPT-2 with placing the first transformer layer on the clients, during different iterations

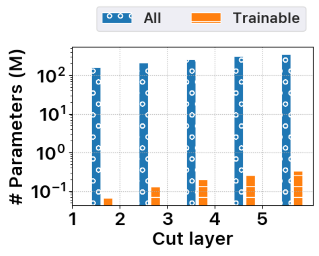
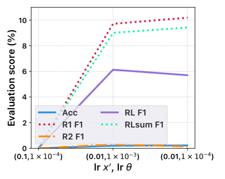


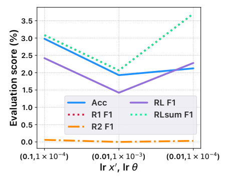
Fig. 5: Parameters and trainable parameters when fine-tuning OPT with LoRA and placing different layers on clients.

TABLE 4: The effectiveness of improved UnSplit attack at the second iteration during split learning on different dataset. Acc, R1 F1, R2 F1, RL F1, and RL sum F1 represent Accuracy, ROUGE-1 F1, ROUGE-2 F1, ROUGE-L F1, and ROUGE-L sum F1 respectively.

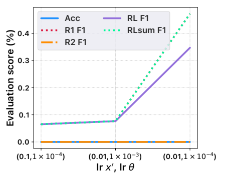
	TextGenerator mini 4					Enron Aslec Emails				
	Acc%	R1 F1%	R2 F1%	RL F1%	RL sum F1%	Acc%	R1 F1%	R2 F1%	RL F1%	RL sum F1%
GPT-2	8.47	0.775	0.0	0.646	0.646	23.83	11.52	1.44	9.94	10.40
OPT	5.18	9.18	0.168	4.45	5.29	0.0	0.44	0.0	0.33	0.38
OpenChat 3.5	0.656	0.703	0.0	0.663	0.584	2.05	2.16	0.0	1.44	2.16
Qwen 1.5	5.37	2.82	3.52	2.53	2.53	3.32	1.93	0.0	1.66	1.66
Llama 2	0.879	1.01	0.0	0.864	1. 01	4.69	0.536	0.0	0.536	0.536
Llama 3	3.91	12.37	4.25	11.20	10.02	2.52	2.39	5.17	2.24	2.24



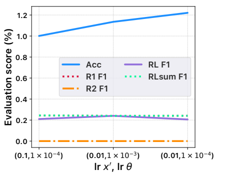
(a) GPT-2



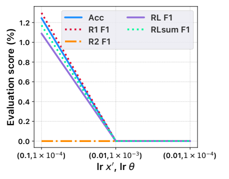
(b) OPT



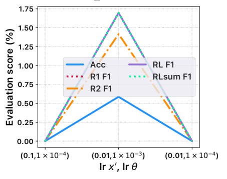
(c) OpenChat



(d) Qwen



(e) Llama2



(f) Llama3

Fig. 6: The attacking results over different models with placing the first transformer layer on the clients, in the second iteration, setting different attacking hyper parameters.

even with the smallest GPT-2 model.

Attack effectiveness along number of trainable parameters. We next study whether the number of the trainable parameters is the factor of the effectiveness of UnSplit. Different from the first iteration, where the model is not fine-tuned, the weights of the client models are updated. We can also fine-tune the model with the LoRA method which makes the model have much fewer trainable parameters. The trainable parameters decrease directly from the million scale into the thousand scale. It is intuitive that with fewer trainable parameters, it is easier to recover the weights of the guessed client models as there are fewer parameters to recover. For example, as shown in Fig. 5, the trainable parameters in OPT are much fewer than all parameters, with having different number of transformer layers placed on the clients. We can see from the Table 3, though with fewer trainable parameters, the UnSplit attack can achieve higher attack accuracy and ROUGE score, it fails to effectively reconstruct the private data. The attacking accuracy and ROUGE score is around or above 10%, not able to reveal privacy of local data.

Takeaway 3.2. *At iteration after iteration 1, if the client model has more model parameters, more trainable parameters and more layers placed on the clients, the UnSplit is less likely to be effective.*

To verify our discovery, we further conducted the experiments with different hyper-parameters and used different datasets as the private datasets. We revisit the experiments by placing a first layer on the clients and using the text-generator dataset. As we can see from the Table 4, though the text-generator and eron-emails have simpler contents comparing to the WikiText-103, the Unsplit attack is still not effective to successfully reconstruct the private data.

3.3.5 *Impacts of UnSplit Hyper-Parameters*

We further investigate that whether the setting of hyper-parameters, including learning rate of updating θ and x' will have effects on the effectiveness. Hence, we repeat the previous experiments using different hyper-parameters. Apart from our default setting, we also set the learning rate of updating x' as 0.01 and set the learning rate of updating θ as 1×10^{-4}. As we can see from Fig. 6, it is noteworthy that the effectiveness remains consistently low across different configurations, as measured by accuracy and ROGUE scores. Thus, the observed results suggest that variations in hyper-parameter values do not significantly impact the efficacy of the UnSplit attack.

Takeaway 3.3. *At iteration after iteration 1, the UnSplit attack is not effective enough to reconstruct private data even if we place only the first transformer layer on the clients.*

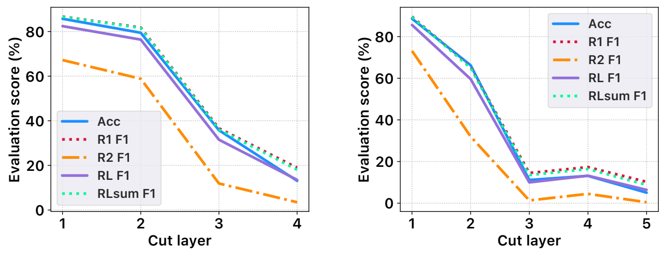
4 Analysis over Attacking Results

A naturally raised question is why the UnSplit attack is not as effective as it was in image classification tasks and ECG data classification tasks. A common observation is that if we place more layers on the clients, it is more possible for the UnSplit to fail to attack, in all cases. However, for image classification with CNNs and vision image transformers, and ECG data classification with 1D CNNs, even after iteration 1, they can reconstruct the original data. Consequently, it becomes imperative to investigate the distinctive characteristics of text generation tasks and language models. In this section, we will deliberately construct some special cases which are not real in actual applications, to make the UnSplit attack work. These constructed scenarios help us gain insights into the underlying reasons for the effectiveness of such attacks.

4.1 Model Difference

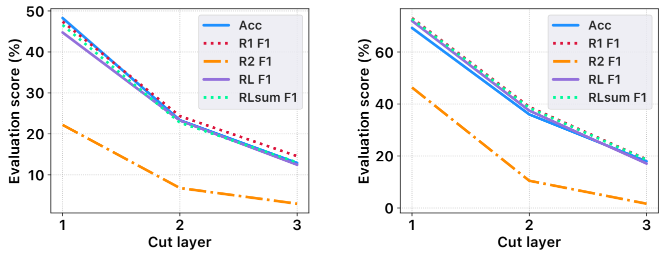
4.1.1 *Special Layers*

An important factor leading to the unsuccessful attack after iteration one is that the training modes on the clients and the server are set different. On the clients, the models are all set in the mode of training. For the guessed client model on



(a) Iteration 1 (b) Iteration 2.

Fig. 7: The attacking results over GPT-2 with placing different number of transformer layers on the clients, in the first and second iteration when dropout layers are disabled.



(a) Iteration 1 (b) Iteration 2.

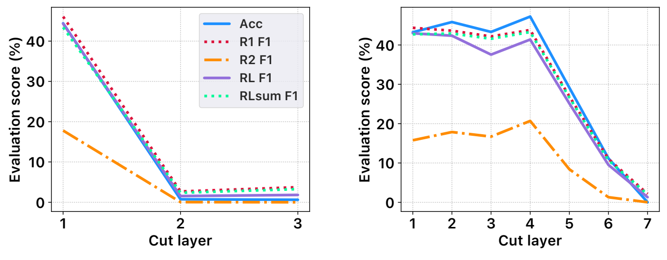
Fig. 8: The attacking results over OPT with placing different number of transformer layers on the clients, in the first and second iteration when dropout layers are disabled.

the server, it can be set either in the mode of training or the mode of evaluation. Importantly, when set in training mode, certain layers such as layer normalization [27] and dropout layer [28] introduce variability in outputs for identical inputs across different runs. It is noteworthy that the model described in the original UnSplit paper lacks normalization layers and dropout layers. However, these components play critical roles in the architecture of large language models. We will next analyze these layers in more detail.

The **dropout layer** serves as a technique employed to address overfitting in neural networks. During the training process, it selectively zeroes out certain elements within the output of the preceding layer prior to the dropout layer, utilizing a probability denoted as p. We also call p as dropout rate. After that, we scale the output with $\frac{1}{1-p}$. During training, dropout layers introduce randomness. By randomly zeroing out elements, outputs for identical inputs vary across runs. Additionally, the zeroing-out operation does not facilitate gradient propagation. The gradients associated with elements set to zero are nullified.

In large language models, as the parameters are often of million and billion scale, dropout is an indispensable tool for mitigating over fitting. As shown in the Fig. 2, we usually have an attention dropout after the attention layers and an MLP dropout after the MLP layer. If we fine-tune the model with the LoRA method, there is usually an LoRA dropout layer. Besides that, there is usually a dropout layer after the position embedding layer and a dropout layer after the language model head, which is a linear layer.

We initiate a comparative analysis to assess the impact of omitting dropout in the first iteration. We repeat the



(a) No LoRA dropout. (b) No other dropout

Fig. 9: The attacking results of UnSplit over GPT-2 with placing different number of transformer layers on the clients, in the second iteration and fine-tune with LoRA method. The LoRA dropout layers are disabled and the other dropout layers are disabled respectively in two cases.

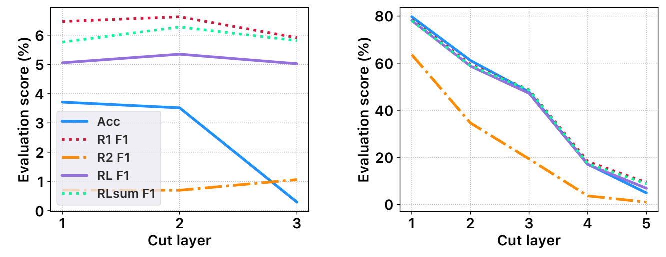
previous experiments, but set the dropout rate of all dropout layers to zero. We can see from the Fig. 7a and Fig. 8a that with setting drop out rate as zero, there is not much improvement in the attacking performance. We then repeat the experiments at iteration 2 with fine-tuning the whole model and setting the dropout rate to zero. It is interesting that the attacking accuracy increases to over 80% on GPT2. On OPT, the attacking accuracy also increases to over 50%. As shown in the Fig. 7b and Fig. 8b, the UnSplit attack becomes effective again when dropout rate is zero and the attacking accuracy is even higher than in the first iteration.

The conclusion that with more layers placed on client models, it is harder for an UnSplit attack to work still holds, even if we disable the dropout layers. From the observation of iteration 1 and iteration 2, we can find that the randomness introduced by dropout layers will affect the recovery of the guessed client models. It is hard for the server to recover the weights of the models on the clients and it leads to the result that we cannot reconstruct the original private data.

As we can also fine-tune the model with the LoRA method, we further study whether the usage of the LoRA dropout layer will affect the UnSplit attack. In a real application, we need to use both the LoRA dropout and other dropout layers. We here consider two cases: only LoRA dropout layers are used and only other dropout layers are used. Regarding the GPT-2 model, from Fig. 9, we can see that comparing to Fig. 7b, both dropout in original transformers and LoRA dropout have impacts on the effectiveness of attacks. If the LoRA drop out rate is set to zero, the attacking accuracy will increase.

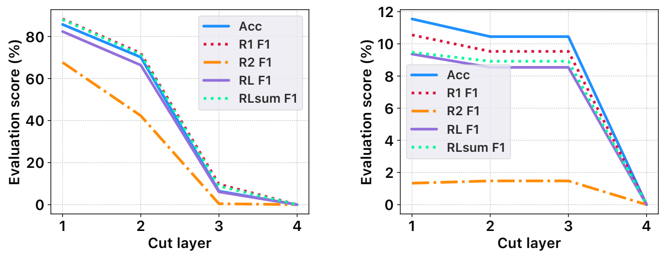
Regarding the OPT model, if we fine-tune with the LoRA method and set the LoRA dropout to 0. From Fig. 10, we can see that with larger models, the dropout layers in the original model plays a more critical factor. We can see that when we use dropout only in LoRA layers, the attacking accuracy is much higher because of fewer trainable parameters, exceeding the accuracy when fine-tuning the whole model. We can conclude that the widely used dropout layers in transformers are a factor in making UnSplit less effective.

Layer normalization is a technique widely used in transformer structured model including language models and vision image transformers [29]. Different from batch normalization, which is widely used in convolutional neural



(a) No LoRA dropout.
(b) No other dropout

Fig. 10: The attacking results of UnSplit over OPT with placing different number of transformer layers on the clients, in the second iteration and fine-tune with LoRA method. The LoRA dropout layers are disabled and the other dropout layers are disabled respectively in two cases.



(a) Fine-tune whole model
(b) Fine-tune with LoRA

Fig. 11: The attacking results of UnSplit over GPT-2 with placing different number of transformer layers on the clients, in the second iteration during studying the impacts of layer normalization layers.

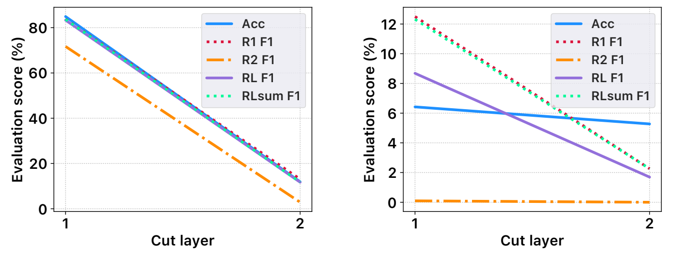
networks, layer normalization does normalization on a scale of samples. The formulation of layer normalization is

$$f(x) = \frac{x - \mathbb{E}[x]}{\sqrt{Var[x]} + \epsilon} * \gamma + \beta \tag{1}$$

In the formulation, ϵ is a constant to avoid zero being denominator; γ and β are two parameters. $\mathbb{E}[x]$ and $Var[x]$ represent the expectation and variance during the training. During the training, layer normalization layers will update $\mathbb{E}[x]$ and $Var[x]$ with training samples. However, as the guessed client model on the server does not know about training samples, it will have different $\mathbb{E}[x]$ and $Var[x]$ as the client models. The calculation of $\mathbb{E}[x]$ and $Var[x]$ is not through updates by gradients but direct calculation over training samples passed through layer normalization layer. As a result, the outputs of layer normalization layers will be different among the between the client models and the guessed client models even if they have the same weights.

There is another difference between layer normalization and batch normalization. In batch normalization, γ and β are scalars, which are the same for all elements in the inputs. While in the layer normalization, they are element wise parameter. So there are more parameters for the server to infer the weights in the client models.

To investigate the impacts of layer normalization without affecting the output distribution, layer normalization layers are not removed directly. Removing them would prevent generating the correct intermediate features. Instead,



(a) Fine-tune whole model
(b) Fine-tune with LoRA

Fig. 12: The attacking results of UnSplit over OPT with varying numbers of transformer layers on the clients in the second iteration, studying the impacts of layer normalization layers. We fine-tuned the model or used LoRA.

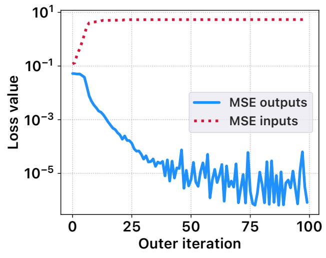
we let the guessed client model process the same private data in the first iteration as the client model. This ensures the guessed client model has the same $\mathbb{E}[x]$ and $Var[x]$ as the client model. We see from Fig. 11 and Fig. 12 that when we fine-tune the whole model, if $\mathbb{E}[x]$ and $Var[x]$ are known to the server, the UnSplit attack works. An interesting observation is that fine-tuning with the LoRA method keeps accuracy low but higher than in Table 3. Hence, we conclude that layer normalization is another factor that makes the UnSplit attack less effective. Additionally, we see that the LoRA layers also play critical roles in determining whether the UnSplit attack can be effective. With the LoRA method, whether UnSplit works depends not only on the original model but also on the LoRA layers.

4.1.2 Model Scale

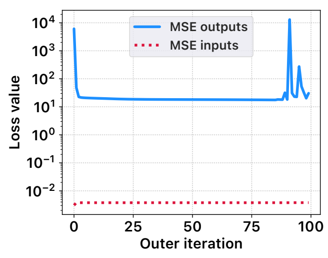
Model scale is another important factor as discussed in the takeaways. To broaden the conclusion, we investigate the number of parameters in CNNs. The model used in the original UnSplit paper has only 1.61 million parameters, and one layer has 1792 parameters. The ResNet152 model has 58.16 million parameters, similar to one transformer layer of GPT-2. One layer of a ResNet152 model has only 75K parameters. The scale of CNN models is much smaller than that of language models, making successful attacks easier.

The theoretical reason why the privacy-preserving ability of LLMs follows scaling low comes from the property that a deep learning model is not a bi-jective function. Since the model is not a bi-jective function, different inputs can lead to the same intermediate features. Due to the special layers which zero-out the propagation of gradients, the attacker has to have the guess on specific weights. When the model has more parameters, with the same model structure or same drop-out rate, the attacker needs to guess more values, leading to a lower success rate. As a result, the bigger the model is, the harder the attack will be.

Apart from considering the number of parameters, each transformer layer in a language model contains two layer normalization layers and two dropout out layers. This is also a factor of why when we place more transformer layers on clients, the UnSplit is more likely to be ineffective. With placing more layers on the clients, we introduce more dropout and layer normalization layers which makes it harder for the UnSplit to recover the weights of the clients models and get the reconstructed private data.



(a) OPT text generator



(b) OpenChat text generator

Fig. 13: The change in MSE between outputs of the guessed client model and the intermediate features, and between private embedding features and reconstructed embedding during the attack, in the experiments in Table 4.

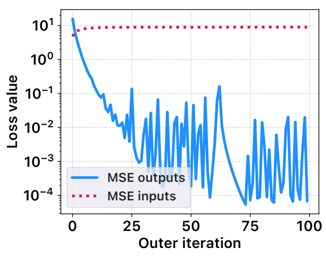


Fig. 14: The change in MSEs during the attack, in the experiments of the second iteration in Table 2 over GPT-2.

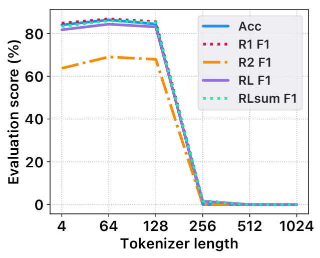


Fig. 15: Repeat the experiments in Table 2 over GPT-2 by using different tokenizer lengths.

4.2 Data Difference

Besides the difference in model structures, another difference between text generation and image classification is the type of data itself. In the UnSplit attack, we update x' and θ according to the MSE between outputs of the guessed client models and intermediate features. However, such an MSE loss between the outputs does not necessarily correlate with the MSE between private data and reconstructed data. We show these two types of MSE in Fig. 13a, Fig. 13b, and Fig. 14, with the example of attacking over OPT and OpenChat on the TextGenerator mini 4. The MSE between the outputs of the client models and those of the estimated client models is denoted as MSE outputs, while the MSE between inputs is labeled as MSE inputs. We can see they do not have a particular correlation. In Fig. 13a and Fig. 14, even if the MSE outputs is trained very low, the MSE inputs can be still high. While in Fig. 13b, it is also possible that the MSE of inputs is higher than the MSE of outputs.

We would like to explain the reason that these two MSE do not necessarily have a correlation in the language model. If we represent a neural network as a function, for most of the neural networks including language models, Convolutional Neural Networks (CNNs), and Vision Transformers (ViTs), it is not a bi-jective function. With the same intermediate features, there can be multiple possible inputs. Even if we use the gradient descent method to update the x' and minimize the loss between the outputs and intermediate features, it is possible that two different inputs can lead to the same intermediate features. With larger input dimension spaces, the number of potential distinct inputs increases.

TABLE 5: The distance correlation and cosine similarity between input features and output features of a given layers.

Layers	DCor	Cos-similarity
Unsplit model, second layer	0.33	0.24
ViT, first layer	0.52	0.29
1D CNN, first two layers	0.47	0.02
GPT-2, first layer	0.19	0.03
GPT-2, first two layers	0.04	0.01
OPT, first layer	0.008	0.006

For instance, the CIFAR10 dataset [30] in the original UnSplit and PatchShuffling papers has an input dimension of $3 \times 32 \times 32$. For the text generation task, the dimension of embedding features, which are the inputs, is $N \times C$. The feature dimension C is set to 768 in our experiments, the default setting in the models we used. The N is called tokenizer length, the length of tokens in each sample. We can understand it as the number of words. In our setting, it is 1024 by default. For the models Llama 2 and OpenChat, it is by default 1×10^{30}. In production-level language models, the tokenizer length can be very large, allowing us to input the contents of an entire book as one sample. For image classification, we do not have images with such a high dimension. But for text generation, we can expand the N according to the actual use cases.

To verify this, we change the tokenizer length and repeat the experiments in the second iteration in Table 2 over GPT-2. To make the embedding feature have the same dimension as image data, the dimension of it is 4×768. We can see from Fig. 15, the attacking accuracy is high. While when we increase the tokenizer length from 128 to 256, the attacking accuracy suddenly drop from over 80% to lower than 1%. The feature dimension of text data is much larger than image data. In the usual case, we should set a long tokenizer length and this will make UnSplit not effective.

Besides the dimension of the input data, the value of the mean square error also indicates different recognizability over different data. For example, over image classification task, as shown in PatchShuffling [7], the reconstructed image is similar to the private data even the MSE between them is about 0.32. In the experiments over OPT shown in Fig. 3, the MSE between reconstructed embedding and original private data are 0.38, 0.33, 0.37 respectively. However, the attacking accuracies are different, from 43.55% to 17.29% and 3.83%. And as shown in Table 1, when the accuracy is lower than 50%, it is hard to tell what the original private texts are. For the experiment over GPT2 in Table 2, the loss is 0.055 and the MSE between reconstructed and private embedding features is 0.18. But the attacking accuracy is 82.57%. In the experiment in the second iteration over Llama 2 in Table 2, the loss is 0.0055 but the attacking accuracy is only 0.879%. Hence, even with achieving the same MSE loss between outputs, attacking text data is much harder.

4.3 Feature Difference

As we have discussed, the dimension of the input data is one factor determining whether the attacks work. To delve deeper into the feature space, we want to see how features differ in various cases. Intuitively, one factor will be the

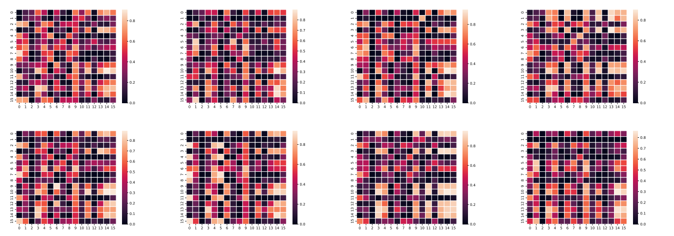


Fig. 16: The similarity of features between the inputs and outputs of the second convolution layer in the model in original UnSplit paper.

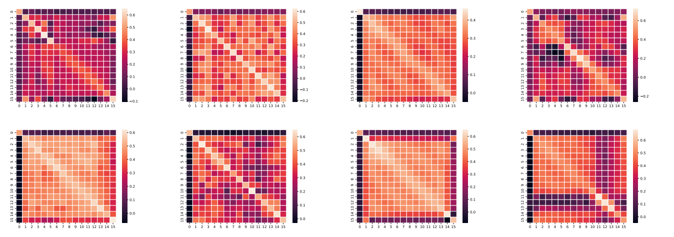


Fig. 17: The similarity of features between the inputs and outputs of the embedding layer plus the first transformer layer in the ViT.

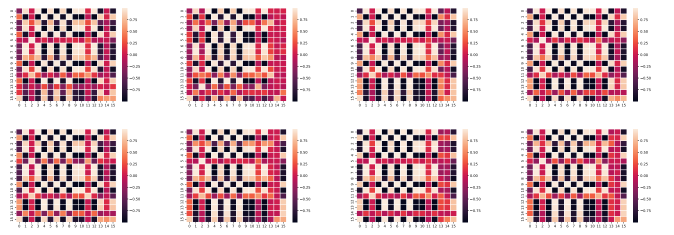


Fig. 18: The similarity of features between the inputs and outputs of the first two layers in the 1D CNN.

similarity between the inputs and outputs of the layers placed on the clients. If they are more similar to each other, it will be easier to infer the inputs (private data) from the outputs (intermediate features). We use cosine similarity to measure this, ranging from 0 to 1. If it equals zero, it means the two features are orthogonal to each other and completely irrelevant. We also measure the distance correlation [31] to see how similar the features are. The distance correlation can measure whether two features are close to each other. The range is also from 0 to 1. Zero means the two features do not have any correlation and one means they are the same. We show the average similarity in Table 5. We also visualize the heat maps of similarity by randomly picking 8 samples from the corresponding datasets. For each grid, the value in the ith row and jth column means the similarity between the features of the ith channel in the inputs and the features of the jth channel in the outputs. For convenience, we show the first 16 channels in the figures.

For the model trained on CIFAR10 in the original paper, we compare the similarity between inputs and outputs of the second layer as they are of the same dimension $64 \times 32 \times 32$. For 64 channels, we compare the similarity of each corresponding channel between input and output. In Fig. 16, we can see some features are very similar to each other and others are completely not.

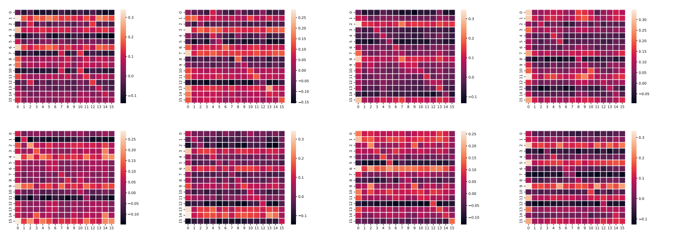


Fig. 19: The similarity of features between the inputs and outputs of the first transformer layer in GPT-2.

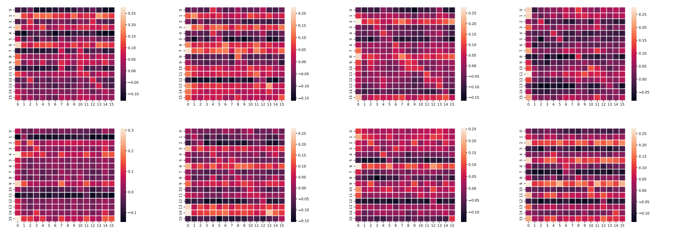


Fig. 20: The similarity of features between the inputs and outputs of the first three transformer layer in GPT-2.

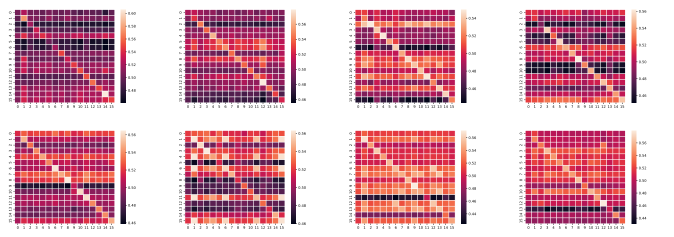


Fig. 21: The similarity of features between the inputs and outputs of the first transformer layer in OPT.

In PatchShuffling [7], the vision transformer model is used. Hence, we visualize the similarity between the output and input features of the first transformer layer in the ViT-patch16–224 model. The weights of ViT are pre-trained on ImageNet [32]. We resize CIFAR10 images to 224×224, leading to a larger feature dimension, which is 196×768. However, the similarity is still high, as shown in Fig. 17. Hence, it is easy to reconstruct private data without protection. This is also why PatchShuffling is a defense method introduced for image classification with ViT using SL.

Apart from image classification, Sharif et al. [11] proposed that it is not safe to train 1D CNN models for ECG data with split learning. We used the same MIT-BIH arrhythmia [33] for ECG classification and their model. We compare the similarity between features of the first two layers. In Fig. 18, we can see the correlation between some feature vectors are very high. This is also why in the paper [11], it is not safe to use split learning for training 1D CNN..

For the GPT-2 model, we compare the similarity between features of the first layer and the first three transformer layers. The text data has a tokenizer length of 1024, and we pick the first 16 for visualization in Fig. 19 and Fig. 20. The cosine similarity between features of the first layer is 0.03, and of the first three layers, it is 0.01. For the OPT model, the similarity between features of the first transformer layer is only 0.006. We visualize this in Fig. 21.

The average similarity metrics for GPT-2 and OPT

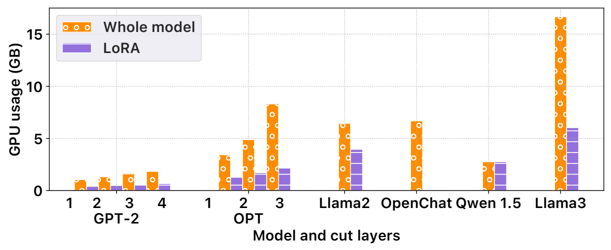


Fig. 22: The GPU memories needed on each clients when training language models with split learning. We place several transformer layers on the clients. The numbers in the x-axis represent the number of layers placed on the clients.

trained on WikiText-103 are smaller than those for ViT and CNN trained on CIFAR10, approaching values close to zero. In scenarios where UnSplit proves to be effective, the similarity tends to be higher. Though the weights of 1D CNN model and the model used in the UnSplit original paper are randomly initialized, the similarity remains high. While for the language model, we use the pre-trained weights. The distribution of similarities is more uniformly distributed and closer to 0. With such a property, training text data on language models is harder to be attacked.

In conclusion, the natural properties in the language models and the properties in the data and features in text generation data help the language model defend against potential attacks itself. The dropout and layer normalization layers, which are necessary parts of a language model for achieving better text generation performance, are also helpful for improving privacy-preserving abilities.

5 Defense

We have shown that in most situations, existing attack methods toward SL are not effective for language models. However, in particular situations such as the first iteration, when we use the GPT-2 model and place only the first transformer layer on the clients, the server can reconstruct the data with over 80% accuracy using the UnSplit attack method. For such special cases, we need a defense solution.

Considering the three takeaways we have concluded, one option is to choose a larger model to fine-tune with the available memory and computation ability on the device. To save memory when fine-tuning large models, users can leverage the LoRA method. A simple defense is to place more transformer layers on clients. Our experiments show that placing two or three layers is sufficient to defend against the attack. We show the GPU memories needed for each client to run first several layers of LLMs in Fig. 22. We can see that a normal personal computer is able to run split learning without leaking privacy.

In situations where clients may face constraints in terms of memory or computational capacity to leverage larger models or place more layers on the clients, a feasible solution is to run one more iteration. We train the model on a public dataset for the first iteration. After the first iteration, users can then fine-tune the model without worrying about privacy leakage. In our setting, to run one iteration with a batch size of 4 and tokenizer length of 1024, the client needs only 4096 tokens retrieved from the Internet. We can also further save the memory usage on the clients through quantization techniques such as QLoRA [14].

There are other efforts trying to prevent the privacy leakage during split learning. One method is to add noise according to differential privacy over the weights of the client models or the intermediate features [34]. However, to achieve certain privacy budget for preventing privacy leakage, the accuracy will decrease. For example, to achieve privacy budget of $\epsilon = 0.67$, the accuracy will drop on a range of **2%** to **4%** on different tasks using RoBERTa [35]. Shuffling-based methods [18] can achieve the same privacy budget with less accuracy loss. However, they apply only to encoder-only transformers, while existing language models involve decoders or are decoder-only transformers. Cryptography based methods such as homomorphic encryption [36] will not harm accuracy too much. However, the latency will be at least **489.07** times of the original split learning. Hence, running the first iteration with public data, though simple, is the most effective way to protect privacy, leveraging the natural property of language models and requiring no change to the original split learning process.

6 Concluding the Remarks

We conduct an extensive examination to find out whether it is privacy-preserving for users to fine-tune large language models through split learning. Our findings indicate that previously employed attack methods, such as methods training inverse-networks and UnSplit attacks, prove ineffective for text generation tasks and large language models in the majority of scenarios. With more than one transformer layer placed on the clients or after the first iteration, an attacker is not able to reconstruct the private data, no matter if the users fine-tune the entire model or with the LoRA method. Our in-depth analysis of the reasons behind this delves into model structures, data types, and feature similarities. For isolated instances where an UnSplit attack might pose a threat, a simple but effective defense method can resolve this, letting the clients leverage public data in the first iteration during split learning. Consequently, an adversary is unable to reconstruct private data from intermediate features. In conclusion split learning can preserve data privacy for fine-tuning pre-trained large language models.

References

[1] G. Wang, S. Cheng, X. Zhan, X. Li, S. Song, and Y. Liu, "OpenChat: Advancing Open-source Language Models with Mixed-Quality Data," *arXiv preprint arXiv:2309.11235*, 2023.
[2] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale *et al.*, "Llama 2: Open Foundation and Fine-tuned Chat Models," *arXiv preprint arXiv:2307.09288*, 2023.
[3] G. Penedo, Q. Malartic, D. Hesslow, R. Cojocaru, A. Cappelli, H. Alobeidli, B. Pannier, E. Almazrouei, and J. Launay, "The RefinedWeb Dataset for Falcon LLM: Dutperforming Curated Corpora with Web Data, and Web Data only," *arXiv preprint arXiv:2306.01116*, 2023.
[4] J. Bai, S. Bai, Y. Chu, Z. Cui, K. Dang, X. Deng, Y. Fan, W. Ge, Y. Han, F. Huang *et al.*, "Qwen technical report," *arXiv preprint arXiv:2309.16609*, 2023.
[5] O. Gupta and R. Raskar, "Distributed Learning of Deep Neural Network over Multiple Agents," *Journal of Network and Computer Applications (JNCA)*, vol. 116, no. 1, pp. 1–8, 2018.

[6] T. Xiao, Y.-H. Tsai, K. Sohn, M. Chandraker, and M.-H. Yang, "Adversarial Learning of Privacy-Preserving and Task-oriented Representations," in *Proc. 2020 AAAI Conference on Artificial Intelligence (AAAI)*, 2020, pp. 12 434–12 441.

[7] D. Yao, L. Xiang, H. Xu, H. Ye, and Y. Chen, "Privacy-Preserving Split Learning via Patch Shuffling over Transformers," in *Proc. 2022 IEEE International Conference on Data Mining (ICDM)*, 2022, pp. 638–647.

[8] Z. Liu, Z. Wu, C. Gan, L. Zhu, and S. Han, "DataMix: Efficient Privacy-Preserving Edge-Cloud Inference," in *Proc. 2020 European Conference on Computer Vision (ECCV)*, 2020, pp. 578–595.

[9] E. Erdoğan, A. Küpçü, and A. E. Çiçek, "UnSplit: Data-Oblivious Model Inversion, Model Stealing, and Label Inference Attacks against Split Learning," in *Proc. 21st Workshop on Privacy in the Electronic Society (WPE)*, 2022, pp. 115–124.

[10] B. Li, N. Su, C. Ying, and F. Wang, "Plato: An Open-Source Research Framework for Production Federated Learning," in *Proc. 2023 ACM Turing Award Celebration Conference (ACM TURC)*, 2023, pp. 1–2.

[11] S. Abuadbba, K. Kim, M. Kim, C. Thapa, S. A. Camtepe, Y. Gao, H. Kim, and S. Nepal, "Can We Use Split Learning on 1D CNN Models for Privacy Preserving Training?" in *Proc. 15th ACM Asia Conference on Computer and Communications Security (ASIACCS)*, 2020, pp. 305–318.

[12] E. J. Hu, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, W. Chen *et al.*, "LoRA: Low-Rank Adaptation of Large Language Models," in *Proc. 2021 International Conference on Learning Representations (ICLR)*, 2021.

[13] S. Zhang, S. Roller, N. Goyal, M. Artetxe, M. Chen, S. Chen, C. Dewan, M. Diab, X. Li, X. V. Lin *et al.*, "OPT: Open Pre-trained Transformer Language Models," *arXiv preprint arXiv:2205.01068*, 2022.

[14] T. Dettmers, A. Pagnoni, A. Holtzman, and L. Zettlemoyer, "QLoRA: Efficient finetuning of quantized llms," in *Proc. Thirty-seventh Conference on Neural Information Processing Systems (NIPS)*, 2023.

[15] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y. Arcas, "Communication-Efficient Learning of Deep Networks from Decentralized Data," in *Proc. 20th International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2017, pp. 1273–1282.

[16] Z. He, T. Zhang, and R. B. Lee, "Model Inversion Attacks Against Collaborative Inference," in *Proc. 35th Annual Computer Security Applications Conference (ACSAC)*, 2019, pp. 148–162.

[17] Y. Zhang, R. Jia, H. Pei, W. Wang, B. Li, and D. Song, "The Secret Revealer: Generative Model-Inversion Attacks against Deep Neural Networks," in *Proc. 2020 IEEE/CVF conference on computer vision and pattern recognition (CVPR)*, 2020, pp. 253–261.

[18] H. Xu, L. Xiang, H. Ye, D. Yao, P. Chu, and B. Li, "Shuffled Transformer for Privacy-Preserving Split Learning," *arXiv preprint arXiv:2304.07735*, 2023.

[19] h. Xu, L. Xiang, H. Ye, D. Yao, and B. Li, "Permutation equivariance of transformers and its applications," in *Proc. 2024 IEEE/CVF conference on computer vision and pattern recognition (CVPR)*, 2024.

[20] J. X. Morris, W. Zhao, J. T. Chiu, V. Shmatikov, and A. M. Rush, "Language model inversion," in *Proc. 2023 International Conference on Learning Representations (ICLR)*, 2023.

[21] O. Li, J. Sun, X. Yang, W. Gao, H. Zhang, J. Xie, V. Smith, and C. Wang, "Label Leakage and Protection in Two-party Split Learning," in *Proc. 2021 International Conference on Learning Representations (ICLR)*, 2021.

[22] D. Pasquini, G. Ateniese, and M. Bernaschi, "Unleashing the Tiger: Inference Attacks on Split Learning," in *Proc. 2021 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2021, pp. 2113–2129.

[23] S. Merity, C. Xiong, J. Bradbury, and R. Socher, "Pointer Sentinel Mixture Models," in *Proc. 2016 International Conference on Learning Representations (ICLR)*, 2016.

[24] Srivatsavaasista, "TextGenerator DS Mini 4," https://huggingface.co/datasets/srivatsavaasista/textgenerator-ds-mini-4.

[25] Snoop2head, "Enron Aeslc Emails," https://huggingface.co/datasets/snoop2head/enron_aeslc_emails.

[26] C.-Y. Lin, "ROUGE: A package for Automatic Evaluation of Summaries," in *Proc. Workshop on Text Summarization Branches Out (WAS)*, 2004, pp. 74–81.

[27] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer Normalization," *arXiv preprint arXiv:1607.06450*, 2016.

[28] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting," *Journal of Machine Learning Research (JMLR)*, vol. 15, no. 1, pp. 1929–1958, 2014.

[29] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly *et al.*, "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale," in *Proc. 2020 International Conference on Learning Representations (ICLR)*, 2020.

[30] A. Krizhevsky, "Learning Multiple Layers of Features from Tiny Images," *Master's thesis, University of Toronto*, 2009.

[31] G. J. Székely, M. L. Rizzo, and N. K. Bakirov, "Measuring and Testing Dependence by Correlation of Distances," *The Annals of Statistics (Ann. Statist.)*, vol. 35, no. 6, pp. 2769–2794, 2007.

[32] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A Large-Scale Hierarchical Image Database," in *Proc. 2009 IEEE/CVF conference on computer vision and pattern recognition (CVPR)*, 2009, pp. 248–255.

[33] G. B. Moody and R. G. Mark, "The impact of the mit-bih arrhythmia database," *IEEE engineering in medicine and biology magazine*, vol. 20, no. 3, pp. 45–50, 2001.

[34] Z. Wang, G. Yang, H. Dai, and C. Rong, "Privacy-Preserving Split Learning for Large-Scaled Vision Pre-Training," *IEEE Transactions on Information Forensics and Security (TIFS)*, vol. 18, no. 1, pp. 1539–1553, 2023.

[35] D. Yu, S. Naik, A. Backurs, S. Gopi, H. A. Inan, G. Kamath, J. Kulkarni, Y. T. Lee, A. Manoel, L. Wutschitz, S. Yekhanin, and H. Zhang, "Differentially private fine-tuning of language models," in *Proc. 2022 International Conference on Learning Representations (ICLR)*, 2022.

[36] T. Khan, K. Nguyen, and A. Michalas, "A More Secure Split: Enhancing the Security of Privacy-Preserving Split Learning," in *Proc. 28th Nordic Conference on Secure IT Systems (NordSec)*, 2023, pp. 307–329.

Dixi Yao is currently a graduate student at the University of Toronto. He received his B.Engr. degree from the Department of Computer Science, Shanghai Jiao Tong University, China, in 2022. He is interested in the intersection of federated learning and large models in terms of efficiency, scalability and privacy concerns. He is a recipient of the Edward S. Rogers Sr. Graduate Scholarships. He was a recipient of the Ewing Rae Graduate Scholarship, a recipient of the Czeslaw And Irene Klawe Scholarship and a recipient of the Kwok Sau Po Scholarship.

Baochun Li received his B.Engr. degree from the Department of Computer Science and Technology, Tsinghua University, China, in 1995 and his M.S. and Ph.D. degrees from the Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, in 1997 and 2000. Since 2000, he has been with the Department of Electrical and Computer Engineering at the University of Toronto, where he is currently a Professor. He holds the Bell Canada Endowed Chair in Computer Engineering since August 2005. His current research interests include cloud computing, security and privacy, distributed machine learning, federated learning, and networking. Dr. Li has co-authored more than 470 research papers, with a total of over 26000 citations, an H-index of 88 and an i10-index of 341, according to Google Scholar Citations. In 2014, he was elevated to the grade of IEEE Fellow, class of 2015. In 2023, he received the Best Paper Award from IEEE INFOCOM 2023, as well as the Best Paper Award from the IEEE International Conference on Metaverse Computing, Networking and Applications (IEEE MetaCom 2023). In June 2023, he was elected as a Fellow of the Canadian Academy of Engineering. In December 2023, he was elected as a Fellow of the Engineering Institute of Canada.