Unraveling Elevated Data Leakage in Split Learning for Fine-Tuning Stable Diffusion Models

Anonymous Author(s)

ABSTRACT

Fine-tuning large models can be computationally expensive, posing a challenge for individual users, especially those lacking expertise and computational resources, to create customized models. While many remote servers are available for large-scale training, directly sending raw data to servers may lead to significant security concerns. To improve data privacy, one can divide the model into a small local module and a large remote module, train the local module with their data and send only intermediate output to a remote server for further training. This approach is called split learning. However, recent studies have shown that split learning, even without raw data sharing, is still not immune to data reconstruction attacks as the intermediate output carries information about the input data. In this paper, we focus on uncovering the elevated data leakage vulnerabilities in large generative AI models like Stable Diffusion, in the context of *split fine-tuning*. We show that the adversary has more advanced knowledge about the client than usual, as demonstrated through our devised data reconstruction attacks. To enhance the security of split fine-tuning when faced with data reconstruction attacks, we design a novel defense based on self-attacking and dropout technique for accommodating varying privacy leakage risks from different cut layers, which protects the intermediate output while having the least impact on model utility and training efficiency compared with the state-of-the-art split learning defenses.

CCS CONCEPTS

• **Computing methodologies** → **Machine learning**; • **Security and privacy**;

KEYWORDS

Model Inversion Attacks, Split Learning, Fine-Tuning, Stable Diffusion

1 INTRODUCTION

With the swift progress of generative artificial intelligence, large text-to-image diffusion models, most notably Stable Diffusion [27] models, have attracted an astonishing level of interest over the past two years, due to their exceptional ability to generate high-quality images simply from a text prompt. Individuals now have numerous access to user-friendly online interfaces that allow them to use pre-trained diffusion models for artistic purposes and creative production with ease, without the need for extensive technical knowledge or expertise. They can further personalize the generated outputs for a specific subject, such as themselves or their pets, by using techniques like DreamBooth [29], which allows large diffusion models to be fine-tuned using just a few (*e.g.*, 3-5) images of that subject.

However, the inherent risk of violating data privacy is a major concern when it comes to uploading personal images to public cloud platforms for remote fine-tuning, as sensitive data can be potentially exposed to unauthorized access or misuse. On the other hand, although locally fine-tuning large Stable Diffusion models guarantees data security, it may require a considerable amount of computational resources and technical expertise. Moreover, many models (such as proprietary models from MidJourney) are not openly available or free for personal use. These factors collectively motivate the deployment of split learning [34], which allows users to fine-tune pre-trained Stable Diffusion models more safely.

Split learning makes clients fine-tune models in a decentralized fashion without sending private data to the server. Instead, each client first sends its data through the first several layers of the model, which are held locally, and then sends the intermediate output of the first several layers to the server, which is responsible for training the remaining portion of the model. Since the client only has access to the first several layers, split learning does not require the entire model to be shared with clients, alleviating potential concerns from the model providers. In addition, split learning is sufficiently general and can be readily applied to any neural network architecture [34], making it feasible for multiple users to collaboratively train or fine-tune any model while preserving data privacy.

However, recent studies have found that, even though only intermediate output is transmitted from client to server with split learning, it is still vulnerable to private data leakage. At first glance, split learning should be privacy-preserving: the server does not have any knowledge of the parameters and structure of the client-side model, or the freedom to query the client. This is contrary to the assumptions commonly made in model inversion attacks, which aim to reconstruct inputs given the outputs of the target model [7, 9, 22, 40, 43]. Whether model inversion attacks are effective in the context of split fine-tuning remains an open question.

In the recent literature, new data reconstruction attacks that were specifically tailored for collaborative inference [13] and split learning [8, 11] were proposed. These attacks allowed an honest-but-curious adversarial server to first reconstruct a functionally similar model to the client model, and then use it to recover the raw data input from the intermediate output received from the client. However, these attacks primarily focused on conventional deep neural networks, such as LeNet-5 [20], ResNet-18 [12], and VGG-16 [30] for image classification tasks. It remains largely unexplored how private data may be effectively reconstructed from

Submitted to ASIA CCS '25, August 25–29, 2025, Ha Noi, Vietnam

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM
https://doi.org/10.1145/nnnnnnn.nnnnnnn

intermediate output when large and complex models, such as Stable Diffusion, are used in split learning.

When fine-tuning Stable Diffusion models, a user has the ability to use any set of private images, which makes it almost impossible for an adversarial server to predict or be aware of the specific dataset used. As a result, the effectiveness of attacks that rely on training a shadow client model [11, 13, 26] may be greatly diminished, as they rely on the availability of public datasets with similar distributions to the target ones. In addition, fine-tuning typically requires fewer training steps compared to training a model from scratch. This may render attacks such as FSHA [26] and PCAT [11] ineffective, as they relied on the ability to train auxiliary models for data reconstruction on the server along with the long-term training process of the primary models. These observations put the adversarial server at a disadvantage, as it may not have the same level of capabilities and knowledge during fine-tuning compared to the process of training a model from scratch.

On the flip side, while split fine-tuning imposes restrictions on the server's ability to launch attacks compared to split learning, it grants the server additional knowledge. Despite the lack of white-box or blackbox access to the client model during fine-tuning, the adversarial server most certainly has the knowledge of the complete pre-trained model before fine-tuning starts. This gives the server a significant advantage, as the network structure is transparent, and the pre-trained client model can be directly used as a functionally similar model, eliminating the need of reconstructing the client model from scratch as in the UnSplit attack [8]. Furthermore, compared to other networks, the unique structure and forwarding path of the U-Net [28] in the Stable Diffusion model pose a greater challenge to preserving data privacy. Previous research [11, 13], which suggested that splitting the neural network at a later layer could reduce the private information in intermediate output, is no longer applicable in this scenario, because the client is required to transmit not only the intermediate output of the cut layer but also the intermediate outputs of previous network blocks to the server.

In this paper, we carry out an in-depth evaluation and analysis of the security of split learning, with a particular focus on preserving data privacy when fine-tuning large generative AI models like Stable Diffusion. We revisit existing threads of model inversion attacks and make adjustments to accommodate the threat models within the context of split fine-tuning, fully considering the advantages and disadvantages of the adversary's knowledge and capabilities. Building upon the refined threat models, we devise model inversion attacks that are specifically tailored for split fine-tuning of the Stable Diffusion model, taking into account the intricate network structure and the model's input, including both image and text prompt. Through this process, we uncover potential vulnerabilities involved in the split fine-tuning process. Our observations have revealed that the risk of data leakage is significantly higher with Stable Diffusion, compared with conventional models.

Drawing from these observations and insights regarding privacy-preserving split learning, there is an urgent need to develop a defense mechanism. In this paper, we propose a new defense mechanism specifically designed to protect the split fine-tuning process over Stable Diffusion models. In particular, we employ the idea of self-attacking, wherein an auxiliary neural network is trained on the client side in advance to reconstruct its own data, as a means to quantify the information leakage within the intermediate output of the cut layer as compared to the input data. On top of it, we apply a dropout layer to the image latents, with an adaptive dropout rate determined by the quantified information leakage. We evaluate the effectiveness of our proposed defense and state-of-the-art defenses especially designed for split learning, including NoPeek [35] and NoiseDefense [32]. Our results demonstrate that our defense mechanism can effectively mitigate privacy risks while achieving fine-tuning performance comparable to that of the model without defense. Our defense exhibits the smallest drop in CLIP score [14] and CLIP directional similarity [10] compared to other defenses, indicating minimal impact on the fine-tuned model's utility, while increasing training time by only 1.8% $\sim$ 4%.

2 PRELIMINARIES AND RELATED WORK

2.1 Split Learning

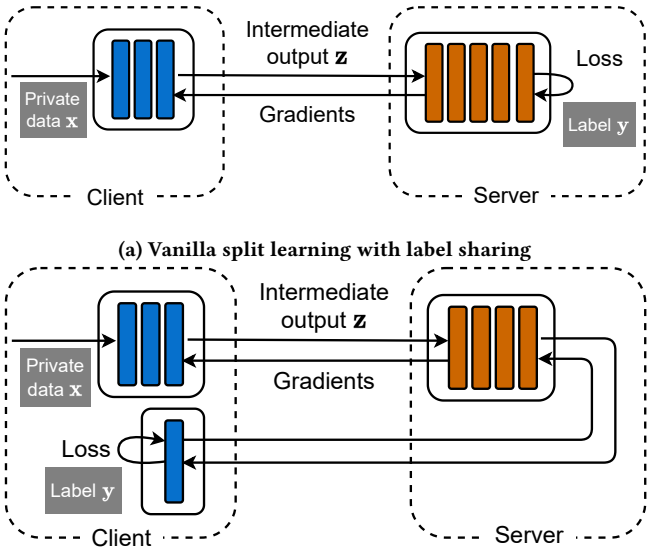
As an alternative to conventional federated learning, split learning [34] advocates that machine learning models should be split between the client and its server, and then trained in a distributed manner. Intuitively, split learning addresses the need of fine-tuning large foundation models (such as Stable Diffusion) well, as these large models may not be trained entirely on the client device. Instead, the client trains a deep neural network up to the cut layer and sends the intermediate output at the cut layer to the server, where forward propagation continues on the remaining portion of the model after the cut layer. The server then calculates the loss for backpropagation and sends the gradient down to the cut layer back to the client to complete the backward pass.

Computing loss on the server side requires the client to share corresponding labels along with the intermediate output. However, this approach may not be secure if the labels contain sensitive information. To eliminate this concern, split learning also allows a U-shaped configuration that ideally protects the labels and the raw data. In this configuration, the server only maintains the middle layers of the model and leaves the end layers to the client. The client is responsible for generating the loss locally and initiating the backpropagation. The client then sends the gradients from the end layers to the server to continue the backward pass through the middle layers. Fig. 1 shows the difference between vanilla and U-shaped split learning configurations. We only consider the U-shaped split learning in this work, as it provides great protection to clients' private labels.

Split learning is a powerful tool for privacy-preserving distributed machine learning that safeguards not only raw data but also model architectures and parameters. Nevertheless, recent work has shown that the intermediate outputs sent from clients to servers can still reveal information about the corresponding training data and can be exploited to reconstruct the raw data samples.

2.2 Data Reconstruction in Split Learning

Model inversion attacks have become a well-established technique for reconstructing data samples using feature representation output from the machine learning model. In the literature, two lines of approaches have been proposed: *optimization-based attacks* that use gradient descent to exploit the target model [9, 22], and *training-based attacks* that involve training a separate attack model [7, 40, 43].



(a) Vanilla split learning with label sharing

(b) U-shaped split learning without label sharing

Figure 1: Split learning with or without label sharing.

However, these standard model inversion attacks are not directly applicable to split learning, primarily because the learning model is divided across multiple devices where the server lacks access to the client model's network structure or parameters, and the ability to query clients with specific data. As a response to this challenge, several model inversion attacks specifically tailored for split learning have been proposed.

In *optimization-based attacks*, the adversary requires whitebox access to the target model — that is, knowledge of its parameters — to perform backpropagation for optimization, but the server involved in split learning has no information about the parameters of the client model during training. To address this limitation, UnSplit [8] incorporates model stealing [25, 33] to allow an honest-but-curious server to construct a functionally similar model to the client model and recover target input samples, with only the knowledge of the client model's network structure.

In *training-based attacks*, the adversary trains a model that learns the mapping from the intermediate output to the input data. The feature-space hijacking attack (FSHA) [26] involves a malicious server completely disregarding primary training tasks and actively hijacking the learning process of the distributed model with the objective of training three additional attack models for reconstructing clients' private data. PCAT [11] trains a pseudo-client model to mimic the functionality of the client model, even in the absence of knowledge about the client model's network structure. This pseudo-client model is then used to build a reverse mapping that enables private data reconstruction. To train those auxiliary attack models, public datasets that are relevant to the same learning task or have a similar distribution to the target dataset are necessary. GLASS [21] utilizes advanced StyleGAN models [18] that are pre-trained on diverse data distributions. This strategy allows GLASS to harness the abundant prior knowledge available in public data, thereby enhancing data reconstruction capabilities for split learning.

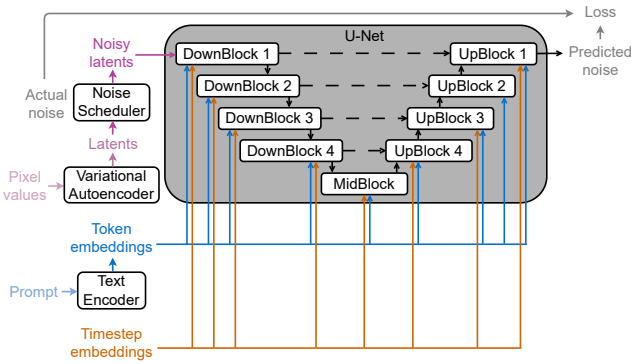


Figure 2: The major components of Stable Diffusion and the forwarding process through DreamBooth fine-tuning.

Optimization-based attacks are advantageous when no auxiliary data is available, as they directly optimize a dummy input without the need for external datasets. However, their reliance on iterative gradient descent makes them slower and more computationally expensive, especially for high-resolution images or large models, since multiple iterations are required to optimize the dummy data effectively. In contrast, training-based attacks allow the decoder to infer input data from intermediate outputs in real time, eliminating the need for repeated optimization as seen in optimization-based attacks. However, the success of this approach largely depends on the quality and relevance of the auxiliary dataset. The server must also have substantial knowledge of the client's data distribution, which may not always be feasible in real-world scenarios. If the auxiliary data poorly reflects the client's data, the decoder may fail to generalize, resulting in inaccurate reconstructions. Moreover, the setup for training-based attacks is more complex, requiring considerable time and computational resources to design and train a model that closely mirrors the client's model.

To mitigate privacy leakage from intermediate outputs in split learning, several research efforts have been proposed. NoPeek [35], which introduces a novel loss function, aims to prevent information leakage by incorporating distance correlation between the raw data and the intermediate output at the cut layer, alongside the conventional classification loss during model training. NoiseDefense [32], which can be regarded as a local differential privacy mechanism [3], applies Laplacian noise to the intermediate output before transmission from clients, preventing data reconstruction by the server.

2.3 Fine-Tuning Stable Diffusion

Stable Diffusion (SD) is an advanced latent diffusion model that excels at generating high-resolution images from text [27]. Unlike operating directly in the high-dimensional image space (with dimensions of 512×512), SD operates in a representative and low-dimensional latent space (with dimensions of $4 \times 64 \times 64$). During the text-to-image generation process, SD begins by generating a random tensor in the latent space, which serves as the initial image. The text prompt is then processed by a text encoder, which converts it into 77 token embedding vectors, each consisting of 768

dimensions. The U-Net model functioning as a noise predictor takes the noisy latent and token embeddings as inputs and iteratively estimates the noise within the latent space and subtracts this estimated noise from the latent. Finally, the decoder of the Variational Autoencoder converts the denoised latent into the final image in pixel space. Additionally, SD is capable of transforming one image into another based on a given text prompt. In this scenario, SD first compresses the input image into the latent space using the encoder of the Variational Autoencoder, without losing any information. It then introduces noise to the latent representation before feeding it into the U-Net for further processing and generation.

DreamBooth is an advanced technique used for fine-tuning a pre-trained Stable Diffusion model, which can implant a subject into the output domain of the model using only 3 to 5 input images of the subject. These images are accompanied by a text prompt containing a unique identifier followed by the class name of the subject [29]. One of the most significant advantages of DreamBooth is its ability to achieve impressive fine-tuning results using such a small dataset, making it ideal for personalized image generation tasks where collecting large amounts of data may be impractical or unnecessary for the user. For example, a user can simply take a few selfies from different angles to fine-tune the Stable Diffusion model for generating an anime version of themselves.

Low-Rank Adaptation (LoRA) is an adapter-based technique commonly used to improve the efficiency of fine-tuning, especially for large models with a vast number of parameters [16]. Rather than interfering with the original model, this method adds small task-specific adapters to the pre-trained model, which can preserve the pre-trained model's general functionality while only adapting to the target task. By leveraging LoRA, we can achieve an additional reduction in training parameters and time for Stable Diffusion on the client side when combined with split learning.

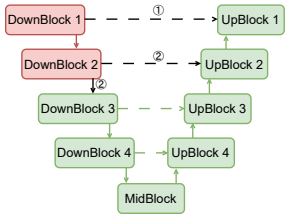
2.4 Splitting The U-Net

The key components of the Stable Diffusion v1.4 model and the forwarding process involved in fine-tuning with DreamBooth are illustrated in Fig. 2. The resulting loss obtained in each training iteration will be used to update the parameters of the U-Net. The solid black arrows in the U-Net's data forwarding flow represent the intermediate outputs that are propagated from one DownBlock to the next through max-pooling. Meanwhile, the dashed arrows between each DownBlock and its corresponding UpBlock indicate that the output from the DownBlock is also forwarded to its symmetric UpBlock.

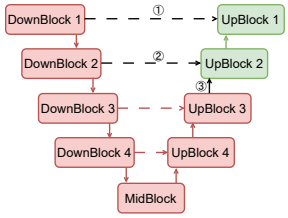
When applying split learning to Stable Diffusion, it is crucial to recognize the symmetric structure and forwarding path of the U-Net. The choice of where to split the network has a direct impact on three key factors: the computational burden on the client, the communication cost required to transmit intermediate outputs between the client and server, and the potential for data leakage from the transmitted outputs.

If the network is split after any DownBlock, the intermediate data sent from the client to the server includes the outputs of both the current DownBlock and all preceding ones. Splitting at a later DownBlock results in more intermediate data being transmitted, which increases communication costs and heightens the risk of data leakage. For example, as illustrated in Fig. 3a, when the client handles the prior components along with DownBlocks 1 and 2, while the server handles the remaining network, the client sends the outputs of both DownBlock 1 and DownBlock 2 to the server.

On the other hand, if the network is split after any UpBlock, the intermediate data transmitted includes the output of the current UpBlock and the outputs of the DownBlocks that correspond to the remaining UpBlocks. In Fig. 3b, for instance, when the network is split after UpBlock 3, the client sends the outputs of DownBlocks 1 and 2, along with the output of UpBlock 3, to the server. Unlike splitting at a DownBlock, splitting at a later UpBlock reduces the amount of intermediate data transmitted but places a heavier computational burden on the client, as fewer operations are offloaded to the server. This compromises one of the key advantages of split learning — transferring a significant portion of the computational workload to the server to ease the client's load.



(a) Splitting the U-Net after DownBlock 2



(b) Splitting the U-Net after UpBlock 3

Figure 3: Examples of splitting the U-Net in Stable Diffusion. Components highlighted in red are maintained by the split learning client, while components highlighted in green are maintained by the server.

Therefore, splitting the network after an early DownBlock is more practical and beneficial for the client, considering both its limited computational resources and the need for privacy preservation, as it significantly reduces the local computational burden and requires the transmission of only a minimal amount of intermediate results. It becomes particularly interesting to investigate whether the output of DownBlock 1 reveals any information about the input data, as this output must be sent to the server regardless of the chosen splitting strategy.

Given these considerations, our study focuses on splitting the model at DownBlock 1 and examining potential data leakage from its intermediate output. Additionally, we explore the data leakage

from subsequent DownBlocks to assess the relationship between network depth and the informativeness of the outputs.

3 DATA RECONSTRUCTION ATTACKS ON SPLIT FINE-TUNING STABLE DIFFUSION

3.1 Threat Model

The adversary can be any participant in the collaborative learning that has access to the corresponding intermediate output. In our evaluation, we focus on the split learning server as the potential adversary who directly receives the intermediate outputs from the split learning client. The goal of the adversary is to reconstruct the client's private data for fine-tuning from the received intermediate output during the fine-tuning process. Specifically for Stable Diffusion, our focus is on reconstructing the subject-oriented image data rather than the associated text prompts. During the fine-tuning process, we assume that the server is honest-but-curious, who adheres to the split learning protocol and does not disrupt the collaborative training for the sake of realizing the data reconstruction. We do not consider a malicious server, as in FSHA [26], to be an adversary since the client can easily detect and identify such behavior.

Split fine-tuning provides "pseudo-whitebox" access to the client model. While split learning effectively hides the model architectures and parameters between the client and server, as mentioned in Section 2, this feature diminishes during split fine-tuning. Before the initiation of split fine-tuning, the layers on both the client and server belong to the same pre-trained model. The server may possess knowledge of the network structure and the model parameters of the pre-trained model. This is particularly relevant for large language or vision models, where numerous checkpoints are readily available online. For instance, thousands of Stable Diffusion model checkpoints can be directly downloaded from the HuggingFace Hub [2]. Consequently, even though the server does not have direct access to the client model during fine-tuning due to model splitting, it still enjoys full access to the initial pre-trained model or any other available model checkpoints.

We use the term "pseudo-whitebox" to denote the whitebox access to the pre-trained version of the client's model prior to the split fine-tuning process. This pre-trained model has the same structure and similar parameters as the model during or after fine-tuning. Specifically, when employing LoRA [16] to fine-tune Stable Diffusion, the parameters of Stable Diffusion remain frozen while the parameters of the LoRA adapter are trained. In this scenario, the disparity between the fine-tuned model and the pre-trained model lies within the LoRA adapter trained with the client's data and the initial LoRA adapter with random parameters. Since the structure of the LoRA adapter depends on the structure of the pre-trained model, attackers do not need to know the LoRA adapter beforehand, and a pseudo-whitebox access can be performed by utilizing the pre-trained model and a randomly initialized LoRA adapter. By having pseudo-whitebox access to the client model, the server eliminates the necessity of stealing the client's model as Unsplit does [8] or constructing a functionally similar pseudo client model as PCAT does [11].

The subject-oriented data for fine-tuning is unpredictable. In split fine-tuning, a client and server can perform fine-tuning on a pre-trained model without the server having any knowledge of the downstream task or data. For instance, when utilizing DreamBooth [29] to fine-tune Stable Diffusion, a client can use a few images of any subject for personalized text-to-image generation, and these images may have significantly divergent data distributions compared to the available public datasets. This can have a detrimental impact on the performance of data reconstruction attacks in [7, 11, 13, 40], where it is assumed that the server can access a few data samples for the same learning task. Similarly, data reconstruction attacks such as GLASS [21], which rely on public datasets with similar distributions to the client's data, can also be affected.

Nevertheless, in order to explore the potential risks in worst-case scenarios, we assume that the adversary has the capability to leverage publicly available datasets to aid in data reconstruction. These public datasets may exhibit either similar or distinct distributions compared to the client's private data for fine-tuning. Additionally, during the fine-tuning process of the Stable Diffusion model through DreamBooth, the input consists of both subject-oriented images and a corresponding text prompt describing them. The resulting intermediate output from the cut layer also encodes information derived from the text prompt. We make the realistic assumption that the server has no prior knowledge of the text prompts used during the fine-tuning process.

In Table 1, we present a summary of threat models we have derived for fine-tuning Stable Diffusion, in addition to the threat models for model inversion attacks against centralized learning and split learning as described in the existing literature. This summary highlights the advantages and disadvantages in terms of the adversary's knowledge of the model and data, providing insights into the unique risks associated with fine-tuning Stable Diffusion.

Optimization-based attacks present a greater risk to split fine-tuning compared to traditional split learning, largely because the server in split fine-tuning has a more extensive understanding of the client-side model — nearly reaching the level of a centralized learning setting. The server can gain pseudo-whitebox access to the client's model, meaning it possesses substantial knowledge of the pre-trained model's structure and parameters, even if these are not fully up-to-date or evolving with training.

Training-based attacks require auxiliary data to train a model that can reconstruct input data from intermediate outputs. The less the server knows about the client model, the stronger the assumptions it must make regarding the target data. When facing training-based attacks, split fine-tuning presents both heightened and reduced vulnerabilities compared to traditional split learning. On one hand, the server's better understanding of the client-side model simplifies the construction of an auxiliary model. On the other hand, the server's limited knowledge of the client's training data makes it more difficult to effectively train the auxiliary model.

3.2 Attack Construction

As depicted in Fig. 4, we have developed both optimization-based and training-based attacks within the split fine-tuning of Stable Diffusion framework, in accordance with the practical threat model discussed earlier. These attacks leverage different levels of knowledge about the client model and its private data, as indicated by the last two rows in Table 1. During any iteration of fine-tuning, once

Table 1: The threat models of data reconstruction attacks in split fine-tuning, compared to existing threat models of successful data reconstruction attacks in other learning settings. Stronger capabilities or knowledge are indicated by darker colors. The threat models of optimization-based attacks (`Opt.`) and training-based attacks (`Tra.`) for split fine-tuning of Stable Diffusion are presented in separate rows.

Setting		Knowledge of model					Knowledge of data			
		None	Network structure	Blackbox	Pseudo-whitebox	White-box	No data	Different distribution	Similar distribution	Training data subset
Centralized learning	[22]					✓	✓			
Split learning	[8, 13]		✓				✓			
Split fine-tuning	`Opt.`				✓		✓			
Centralized learning	[7, 40]			✓					✓	✓
Split learning	[13]		✓	✓					✓	✓
	[11]	✓								✓
	[21]			✓					✓	
Split fine-tuning	`Tra.`				✓			✓	✓	

the server receives an intermediate output z^* from the client, it can attempt to reconstruct the corresponding raw data input x^* using either of these two attacks.

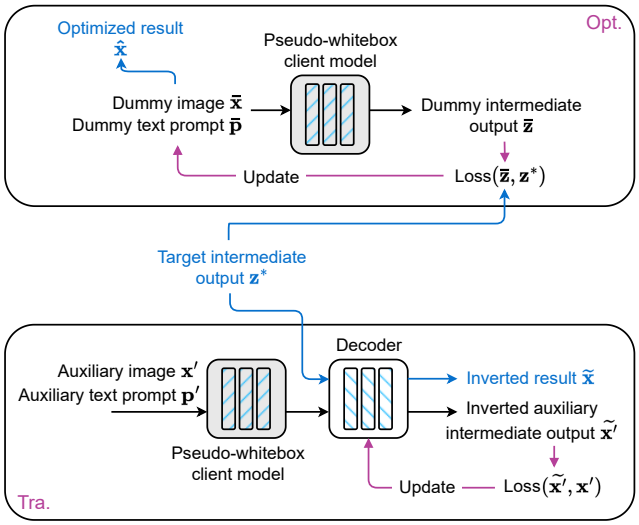


Figure 4: The optimization-based (`Opt.`) and the training-based (`Tra.`) model inversion attacks in split fine-tuning.

(`Opt.`) Our optimization-based attack. In the optimization-based attack, the server has pseudo-whitebox access to the client model $\hat{f}_c$ and the intermediate output z^* transmitted from the client. To execute the attack, the server initializes a dummy input composed of a dummy image $\bar{x}$ and a dummy text prompt $\bar{p}$. This input is then fed into the pseudo-whitebox client model $\hat{f}_c$ to generate the corresponding dummy intermediate output $\bar{z}$. A loss function is employed to calculate the Euclidean distance between the dummy intermediate output $\bar{z}$ and the target intermediate output z^*. By freezing the parameters of $\hat{f}_c$, the server can perform gradient descent on the dummy data to minimize the loss. As a result, the dummy data is optimized towards the target data.

In our initialization of the dummy image $\bar{x}$, we set all pixel values to 0.5, providing a neutral starting point for optimization. For the dummy text prompt $\bar{p}$, we initialize it using padding values, specifically the token 49407, which represents the end-of-text token in Stable Diffusion. This token is used by the model when the input prompt contains fewer tokens than its expected output dimension of 77. In such cases, Stable Diffusion fills the gap by repeatedly appending the 49407 token to ensure the prompt reaches the required length. We have two choices for the optimization objective: either optimize the dummy image alone or optimize both the dummy image and the dummy text prompt. The effectiveness of these choices will be compared later in Section 4.1. The algorithm for this attack is described in Algorithm 1.

(`Tra.`) Our training-based attack. In the training-based attack, the server leverages the pseudo-whitebox client model $\hat{f}_c$ and auxiliary data to train an inversion model f_c^{-1}. In this setup, the pseudo-whitebox client model $\hat{f}_c$ works as an encoder that converts raw data into intermediate outputs, while the inversion model f_c^{-1} acts as a decoder that converts intermediate outputs back into raw data. To execute the attack, the server inputs the auxiliary images x' and auxiliary text prompts p' to $\hat{f}_c$ and uses the corresponding auxiliary intermediate outputs z' to train f_c^{-1}. The loss function used in this process is the Euclidean distance between the output of the decoder f_c^{-1}, denoted as $\widetilde{x'}$, and the original images x'. Once the

Algorithm 1 (`Opt.`) Optimization-based model inversion attack against the split fine-tuning of Stable Diffusion

Input: Target intermediate output z^*
Output: Reconstructed image $\hat{x}^*$
Initialize: Initial dummy image $\bar{x}_1$ and dummy text prompt $\bar{p}_1$; pseudo-whitebox client model $\hat{f}_c$ with pre-trained parameters $\hat{\theta}_c$

for $i = 1$ to N **do**
  $\bar{z}_i = \hat{f}_c(\bar{x}_i, \bar{p}_i)$
  $\bar{x}_{i+1} \leftarrow \bar{x}_i - \eta \nabla_{\bar{x}_i} \|\bar{z}_i, z^*\|^2$ or $(\bar{x}_{i+1}, \bar{p}_{i+1}) \leftarrow (\bar{x}_i, \bar{p}_i) - \eta \nabla_{\bar{x}_i} \|\bar{z}_i, z^*\|^2$
end for
$\hat{x}^* \leftarrow \bar{x}_{n+1}$

decoder is trained, the server can utilize it to infer the target intermediate output z^* and generate the reconstructed target image $\widetilde{x}^*$. The algorithm for this attack is described in detail in Algorithm 2.

Algorithm 2 (`Tra.`) Training-based model inversion attack against the split fine-tuning of Stable Diffusion

Input: Target intermediate output z^*
Output: Reconstructed raw data $\widetilde{x}^*$; Trained decoder f_c^{-1}
Initialize: Pseudo-whitebox client model $\hat{f}_c$ with pre-trained parameters $\hat{\theta}_c$; decoder model f_c^{-1} with randomly generated parameters θ_c^{-1}; auxiliary text prompt p'

Decoder training phase:
for each epoch **do**
  for each batch of auxiliary data x' **do**
    $z' = \hat{f}_c(x', p')$
    $\widetilde{x'} = f_c^{-1}(z')$
    $\theta_c^{-1} \leftarrow \theta_c^{-1} - \eta \nabla_{\theta_c^{-1}} \left\| \widetilde{x'}, x' \right\|^2$
  end for
end for
Decoder inference phase:
$\widetilde{x}^* = f_c^{-1}(z^*)$

As the encoder and decoder serve opposite purposes — one encoding data into intermediate outputs, the other reconstructing the raw data — the decoder can be designed by reversing the structure of the encoder or by using layers that operate in the opposite direction. For example, in PyTorch, the `Conv2d` and `ConvTranspose2d` layers are often paired in encoder-decoder architectures, where the `ConvTranspose2d` layers upscale the latent representation back to the original image dimensions, effectively reversing the downsampling performed by the `Conv2d` layers.

In designing the decoder model architecture, since the adversary has pseudo-whitebox access to the client-side model, we can leverage the transparency of the encoder's architecture to construct a decoder by mirroring the encoder's structure. By reversing the order of these layers, the decoder can be constructed to effectively learn the inverse mapping from intermediate outputs back to the original data. In Table 2, we provide a brief overview of our decoder model design for Stable Diffusion when the U-Net is split after a specific DownBlock. This design will be used in the subsequent evaluations to assess its performance in reconstructing client's data used for fine-tuning.

Table 2: The structure of the decoder network for Stable Diffusion in our training-based attack.

Target model	Cut layer (after)	Decoder structure

Stable Diffusion	U-Net's `DownBlock` 1/2/3	`UNetMidBlock2DCrossAttn`
		`CrossAttnUpBlock2D`
		`nn.GroupNorm`
		`nn.SiLU`
		`nn.ConvTranspose2d`
		`nn.ConvTranspose2d`
		`nn.Conv2d`

4 EVALUATION OF DATA RECONSTRUCTION ATTACKS IN SPLIT FINE-TUNING OF STABLE DIFFUSION

In this section, we conduct separate evaluations of the performance of our optimization-based and training-based data reconstruction attacks. Our analysis reveals that split fine-tuning with the Stable Diffusion model exhibits distinctive vulnerabilities to data reconstruction compared to conventional scenarios.

The pre-trained Stable Diffusion v1.4 model used in our experiments is directly loaded from HuggingFace's `diffusers` library. The fine-tuning process, employing DreamBooth [29] and LoRA [16] techniques, is facilitated by the PEFT framework [23]. To prepare the client's data for fine-tuning, we select high-quality images with a resolution of at least 512×512 from the DreamBooth Dataset [29] and online resources [1]. In our experiments, we utilize four datasets for different fine-tuning tasks, each consisting of N images focused on a specific subject: a duck toy ($N = 4$), a dog ($N = 5$), Elon Musk ($N = 20$), and Tim Cook ($N = 20$). In each fine-tuning task, the model is trained iteratively using one image of the subject per step, cycling through the dataset. Each image is used for training a total of 100 times, meaning there are $100 \times N$ training steps in total. While data reconstruction can occur at any training step, we choose to perform the attack around step 400 in all experiments to ensure consistency, where each image in the dataset has been used approximately $400/N$ times for training.

4.1 The Optimization-Based Attack

In all of the subsequent experiments, we have set the number of optimization iterations to 20,000, which is adequate for achieving convergence through our observations.

Varying levels of knowledge about the client model. In Fig. 5, we demonstrate the benefits of split fine-tuning when the server has pseudo-whitebox access to the client model, which refers to having access to the pre-trained version of the client model before fine-tuning. We compare this scenario to having full whitebox access, where the server has complete access to the latest client model

at any given point during the fine-tuning process. Additionally, we consider the case where the server is aware of the client model's structure and employs UnSplit [8] to construct a functionality similar to that of the client model for data reconstruction.

UnSplit typically starts with a clone model mirroring the structure of the client model, yet with random parameters. It then reconstructs both the parameters and the target images from the intermediate outputs. However, this attack is likely to fail when dealing with more complex client model structures and learning tasks, as the solution space could become excessively expansive for convergence through gradient descent optimization [11]. In this experiment, we employ Stable Diffusion v1.1 as the starting point for UnSplit's clone model to facilitate its performance.

We note that pseudo-whitebox access renders split learning more vulnerable to model inversion attacks compared to conventional split learning setups where the server may or may not even know the network structure. The efficacy of the pseudo-whitebox attack is comparable to that of the whitebox attack. Therefore, in contrast to conventional split learning, split fine-tuning favors the adversary by obviating the necessity to train a functionally similar client model from scratch.

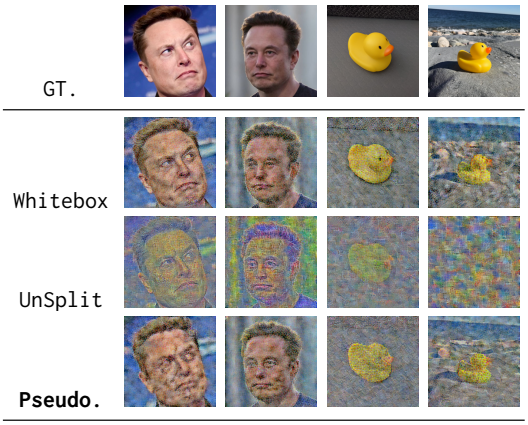


Figure 5: A comparison of reconstructed images with different levels of the adversary's knowledge of the client's model.

Different choices for dummy text prompts and optimization objectives. We investigate the effectiveness of our two methods for initializing the dummy text prompt and two options for optimization objectives, compare them with the direct use of the actual text prompt employed during fine-tuning as the dummy text prompt, and present a representative example in Fig. 6. The actual text prompt used for fine-tuning the target image on the left is a sequence of 7 tokens, while the one for the right image is twice as long, including 14 tokens.

As we can see from Fig. 6, optimizing the image and text prompt together can harm performance, especially when the text prompt is already optimal. When the text prompt used for fine-tuning is longer, the reconstruction quality tends to be relatively worse. We attribute this to the fact that the text encoder of Stable Diffusion produces a fixed set of 77 token embedding vectors for a given text prompt; if the given text prompt has fewer tokens, the remaining

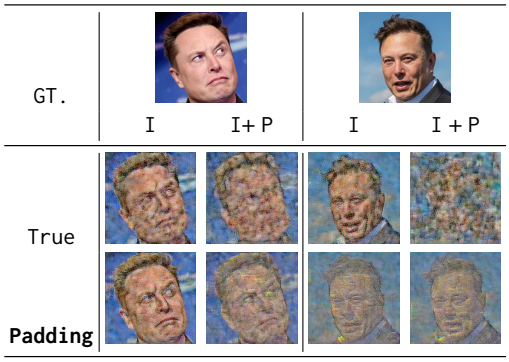


Figure 6: A comparison of reconstructed images considering the adversary's varying knowledge of the text prompts and different optimization objectives. The two ways of initializing the dummy text prompt used for the optimization-based attack are represented by `True` and `Padding`. The optimization of the dummy image only is denoted by `I`, while the optimization of both the dummy image and dummy text prompt is denoted by `I+P`.

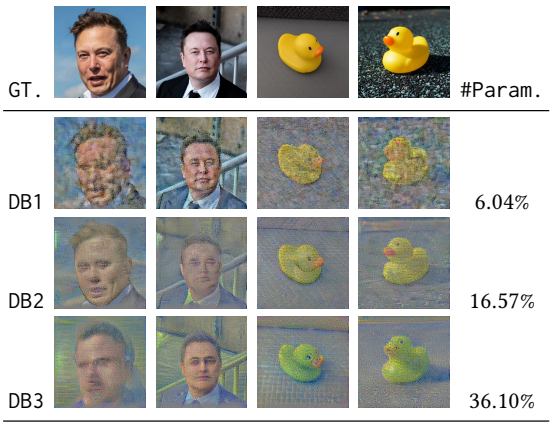


Figure 7: A comparison of reconstructed images when the model is split at different layers. `DB` stands for DownBlock. The numbers in the column of `#Param.` represents the ratio of the training parameters at the client to the total number of training parameters under different cut layers.

spaces are filled with padding values, i.e., 49407. If the actual text prompt used for fine-tuning is short, it results in numerous padding values in the token embeddings. In such cases, `Padding` dummy text prompt closely resembles the token embeddings of the actual text prompt. Conversely, if the actual prompt is sufficiently long, a `Padding` dummy text prompt may not assist the attacker in reconstruction. Given this finding, we choose to employ the `Padding` dummy text prompt and optimize the dummy image only for all subsequent evaluations of our optimization-based attack.

Different cut layers. We examine the risk of information leakage when the network is cut at different layers and show the reconstructed images in Fig. 7. The numbers listed under `Params.`

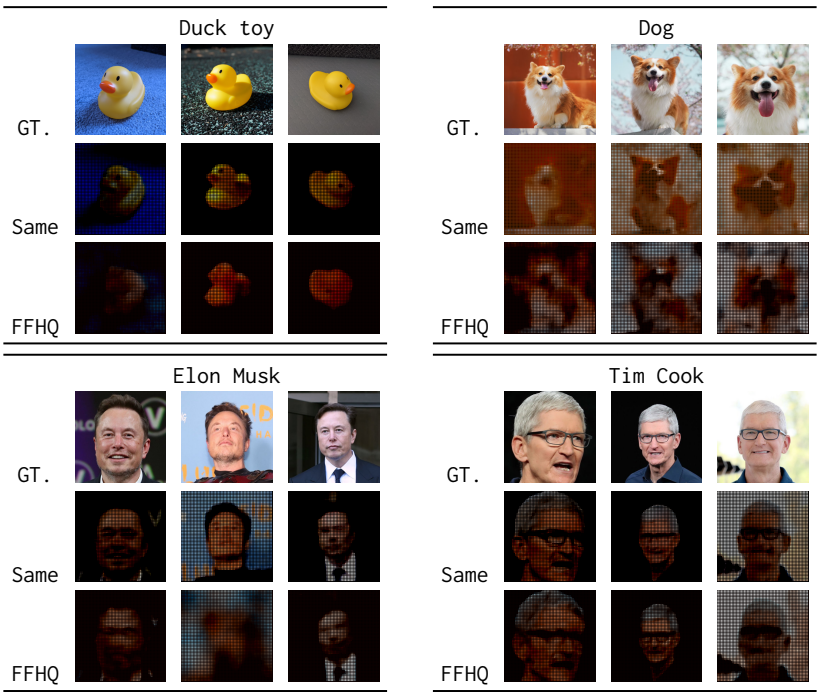


Figure 8: Results of our training-based attack (`Tra.`) on different fine-tuning tasks with different auxiliary data. In the row labeled `Same`, the auxiliary dataset used is identical to the target private dataset used for fine-tuning.

represent the ratio of training parameters at the client to the total number of parameters in the model for different cut layers. We present this to illustrate the trade-off between data leakage from intermediate output and computational overhead on the client side when the model is split at various cut layers.

Unlike the findings in previous attacks on traditional convolutional neural networks [8, 11, 13], we observe that in Stable Diffusion, as the model split depth increases, the data leakage from the intermediate output does not decrease. The reconstructed images exhibit fewer noises, and features become more pronounced. What's worse, the forwarding mechanism of the U-Net necessitates the sharing of intermediate outputs from previous network blocks with the server, even when the model is split at a later cut layer. This suggests that, whether considering reducing computation overhead or enhancing privacy preservation for the client, splitting the Stable Diffusion model after a shallow layer like DownBlock 1 is an optimal choice. Despite this, it highlights the need for a defense mechanism to protect the intermediate outputs from all blocks.

4.2 The Training-Based Attack

Auxiliary public data. We investigate the training-based attack on four fine-tuning tasks and compare the reconstruction performance when employing a public dataset FFHQ [18] versus directly using the same fine-tuning dataset to train the decoder. FFHQ is a high-quality (at 1024×1024 resolution) image dataset containing human faces with considerable variations in age, ethnicity, and image background. It also provides extensive coverage of facial accessories such as eyeglasses, sunglasses, and hats. The data distribution of FFHQ is expected to be similar to that of celebrities' faces but distinct from that of objects and animals.

As depicted in the row labeled `Same` in Fig. 8, when the server can acquire a subset of the fine-tuning data for its decoder training, the reconstruction precision is very promising. However, the reality is that subject-oriented fine-tuning data can be heterogeneous, and the server may only have the assistance of public datasets where the data distribution can vary. This point is supported by the results of using FFHQ as the auxiliary data to reconstruct the duck toy and the dog. The reconstructed duck toy and dog images using FFHQ data exhibit noticeably worse quality than the images reconstructed using the same fine-tuning dataset, indicating the negative impact of using heterogeneous fine-tuning datasets. Even when the data distribution is similar, as in the case of using FFHQ to reconstruct celebrities, the disparities between the `Same` and `FFHQ` reconstructed images are smaller compared to the previous case yet not all facial features can be restored, recognizing the identity remains challenging.

Different cut layers. We also analyze the reconstruction performance of the training-based attack at different model split depths, as shown in Fig. 9. For this experiment, we use celebrities' faces as the fine-tuning data and FFHQ as the auxiliary dataset, both sharing a similar distribution. It's noticeable that as the cut layer deepens, faces in the images become more distorted, and features appear more fuzzy. Overall, the training-based attack exhibits lower reconstruction capability than the optimization-based attack, especially when the distribution of the auxiliary data diverges significantly from that of the actual fine-tuning data.

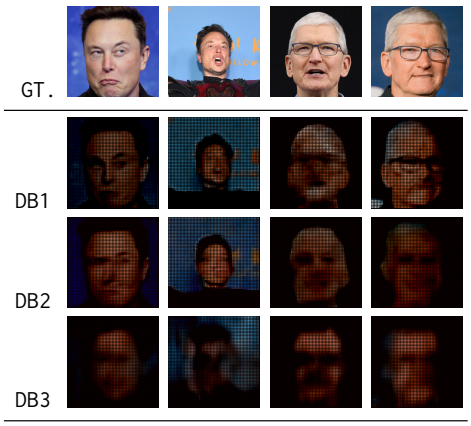


Figure 9: A comparison of reconstructed images when the model is split at different layers. The auxiliary dataset used in this scenario is FFHQ.

5 OUR PROPOSED DEFENSE

Thus far, it has become clear that split fine-tuning of large generative models, such as Stable Diffusion, does not fully preserve privacy. As a result, developing a robust defense mechanism to protect the intermediate outputs, which may contain sensitive information from private input data, is crucial. To address the heightened risk of data leakage in split fine-tuning, we have designed a defense mechanism that strikes a balance between ensuring privacy protection and maintaining minimal impact on model utility and training efficiency.

5.1 The Design

Recent work such as [39, 44] employed dropout regularization techniques, which was initially introduced in [31] to prevent deep neural networks from overfitting, as a way to protect shared gradients from gradient leakage attacks. To this end, we propose to apply an additional dropout layer at the client side on the image latents after the Variational Autoencoder described in Fig. 2. Instead of setting a fixed dropout rate, we devise a method to quantify the information leakage within the intermediate output in comparison to the input image, in order to provide adaptive protection.

Quantifying information leakage through self-attack. It is crucial to quantify the information leakage to accurately assess the extent to which sensitive information from the input data could be inferred by an adversary. However, quantifying the information leakage $\mathcal{I}(z, x)$ between the intermediate output z and the input data x is inherently complex, particularly when working with high-dimensional data and sophisticated models like Stable Diffusion [15]. Inspired by a separate study on locally supervised learning [37], we model the relationship between information leakage $\mathcal{I}(z, x)$ and the expected reconstruction error $\mathcal{R}(x|z)$, which can be approximated as follows:

$$\begin{aligned} \mathcal{I}(z, x) &= \mathcal{H}(x) - \mathcal{H}(x|z) \geq \mathcal{H}(x) - \mathcal{R}(x|z) \\ &\approx \max[\mathcal{H}(x) - \mathcal{R}(x|z)], \end{aligned} \tag{1}$$

where $\mathcal{H}(x)$ and $\mathcal{H}(x|z)$ denote the marginal and conditional entropy of s, respectively.

Since $\mathcal{H}(x)$ can be treated as a constant [15, 37], we focus on estimating the reconstruction error $\mathcal{R}(x|z)$, which serves as a proxy for the amount of information that can be leaked from the intermediate output. To achieve this, we let the client train an auxiliary decoder f_R^{-1} to reverse the operation of the client-side model and reconstruct the original input from the intermediate output z. This decoder is trained on the client's own data before fine-tuning begins, making it highly effective at estimating the reconstruction error for the specific data distribution of the client.

The architecture of the decoder follows the same structure, as shown in Table 2, and is designed to efficiently map the intermediate output z back to the input x. We adopt the mean square error between the reconstructed data and the input data to represent $\mathcal{R}(x|z)$. This functions as if the client conducts a self-attack and utilizes the reconstruction result as a precaution. As the client has comprehensive knowledge of their data, they can train a more efficient decoder compared to other potential attackers. By leveraging the self-attacking mechanism, the client can measure the level of potential information leakage before sending intermediate results to the server.

Adaptive dropout layer for image latents. During each step of fine-tuning, the client forwards the input data to the cut layer and obtains the reconstruction error $\mathcal{R}(x|z)$ by inferencing the trained decoder f_R^{-1}. Once the reconstruction error $\mathcal{R}(x|z)$ is determined, we use $-\mathcal{R}(x|z)$ to represent $\mathcal{I}(z, x)$ based on Eq. (1). The next step is to adaptively protect the intermediate outputs by adjusting the dropout rate based on the estimated risk of information leakage.

To achieve this, we apply the Sigmoid function to convert the information leakage to the dropout rate p as shown below:

$$p = \frac{1}{1 + e^{-\mathcal{R}(x|z)}}, \tag{2}$$

where the value of p is restricted to the range $[0, 0.5]$, ensuring that the dropout rate remains non-negative and capped. A higher reconstruction error indicates lower information leakage, leading to a smaller dropout rate, which preserves more information for model utility. Conversely, a smaller reconstruction error implies greater potential for leakage, prompting the use of a higher dropout rate to mask the sensitive information more effectively.

Unlike other defenses that perturb the intermediate output of the cut layer [32], we opt to apply the dropout layer to the image latents before they are fed to the U-Net, to ensure that sensitive information is obscured at an earlier stage. This decision stems from the understanding that focusing solely on safeguarding the intermediate output at the cut layer may not be sufficient to fully prevent data leakage in Stable Diffusion models, where intermediate outputs from preceding network blocks must also be transmitted to the server, potentially exposing sensitive information. By applying the dropout earlier, we address the risk of leakage from multiple stages of the network, thereby enhancing overall privacy.

5.2 Evaluation of Defenses

For all scenarios, the NoiseDefense's noise and the dropout layer in our defense are applied to the hidden states or pixel-related tensors only in the intermediate output. Following the experimental setup of the original paper, we set the weight of the distance correlation

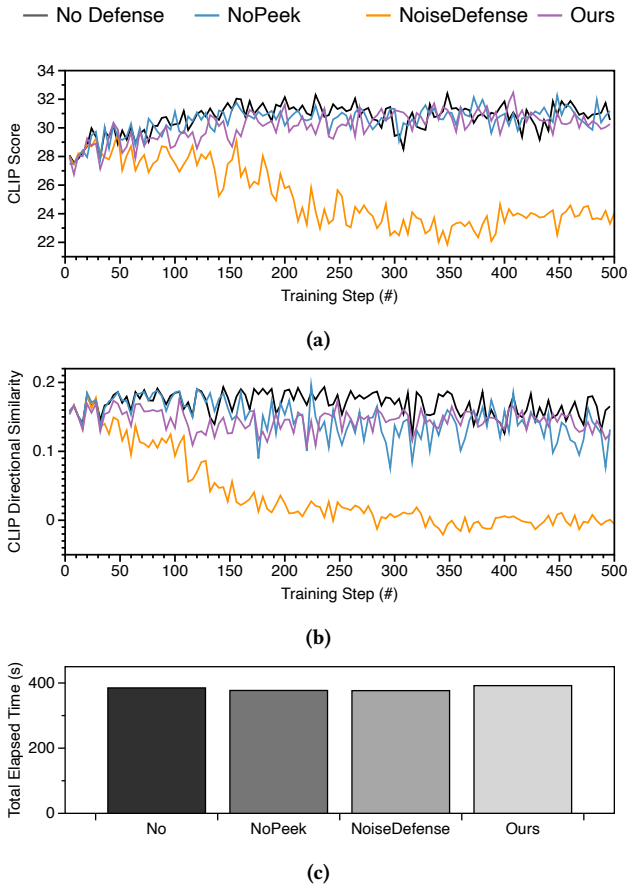


Figure 10: The impact of employing different defense mechanisms on the fine-tuning of Stable Diffusion with the duck toy images.

term in the NoPeek loss function to 1. For NoiseDefense, we set the noise scale to 10.

Impact of defenses on model utility and training efficiency. Fig. 10 plots the evolution of different metrics over the training steps during the fine-tuning of the Stable Diffusion model on the duck toy images. We apply the CLIP score to assess the consistency between the text prompt for inference and the image generated by the model from a specific checkpoint [14]. In addition, we utilize CLIP directional similarity to evaluate the coherence between the ground truth image and the generated image, paired with their respective ground truth description and the prompt for inference [10]. The higher the CLIP score or CLIP directional similarity, the greater the coherence, and consequently, the better the model utility through fine-tuning. The prompt used for DreamBooth fine-tuning is "A photo of a [V] duck toy", and the prompt for inference is "A photo of a [V] duck toy swimming in a pool". The scores are averaged over 10 generated images corresponding to the inference prompt, sampled at every 5th checkpoint throughout the 500 fine-tuning steps, to reduce bias. All the experiments are conducted on the same machine equipped with an Intel i7-13700K 16-core processor, 128GB DDR5 physical memory, and an NVIDIA GeForce RTX 4090 24GB GDDR6 video card, in the environment of Ubuntu Linux 22.04 with CUDA version 12.3.

As shown in Figs. 10a and 10b, integrating NoiseDefense into fine-tuning has the most detrimental effect on learning, significantly reducing both CLIP score and CLIP directional similarity. In contrast, NoPeek and our defense cause only a slight decline in CLIP directional similarity compared to the baseline with no defense. As shown in Fig. 10c, the total elapsed time remains similar across all methods, with our defense causing a slight slowdown of 1.8%, though the increase is marginal. This suggests that while our method introduces minor computational overhead, it does not significantly impact fine-tuning efficiency. The same trend is observed for the brain MRI dataset, as demonstrated in Table 4 and Fig. 15 in Appendix A.3. In addition, more results on the impact of fine-tuning effectiveness and efficiency for two smaller models, ResNet-18 [12] and ViT [6], can be found in Appendix A.3, serving as supporting evidence for our analysis.

The effectiveness of defenses against data reconstruction attacks. In the effectiveness evaluations, we consider realistic settings for the server, which has a pseudo-whitebox client model and access to public datasets such as FFHQ. The Stable Diffusion model is split after DownBlock 1. While DreamBooth fine-tuning focuses on a small, subject-oriented dataset, these images predominantly feature a single object against a simple background. A more complex dataset may influence the effectiveness of the attacks and defenses we have studied. To investigate this, we conduct additional experiments on fine-tuning Stable Diffusion using medical diagnostic images, which typically contain multiple structures and more complex spatial relationships. Specifically, we utilize a high-quality dataset of 13.4K brain MRI images (512×512) with captions from HuggingFace[1]. This dataset was released after the pre-trained Stable Diffusion checkpoint and was included in its pre-training phase. For this fine-tuning task, we set the batch size to 4 and the training epoch to 1.

Brain MRI					
Attack	**Defense**	**MSE↑**	**LPIPS↑**	**PSNR↓**	**SSIM↓**
`Opt.`	No Defense	1.7063	0.6011	-2.3206	-0.2346
	NoPeek	1.7202	0.7079	-2.3557	-0.0905
	NoiseDefense	1.7231	0.7118	-2.363	-0.0781
	Ours	1.7046	0.6349	-2.3163	-0.2161
`Tra.`	No Defense	0.0639	0.5888	12.1806	0.4149
	NoPeek	0.0642	0.5901	12.1437	0.4254
	NoiseDefense	0.1158	0.6488	9.4463	0.1178
	Ours	0.0658	0.5866	12.0605	0.3667

Table 3: Evaluation of different defense mechanisms against optimization-based attack (`Opt.`) and training-based attack (`Tra.`) on brain MRI images. Higher MSE and LPIPS values indicate stronger defenses, while lower PSNR and SSIM values suggest increased resistance to reconstruction.

[1]https://huggingface.co/datasets/txz32102/MRI_512

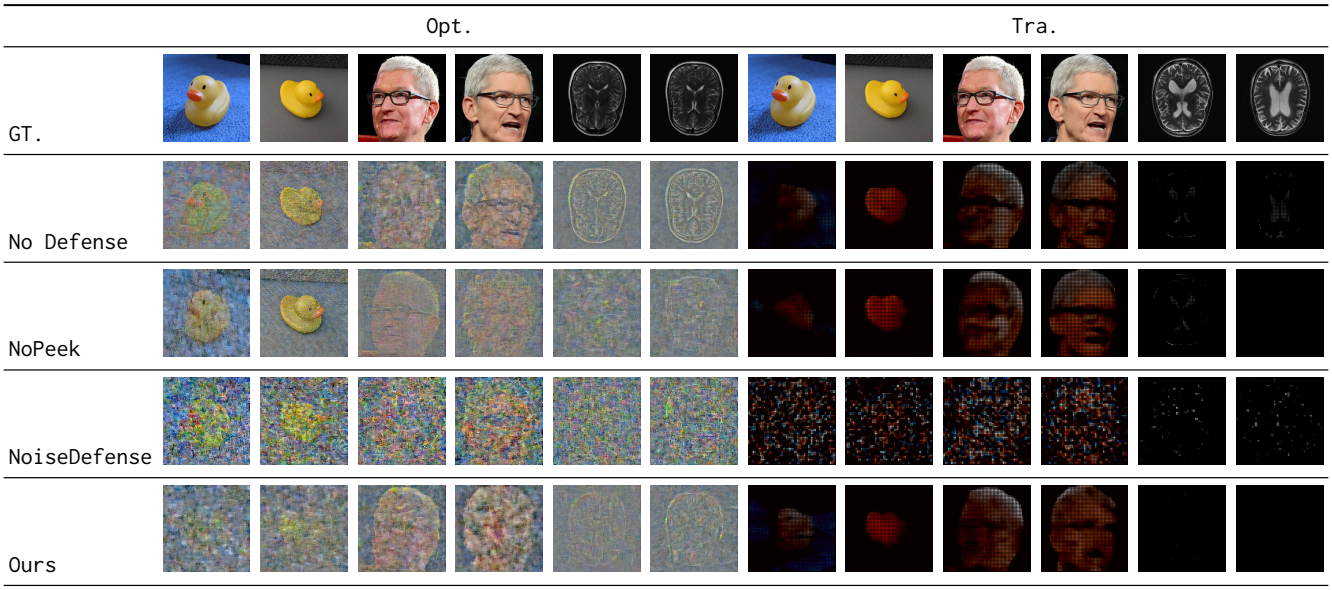


Figure 11: Comparison of reconstructed images from the optimization-based attack (`Opt.`) and training-based attack (`Tra.`) under different defense mechanisms.

We present example reconstructed images from the duck toy, Tim Cook face, and brain MRI datasets, generated by the two attacks under different defenses, in Fig. 11. To provide a more comprehensive analysis across a broader dataset, we also report statistical results over 100 reconstructed brain MRI images per scenario in Table 3, comparing them to the ground truth using multiple evaluation metrics, including mean squared error (MSE), learned perceptual image patch similarity (LPIPS), peak signal-to-noise ratio (PSNR), and structural similarity index measure (SSIM). These metrics evaluate the difference between the reconstructed and target images from multiple perspectives: MSE quantifies pixel-level difference between images, LPIPS quantifies perceptual dissimilarity based on deep neural network embeddings [42], PSNR measures signal fidelity relative to noise, and SSIM captures structural similarity by analyzing brightness, contrast, and spatial patterns [38]. These metrics are widely used in prior studies to assess image quality and similarity in data reconstruction attacks and defenses [11, 17, 21, 36]. Additional relevant results can be found in Appendix A.4.

Without any defense, in Fig. 11, brain MRI images exhibit the highest resistance to reconstruction attacks compared to the duck toy and Tim Cook face datasets, particularly in the training-based attack (`Tra.`), where the reconstructed images appear mostly black, with only faint outlines and minimal structural details visible. Unlike single-object images or human faces, which have well-defined shapes and high-contrast features that aid reconstruction, MRI images consist of complex textures and subtle variations, making them significantly harder to recover. However, the optimization-based attack (`Opt.`) remains more effective, managing to recover blurred yet discernible structures.

NoiseDefense provides the strongest protection, as seen in the heavily distorted reconstructed images, making it nearly impossible to extract any meaningful features from the original data. This is further confirmed by the high MSE and LPIPS values in Table 3. However, this comes at a severe cost to model utility, as NoiseDefense injects excessive noise into intermediate outputs, significantly degrading fine-tuning performance. In contrast, NoPeek offers moderate protection by enforcing distance correlation constraints, reducing reconstruction quality compared to no defense. However, as seen in Fig. 11, it does not fully obfuscate intermediate representations, allowing some details to remain recoverable. Our defense, on the other hand, provides stronger protection than NoPeek by further degrading reconstruction quality, ensuring that critical image features cannot be extracted. At the same time, unlike NoiseDefense, which achieves the strongest privacy protection at the expense of fine-tuning effectiveness, our defense maintains a balanced trade-off, preserving model utility while preventing key feature extraction. This makes our approach a more practical and efficient defense strategy, offering strong privacy protection without compromising learning performance.

6 CONCLUDING REMARKS

In this paper, we conducted a thorough evaluation of the risk of data leakage when using split learning to fine-tune large generative AI models, with a specific focus on Stable Diffusion models. We show that the attack is more effective as the adversary has access to the pre-trained client model, and due to the unique U-Net structure of Stable Diffusion models, splitting the model at a later layer may not help mitigate data leakage. With our proposed defense mechanism, which can be extended to safeguard other models as well, we are able to ensure that split learning remains a privacy-preserving training paradigm for collaborative fine-tuning when it comes to large generative AI models, even under sophisticated attacks.

REFERENCES

[1] [n. d.]. gettyimages. https://www.gettyimages.ca/.

[2] [n. d.]. The Hugging Face Hub. https://huggingface.co/models.

[3] Martin Abadi, Andy Chu, Ian Goodfellow, H. Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. 2016. Deep Learning with Differential Privacy. In *Proc. the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS)*. 308–318.

[4] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. ImageNet: A Large-Scale Hierarchical Image Database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*. 248–255.

[5] Tim Dockhorn, Tianshi Cao, Arash Vahdat, and Karsten Kreis. 2023. Differentially Private Diffusion Models. *Transactions on Machine Learning Research* (2023).

[6] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. 2021. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. In *Proc. International Conference on Learning Representations (ICLR)*.

[7] Alexey Dosovitskiy and Thomas Brox. 2016. Inverting Visual Representations with Convolutional Networks. In *Proc. the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 4829–4837.

[8] Ege Erdoğan, Alptekin Küpçü, and A Ercüment Çiçek. 2022. Unsplit: Data-Oblivious Model Inversion, Model Stealing, and Label Inference Attacks against Split Learning. In *Proc. the 21st Workshop on Privacy in the Electronic Society (WPES)*. 115–124.

[9] Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. 2015. Model Inversion Attacks That Exploit Confidence Information and Basic Countermeasures. In *Proc. the 22nd ACM SIGSAC Conference on Computer and Communications Security (CCS)*. 1322–1333.

[10] Rinon Gal, Or Patashnik, Haggai Maron, Amit H Bermano, Gal Chechik, and Daniel Cohen-Or. 2022. StyleGAN-NADA: CLIP-Guided Domain Adaptation of Image Generators. *ACM Transactions on Graphics (TOG)* 41, 4 (2022), 1–13.

[11] Xinben Gao and Lan Zhang. 2023. PCAT: Functionality and Data Stealing from Split Learning by Pseudo-Client Attack. In *Proc. the 32nd USENIX Security Symposium*. 5271–5288.

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. In *Proc. the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 770–778.

[13] Zecheng He, Tianwei Zhang, and Ruby B. Lee. 2019. Model Inversion Attacks against Collaborative Inference. In *Proc. the 35th Annual Computer Security Applications Conference*. 148–162.

[14] Jack Hessel, Ari Holtzman, Maxwell Forbes, Ronan Le Bras, and Yejin Choi. 2021. CLIPScore: A Reference-free Evaluation Metric for Image Captioning. In *Proc. Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

[15] R Devon Hjelm, Alex Fedorov, Samuel Lavoie-Marchildon, Karan Grewal, Phil Bachman, Adam Trischler, and Yoshua Bengio. 2019. Learning Deep Representations by Mutual Information Estimation and Maximization. In *Proc. International Conference on Learning Representations (ICLR)*.

[16] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. LoRA: Low-Rank Adaptation of Large Language Models. In *Proc. International Conference on Learning Representations (ICLR)*.

[17] Yangsibo Huang, Samyak Gupta, Zhao Song, Kai Li, and Sanjeev Arora. 2021. Evaluating Gradient Inversion Attacks and Defenses in Federated Learning. *Advances in Neural Information Processing Systems (NeurIPS)* 34 (2021), 7232–7241.

[18] Tero Karras, Samuli Laine, and Timo Aila. 2019. A Style-Based Generator Architecture for Generative Adversarial Networks. In *Proc. the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 4401–4410.

[19] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. [n. d.]. CIFAR-100 (Canadian Institute for Advanced Research). https://www.cs.toronto.edu/~kriz/cifar.html. Accessed: 2023-10.

[20] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. 1998. Gradient-Based Learning Applied to Document Recognition. *Proc. the IEEE* 86, 11 (1998), 2278–2324.

[21] Ziang Li, Mengda Yang, Yaxin Liu, Juan Wang, Hongxin Hu, Wenzhe Yi, and Xiaoyang Xu. 2023. GAN You See Me? Enhanced Data Reconstruction Attacks against Split Inference. In *Advances in Neural Information Processing Systems (NeurIPS)*.

[22] A. Mahendran and A. Vedaldi. 2015. Understanding Deep Image Representations by Inverting Them. In *Proc. the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 5188–5196.

[23] Sourab Mangrulkar, Sylvain Gugger, Lysandre Debut, Younes Belkada, and Sayak Paul. 2022. PEFT: State-of-the-Art Parameter-Efficient Fine-Tuning Methods. https://github.com/huggingface/peft.

[24] Ilya Mironov. 2017. Rényi Differential Privacy. In *IEEE 30th Computer Security Foundations Symposium (CSF)*. 263–275.

[25] T. Orekondy, B. Schiele, and M. Fritz. 2019. Knockoff Nets: Stealing Functionality of Black-Box Models. In *Proc. the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 4949–4958.

[26] Dario Pasquini, Giuseppe Ateniese, and Massimo Bernaschi. 2021. Unleashing the Tiger: Inference Attacks on Split Learning. In *Proc. the 28th ACM SIGSAC Conference on Computer and Communications Security (CCS)*. 2113–2129.

[27] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. 2022. High-Resolution Image Synthesis with Latent Diffusion Models. In *Proc. the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 10684–10695.

[28] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. 2015. U-Net: Convolutional Networks for Biomedical Image Segmentation. In *Proc. International Conference on Medical Image Computing and Computer-Assisted Intervention (MICCAI)*. 234–241.

[29] Nataniel Ruiz, Yuanzhen Li, Varun Jampani, Yael Pritch, Michael Rubinstein, and Kfir Aberman. 2023. Dreambooth: Fine Tuning Text-to-Image Diffusion Models for Subject-Driven Generation. In *Proc. the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 22500–22510.

[30] Karen Simonyan and Andrew Zisserman. 2014. Very Deep Convolutional Networks for Large-Scale Image Recognition. In *Proc. International Conference on Learning Representations (ICLR)*.

[31] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research* 15, 56 (2014), 1929–1958. http://jmlr.org/papers/v15/srivastava14a.html

[32] Tom Titcombe, Adam J Hall, Pavlos Papadopoulos, and Daniele Romanini. 2021. Practical Defences against Model Inversion Attacks for Split Neural Networks. *arXiv preprint arXiv:2104.05743, ICLR Workshop on Distributed and Private Machine Learning* (2021).

[33] Florian Tramèr, Fan Zhang, Ari Juels, Michael K. Reiter, and Thomas Ristenpart. 2016. Stealing Machine Learning Models via Prediction APIs. In *Proc. the 25th USENIX Security Symposium*. 601–618.

[34] Praneeth Vepakomma, Otkrist Gupta, Tristan Swedish, and Ramesh Raskar. 2018. Split Learning for Health: Distributed Deep Learning without Sharing Raw Patient Data. *arXiv preprint arXiv:1812.00564, ICLR AI for Social Good Workshop* (2018).

[35] Praneeth Vepakomma, Abhishek Singh, Otkrist Gupta, and Ramesh Raskar. 2020. NoPeek: Information Leakage Reduction to Share Activations in Distributed Deep Learning. In *Proc. the 20th International Conference on Data Mining Workshops (ICDMW)*. 933–942.

[36] Fei Wang, Ethan Hugh, and Baochun Li. 2023. More than Enough is Too Much: Adaptive Defenses against Gradient Leakage in Production Federated Learning. In *Proc. IEEE Conference on Computer Communications (INFOCOM)*. 1–10.

[37] Yulin Wang, Zanlin Ni, Shiji Song, Le Yang, and Gao Huang. 2021. Revisiting Locally Supervised Learning: an Alternative to End-to-end Training. In *Proc. International Conference on Learning Representations (ICLR)*.

[38] Zhou Wang, A.C. Bovik, H.R. Sheikh, and E.P. Simoncelli. 2004. Image Quality Assessment: From Error Visibility to Structural Similarity. *IEEE Transactions on Image Processing* 13, 4 (2004), 600–612.

[39] Wenqi Wei, Ling Liu, Margaret Loper, Ka-Ho Chow, Mehmet Emre Gursoy, Stacey Truex, and Yanzhao Wu. 2020. A Framework for Evaluating Client Privacy Leakages in Federated Learning. In *Proc. 25th European Symposium on Research in Computer Security (ESORICS)*. 545–566.

[40] Ziqi Yang, Jiyi Zhang, Ee-Chien Chang, and Zhenkai Liang. 2019. Neural Network Inversion in Adversarial Setting via Background Knowledge Alignment. In *Proc. the 26th ACM SIGSAC Conference on Computer and Communications Security (CCS)*. 225–240.

[41] Ashkan Yousefpour, Igor Shilov, Alexandre Sablayrolles, Davide Testuggine, Karthik Prasad, Mani Malek, John Nguyen, Sayan Ghosh, Akash Bharadwaj, Jessica Zhao, Graham Cormode, and Ilya Mironov. 2021. Opacus: User-Friendly Differential Privacy Library in PyTorch. In *NeurIPS 2021 Workshop Privacy in Machine Learning*.

[42] R. Zhang, P. Isola, A. A. Efros, E. Shechtman, and O. Wang. 2018. The Unreasonable Effectiveness of Deep Features as a Perceptual Metric. In *Proc. the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 586–595.

[43] Y. Zhang, R. Jia, H. Pei, W. Wang, B. Li, and D. Song. 2020. The Secret Revealer: Generative Model-Inversion Attacks Against Deep Neural Networks. In *Proc. the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 250–258.

[44] Yanchong Zheng. 2021. Dropout against Deep Leakage from Gradients. *arXiv preprint arXiv:2108.11106* (2021).

A APPENDIX

A.1 Further Discussion on the Threat Model Assumption

In the attacker's assumption of knowledge about the client-side model, model architecture is a crucial factor [8, 26, 33]. Minor modifications, such as small changes in layer width, activation functions, or dropout rates, may still allow the attack to succeed but with degraded reconstruction quality, as the attacker's model only approximates the intermediate features. In contrast, major modifications, including reordering layers, adding non-linear transformations, or changing feature dimensionality, can significantly disrupt feature alignment, often rendering the attack ineffective. The greater the architectural divergence, the harder it becomes for the attacker to reconstruct meaningful data, highlighting architectural modifications as a viable defense mechanism.

For the optimization-based attack, the attacker refines a dummy image using a pseudo-whitebox client model to match the target intermediate outputs. Significant architectural modifications disrupt this alignment, degrading accuracy or making optimization infeasible. Similarly, in the training-based attack, the attacker trains a decoder to map the client's intermediate outputs back to training data by leveraging a public dataset and a pseudo-whitebox client model. If the client modifies its architecture, the feature space of the intermediate outputs shifts, preventing the decoder from generalizing. However, the attacker could train a functionally similar model, even without direct knowledge of the client's architecture [11]. In this case, the learned mapping remains feasible but still less effective compared to having full architectural knowledge. Maintaining or improving attack effectiveness while relaxing the attacker's assumptions about data or model knowledge remains a challenging but important direction for further research.

A.2 Discussion on the Application of DP-SGD in Split Learning for Fine-Tuning Stable Diffusion Models

We evaluate the use of differentially private stochastic gradient descent (DP-SGD) citeabadi2016dp as a general defense for protecting Stable Diffusion against data reconstruction attacks during its split fine-tuning. We implement (ε, δ)-DP using the DP-SGD library in PyTorch, Opacus [41], for both training and privacy accounting [5], setting the target failure probability to $\delta = 10^{-5}$. By adjusting the noise multiplier σ in Opacus, which controls the amount of noise added to gradients, the framework provides an estimate of ε using the Rényi differential privacy [24] accountant. We present attack performance under different noise multiplier σ settings in Fig. 12, along with the corresponding evolution of ε in Fig. 13. On top of Fig. 10, we also overlay the CLIP score and CLIP directional similarity for checkpoints fine-tuned with DP-SGD using $\sigma = 20$, as shown in Fig. 14.

The attack performance under DP-SGD fine-tuning remains largely unaffected, regardless of the privacy budget ε. Even with strong privacy protection given by low ε (high noise multiplier σ), the reconstructed images still retain distinguishable object shapes and textures, indicating that DP-SGD does not effectively prevent

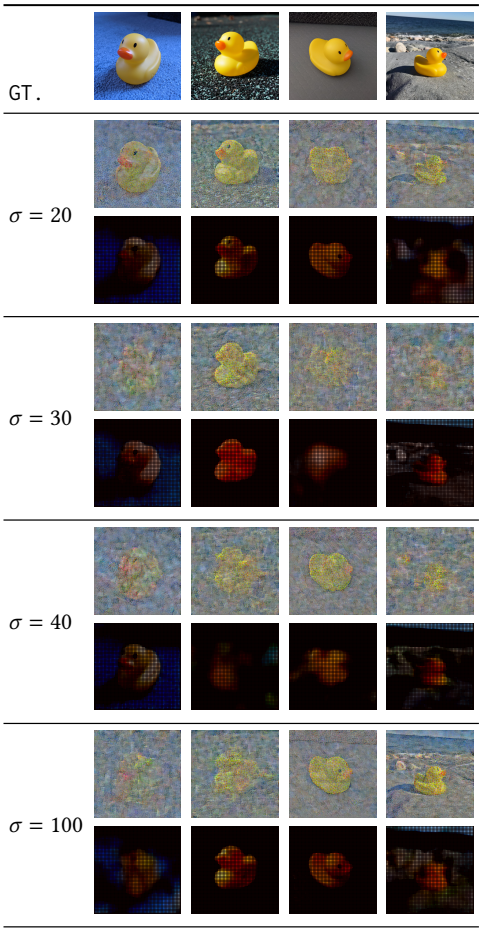


Figure 12: Reconstructed duck toy images from attacks under DP-SGD fine-tuning with varying levels of added noise.

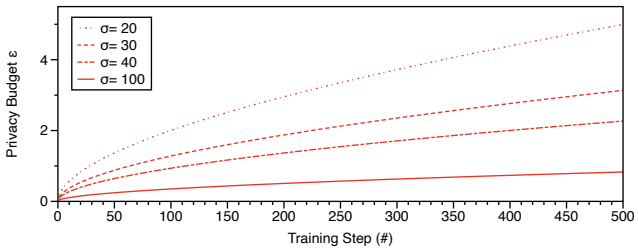


Figure 13: Estimate of ε during DP-SGD fine-tuning for different noise multipliers σ, with $\delta = 10^{-5}$.

data reconstruction in split learning. This aligns with a fundamental limitation of DP-SGD in this split fine-tuning setting: it applies noise to gradients rather than directly perturbing the intermediate outputs. Since split fine-tuning transmits unaltered intermediate activations to the server, the added noise in gradient updates does not sufficiently disrupt the reconstruction process. As a result, while DP-SGD may protect model parameters and prevent gradient-based

privacy attacks, it does not mitigate leakage from intermediate representations, leaving split learning models vulnerable to data reconstruction attacks from intermediate outputs.

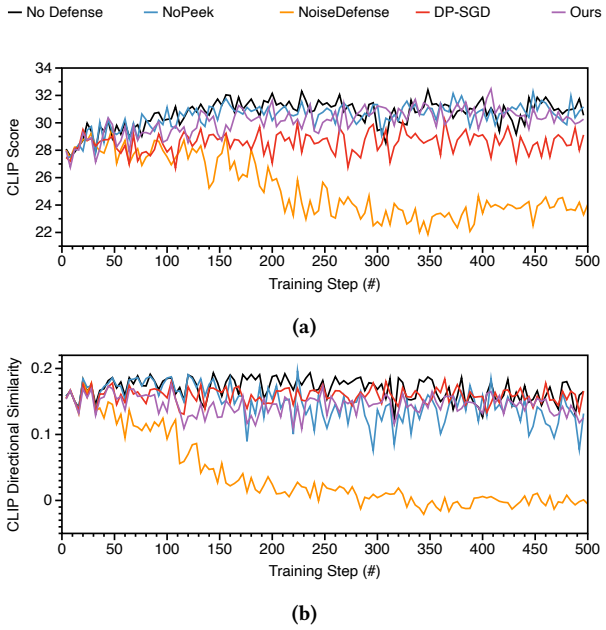


Figure 14: The model utility when applying DP-SGD for fine-tuning Stable Diffusion on the duck toy dataset.

A.3 More Results on Impact of Defenses on Model Utility and Training Efficiency

Table 4 and Fig. 15 present the impact of different defenses on model utility during the fine-tuning of Stable Diffusion on the brain MRI dataset, serving as a complementary analysis to Fig. 10 in Section 5.

Defense	CLIP Score	CLIP Directional Similarity
No Defense	33.9729	0.014
NoPeek	33.489	-0.002
NoiseDefense	30.9157	0.0066
Ours	32.7914	0.0393

Table 4: Model utility, measured by CLIP score and CLIP directional similarity, when applying different defenses after 250 fine-tuning steps on brain MRI images.

To assess the impact on other models and datasets, we present additional results in Fig. 16, which shows the increase in model accuracy and the decrease in training loss over time for two smaller models, ResNet-18 [12] and ViT [6]. In these cases, the CIFAR-10 [19] and Mini ImageNet [4] datasets are used, both of which are much larger in scale compared to the DreamBooth dataset employed for fine-tuning Stable Diffusion. In Fig. 16a, we can see that both NoiseDefense and our defense caused a 3.0% drop in ultimate

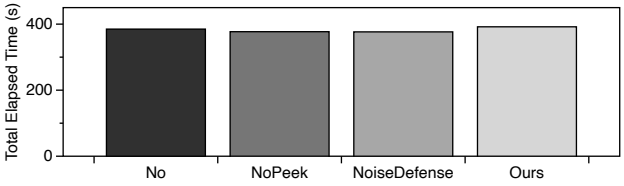
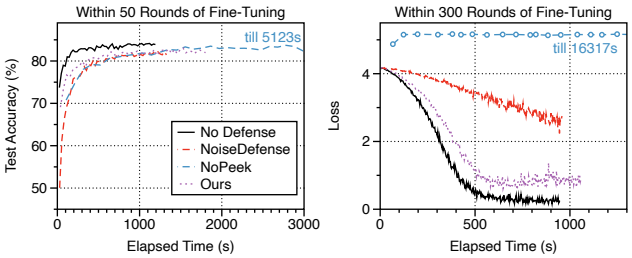


Figure 15: Total elapsed time when applying different defenses after 250 fine-tuning steps on brain MRI images. Our defense has caused a slight slowdown of 3.97%.

accuracy, while NoPeek achieved similar accuracy compared to having no defense. However, NoPeek drastically slowed down the training process because the pairwise distance correlation included in their loss function requires significant computation, especially for high-dimensional input and smashed data.

The gap between the impact of different defenses becomes more apparent in the more complex model shown in Fig. 16b. Even after the fine-tuning is complete, we observed that NoPeek experienced only a 19.46% drop in loss while taking approximately 16× longer. This could be due to an imbalance between the weights of the distance correlation loss and the classification loss in the loss function. Similarly, in the case of NoiseDefense, a fixed noise scale can have completely different effects on the two models, resulting in a 36% lower drop in loss compared to having no defense for ViT. Therefore, it is important carefully tune the hyperparameters in these two defenses to achieve optimal performance. Overall, our defense with adaptive dropout rates for different models and data has the least impact on model utility compared to other defenses.



(a) ResNet-18 & CIFAR-10 **(b) ViT & Mini ImageNet**

Figure 16: The impact of different defense strategies on the fine-tuning of two smaller models with larger datasets. Each circle along the blue curve in (b) represents the loss for each step in the fine-tuning process.

A.4 More Results on the Effectiveness of Defenses against Data Reconstruction Attacks

Fig. 17 provides additional examples of reconstructed images from the optimization-based attack on brain MRI dataset, serving as a complementary analysis in Section 4.1.

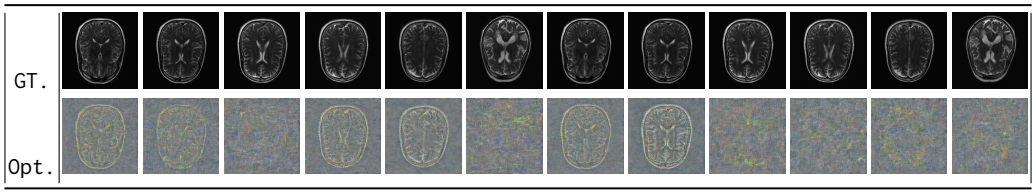


Figure 17: Additional reconstructed images from the optimization-based attack (`Opt.`) on the Brain MRI dataset. The Stable Diffusion model is split after DownBlock 1.

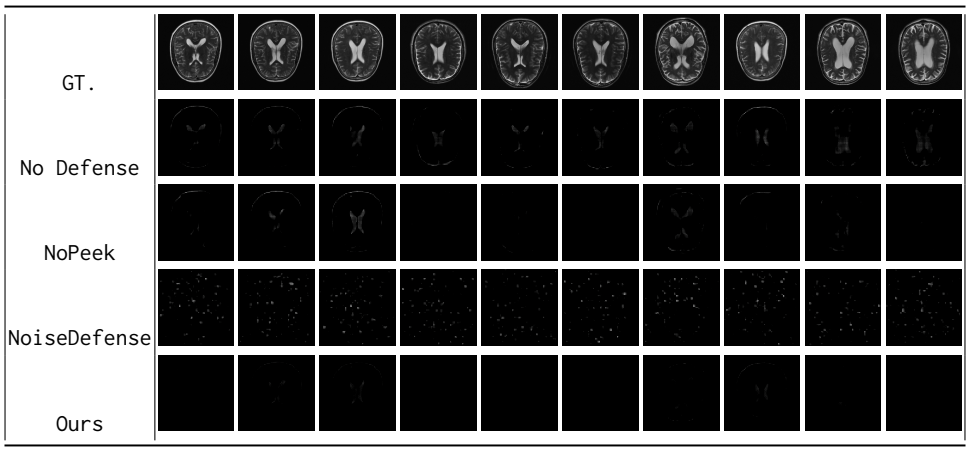


Figure 18: Additional comparisons of reconstructed images from the training-based attack (`Tra.`) using different defense mechanisms on the Brain MRI dataset.

Fig. 18 provides additional examples of reconstructed brain MRI images from the training-based attack under different defenses, serving as a complementary analysis to Fig. 11 and Table 3 in Section 5. The increased complexity of MRI images reduces the effectiveness of the training-based attack compared to subject-oriented or face datasets.

Table 5 presents quantitative evaluations for attacks on the duck toy dataset, serving as a complementary analysis to Fig. 11 and Table 3 in Section 5.

Duck toy					
Attack	**Defense**	**MSE↑**	**LPIPS↑**	**PSNR↓**	**SSIM↓**
`Opt.`	No Defense	0.7250	0.7093	1.6961	-0.0653
	NoPeek	0.6583	0.7266	1.8759	-0.1442
	NoiseDefense	0.7676	0.7846	1.6615	-0.0652
	Ours	0.7907	0.8010	1.8712	-0.1379
`Tra.`	No Defense	0.1674	0.7637	7.9482	0.3181
	NoPeek	0.1504	0.7492	8.4165	0.4026
	NoiseDefense	0.2001	0.7951	7.1373	0.1294
	Ours	0.1708	0.7900	7.8887	0.3707

Table 5: Evaluation of different defense mechanisms against optimization-based attack (`Opt.`) and training-based attack (`Tra.`) on duck toy images.