



Figure 6. The image on the right is generated from the ControlNet with a condition image on the left. The condition is an image of depth maps. The text prompt is: *Stormtrooper’s lecture.*

Figure 7. The image on the right is generated from the ControlNet trained by FedAvg using the condition image on the left. The text prompt is: *A skier poses for a shot on the night time slopes.*

## A. Details about Experimental Settings in Section 3.2

The diffusion model is stable diffusion V-1.5, ControlNet is of version 1.1, and the autoencoder is ViT-Large-Patch14 CLIP model [30]. The input resolution is  $512 \times 512$ . The NVIDIA A100 serves as the server’s device, and the NVIDIA A4500 is used for clients’ devices. The fine-tuning batch size is 4, and the model is trained for  $2.5 \times 10^3$  iterations. The number of clients is 50, with each client having 1000 training samples.

## B. Alternative Distributed Training Paradigms

Federated learning (FL) is an alternative distributed training paradigm that preserving users privacy by training directly on client devices and aggregating local training updates using a federated learning server. However, conventional federated learning may not be suitable for fine-tuning large ControlNet and diffusion models for three important reasons. *First*, ControlNets and diffusion models are large generative models, requiring formidable GPU resources on client devices for local fine-tuning of pre-trained models. *Second*, even if such GPU resources were available on client devices, pre-trained ControlNet and diffusion models may not be accessible as open-source due to commercial interests. For example, neither OpenAI nor Midjourney has open-sourced models like DALL-E 2 [31]. *Finally*, our experimental results presented here indicate that large ControlNet models fine-tuned with conventional federated averaging [25] as the aggregation mechanism experienced severely degraded performance compared to centralized training.

We follow the standard federated averaging scheme to train the ControlNet with 50 clients, each having 1000 training samples. We train for a total of 100 rounds and aggregate weights after every 250 local iterations. We evaluate the performance on the MS-COCO [21] validation set. An example of successful fine-tuning of a ControlNet [45] is shown in Figure 6. We can generate a stormtrooper with the same skeletons as in the left image of the depth maps. However, as shown in Figure 7, even under the assumption

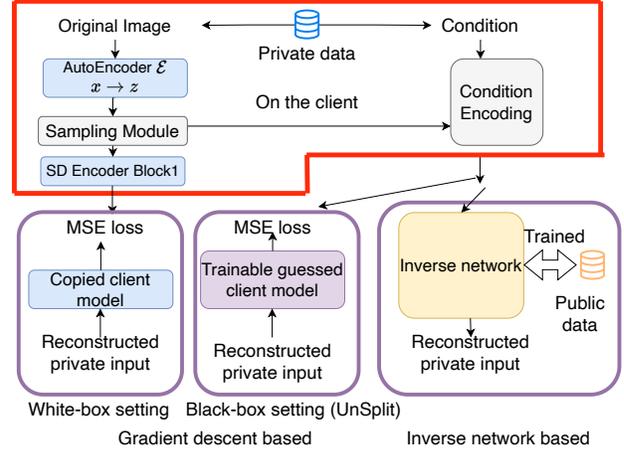


Figure 8. The illustration of different inversion attacks.

that clients have powerful computing units and the weights of the diffusion model are available, the ControlNet trained by FedAvg [25] fails to learn the conditions. The generated image does not match the condition at all; for example, the posture of the person in the generated image differs from that in the left condition images.

Although there may exist other aggregation scheme in federated learning and privacy-preserving methods in federated learning, considering the existing challenges of training models on the clients and unavailability of the whole pre-trained models, we leave the exploration of federated learning and other distributed paradigms for future work. In this paper’s scope, we focus on split learning.

## C. Re-evaluating Potential Attacks

### C.1. Potential Threats in Split Learning

#### C.1.1 Complementary about Threat Modeling

In our evaluation, we do not consider clients to be malicious. In split learning, a client receives no data if using our proposed framework. Therefore, malicious or colluding clients cannot obtain data related to constructing private images from other clients. However, malicious or colluding clients may send crafted data to the server to launch attacks, such as harming model utility. Such cases are detectable since the model cannot generate correct results. As our focus is on adversaries attempting to reconstruct private images, we do not consider this threat.

#### C.1.2 Complementary about Attacking Methods

Several threats have been specifically proposed in split learning, ranging from the leakage of inputs to labels. The most threatening attack is the *inversion attack*, which attempts to reconstruct original private data based on the received

intermediate feature. We summarize typical inversion attack methods proposed in previous literature in Figure 8. There are two typical methods to do such an attack.

The first method is based on gradient descent. In this type of attack, the adversary first constructs a randomized input or an input with prior knowledge about the private data. This input is then forwarded through a saved client model on the server, and the reconstruction loss (usually MSE loss) between the output from the randomized input and the received intermediate features is minimized. After several gradient descent iterations, the randomized input optimizes to resemble the private data, which we consider a reconstruction of private data. The attacker can launch these attacks under a white-box setting [46] if it knows the client model parameters; otherwise, it operates in a black-box setting.

In the black-box setting, the first method is a query-based attack [13] where the server sends specific designed inputs to the clients and observes the corresponding intermediate feature output. The second attack method, UnSplit [10], does not require such queries. In the UnSplit attack methodology, a client model replica, denoted as  $M$ , is initialized on the server along with a training sample represented as  $x$ . The parameters of the guessed client model are designated as  $\theta$ . Following the completion of the UnSplit attack, the converged training samples are utilized as the desired reconstruction of private data. During each iteration of split learning, upon receiving intermediate features denoted as  $\hat{h}$ , the server feeds the training sample into the guessed client model to obtain the output. Subsequently, the server undergoes multiple inner iterations to update  $x$  using  $\nabla_x L_{\text{MSE}}(M_\theta(x), \hat{h})$ , followed by several inner iterations to update  $\theta$  using  $\nabla_\theta L_{\text{MSE}}(M_\theta(x), \hat{h})$ . These steps are iteratively performed until convergence is achieved.

The second type of inversion attack is based on training an inverse network [13, 23, 43]. In this approach, the attacker first trains an inverse network on a public dataset, which is assumed to have a similar distribution as the private dataset. The inverse network takes the intermediate features as inputs and outputs the reconstructed private data. During the training of inverse networks, if it is under a white-box setting, the attacker will directly use known client model weights to train an inverse network. Otherwise, the attacker will first train an estimated client model using known server model weights on the same public dataset and then use this estimated client model to train an inverse network.

Beyond leakage from the most threatening inversion attack, other concerns exist about data privacy. Label leakage [20] assumes labels contain private information, allowing the server to infer private labels by observing gradient distributions returned to clients. However, this attack applies only to binary classification in split learning. Inference attacks [29] steal private data by sending attacker-designed

gradients to trick client models into returning features that enable data reconstruction.

Another potential privacy risk involves text prompt leakage. As shown in Figure 1, fine-tuning ControlNet requires the server to input text prompts into the encoders and decoders of both the diffusion model and control network. Consequently, clients must upload their private text prompts to the server. Although some may argue prompts are short, descriptive texts with limited private content, the server could still use known prompts to extract a training dataset [2]. Therefore, clients must keep prompts confidential from the server.

## C.2. Re-evaluating the Validity of Assumptions

### C.2.1 The client model weights can be kept secretly.

In a white-box setting [46], the client model weights are known. However, in real-world scenarios, the client does not need to disclose the model weights to the server for split learning to function. Even if an adversary manages to steal the client model weights, clients can simply re-initialize the model with different parameters. During the training process, if the client model is trainable, its weights will change in each iteration, making such an assumption invalid. The potential vulnerability arises if the client model is pre-trained. Since pre-trained weights are typically publicly available on the Internet, such an attack could pose a threat.

### C.2.2 The client can do split learning without providing prior knowledge about private data to the server

In real-world split learning scenarios, the server only requires the client model for training, operating without any knowledge of the private data. In a black-box setting, it is assumed that the adversary possesses prior knowledge about the private data, enabling it to train an inverse network on public data. For instance, Yao et al. [43] employed CelebA [22] as the public dataset and LFWA [17] as the private dataset, both containing human faces. However, in practical contexts, the availability of a public dataset exhibiting such a correlation with private datasets remains uncertain.

### C.2.3 The client can reject the query request.

In a specific inversion attack, an adversary must query the client model with samples supplied by the server [13]. However, in the standard split learning setup, clients do not need to respond to any queries from the server; the split learning still works. Therefore, to counter such an attack, clients can simply reject all queries originating from the server. One may argue that the server could construct these queries in a manner resembling gradients, making them indistinguishable to clients. However, with our structure that eliminates the need for gradient back-sending, such concerns are mitigated.

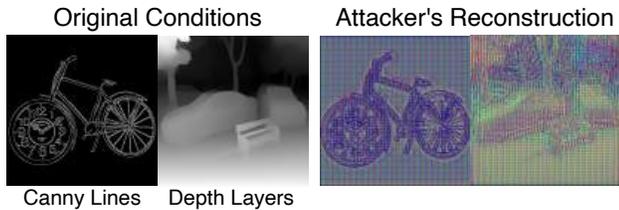


Figure 9. **Privacy preserving:** Higher distortion means better privacy preservation. Randomly selected and non-cherry-picked examples of reconstructed images by UnSplit attack.

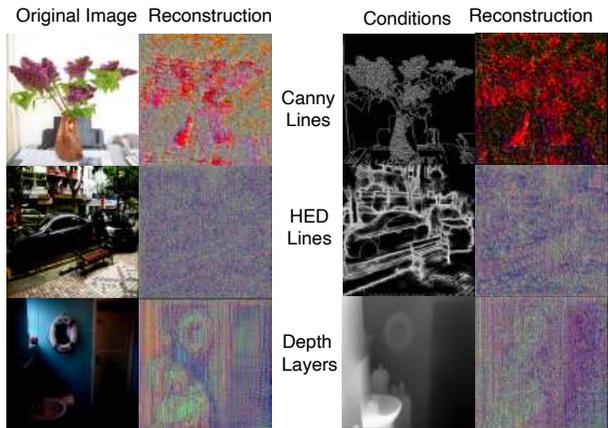


Figure 10. **Privacy preserving:** Higher distortion means better privacy preservation. Randomly selected and non-cherry-picked examples of reconstructed images by attacks optimizing MSE loss under white-box setting.

In inference attacks [29], if the client model is trained with gradients designed by the attacker, the resulting model will inevitably experience a performance decline. Users can easily detect this degradation in performance and cease using the compromised server. Furthermore, our designed structure offers a straightforward defense against such attacks as we do not need to train client models.

### C.3. Re-evaluating the Effectiveness of Attacks

In summary, practical applications of split learning face four threats. The first is a potential attack using gradient descents in a white-box scenario, particularly if clients utilize pre-trained weights. The second threat is an UnSplit attack, while the third involves training inverse networks to infer private data without prior knowledge of the data. The fourth threat is the leakage of text prompts.

#### C.3.1 Metrics for Privacy-preserving Effectiveness

An honest-but-curious server aims to reconstruct private data based on intermediate results. We evaluate the similarity between reconstructed images and private images using peak

signal-to-noise ratio (PSNR) and the structural similarity index measure (SSIM) [16]. Private images encompass users' natural and conditional images. Both SSIM and PSNR utilize image pixel values ranging from 0 to 255. PSNR assesses image reconstruction quality, while SSIM gauges image similarity. Lower SSIM and PSNR values signify decreased image similarity, indicating improved privacy preservation.

#### C.3.2 Attack by Gradient Descents

In the original structure, since the server lacks knowledge of the condition encoder weights, we resort to the UnSplit attack method, following the procedure outlined in UnSplit [10]. This attack involves updating the inputs based on the mean squared error (MSE) loss between intermediate results and outputs generated by randomly initialized inputs, iterated over 100 loops. Subsequently, these inputs are used to update the weights of the guessed client model, which is also initialized randomly on the server, for another 100 loops. This process is repeated for a total of 100 outer loops. We optimize the randomized model weights and inputs using the Adam optimizer with a learning rate of 0.001. The loss function utilized is  $\mathcal{L} = \mathcal{L}_{MSE} + \mathcal{L}_{L_2}$ . For fine-tuning the ControlNet, the dataset used is MS-COCO [21]. The successful reconstruction of images using the UnSplit method is illustrated in Figure 9.

In the gradient back-sending free structure, the server possesses knowledge of the weights of the pre-trained condition encoder. Consequently, the server can launch attacks in a white-box setting. For each attack, we conduct 1000 iterations using the Adam optimizer with a learning rate of 0.001 and MSE as the loss function. Regarding the reconstruction of the original image with the output of the SD encoder block 1, the PSNR is 3.39, and the SSIM is 0.12. For reconstructing the condition image, the PSNR is 5.95, and the SSIM is only 0.002. As depicted in Figure 10, the reconstructed images are far from recognizable. This ineffectiveness is attributed to the pre-trained autoencoder's complex model structure, which incorporates dropout layers and batch normalization layers. Between the two runs, even with identical inputs, variations in outputs occur due to dropout layers. In dropout layers, the operation of zeroing elements also nullifies the gradient, making methods relying on gradient descent ineffective. Given the ineffectiveness of the white-box setting, we do not need to test the black-box setting Unsplit attack.

#### C.3.3 Attack using Inverse Networks

For this attack, we examine two aspects: reconstructing the original image and the condition image. Since the server knows the diffusion model, it can directly use it to train an inverse network. The key question is how similar private and public datasets are. We choose MS-COCO as the public dataset and CelebA [22] and ImageNet [4] as the private

Table 4. The inverse networks for inversion attacks have two types: Type 1 reconstructs the original image; Type 2 reconstructs the condition image. Type 1&2 denotes layers used in both structures. Padding size is 1 and kernel size is 3.

Input	Operator	Stride	#Out	Structure	Activation
$64^2 \times 320$	Conv2d	1	320	Type 1	SiLU
$64^2 \times 320$	Conv2d,	1	256	Type 1	SiLU
$64^2 \times 256$	Upsample	2	96	Type 1	SiLU
$64^2 \times 4$	Upsample	2	96	Type 2	SiLU
$128^2 \times 96$	Conv2d	1	96	Type 1&2	SiLU
$128^2 \times 96$	Upsample	2	32	Type 1&2	SiLU
$256^2 \times 32$	Conv2d	1	32	Type 1&2	SiLU
$256^2 \times 32$	Upsample	2	16	Type 1&2	SiLU
$512^2 \times 16$	Conv2d	1	16	Type 1&2	SiLU
$512^2 \times 16$	Conv2d	1	3	Type 1&2	Sigmoid

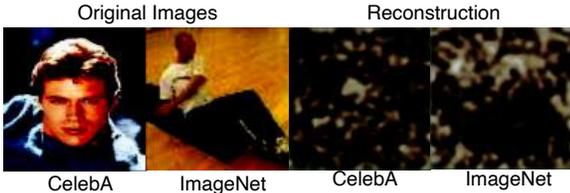


Figure 11. **Privacy preserving:** Higher distortion means better privacy preservation. Randomly selected and non-cherry-picked examples of reconstructed images from the outputs of the SD Encoder Block 1 using an inverse network.

datasets. MS-COCO and ImageNet contain images of various categories, while CelebA comprises over 200K faces from more than 10K celebrities.

The inverse network is trained on the public dataset and then evaluated on the private dataset. We use the AdamW optimizer with a learning rate of  $1 \times 10^{-5}$ , a batch size of 8, and train it for  $7.5 \times 10^4$  iterations. The structure of the inverse network is shown in Table 4. As illustrated in Figure 11, this attack is ineffective on both datasets. For CelebA, the PSNR is 5.87 and SSIM is 0.73, while for ImageNet, the PSNR is 6.56 and SSIM is 0.71. The reconstruction results are barely recognizable as the original private images.

Secondly, for the reconstruction of the condition image, if we employ our proposed structure, the server can directly train the inverse network. However, in the original structure, the server uses its model weights to train an estimated client model on the public dataset and then trains the inverse network. Unfortunately, this attack is effective for both structures with condition images. We will present the results and defense mechanisms in the following sections.

### C.4. Summary

We summarize potential split learning attacks in Table 5 and their effectiveness. The remaining effective method is inverse network-based attacks for reconstructing condition images. Another valid threat is text prompt leakage. Thus, in this paper, we emphasize defending against these two threats.

### D. Details about Experimental Settings in Section 5

**Execution Environment.** Our experiments are conducted on a server with A100 GPUs. We use NVIDIA A4500 GPUs as the client devices.

**Dataset.** The MS-COCO dataset contains over 120K images with proper prompts. The model is fine-tuned over MS-COCO for 25000 iterations with a batch size of 4. The remaining settings is the same as default implementation of ControlNet [45] where we use AdamW optimizer with learning rate of  $1 \times 10^{-5}$ . The noise coefficient  $\lambda_t$  is 0. We use the MS-COCO validation set with over 5000 images to evaluate the quality of generated images.

**Other Settings.** We have 50 clients in total and each has 1000 training samples. The number of clients will effect efficiency and scalability but will not effect image generation performance or privacy-preserving ability. Since the main focus of this paper is on the latter two aspects, we do not particularly study other settings. The resolution of the input and generate images is  $512 \times 512$ . The images are generated with the same random seed. The settings for evaluating privacy against reconstructing private data is the same as in Appendix C.

Table 5. Summary of existing attacks in split learning, assessing validity and effectiveness in the diffusion model scenario, using  $\checkmark$  for successful data reconstruction and  $\times$  otherwise;  $-$  denotes N/A.

		Original structure			Our structure		
		Valid?	Raw image	Condition image	Valid?	Raw image	Condition image
Gradient descent	White-box	$\checkmark$	$\times$	$-$	$\checkmark$	$\times$	$\times$
	Query-based	$\times$	$-$	$-$	$\times$	$-$	$-$
	Black-box	$\checkmark$	$-$	$\times$	$\times$	$-$	$-$
Inverse network	White-box	$\checkmark$	$\times$	$-$	$\checkmark$	$\times$	$\checkmark$
	Black-box	$\checkmark$	$-$	$\checkmark$	$\times$	$-$	$-$
Label leakage	$-$	Invalid: only applicable to binary image classification.					
Inference attack	$-$	Invalid: detectable as the model cannot generate the correct results.					
Text prompt leakage	$-$	The assumption is valid.					

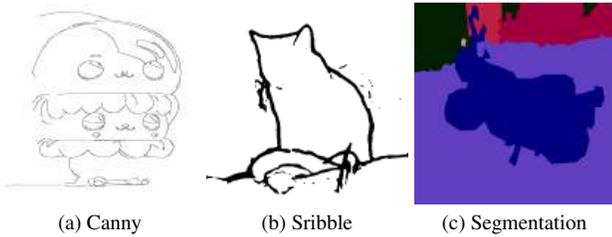


Figure 12. Examples of different conditions.

### E. Examples of Different Conditions

Figure 13 shows the examples of three different conditions: canny, segmentation, and scribble, studied in the paper.

### F. More Details about Implementation

For our and other privacy-preserving methods, we implement them with our designed gradient sending-back free structure. For  $(\epsilon, \Delta)$ -LDP mechanism,  $\Delta = 1 \times 10^{-4}$  and we calculate that  $\alpha \approx 0.16$ .

### G. Hyperparameter Tuning for Our Privacy-Preserving Methods

Before we choose the  $k$  and  $\beta_0$  for our methods, we try to use different values and compare their performance regarding the quality of image generation. In Remark 4.2, we can set different privacy budgets with proper  $k$  and  $\beta_0$ . In Figure 13, we change privacy budgets by setting different scheduling parameters  $k$  and  $\beta_0$  respectively. In the default setting of our method, the privacy budget is 8. We try the other two cases of setting privacy budgets as 0.3 and 2. As shown in Figure 13, our method can still generate images of good quality. However, for the baseline methods, they fail to

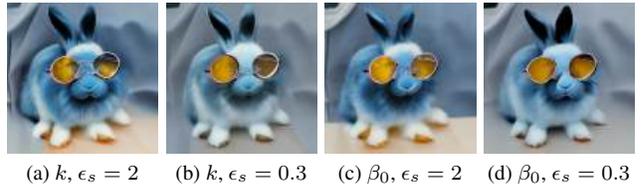


Figure 13. Randomly selected and non-cherry-picked examples of generated images varying privacy budget with different  $k$  and  $\beta_0$  in our method (Ours++).  $k$  and  $\beta_0$  indicate that the privacy budgets change by altering only  $k$  or  $\beta_0$  respectively compared to the default settings.

generate good images and protect privacy when they use the same privacy budgets of 0.3 and 2.

### H. More Visualization of Image Generation

Figure 15 and Figure 16 show more visualization result of image generation when different privacy-preserving methods are applied.

### I. More Visualization of Reconstruction

Figure 17 show more visualization results of reconstructed images by attacks using inverse networks, where different privacy-preserving methods are applied.

### J. Discussion

In this paper, we resolve the question of how we can train ControlNet and diffusion models while keeping users' data privacy. Besides the aspect of preserving privacy, there are other issues worth studying in production level split learning with ControlNet and stable diffusion. In this paper, we focus on ControlNet and stable diffusion while in future work,

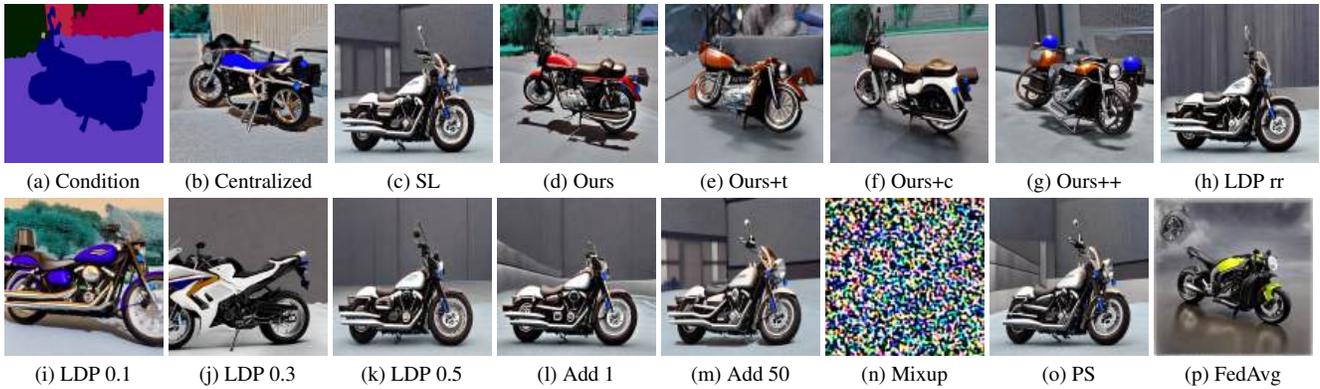


Figure 14. **Image generation:** Images of higher quality means better. Randomly selected and non-cherry-picked examples of generated images with the given condition of Segmentation under different methods. The text prompt is: *a motorcycle*.

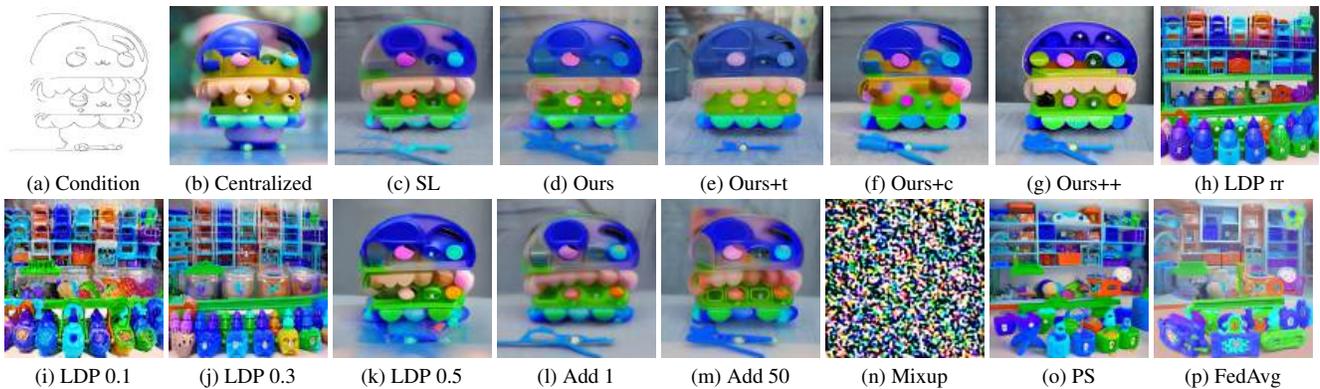


Figure 15. **Image generation:** Images of higher quality means better. Randomly selected and non-cherry-picked examples of generated images with the given condition of Canny under different methods. The text prompt is: *cute toys for kids*.

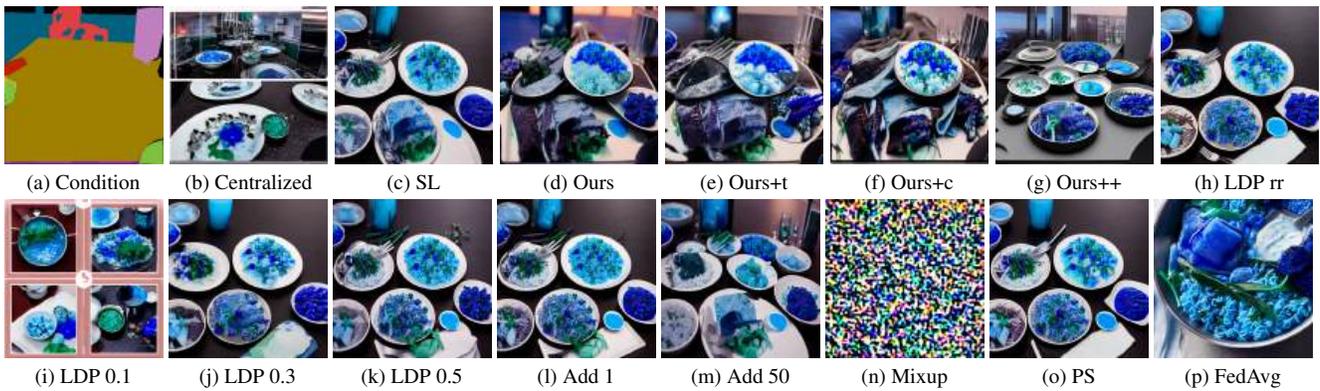


Figure 16. **Image generation:** Images of higher quality means better. Randomly selected and non-cherry-picked examples of generated images with the given condition of Segmentation under different methods. The text prompt is: *dinner with food in blue*.

we hope we can extend our methods to other fine-tuning methods for diffusion models such as T2I-Adapters etc.

Another challenging question is how we can keep users' data privacy during the inference stage after deploying trained ControlNet and diffusion models. The inference

process is different from the training. A trivial solution is to run the inference completely on the edge device, which needs about 7.5GB of memory. The memory requirement is much less than that of training, which is feasible. However, maybe not all clients have enough memory. It is a challenge

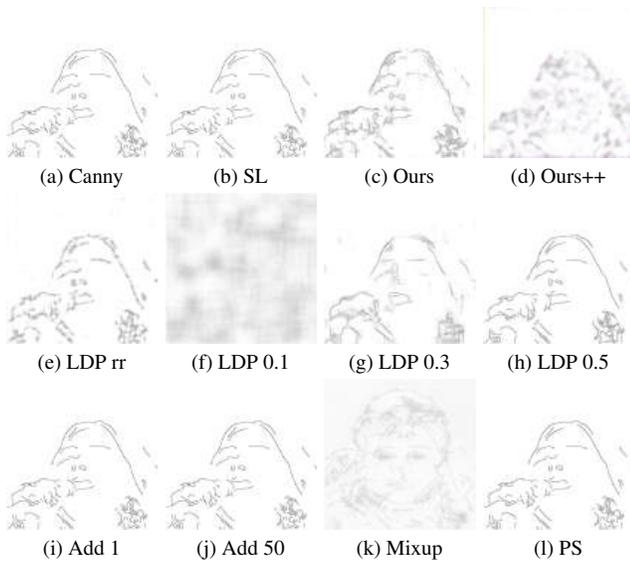


Figure 17. **Privacy preserving:** Higher distortion means better privacy preservation. Randomly selected and non-cherry-picked examples of reconstructed condition images by inverse network based attacks when fine-tuning the ControlNet with condition canny. The private dataset is CelebA.

that how we can still keep user data privacy if we deploy a ControlNet across the clients and the server. From related work, we can see large efforts are being put into privacy-preserving inference in split learning. It is worth studying whether these methods are helpful during the inference stage.

In this paper, the target is towards privacy-preserving split learning with ControlNet and diffusion model. In a broader research topic, one question is how we can safely do split learning. In such a case, we may not assume every client is honest, which means some clients are malicious and not sending the correct intermediate features. To harm the interests of other clients, some clients may do backdoor attacks or adversarial attacks, diminishing the utility of the fine-tuned ControlNet and diffusion model.

In our experiments, we deploy split learning with 50 clients. We can increase the number of clients if we want, but since  $T_s$  is much larger than  $T_c$ , the whole training time is the same. Therefore, we do not increase the number. On the production level, it is possible that there are more than 50 clients. With our methods, we can still train ControlNet with split learning over them while preserving data privacy. A minor issue is that since the clients only need to do inference, they may send intermediate features of large amounts continuously and simultaneously. It is worth studying how the server deals with a large scale of requests simultaneously. We can expand the client number to hundreds or thousands to evaluate the scalability.