Supplementary Material

DecentNeRFs: Decentralized Neural Radiance Fields from Crowdsourced Images

A Supplementary Video

In the supplementary material, we have included a video that visually illustrates the key features and results of our DecentNeRF approach, providing compelling evidence to support our claims. This video serves as a powerful complement to the written content, offering a more immersive and intuitive understanding of our work.

B Compute Comparisons

B.1 Complexity Comparison

Offline Communication and Computation For each round of aggregation, each user prepares offline coded random masks and distributes them to the users.
Reconstruction Decoding process to reconstruct the weighted average of the obfuscated weights sent by the users.
Online Communication Sending and receiving of obfuscated and receiving of weights and gradients.

In Table 3 we note that DecentNeRF has the same server complexity as FedAvg [23] at the cost of incurring more user complexity.

	FedNeRF	**DecentNeRF**
offline comm. (U)	$O(sN)$	$O(sN^2 + dN)$
offline comp. (U)	$O(dN + sN^2)$	$O(dN^2)$
online comm. (U)	$O(d + sN)$	$O(dN + sN^2)$
online comm. (S)	$O(dN + sN^2)$	$O(dN + sN^2)$
online comp. (U)	$O(d)$	$O(dN^2)$
reconstruction (S)	$O(dN^2)$	$O(dN^2)$

Table 3: Complexity comparison between FedAvg [23] and DecentNeRF. In terms of server computing, DecentNeRF has the same complexity but utilizes higher user computing. Here **N** is the total number of users, **d** is the model size, and **s** is the length of the secret keys as the seeds for Pseudo-Random Generator ($\mathbf{s} \ll \mathbf{d}$). U is for User and S is for Server

B.2 Server Compute

Here we show how server compute (in terms of FLOPs) is estimated for one of the example blender scenes in Table 1.

NeRF-W: For rendering one batch of rays (4096), SOTA NeRF algorithms such as the one by Müller et al. [25] take about $\approx 350 \times 10^6$ FLOPs (approximately 10^4 times faster than vanilla NeRF [24]). For the Lego dataset, which has 100 images of size 800×800 pixels, this leads to 12,500 batches for one epoch, resulting in a total cost of approximately 4.8×10^{12} FLOPs per epoch. Training the scene for 30 epochs requires around 130×10^{12} FLOPs of computing on the server. Here, we note that the compute cost for the server scales with the number of pixels in the complete dataset.

FedNeRF: It requires the addition of 20 (number of users) Global MLP weights per round, each MLP having around 1 million parameters. 1 addition is equivalent to 1 FLOP. For 30 federation rounds, the total computation cost at the server is approximately 0.6×10^9 FLOPs. In a decentralized setting, we note that the computing cost scales by number of Users.

DecentNeRF: In addition to the aggregation of user weights, similar to FedNeRF, the server computes the aggregate of gradients, which incurs the same computational cost as the aggregation of weights in FedNeRF, approximately 0.6×10^9 FLOPs. The computation of the dot product between each user's weights and the aggregated sum of gradients incurs an additional 1.2×10^9 FLOPs (accounting for parameter-wise multiplication and summation). For the reconstruction cost, we utilize Secure Aggregation (SecAgg+) [3], which has $O(dN \log N)$ complexity for the previous 4 computations, incurring an additional 3×10^9 FLOPs.

C Implementation Details

Training Details We use a PyTorch implementation of NeRF-W [19] for centralized evaluation, which is a variant of vanilla NeRF [24]. As shown in Figure 3, the Global Radiance Field **G**, consists of eight layers with 256 hidden units each, followed by an MLP that outputs RGB. This MLP comprises four layers of 128 hidden units each. The Personal MLP, consisting of four layers of 128 hidden units, outputs both σ and color. Details on positional encoding, encoding viewing direction, and other aspects are in line with NeRF-W and are mentioned in the supplementary material. For DecentNeRF, we implement the same architecture on the user device, which we simulate using the open-source Flower library [4]. This implementation includes the secure aggregation protocol of SecAgg+ [6], a feature of Flower. The FedNeRF implementation is akin to DecentNeRF but omits the Personal MLP, appearance embedding, secure aggregation, and learned weighted averaging.

During training, we sample 64 points for coarse and 64 for fine part of the network while for inference we sample 128 fine samples for all implementations. Models are trained with Adam for the same number of total iterations (sum of all user iterations for decentralized) for NeRF-W and DecentNeRF while FedNeRF is trained for lower rounds as the validation loss doesn't improve beyond certain rounds.

D Model Extraction Attacks and Defenses

There are model-extraction attacks that attempt to recover individual weights of the model reported by any client [30, 38] even if the weights were to be securely aggregated to compute the averaged weight. These attacks have been studied in the context of federated learning along with other attacks such as backdoor attacks [2] for adversarially poisoning a federated learning system. Similarly, there have been several defenses based on differential privacy, such as the PrivUnit mechanism [5] to prevent such attacks. The study of federated learning systems in the context of various such attacks is out of scope of this paper. The contributions of our paper hold independently to help enable a real-world vision application. That said, our method is in fact forward-compatible with the PrivUnit defense to prevent reconstruction of client weights.

E Dataset Details

Phototourism dataset Even though the scenes from the phototourism dataset vary in number of images we only use the first 200 images for all approaches since for our decentralized approaches we need to simulate multiple clients and a single machine. The real implementation of our decentralized wouldn't have this restriction and can learn on more than 200 images.

F DecentNeRF pseudocode

Algorithm 1 DecentNeRF algorithm. Data transfers are marked by *.

Require: Hyperparameters:
Weighted Averaging Learning Rate η,
Number of merge rounds M,
Number of training epochs per merge Γ.

$\{g^{(0)}, p^{(0)}\}$ are initial Shared and Public MLP weights
$\alpha^{(0)} \rightarrow \{\alpha_1, \alpha_2, \ldots, \alpha_K\}$ are initial client contributions
Distribute $\{g^{(0)}, p^{(0)}\}$ to all clients $k = 1, \ldots, K$ *
for merge round $m = 1$ to M **do**
  for client $k = 1$ to K in Parallel **do**
    $g^{(m-1)}$ is received from the server *
    if $m > 1$ **then**
      freeze $\{g_k^{(m-1)}, p_k^{(m-1)}\}$
      $\frac{\partial L_k}{\partial g_k^{(m-1)}} \leftarrow$ *Calc. Gradients*
      $\frac{\partial L_k}{\partial g_k^{(m-1)}}$ transmitted to the server *
      $\sum \frac{\partial L_k}{\partial \mathbf{g}^{(m-1)}}$ is recieved from the server *
      $\alpha_k^{(m)} \leftarrow$ Alpha Update$(\left\{ \frac{\partial L_k}{\partial g_k^{(m-1)}}, g_k^{(m-1)}, \eta \right\})$
    end if
    $\{g_k^{(m)}, p_k^{(m)}\} \leftarrow$ NeRF$(g_k^{(m-1)}, p_k^{(m-1)}, \Gamma)$
    $\alpha_k^{(m)} \times g_k^{(m)}$ is transmitted to the server *
  end for
  for server in Parallel **do**
    $g^{(m)} \leftarrow$ Secure Aggregation$(\{\alpha_k \times g_k\}_{k=1,\ldots,K}^{(m)})$
    $g^{(m)}$ is transmitted to the clients *
    $\sum \frac{\partial L_k}{\partial \mathbf{g}^{(m-1)}} \leftarrow$ Secure Aggregation$(\{\frac{\partial L_k}{\partial g_k^{(m-1)}}\}_{k=1,\ldots,K})$
    $\sum \frac{\partial L_k}{\partial \mathbf{g}^{(m-1)}}$ is transmitted to the clients *
  end for
end for
return $g^{(M)}$