Federated Learning for Large-Scale Scene Modeling with Neural Radiance Fields

Teppei Suzuki[1]

***Abstract*— We envision a system to continuously build and maintain a map based on *earth-scale* neural radiance fields (NeRF) using data collected from vehicles and drones in a lifelong learning manner. However, existing large-scale modeling by NeRF has problems in terms of scalability and maintainability when modeling earth-scale environments. Therefore, to address these problems, we propose a federated learning pipeline for large-scale modeling with NeRF. We tailor the model aggregation pipeline in federated learning for NeRF, thereby allowing local updates of NeRF. In the aggregation step, the accuracy of the clients' global pose is critical. Thus, we also propose global pose alignment to align the noisy global pose of clients before the aggregation step. In experiments, we show the effectiveness of the proposed pose alignment and the federated learning pipeline on the large-scale scene dataset, Mill19.**

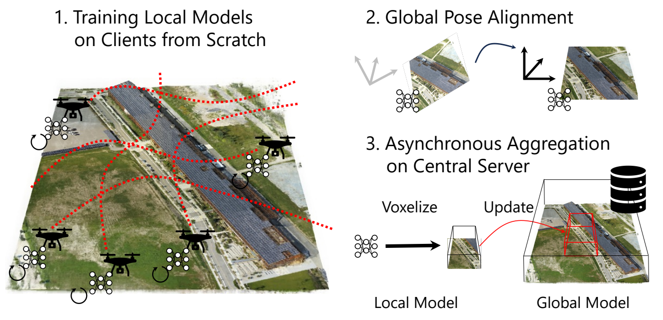

1. Training Local Models on Clients from Scratch
2. Global Pose Alignment
3. Asynchronous Aggregation on Central Server
Voxelize
Update
Local Model
Global Model

Fig. 1: Overview of the proposed federated learning pipeline for neural radiance fields.

I. Introduction

Neural radiance fields (NeRF) [1] have emerged to represent scenes for view synthesis and have been used as maps for many robotics applications, such as vision-based localization, navigation, and SLAM [2], [3], [4], [5], [6], [7]. Although these studies have been evaluated in relatively small-scale scenes, we believe that such NeRF-based robotics applications will be available to the systems operated in large-scale environments, such as self-driving cars and delivery drones. In addition, since NeRF can lift arbitrary 2D information to 3D (*e.g.*, semantic labels [8], and feature vectors [9]), it will facilitate a higher-level environmental recognition and understanding in robotics applications.

Nowadays, a large number of vehicles are on the road, and in the near future, many unmanned aerial vehicles, such as delivery drones, will be in the sky. In such a world, we envision the system to continuously build and maintain an *earth-scale* NeRF-based map using data collected from vehicles and drones in a life-long learning manner and to leverage the map created and maintained with such a system.

Large-scale scene modeling with NeRF has been studied in recent years [10], [11]. These methods divide a large-scale environment into many small areas and model the small areas with multiple NeRF models. Although this pipeline efficiently models a large-scale scene in a distributed training manner, it has three problems in realizing our envisioned system: (i) The existing pipelines need to collect data in a server once, but they need high communication costs to collect data from clients on a server and need a large number of memory budgets to save the earth-scale data; (ii) these pipelines require tremendously large computational resources for training a large number of models to cover an earth-scale environment; and (iii) they can only update the model for each pre-divided area for the maintenance of the NeRF-based map, and if the training data is collected only in a part of the area, the model may suffer from the forgetting problem [3].

To develop a more scalable and maintainable method, we propose the federated learning pipeline for large-scale modeling with NeRF, as depicted in Fig. 1. Federated learning (FL) [12] is a data-decentralizing training protocol that enables clients, such as mobile devices, to collaboratively train a model while keeping the training data decentralized. Since FL does not need to aggregate data collected by clients on the server, it can alleviate the communication cost problem and the data storage problem. FL also alleviates the computational resource problem because it can leverage the resources of clients, thereby enabling their overall FLOPs to outperform those of supercomputers.

To train NeRF in a federated learning manner, we tailor the model aggregation pipeline, which allows the global model to be locally updated based on the client data, avoiding the forgetting problem and improving maintainability. Specifically, we cache the outputs of each local model as the 3D voxel grid, and then the global model is updated by aggregating cached representation based on the clients' pose in the global coordinate. This pipeline does not affect the outputs of the global model beyond the region modeled by the local models. In addition, to align the inaccurate global pose of clients due to sensor noise, we propose the global pose alignment step before the aggregation step, which is inspired by the vision-based localization using NeRF [2], [7], [13].

Our contribution is summarized below:

- We propose the federated learning pipeline for large-

[1]Denso IT Laboratory, Inc., Tokyo, Japan
suzuki.teppei@core.d-itlab.co.jp

scale scene modeling with NeRF. By caching the outputs of each local model as the voxel grid, we realize the local update of the global model.
- We also propose the global pose alignment. To alleviate the sensor noise in the clients' pose in the global coordinate, we align the global pose of each client before the aggregation step by minimizing the difference between the images rendered by the global and local models.
- We assess the effectiveness of our training pipeline on the Mill19 dataset [11], which contains thousands of HD images collected from drone footage over 100,000m^2 of terrain near an industrial complex.

II. Related Work

A. Neural Radiance Fields

3D reconstruction from image collections [14], [15], [16] is an important task in computer vision fields, and the reconstruction methods have been applied to many robotics applications, particularly SLAM systems [17], [18], [19]. In principle, objects or whole scenes are reconstructed as 3D points based on multiple-view geometry.

In recent years, NeRF [1] has emerged for novel view synthesis. Unlike 3D point-based reconstruction, NeRF models the target scene by a continuous function and its image rendering process is differentiable. Taking advantage of these properties, some vision-based robotics systems have been proposed. iNeRF [2], a vision-based localization method, estimates the camera pose through the minimization of the loss between the target image and the rendered image with the gradient method. Maggio *et al*. [7] and Lin *et al.* [13] improved the robustness of iNeRF by leveraging the advantage of Monte Carlo localization [20]. Adamkiewicz *et al*. [5] utilized this localization scheme for robot navigation. Kruzhkov *et al*. [21] incorporated NeRF into the SLAM system. They used NeRF as a mapping module and reduced the memory budget for the map. Moreover, pure NeRF-based SLAM systems, such as iMap [3] and Nice-SLAM [4], were studied.

As reviewed above, NeRF is applied to robotics applications that use NeRF as a map. When considering systems using NeRF as a map in a large environment, such as self-driving cars and delivery drones, we need to model large-scale scenes. There are certain studies for large-scale scene modeling by NeRF. Block-NeRF [10] and Mega-NeRF [11] have modeled urban-scale environments. Both methods divide the target scene into small areas and then model the small areas with multiple NeRF models. They can efficiently model a large-scale scene in a distributed training manner.

However, the existing large-scale modeling methods have some problems in terms of maintainability and computational costs. When part of the scene is changed, they need to update all models representing the changed scene, even if the changed area is a part of the modeled area. In addition, due to the forgetting problem [3], the model needs to be trained with all data in the corresponding area if a part of the area is changed. Therefore, maintaining the map created by existing methods is costly. Moreover, these methods require tremendously large memory budgets to save data and large computational resources to model the environment if one builds an earth-scale NeRF-based map. Hence, we approach these problems by adopting a federated learning scheme.

B. Federated Learning

Federated learning (FL) [12] is a data-decentralizing training process for machine learning models. It distributes the model to clients such as mobile devices or organizations, trains the model on clients using their own data, and aggregates the models on the server. By repeating the above process, FL collaboratively trains a global model under the orchestration of a central server. FL has some advantages: preserving privacy since it does not collect user data on the server and leveraging the computational resources of many clients (millions or more in our scenario). Because of these advantages, FL has been studied for training machine learning models in various tasks, such as image classification [12], [22], image segmentation [23], [24], and self-supervised learning [25], [26].

Many FL approaches update models using a synchronous protocol, which needs to wait for all updates on clients before aggregation and delays training. The existence of lagging devices (i.e., stragglers, stale workers) is inevitable due to device heterogeneity and network unreliability.

To mitigate this problem, asynchronous federated learning (AFL) has been proposed [27], which aggregates local models asynchronously. Xie *et al.* [27] aggregates models by exponential moving averaging. Chen *et al.* [28] proposed a method to aggregate the model with a decay coefficient to balance the previous and current gradients on clients. Since AFL improves the communication efficiency of FL, our pipeline also adopts the AFL strategy.

III. Federated Learning for NeRF

We present the overall pipeline of our method in Fig. 1, which consists of three steps, training local models, global pose alignment, and an aggregation step. We consider the AFL setting; namely, the server updates the global model as soon as it receives the local model.

We first describe the assumptions of this study in Sec. III-A. Then, in Secs. III-B and III-C, we describe the training pipeline for the local models and the aggregation step. Finally, we describe the global pose alignment in Sec. III-D.

A. Assumptions

We assume each client knows the relative camera poses between collected images. This is a realistic assumption because relative poses can be obtained from some sensors, SLAM, or SfM. In the experiments, we use the camera poses given by Mill19 [11] that are estimated by PixSfM [29]. Note that if the obtained poses are noisy, we would be able to correct them by using the methods that simultaneously optimize poses and the model (*e.g.*, BARF [30] and NoPe-NeRF [31]).

We also assume that the global pose of each client is known but *noisy*. Since the goal of this study is to train the large-scale NeRF model as a map, there is no map to localize the position[1]. In this situation, we need to use sensors such as GPS and IMU to obtain the global pose, but the pose obtained from the sensors basically contains noise.

In addition, we assume that images collected by clients are captured under the same lighting and have no dynamic and/or transient object. This assumption is unreasonable, particularly for outdoor scenes; we will study more realistic scenarios in future work.

B. *Training Local Models*

For simplifying the pipeline, we assume that client models are trained from scratch because NeRF basically "overfits" training data and it does not care how the model is initialized. Of course, we can consider the standard pipeline; namely, clients download the global model and then train it with the local data, which may make convergence faster but increase communication costs.

The local model consists of two models: the density model f_σ and the color model f_c. Let $\mathbf{r} = \{\mathbf{r}_i \in \mathbb{R}^3 | \mathbf{r}_i = \mathbf{o} + s_i\mathbf{d},\ s_i \in \mathbb{R}\}_i$ be a set of sampled points along the ray passing through a pixel, where $\mathbf{o}$ and $\mathbf{d}$ are a camera origin and a ray direction. Following [11], we sample 512 points per ray for rendering images in our experiments. The density at $\mathbf{r}_i$ is computed as $\sigma_i = \phi \circ f_\sigma(\mathbf{r}_i)$, where ϕ denotes the softplus function. For representing RGB, we use the spherical harmonics (SH) as in PlenOctrees [32], which enables caching outputs of the model on the grid voxels in the aggregation step described in Sec. III-C. Specifically, f_c outputs SH coefficients $\mathbf{k}(\mathbf{r}_i) = f_c(\mathbf{r}_i)$, where $\mathbf{k}(\mathbf{r}_i) = \{k_\ell^m \in \mathbb{R}^3\}_{\ell:0\le\ell\le\ell_{\max}}^{m:-\ell\le m\le\ell}$; ℓ and m are a degree and an order of the SH function, respectively. Each k_ℓ^m is a set of three coefficients corresponding to RGB. Then, the view-dependent color at r_i is computed by

$$\mathbf{c}_i = S\left(\sum_{\ell=0}^{\ell_{\max}} \sum_{m=-\ell}^{\ell} k_\ell^m Y_\ell^m(\mathbf{d})\right), \tag{1}$$

where $S(\cdot)$ is the sigmoid function and $Y_l^m(\mathbf{d}) : \mathbb{S}^2 \to \mathbb{R}$ is the SH function at the viewing angle $\mathbf{d}$. Following [33], we use spherical harmonics of degree 2, which has 27 harmonic coefficients.

The rendering and training procedure is the same as in the original NeRF [1]. Specifically, the pixel color corresponding to the ray $\mathbf{r}$ is computed as follows:

$$\hat{\mathbf{C}}(\mathbf{r}) = \sum_i \exp(-\sum_{j=1}^{i-1} \sigma_j\delta_j)(1 - \exp(-\sigma_i\delta_i))c_i, \tag{2}$$

where $\delta_i = t_{i+1} - t_i$ is the distance between adjacent samples on the ray. Then, the NeRF model is trained by minimizing the following loss:

$$\mathcal{L} = \mathop{\mathbb{E}}_{\mathbf{r}\in\mathcal{R}} \left\| \hat{C}(\mathbf{r}) - C(\mathbf{r}) \right\|_2^2, \tag{3}$$

[1]Once the map (*i.e.*, the global model) is initialized, we can use it to localize each client in the same manner as that for iNeRF [2].

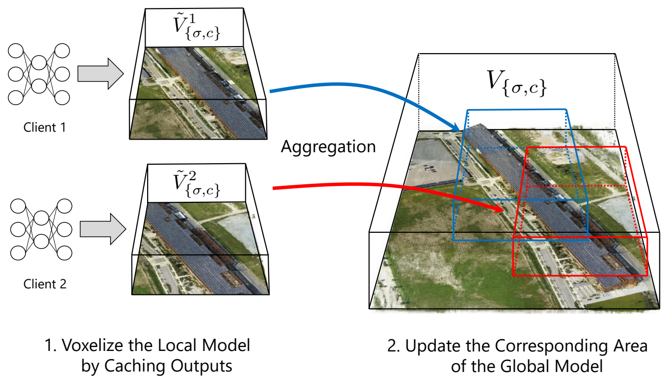

$\tilde{V}^1_{\{\sigma,c\}}$
Client 1
Aggregation
$V_{\{\sigma,c\}}$
$\tilde{V}^2_{\{\sigma,c\}}$
Client 2
1. Voxelize the Local Model by Caching Outputs
2. Update the Corresponding Area of the Global Model

Fig. 2: Overview of the proposed aggregation step. We first cache outputs of the local model on the grid voxels, $\tilde{V}_\sigma^i$ and $\tilde{V}_c^i$, and then add them to the global voxel grid, V_σ and V_c.

where $C(\mathbf{r}) \in \mathbb{R}^3$ denotes the ground-truth pixel color corresponding to $\mathbf{r}$, and $\mathcal{R}$ is the set of rays in the training data. Note that we do not consider the coarse-to-fine rendering pipeline [1] in this study, but it can be made available with a larger memory budget and more computational resources.

C. *Updating Global Model*

The aggregation step in federated learning often averages the local models. However, this averaging strategy cannot work for NeRF due to its properties; NeRF represents only the scene in the training data and assigns random color and density to the outside of the scene. In addition, the 3D position (x, y, z) may indicate a different location for each client because clients' coordinates may not be aligned due to the sensor noise. For these reasons, the simple averaging strategy will degrade the rendering quality of the global model.

To avoid the aforementioned problem, we tailor the aggregation step for NeRF. We represent the global model as the voxel grid, as in Plenoxels [32]. We cache the output of the local models on the voxel grid, as in FastNeRF [34], and then aggregate the cached voxel grid and the global model, as shown in Fig. 2. Since we only cache the outputs in the area modeled by the local model and aggregate them, this procedure does not affect the output outside the area. In addition, the voxel grid representation allows us to align the global pose of the clients before aggregation (Sec. III-D).

Let $\tilde{V}_\sigma^n \in \mathbb{R}^{X\times Y\times Z\times 1}$ and $\tilde{V}_c^n \in \mathbb{R}^{X\times Y\times Z\times 27}$ be a cached density grid voxel and a cached SH coefficient grid voxel of the n-th client, where X, Y, and Z are the voxel resolution and 27 in V_c corresponds to the number of the coefficients. Also, we define the global model's voxel grids as $V_\sigma \in \mathbb{R}^{X_g\times Y_g\times Z_g\times 1}$ and $V_c \in \mathbb{R}^{X_g\times Y_g\times Z_g\times 27}$, where X_g, Y_g, and Z_g are the global model's voxel resolution. Note that the global model is defined on the global coordinate and the local voxel grid is defined on the local coordinate, respectively; hence, the position on the local coordinate, $P \in \mathbb{R}^3$, is mapped to the global coordinate using the global pose, as in $P_g = R_nP + t_n$, where $R_n \in \mathrm{SO}(3)$ and $t_n \in \mathbb{R}^3$ denote the relative rotation and translation between the n-

th client's local and global coordinates, respectively, which are obtained from sensors such as GPS and IMU. Then, we update the density grid voxel of the global model with exponential moving averaging in the asynchronous federated learning manner as

$$V_\sigma[P_g] \leftarrow \eta V_\sigma[P_g] + (1-\eta)\tilde{V}_\sigma^n[P], \tag{4}$$

where $V_\sigma[P]$ denotes an element of V_σ at P; η denotes a mixing value that is set to 0.9 in the experiments. The SH coefficient grid is also updated in the same manner.

The rendering process of the global model is the same as Plenoxels [33]; namely, the density and SH coefficient are sampled from the voxel grids by trilinear interpolation and then the RGB color is computed following eq. (1) with the sampled coefficients. Finally, pixel colors are rendered by eq. (2) with the sampled density and the color. We define the rendered pixel color through the global model as $\hat{C}_g(\mathbf{r})$.

D. Global Pose Alignment

As we mentioned in Sec. III-A, the global pose of the clients contains noises. Thus, we need to correct it before aggregation because inconsistency between the local and global models occurs in eq. (4) if the global pose is incorrect.

To align the pose, we optimize it to minimize the difference between the reference and target images, which are rendered by the local and global models, respectively. Let $t^* \in \mathbb{R}^3$ and $\{R_j^* \in \mathrm{SO}(3)\}_j$ be a translation vector and rotation matrices for the target views, which are obtained from the area modeled by both the global and local models. Then, we align the client's global pose by solving the following minimization problem:

$$\begin{aligned}(\hat{t},\ \hat{R}) = \underset{\{t,R\}}{\arg\min} \sum_j \mathop{\mathbb{E}}_{\substack{\mathbf{r}_{t^*,R_j^*},\\ \mathbf{r}_{t^*+t,RR_j^*}}} & \lambda|\hat{C}_g(\mathbf{r}_{t^*,R_j^*}) - \hat{C}(\mathbf{r}_{t^*+t,RR_j^*})| \\ & + (1-\lambda)|\hat{D}_g(\mathbf{r}_{t^*,R_j^*}) - \hat{D}(\mathbf{r}_{t^*+t,RR_j^*})|, \end{aligned} \tag{5}$$

where $\mathbf{r}_{t,R}$ denotes the ray depending on the camera pose (t, R), and $\hat{D}_g$ and $\hat{D}$ denote the rendered depth through the global and local models, respectively, which are computed by replacing color values, c_i, in eq. (2) with the sampled point position, s_i, as in [3]; $\lambda \in [0, 1]$ is a weight parameter. After optimization, we align the global pose of the local model based on $\hat{t}$ and $\hat{R}$. Since the rendering process can be differentiable with respect to t and R, we optimize them using the gradient method, as in the existing NeRF-based localization methods [2], [7], [13]. In our experiments, we used the same optimization pipeline as that in Lin *et al.* [13] to optimize the pose, which is a Monte Carlo-based method. Note that, unlike existing methods [2], [7], [13], we leverage multiple views and rendered depth to make optimization stable. We assess the effectiveness of the multiple target views and the depth loss in the experiment section.

IV. Experiments

A. Experimental Setup

To simulate our method, we use the Mill19 dataset [11] that includes two scenes, `building` and `rubble`, which have thousands of HD images collected from drone footage over 100,000m^2 of terrain close to an industrial complex. We first generate the data owned by clients in the following procedure: first, we randomly select one image from training data and compute the Euclidean distance between the camera position of the selected image and the camera positions of the other images, and then we collect k-nearest neighbors as the data owned by a client[2]. We repeat this procedure for the number of clients, N, that is set to 100 in the experiments. In this experiment, we randomly set k for each client in a range of 100 to 200. After training N local models, we update the global model with the pose alignment and evaluate it with the validation data of Mill19. We set the voxel size for caching the output of the local model to 0.25 m. Following [11], we use the appearance vector to model the color of each image and optimize it in test time. The experiments were conducted using the NVIDIA V100 GPU.

We use InstantNGP [35] as the local model because it converges faster and has fewer FLOPs than the original NeRF architecture, which is suitable if the computational resources of the clients are limited. Each local model is trained for one epoch[3] with a batch size of 8192. We optimize the local models with Adam [36], whose hyperparameters are the same as those of InstantNGP[35], except for the learning rate; it is set to 5e-3 only for the hash encoding and 5e-4 for the other parameters.

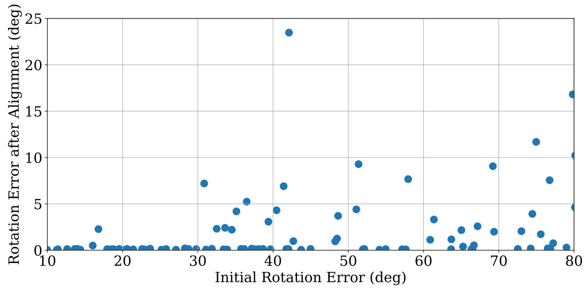
B. Effect of Pose Alignment

To evaluate our global pose alignment framework, we randomly select one local model as the global model. We also randomly select another model as a local model from the models whose training data partially overlap the area modeled by the global model. After selecting the models, we compute a center position of the overlapped area as a translation vector, $t^* \in \mathbb{R}^3$, and obtain rotation matrices, $\{R_j^* \in \mathrm{SO}(3)\}_j$, from camera poses of the training data close to t^*. We render the images at $\{(t^*, R_j^*)\}_j$ as the target values through the global model. Finally, we randomly sample a translation vector, $t \in \mathbb{R}^3$, and a rotation matrix, $R \in \mathrm{SO}(3)$, which can be regarded as the sensor noises, and optimize them by solving eq. (5). Thus, the optimal values will be $t = (0, 0, 0)$ and $R = I$, and we report the gap from them. Note that since we use the appearance vector, as described in IV-A, we also optimize it in addition to the pose. For optimization, we use the Adam optimizer [36] with a batch size of 4096. We set an initial learning rate to 5e-4 and decay it to 5e-5.

As an ablation study, we show the rotation errors and translation errors with various λ in eq. (5) and a various number of target views in Tab. I. We randomly sample noises in a range of [-20 m, 20 m] and [-20°, 20°]. As presented on the left-hand side of Tab. I, RGB information is critical for the pose alignment, but the depth information

[2]In our envisioned scenario, the clients are basically drones and vehicles, and the collected data should be sequential frames. Thus, we generate the client data from images that are physically close to each other.

[3]One epoch corresponds to the iterations of `#pixels` / `#batchsize`.



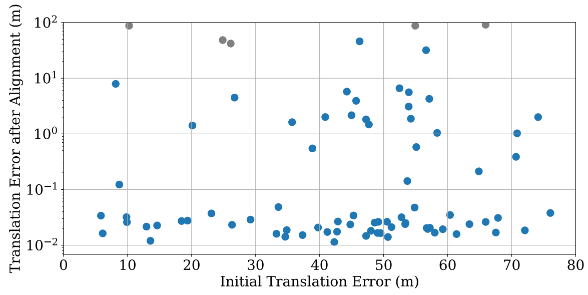


Fig. 3: The alignment results for various initial errors. The gray dots denote failure cases that increase the initial errors.

TABLE I: Rotation and translation errors with various λ and a various number of target views. We report the mean and standard deviation over 10 trials. We use four target views on the left-hand side of the table and set λ to 0.75 on the right-hand side of the table.

λ	Rot. (deg)	Trans. (m)
0.0	24.3±15.5	32.3±18.7
0.25	19.9±16.7	17.7±11.2
0.5	12.9±16.3	9.62±11.7
0.75	**0.53±0.96**	**0.19±0.33**
1.0	0.92±1.63	0.43±0.78

#views	Rot. (deg)	Trans. (m)
1	0.34±0.24	0.97±0.68
2	0.33±0.24	0.49±1.18
4	0.53±0.96	0.19±0.33
8	**0.12±0.09**	**0.02±0.01**

helps marginally with error reduction. The multiple target views also contribute to reducing errors, as depicted on the right-hand side of Tab. I.

We also show the evaluation results of our alignment framework with various magnitudes of initial errors in Fig. 3. We randomly sample the pose noises and then align them. We plot the results for 100 trials in Fig. 3. Note that a few trials with various translation errors have failed, and the failure rate is 5%. Our method can align the translation error up to 75 m, which is comparable to a typical GPS error. Also, it can correct the rotation error that is larger than errors in a typical IMU. A few plots in Fig. 3 indicate relatively large errors after alignment, but we can decrease the errors by increasing the number of particles and resampling rounds in the Monte Carlo-based optimization; in fact, our method can reduce errors even when the initial errors are larger than that of the failure cases. In our method, the increase in computation time is acceptable because alignment will be performed offline and does not require real-time processing.

C. Comparison with Other Training Protocols

To evaluate the rendered image quality of our method, we compare the proposed training pipeline with two data-centralized training protocols, baseline and distributed training. The baseline indicates training one model with all data, which corresponds to the ordinary NeRF training pipeline. The distributed training indicates the Mega-NeRF's training pipeline [11]; namely, the scene is divided into grids, and then models are trained with data corresponding to each grid. We divide the scene into 4×4 grids for the distributed training; that is, 16 models are trained, and the number of data for training each model is approximately 150, which is the same as the expected number of data used to train each model of our pipeline. For a fair comparison, we evaluate these training pipelines with the same model and the same hyperparameters as ours, except for the model size in the baseline. Since the baseline models the scene using one model, while other training protocols use multiple models, we double the grid size of the hash encoding for the baseline to be fair in terms of the model capacity. Note that we assume that the global position is correctly aligned by our position alignment algorithm in this experiment.

We show the evaluation results in Tab. II. Note that since the models can be trained in parallel, the longest training time out of all local models is reported as the training time in the distributed learning and ours. The training time of distributed training and ours is much shorter than that of the baseline because of the distributed training protocol. Since the scale of Mill19 is relatively small compared to our envisioned scenario, the training time of the baseline is acceptable. However, if we scale it to an earth-scale, it is impossible to train a model with the standard training pipeline. In addition to the reduction in training time, our method can alleviate bandwidth consumption because the size of the local model is 0.1 GB while that of the local data is up to 1 GB. The rendering speed for the distributed training is slower than the others because the distributed training combines the outputs of the multiple models to render one image. Rendering by ours is the fastest because it only samples cached outputs from the voxel grids for rendering, unlike the others that consist of hash encoding and MLP.

The PSNR for our method is worse than that for the baseline for two reasons: each client trains a local model with a relatively small number of data, while the baseline trains the model with a sufficiently large number of data. Basically, NeRF requires a sufficiently large number of viewpoints to correctly represent the scene in 3D. However, some clients do not satisfy this requirement around the test view and the local models trained by such clients cannot represent the scene correctly, as shown in Fig. 4. Consequently, such local models degrade the global model performance. This disadvantage is also found in distributed training; in fact, PSNR for distributed training is worse than that for the baseline. The other reason is the quantization error in the caching process. We cache the outputs of the local models on the voxel grid in the aggregation step, and such a quantization

TABLE II: Performance for different training protocols on building and rubble scenes of Mill19.

	Building		Rubble		Rendering Speed
	PSNR	Training Time (h)	PSNR	Training Time (h)	(Pixel/Second)
Baseline	22.29	32.4	24.93	28.0	80.4K
Distributed Training	18.93	0.75	21.24	0.67	69.2K
Ours	18.06	0.97	23.01	0.95	342.6K

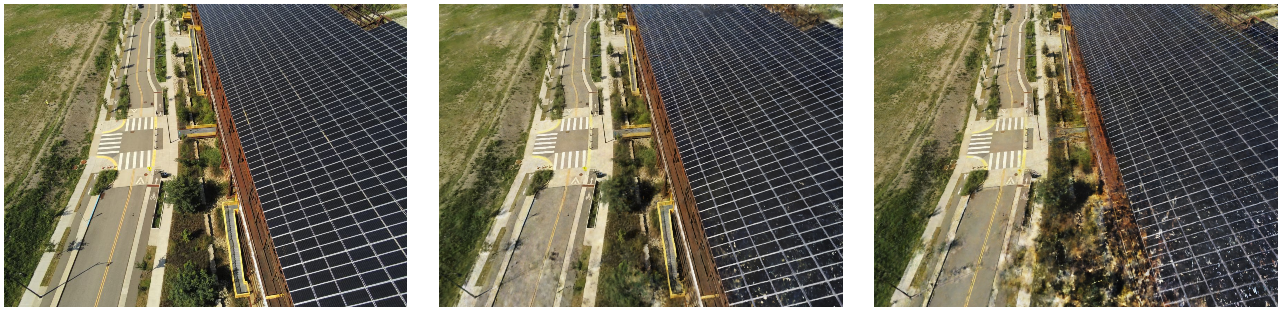


Fig. 4: From left to right, the ground-truth image of the test view, the image rendered by the local model trained with a sufficiently large number of viewpoints around the test view, and the image rendered by the local model trained with a relatively small number of viewpoints.

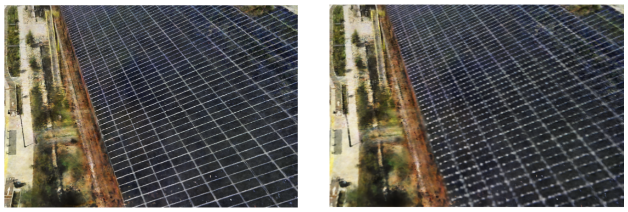


Fig. 5: An image rendered by the client model (left) and an image rendered by the cached voxel grids (right). The lattice pattern composed of high-frequency components is broken.

operation approximates the continuous function by the piece-wise linear function. Thus, there are approximation errors, which cause the gap between PSNR for distributed training and ours, especially for the building scene that includes high-frequency components, as shown in Fig. 5. This error can be reduced by increasing the cached grid size, but it requires larger memory budgets. Improving local model training and alleviating the quantization errors are possible directions for future work.

The PSNR for ours on the rubble scene is better than that for distributed training. The quantization errors in the rubble scene are smaller than those in the building scene because images in the rubble scenes are basically composed of low-frequency components. Therefore, in scenes where the effect of the errors is small, our training protocol would be better than distributed training in this experimental setting.

V. Limitations and Future Work

As we mentioned in Sec. III-A, this work does not consider dynamic and/or transient objects and lighting changes. It may raise privacy concerns because if a person in the scene is modeled with the lighting, the model may show where and when the person was doing what. However, there are some approaches to modeling the scene by ignoring such objects and lighting effects (*e.g.*, NeRF-W [37] and Block-NeRF [10]); by exploiting the advantage of such methods, the proposed pipeline will leverage the privacy-preserving aspect of the federated learning. Therefore, we believe that considering the transient objects and lighting changes in the federated learning pipeline is an important research direction.

Moreover, as we mentioned in Sec. IV-C, the quality of the rendered images is worse than the images rendered by the baseline. Fortunately, there are many studies on training NeRF with limited viewpoints [38], [39], [40]. We believe that we can improve the performance of the local models by leveraging the training protocol of such methods; consequently, the global model can be improved. Another approach is to consider the number of viewpoints for aggregating models. Specifically, by adjusting the mixing coefficient in the exponential moving average according to the number of viewpoints, we may be able to ignore the low-quality outputs of the local model in the aggregation step.

VI. Conclusion

In this study, we proposed the federated learning pipeline for large-scale scene modeling with NeRF. We designed the aggregation step for training NeRF in a federated learning manner: The global model is defined as the voxel grids and the local models are aggregated by caching the outputs of the local model on the voxel grids, which allows us to update the global model locally. In addition, we proposed the global pose alignment step to alleviate the sensor noise in the global pose of each client. We assessed the proposed method on the Mill19 dataset containing thousands of images collected from drone footage over 100,000m^2 of terrain near an industrial complex and verified its effectiveness.

We believe our work opens new avenues for large-scale scene modeling using NeRF and is an important step towards collaboratively training for NeRF in a federated learning manner.

References

[1] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng, "Nerf: Representing scenes as neural radiance fields for view synthesis," *Communications of the ACM*, vol. 65, no. 1, pp. 99–106, 2021.

[2] L. Yen-Chen, P. Florence, J. T. Barron, A. Rodriguez, P. Isola, and T.-Y. Lin, "inerf: Inverting neural radiance fields for pose estimation," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2021, pp. 1323–1330.

[3] E. Sucar, S. Liu, J. Ortiz, and A. J. Davison, "imap: Implicit mapping and positioning in real-time," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 6229–6238.

[4] Z. Zhu, S. Peng, V. Larsson, W. Xu, H. Bao, Z. Cui, M. R. Oswald, and M. Pollefeys, "Nice-slam: Neural implicit scalable encoding for slam," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 12 786–12 796.

[5] M. Adamkiewicz, T. Chen, A. Caccavale, R. Gardner, P. Culbertson, J. Bohg, and M. Schwager, "Vision-only robot navigation in a neural radiance world," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 4606–4613, 2022.

[6] O. Kwon, J. Park, and S. Oh, "Renderable neural radiance map for visual navigation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 9099–9108.

[7] D. Maggio, M. Abate, J. Shi, C. Mario, and L. Carlone, "Loc-nerf: Monte carlo localization using neural radiance fields," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2023, pp. 4018–4025.

[8] A. Kundu, K. Genova, X. Yin, A. Fathi, C. Pantofaru, L. J. Guibas, A. Tagliasacchi, F. Dellaert, and T. Funkhouser, "Panoptic neural fields: A semantic object-aware neural scene representation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 12 871–12 881.

[9] S. Kobayashi, E. Matsumoto, and V. Sitzmann, "Decomposing nerf for editing via feature field distillation," *Advances in Neural Information Processing Systems*, vol. 35, pp. 23 311–23 330, 2022.

[10] M. Tancik, V. Casser, X. Yan, S. Pradhan, B. Mildenhall, P. P. Srinivasan, J. T. Barron, and H. Kretzschmar, "Block-nerf: Scalable large scene neural view synthesis," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 8248–8258.

[11] H. Turki, D. Ramanan, and M. Satyanarayanan, "Mega-nerf: Scalable construction of large-scale nerfs for virtual fly-throughs," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 12 922–12 931.

[12] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Artificial intelligence and statistics*. PMLR, 2017, pp. 1273–1282.

[13] Y. Lin, T. Müller, J. Tremblay, B. Wen, S. Tyree, A. Evans, P. A. Vela, and S. Birchfield, "Parallel inversion of neural radiance fields for robust pose estimation," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2023, pp. 9377–9384.

[14] J. L. Schönberger and J.-M. Frahm, "Structure-from-motion revisited," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[15] S. Agarwal, Y. Furukawa, N. Snavely, I. Simon, B. Curless, S. M. Seitz, and R. Szeliski, "Building rome in a day," *Communications of the ACM*, vol. 54, no. 10, pp. 105–112, 2011.

[16] M. Pollefeys, D. Nistér, J.-M. Frahm, A. Akbarzadeh, P. Mordohai, B. Clipp, C. Engels, D. Gallup, S.-J. Kim, P. Merrell, *et al.*, "Detailed real-time urban 3d reconstruction from video," *International Journal of Computer Vision*, vol. 78, pp. 143–167, 2008.

[17] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardos, "Orb-slam: a versatile and accurate monocular slam system," *IEEE transactions on robotics*, vol. 31, no. 5, pp. 1147–1163, 2015.

[18] Z. Teed and J. Deng, "Droid-slam: Deep visual slam for monocular, stereo, and rgb-d cameras," *Advances in neural information processing systems*, vol. 34, pp. 16 558–16 569, 2021.

[19] J. Engel, T. Schöps, and D. Cremers, "Lsd-slam: Large-scale direct monocular slam," in *European conference on computer vision*. Springer, 2014, pp. 834–849.

[20] F. Dellaert, D. Fox, W. Burgard, and S. Thrun, "Monte carlo localization for mobile robots," in *Proceedings 1999 IEEE international conference on robotics and automation (Cat. No. 99CH36288C)*, vol. 2. IEEE, 1999, pp. 1322–1328.

[21] E. Kruzhkov, A. Savinykh, P. Karpyshev, M. Kurenkov, E. Yudin, A. Potapov, and D. Tsetserukou, "Meslam: Memory efficient slam based on neural fields," in *2022 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. IEEE, 2022, pp. 430–435.

[22] T. Nishio and R. Yonetani, "Client selection for federated learning with heterogeneous resources in mobile edge," in *ICC 2019 - 2019 IEEE International Conference on Communications (ICC)*, 2019, pp. 1–7.

[23] J. Miao, Z. Yang, L. Fan, and Y. Yang, "Fedseg: Class-heterogeneous federated learning for semantic segmentation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 8042–8052.

[24] W. Li, F. Milletarì, D. Xu, N. Rieke, J. Hancox, W. Zhu, M. Baust, Y. Cheng, S. Ourselin, M. J. Cardoso, *et al.*, "Privacy-preserving federated brain tumour segmentation," in *Machine Learning in Medical Imaging: 10th International Workshop, MLMI 2019, Held in Conjunction with MICCAI 2019, Shenzhen, China, October 13, 2019, Proceedings 10*. Springer, 2019, pp. 133–141.

[25] W. Zhuang, Y. Wen, and S. Zhang, "Divergence-aware federated self-supervised learning," *arXiv preprint arXiv:2204.04385*, 2022.

[26] F. Zhang, K. Kuang, Z. You, T. Shen, J. Xiao, Y. Zhang, C. Wu, Y. Zhuang, and X. Li, "Federated unsupervised representation learning," *arXiv preprint arXiv:2010.08982*, 2020.

[27] C. Xie, S. Koyejo, and I. Gupta, "Asynchronous federated optimization," *arXiv preprint arXiv:1903.03934*, 2019.

[28] Y. Chen, Y. Ning, M. Slawski, and H. Rangwala, "Asynchronous online federated learning for edge devices with non-iid data," in *2020 IEEE International Conference on Big Data (Big Data)*. IEEE, 2020, pp. 15–24.

[29] P. Lindenberger, P.-E. Sarlin, V. Larsson, and M. Pollefeys, "Pixel-perfect structure-from-motion with featuremetric refinement," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2021, pp. 5987–5997.

[30] C.-H. Lin, W.-C. Ma, A. Torralba, and S. Lucey, "Barf: Bundle-adjusting neural radiance fields," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 5741–5751.

[31] W. Bian, Z. Wang, K. Li, J.-W. Bian, and V. A. Prisacariu, "Nope-nerf: Optimising neural radiance field with no pose prior," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 4160–4169.

[32] A. Yu, R. Li, M. Tancik, H. Li, R. Ng, and A. Kanazawa, "Plenoctrees for real-time rendering of neural radiance fields," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 5752–5761.

[33] S. Fridovich-Keil, A. Yu, M. Tancik, Q. Chen, B. Recht, and A. Kanazawa, "Plenoxels: Radiance fields without neural networks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 5501–5510.

[34] S. J. Garbin, M. Kowalski, M. Johnson, J. Shotton, and J. Valentin, "Fastnerf: High-fidelity neural rendering at 200fps," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 14 346–14 355.

[35] T. Müller, A. Evans, C. Schied, and A. Keller, "Instant neural graphics primitives with a multiresolution hash encoding," *ACM Transactions on Graphics (ToG)*, vol. 41, no. 4, pp. 1–15, 2022.

[36] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[37] R. Martin-Brualla, N. Radwan, M. S. Sajjadi, J. T. Barron, A. Dosovitskiy, and D. Duckworth, "Nerf in the wild: Neural radiance fields for unconstrained photo collections," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 7210–7219.

[38] K. Deng, A. Liu, J.-Y. Zhu, and D. Ramanan, "Depth-supervised nerf: Fewer views and faster training for free," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 12 882–12 891.

[39] A. Yu, V. Ye, M. Tancik, and A. Kanazawa, "pixelnerf: Neural radiance fields from one or few images," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 4578–4587.

[40] M. Kim, S. Seo, and B. Han, "Infonerf: Ray entropy minimization for few-shot neural volume rendering," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 12 912–12 921.