

# MEIL-NeRF: Memory-Efficient Incremental Learning of Neural Radiance Fields

Jaeyoung Chung<sup>1</sup>Kanggeon Lee<sup>1</sup>Sungyong Baik<sup>2</sup>Kyoung Mu Lee<sup>1</sup><sup>1</sup>Dept. of Electrical and Computer Engineering, ASRI, Seoul National University, Seoul, Korea<sup>2</sup> Dept. of Data Science, Hanyang University, Seoul, Korea

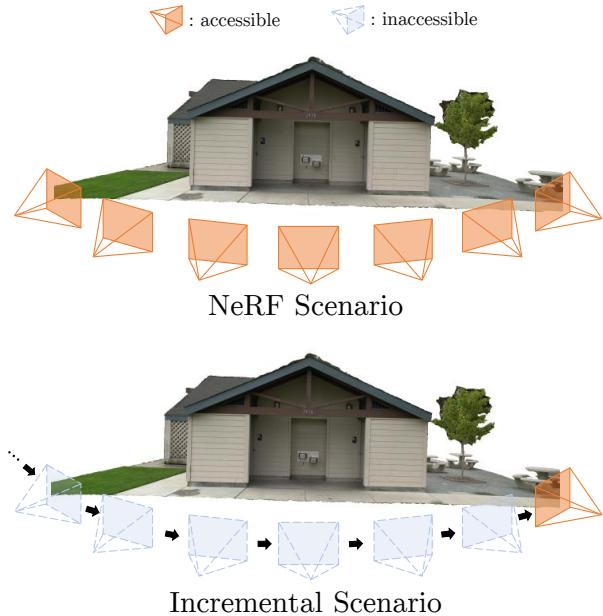
robot0321@snu.ac.kr, dlrkdrjs97@snu.ac.kr, dsybaik@hanyang.ac.kr, kyoungmu@snu.ac.kr

## Abstract

Hinged on the representation power of neural networks, neural radiance fields (NeRF) have recently emerged as one of the promising and widely applicable methods for 3D object and scene representation. However, NeRF faces challenges in practical applications, such as large-scale scenes and edge devices with a limited amount of memory, where data needs to be processed sequentially. Under such incremental learning scenarios, neural networks are known to suffer catastrophic forgetting: easily forgetting previously seen data after training with new data. We observe that previous incremental learning algorithms are limited by either low performance or memory scalability issues. As such, we develop a **Memory-Efficient Incremental Learning algorithm for NeRF (MEIL-NeRF)**. MEIL-NeRF takes inspiration from NeRF itself in that a neural network can serve as a memory that provides the pixel RGB values, given rays as queries. Upon the motivation, our framework learns which rays to query NeRF to extract previous pixel values. The extracted pixel values are then used to train NeRF in a self-distillation manner to prevent catastrophic forgetting. As a result, MEIL-NeRF demonstrates constant memory consumption and competitive performance.

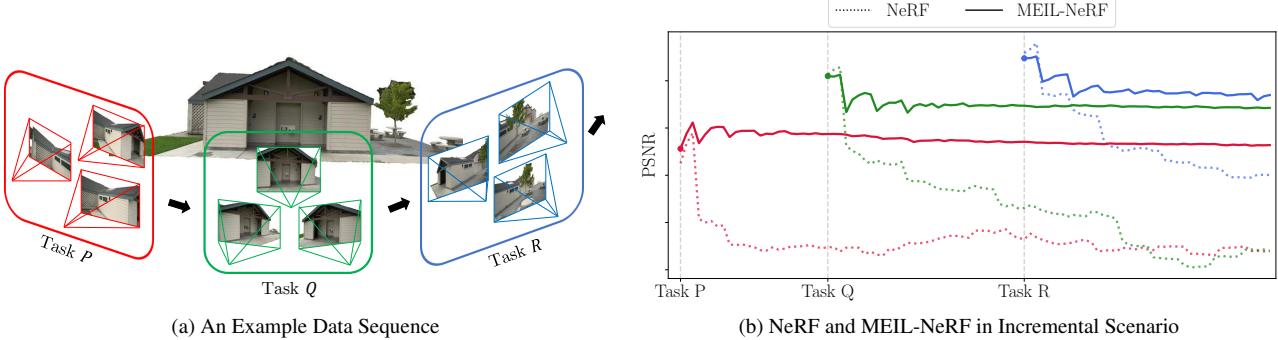
## 1. Introduction

Representation and reconstruction of 3D objects and scenes are important computer vision and computer graphics tasks, with a broad range of applications, such as virtual reality [5, 16], autonomous driving [21, 30], and robotics [2, 14, 40]. Recently, neural radiance fields (NeRF) [36] has brought substantial improvement by exploiting the representation power of neural networks. They represent a static scene with an MLP network by querying spatial location and view direction along camera rays and integrating the



**Figure 1. NeRF under standard scenario vs. incremental scenario.** Unlike vanilla NeRF where all viewpoints are always accessible, we consider incremental scenarios where only recent few viewpoints are accessible at each time. In the incremental scenarios, a set of images and corresponding poses are sequentially given and there is no or limited access to the past viewpoints. Such incremental scenario poses challenges in that NeRF will be susceptible to forgetting of images at previous viewpoints, degrading the overall representation performance.

output colors and densities using volume rendering techniques. To achieve such outstanding performance, however, NeRF assumes access to all data (i.e., RGB values of a scene seen from all viewpoints) at once. Such constraint hinders NeRF from being applied to practical applications—for example, large-scale scenes and edge devices with a limited amount of memory—where data needs



**Figure 2. Catastrophic Forgetting on NeRF in Incremental Scenario.** We present an incremental scenario on NeRF and demonstrate the results. (a) shows an example data sequence, assuming that the NeRF network can only learn the data from the latest task. In this scenario, the performance of NeRF network drops rapidly, and the early tasks are completely ruined, as shown in (b). To alleviate catastrophic forgetting in the network, we propose a learning strategy based on self-distillation, which significantly slows down the forgetting speed.

to be processed sequentially. In other words, NeRF will incrementally learn the scene, the partial information of which will be visible from few viewpoints at each time, as illustrated in Figure 1.

Under such incremental learning scenarios, neural networks are known to suffer catastrophic forgetting [15]: old knowledge gets forgotten while learning new knowledge. Various incremental learning algorithms have been developed to mitigate the adverse effects of catastrophic forgetting [11] for classification problems. Among them, the method of saving a small portion of data in memory for replay has drawn attention for its simplicity and high performance. A few works [50, 62] have employed such replay method along with NeRF on simultaneous localization and mapping (SLAM). Despite the simplicity and notable performance, such replay-based methods have a critical drawback to be utilized in practical applications: scalability issues due to monotonically increasing memory.

Considering the lack of studies on incremental learning problems with NeRF, we first introduce a new benchmark, where the camera does not revisit the previously seen part of the scene, which becomes no longer accessible afterwards (Figure 2a). Then, we train NeRF with several representative incremental learning algorithms to evaluate its effectiveness in the context of 3D representation. We observe that they suffer low performance or memory scalability issues. The low performance is mainly due to the absence of the access to previous data, whereas the memory scalability issue is due to the need for increasing memory to store previous data.

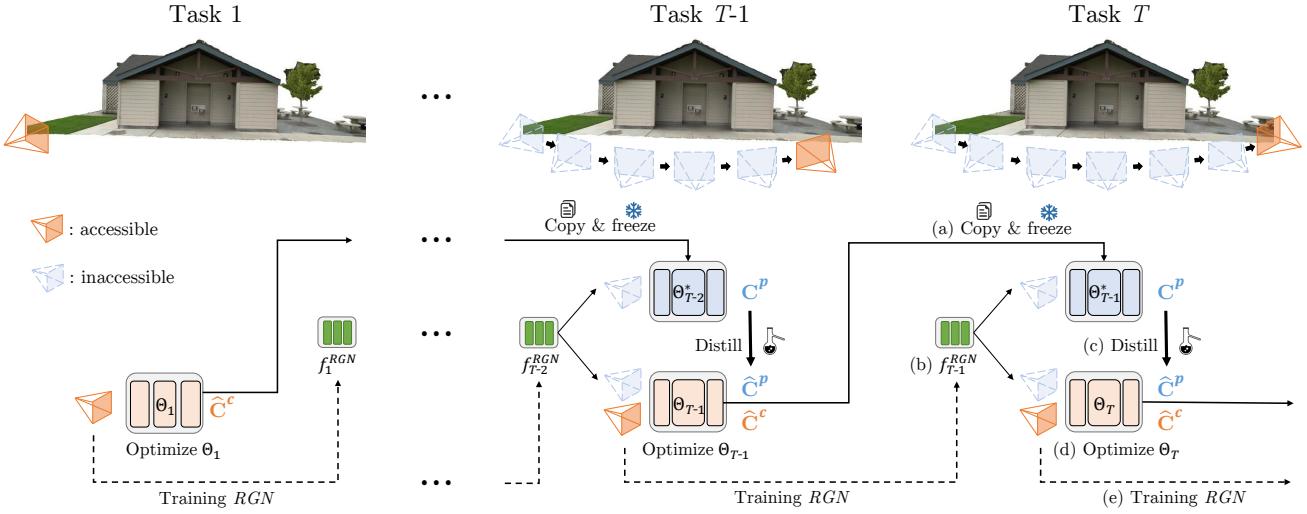
To balance the trade-offs, we introduce a **Memory-Efficient Incremental Learning of NeRF (MEIL-NeRF)**. In order to prevent catastrophic forgetting without increasing memory, we turn our attention to a neural network itself. In NeRF, a neural network produces the pixel RGB values, when rays or viewpoints are given as input. Thus, we consider a neural network as a memory storage for the pixel RGB values of the scene. The proposed perspective raises

a question: which rays should we give to NeRF such that previously seen pixel RGB values of the scene will be retrieved? Only when rays are directed at the scene, we will be able to extract the pixel RGB values of the scene from the network. In this work, we answer the question by introducing another small network, named Ray Generator Network (RGN), that is trained to produce the previously seen rays directed towards the scene. Then, we feed the produced rays into NeRF to obtain previous RGB values of the scene, which are used to train NeRF in a self-distillation [61] manner to prevent forgetting of previous RGB values.

The experimental results demonstrate that our proposed framework greatly reduces the adverse effects of catastrophic forgetting in NeRF without increasing memory, as shown in Figure 2b. The memory-efficient incremental learning algorithm is made feasible by considering NeRF as memory storage and using RGN to remember which rays to feed NeRF to extract previous RGB values.

## 2. Related Works

**Neural Radiance Fields.** There has been several attempts in implicitly expressing 3D geometry either by learning signed distance function [42], occupancy probability [35], or color information at spatial location [36, 47]. Among the attempts, neural radiance fields (NeRF) [36] has emerged as one of promising methods for its high reconstruction quality, spurring a large number of research works on improvements [3, 4, 9, 18, 22, 27, 28, 33, 37, 44, 52–55, 57–59] and a wide range of applications [56]: controllable human avatar [10, 29, 43], realistic game [17], robotics [20, 41, 50, 62], 3D-aware image generation [6, 7], and data compression [13]. Given images and the corresponding camera poses, NeRF casts camera rays, queries sample points and view direction, and finally synthesizes an image by accumulating the output colors and densities along the camera rays. Since NeRF assumes access to all data (i.e., viewpoints and the corresponding scene colors) for training, NeRF faces



**Figure 3. Overall Pipeline of Proposed Method.** We present the learning details of the proposed method. (a) Before starting each task, we copy and freeze the previous parameters for distillation. (b) We create rays for distillation using *Ray Generator Network* (RGN) that remembers which rays are important. (c) We retrieve past color information by querying sample points along the generated rays from the saved parameters which is treated as memory storage. (d) In each task, the network learns current task data(orange) and the distilled past data(blue) at the same time. (e) After the training of each task ends, we update RGN with current task rays and distilled past rays.

challenges in practical applications (e.g., edge devices with a limited amount of memory), where such assumption does not hold. Consequently, NeRF is required to learn the scene with the online stream of data without access to previously observed data.

**Incremental learning.** Under the incremental learning scenario as described above, neural networks are known to forget the previously learned knowledge while learning new knowledge, which is referred to as catastrophic forgetting [15]. There has been many studies to mitigate catastrophic forgetting for classification tasks. Incremental learning algorithms can be divided into three methodologies: regularization, parameter isolation, and replay [11]. Regularization-based methods aim to retain the learned mapping for past data (old knowledge) while learning a mapping for new data (new knowledge). To do so, several works [1, 8, 23, 60] try to find which parameters are important for old knowledge and regularize the changes in those important parameters while learning new knowledge. In contrast, other works aim to regularize the changes in outputs for previously learned classes [12, 26] while learning new classes in the context of classification. Meanwhile, parameter isolation methods [19, 31, 32, 45] attempt to learn a separate sub-network for each type of data (i.e., task). Last but not least, replay-based methods aim to preserve old knowledge by storing the selected samples of past data, which are used together with new data during training.

**Incremental Learning for Neural Radiance Fields.** There has been very few studies on trying to incorporate NeRF with incremental learning tailored for specific applications, such as simultaneous localization and mapping (SLAM).

iMAP [50] continuously select and save few data at each keyframe. On the other hand, NICE-SLAM [62] and NeRF-SLAM [46] employ hierarchical volumetric structures by reserving multi-scale spatial grid locations with feature representation and different NeRF, respectively. However, these methods mostly suffer the memory scalability issues as the required amount of memory increases (either for model or data) with the scale of space.

In this work, we observe that prior works either suffer low performance or memory inefficiency. To balance between the trade-offs, we propose a new perspective that NeRF can act as a storage: given appropriate rays directed at the scene, it gives the corresponding RGB values. Thus, if a framework learns which rays are directed at the scene, past data (RGB values seen from past viewpoints) can be easily retrieved by feeding the rays into NeRF. Then, NeRF can be trained on new data along with retrieved past data to prevent forgetting. Upon the motivation, we introduce another network, which learns which rays are relevant for the given scene. As a result, our framework alleviates catastrophic forgetting for NeRF with constant memory under incremental learning scenarios.

### 3. Problem Statement

Before delving into details of the proposed framework, we start with defining the problem statement of neural radiance fields (NeRF) under incremental learning scenarios. In particular, we consider task incremental learning scenarios [11, 26], where a batch of data (i.e., task) comes in sequentially. In the context of NeRF, each task  $\mathcal{T}$  is con-

structed with  $\mathcal{N}$  number of images and the corresponding camera poses, where  $\mathcal{N} > 1$  to ensure that geometric information can be obtained at each task. Specifically,  $t$ -th task  $\mathcal{T}_t$  is made of a set of paired data  $(\mathcal{X}^{(t)}, \mathcal{Y}^{(t)})$ , where  $\mathcal{X}^{(t)}$  is a set of rays emitted from the given  $\mathcal{N}$  camera views and  $\mathcal{Y}^{(t)}$  is a set of the corresponding RGB colors. To impose incremental learning, only the latest task is available while previous tasks become inaccessible. Overall, given the formulation, the objective of incremental learning is to find the optimal parameters  $\Theta_{\text{opt}}$  that prevent catastrophic forgetting and perform well across all seen tasks:

$$\Theta_{\text{opt}} = \min_{\Theta} \sum_{t=1}^T \mathbb{E}_{(\mathcal{X}, \mathcal{Y} \sim \mathcal{D})} [\mathcal{L}(\mathcal{F}(\mathcal{X}^{(t)}; \Theta), \mathcal{Y}^{(t)})], \quad (1)$$

where  $T$  is the number of seen task;  $\mathcal{L}$  is a loss function;  $\mathcal{F}$  is a forward mapping (consisting of NeRF network and volume rendering) from  $\mathcal{X}^{(t)}$  to  $\mathcal{Y}^{(t)}$ ; and  $\Theta$  denotes the parameters of the network.

## 4. Method

In this section, we start with a formulation of neural radiance fields (NeRF) in Section 4.1. Then, we delineate our proposed framework, MEIL-NeRF, which prevents the catastrophic forgetting by using past task information retrieved from NeRF itself (Section 4.2) with generated rays from our newly introduced ray generator network (Section 4.3). The overall pipeline is illustrated in Figure 3.

### 4.1. Preliminaries

NeRF [36] aims to represent a scene with an MLP network that takes 3D coordinates  $\mathbf{p}$  and view direction  $\mathbf{r}_d$  as input and produces RGB color  $\mathbf{c}$  and volume density  $\sigma$  as output. The corresponding pixel color  $\mathbf{C}$  of the given camera ray  $\mathbf{r}$  is estimated by volume rendering [25], summing with quadrature rule [34] over sampled 3D points along a view direction  $\mathbf{r}_d$  of given camera ray  $\mathbf{r} = (\mathbf{r}_o, \mathbf{r}_d)$ , where each sampled point is obtained by  $\mathbf{p}_i = \mathbf{r}_o + z_i * \mathbf{r}_d$ ;  $\mathbf{r}_o$  is the coordinate of a camera; and  $z_i$  is the distance from a camera to the sampled point  $\mathbf{p}_i$ . Thus, the estimated pixel color  $\hat{\mathbf{C}}$  of the ray  $\mathbf{r}$  is obtained by

$$\hat{\mathbf{C}} = \sum_{i=1}^P \left( \prod_{j=1}^{i-1} (1 - \alpha_j) \alpha_i \mathbf{c}_i \right), \quad (2)$$

where  $\alpha_i = 1 - \exp(-\sigma_i(z_{i+1} - z_i))$  and  $P$  is the number of sampled points. To avoid notation clutter, we refer to the forward mapping from the ray to the pixel color as  $\hat{\mathbf{C}} = \mathcal{F}(\mathbf{r}; \Theta)$ , where  $\Theta$  is the parameters of the MLP network.

### 4.2. MEIL-NeRF

The core idea of the proposed framework is to retrieve the pixel colors of past camera rays by querying the network with the past rays. In this section, we describe how

---

### Algorithm 1 MEIL-NeRF

---

**Require:**  $\Theta_{T-1}$   $\triangleright$  Parameters trained on previous tasks  
1: Copy and freeze as  $\Theta_{T-1}^*$   
2: **for** iteration **do**  
3:     Sample current data  $(\mathbf{r}^c, \mathbf{C}^c)$   
4:     **if**  $T > 1$  **then**  
5:         Generate past rays  $\mathbf{r}^p$   $\triangleright$  Eq. 6, 7  
6:         Obtain past data  $(\mathbf{r}^p, \mathcal{F}(\mathbf{r}^p; \Theta_{T-1}^*))$   $\triangleright$  Eq. 3  
7:     **end if**  
8:     Update NeRF network  $\Theta_T$   $\triangleright$  Eq. 5, 4  
9: **end for**  
10: Update ray generator network  $f^{\text{RGN}}$   $\triangleright$  Eq. 8

---

the proposed framework uses the retrieved pixel colors of past camera rays to prevent catastrophic forgetting, whereas Section 4.3 discusses how the past rays are generated. To ensure that accurate past task information is retrieved from the network, we copy and free the parameters of the network before training the network on the current  $T$ -th task. Since the network has been trained on  $T - 1$  previous tasks, we use  $\Theta_{T-1}^*$  to denote the frozen parameters. At each training iteration for  $T$ -th task, we obtain the pixel colors of past camera rays from the network with  $\Theta_{T-1}^*$  by feeding it with the past rays  $\mathbf{r}^p$ :

$$\mathbf{C}^p = \mathcal{F}(\mathbf{r}^p; \Theta_{T-1}^*). \quad (3)$$

Treating  $\mathbf{C}^p$  as pseudo ground truth, we optimize  $\Theta_T$  to remember past tasks in a self-distillation [61] manner. Thus, the network is trained to learn the current task  $(\mathbf{r}^c, \mathbf{C}^c)$  and past tasks  $(\mathbf{r}^p, \mathbf{C}^p)$ :

$$\mathcal{L} = \frac{1}{m_c} \sum_{\mathbf{r}^c} \|\hat{\mathbf{C}}^c - \mathbf{C}^c\|^2 + \frac{\lambda_p}{m_p} \sum_{\mathbf{r}^p} \rho(\hat{\mathbf{C}}^p - \mathbf{C}^p), \quad (4)$$

where  $\rho(x) = \sqrt{x^2 + \epsilon^2}$  is a Charbonnier penalty function;  $\lambda_p$  is a hyperparameter that controls the trade-off between learning current and past tasks;  $m_c$  and  $m_p$  are the batch size of the current and past rays; and  $\hat{\mathbf{C}}^c$  and  $\hat{\mathbf{C}}^p$  are estimated colors by the network that is being trained:

$$[\hat{\mathbf{C}}^c, \hat{\mathbf{C}}^p] = \mathcal{F}([\mathbf{r}^c, \mathbf{r}^p]; \Theta_T). \quad (5)$$

While we use L2 loss for learning current data as in standard NeRF training [36], we adopt Charbonnier penalty function [51] for learning past data. Since Charbonnier is a differentiable variant of L1 distance, which better helps preserve past data without blurring that may be caused if L2 loss is employed otherwise. When learning each task, we set  $\lambda_p$  to be small at the beginning and then gradually increase it. Such scheduling facilitates training by making the network focus on learning current task first and then preventing catastrophic forgetting.

Table 1. Quantitative results on the datasets. The best and second best performance are **highlighted** and underlined, respectively.

		Tanks and Temples (Custom)					Replica (Custom)					TUM-RGBD (Custom)			
		Barn	Caterpillar	Family	Ignatius	Truck	Office0	Office3	Office4	Room0	Room1	Room2	Seq0	Seq1	Seq2
PSNR (dB)	NeRF-Incre	14.17	15.90	23.38	17.00	16.57	24.44	16.28	19.55	19.18	17.44	19.10	13.15	13.72	12.56
	EWC [23]	14.70	12.35	19.66	13.45	16.80	24.01	16.78	18.31	19.25	17.88	19.30	13.60	14.25	12.56
	PackNet [32]	15.67	12.77	22.91	19.99	16.76	28.68	20.36	24.54	19.28	24.53	22.55	17.66	17.04	17.46
	iMAP <sup>†</sup> [50]	20.70	21.36	28.57	24.05	22.65	33.82	26.28	31.57	25.07	28.14	27.42	21.89	21.52	21.44
	MEIL-NeRF (ours)	<b>22.14</b>	<b>23.41</b>	<b>29.95</b>	<b>25.03</b>	<b>24.31</b>	<b>34.35</b>	<b>26.52</b>	<b>32.72</b>	<b>25.73</b>	<b>29.47</b>	<b>28.23</b>	<b>24.12</b>	<b>23.78</b>	<b>23.59</b>
	iMAP* [50]	22.05	21.99	30.88	24.85	23.79	35.98	28.97	34.07	26.05	30.16	29.71	23.93	23.61	23.30
	NeRF-Joint [36]	24.51	25.91	33.89	26.59	27.12	39.99	32.85	37.67	29.35	34.36	28.79	25.91	25.86	25.96
MS-SSIM	NeRF-Incre	0.394	0.657	0.890	0.747	0.649	0.679	0.508	0.649	0.552	0.558	0.499	0.345	0.385	0.299
	EWC	0.405	0.531	0.755	0.514	0.657	0.663	0.526	0.629	0.558	0.515	0.504	0.334	0.382	0.281
	PackNet	0.423	0.414	0.877	0.761	0.549	0.775	0.558	0.716	0.487	0.677	0.631	0.517	0.480	0.466
	iMAP <sup>†</sup>	0.756	0.874	0.962	0.940	0.890	<b>0.948</b>	0.895	0.948	0.773	0.891	0.875	0.813	0.782	0.774
	MEIL-NeRF (ours)	<b>0.836</b>	<b>0.909</b>	<b>0.972</b>	<b>0.953</b>	<b>0.925</b>	<b>0.948</b>	<b>0.908</b>	<b>0.960</b>	<b>0.831</b>	<b>0.914</b>	<b>0.896</b>	<b>0.882</b>	<b>0.864</b>	<b>0.858</b>
	iMAP*	0.838	0.841	0.981	0.952	0.917	0.970	0.934	0.968	0.828	0.919	0.917	0.878	0.853	0.835
	NeRF-Joint	0.907	0.944	0.992	0.966	0.959	0.987	0.972	0.984	0.910	0.972	0.904	0.919	0.910	0.911

<sup>†</sup> reproduced to have similar memory usage to ours

<sup>\*</sup> reproduced to have similar performance to ours without restricting the memory usage

### 4.3. Ray Generator Network

Thus, the naïve random generation of rays will most likely not point at the scene of interest, failing to provide data of past tasks. Another possible alternative is to store the camera rays regarding past tasks, but it has the memory scalability issues. Instead, we introduce a ray generator network (RGN), which is trained to produce the past camera rays that are directed to the scene of interest. RGN is implemented by a small MLP network  $f^{\text{RGN}}$  that learns to map from a real number  $x \in [0, 1]$  to the origin  $\mathbf{r}_o^*$  and the direction  $\mathbf{r}_d^*$  of a ray of unit length:

$$(\mathbf{r}_o^*, \mathbf{r}_d^*) = f^{\text{RGN}}(x). \quad (6)$$

Since we have  $\mathcal{N}$  number of cameras and their corresponding principal rays for each task, the  $(T - 1)\mathcal{N}$  equally spaced numbers  $\mathbf{x}^{T-1} = [0, \frac{1}{((T-1)\mathcal{N}-1)}, \frac{2}{((T-1)\mathcal{N}-1)}, \dots, 1]$  are assigned to the corresponding past principal rays, where  $x = 0$  and  $x = 1$  correspond to the initial and latest principal rays, respectively. Centered around the generated principal rays  $(\mathbf{r}_o^*, \mathbf{r}_d^*)$ , we use camera intrinsics to create  $m_p$  non-principal rays  $(\mathbf{r}_o, \mathbf{r}_d)$  around the principal rays but with the same origin  $\mathbf{r}_o = \mathbf{r}_o^*$ :

$$\mathbf{r}_d = \frac{\mathbf{f} \cdot \mathbf{r}_d^* + s \cdot \cos(u_\theta) \cdot \mathbf{p}_1 + s \cdot \sin(u_\theta) \cdot \mathbf{p}_2}{|\mathbf{f} \cdot \mathbf{r}_d^* + s \cdot \cos(u_\theta) \cdot \mathbf{p}_1 + s \cdot \sin(u_\theta) \cdot \mathbf{p}_2}, \quad (7)$$

where  $\mathbf{p}_1, \mathbf{p}_2$  are unit vectors perpendicular to the principal ray obtained by the Gram-Schmidt process;  $\mathbf{f}$  is a focal length; and  $s, u_\theta$  are random variables with uniform distributions as  $s \sim \mathcal{U}[0, \sqrt{W^2 + H^2}/2]$  and  $u_\theta \sim \mathcal{U}[0, 2\pi]$ . Such formulation enables us to create a substantial amount of rays while reducing the learning complexity of RGN. Overall, past rays can be retrieved from RGN by feeding it with  $x \sim \mathcal{U}[0, 1]$ , along with non-principal rays by Equation (7).

Since new rays likewise come in sequentially, we perform incremental learning of RGN via self-distillation. At the end of each task, we update RGN with the current principal rays  $(\mathbf{r}_o^*, \mathbf{r}_d^*)_T$  and the past principal rays  $(\mathbf{r}_o^*, \mathbf{r}_d^*)_{1:T-1}$

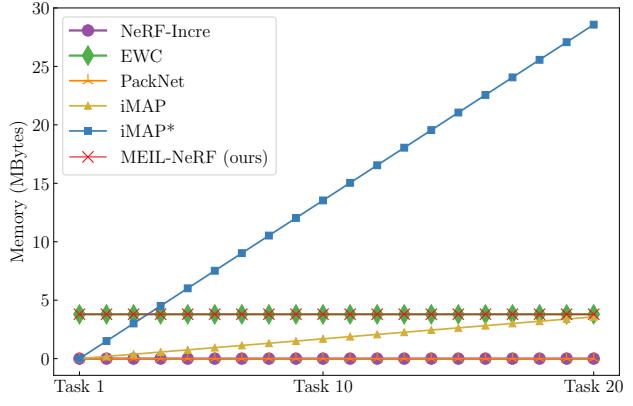


Figure 4. **Additional Memory Usage.** iMAP and iMAP\* increases memory storage as it saves the ground truth data steadily. PackNet does not use additional memory, and EWC and our method require additional memory as much as the size of the network. The proposed method utilizes the information stored in the network and achieve best performance with a small fixed memory.

that are retrieved from RGN conditioned on  $\mathbf{x}^{T-1}$ . Then, we train RGN to map  $T \cdot \mathcal{N}$  equally spaced numbers  $\mathbf{x}^T \in [0, 1]$  to  $(\mathbf{r}_o^*, \mathbf{r}_d^*)_{1:T}$  via MSE loss:

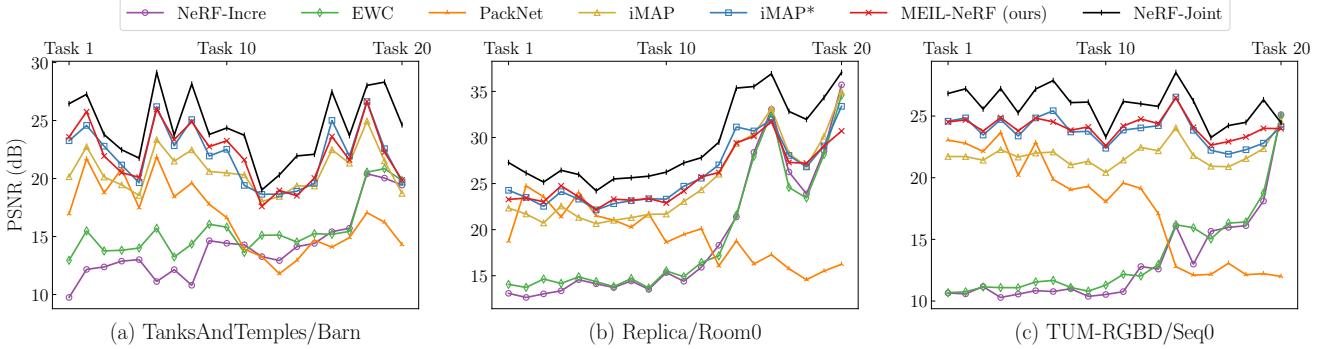
$$\mathcal{L}_{\text{RGN}} = \|(\mathbf{r}_o^*, \mathbf{r}_d^*)_{1:T} - f^{\text{RGN}}(\mathbf{x}^T)\|^2. \quad (8)$$

## 5. Experiment

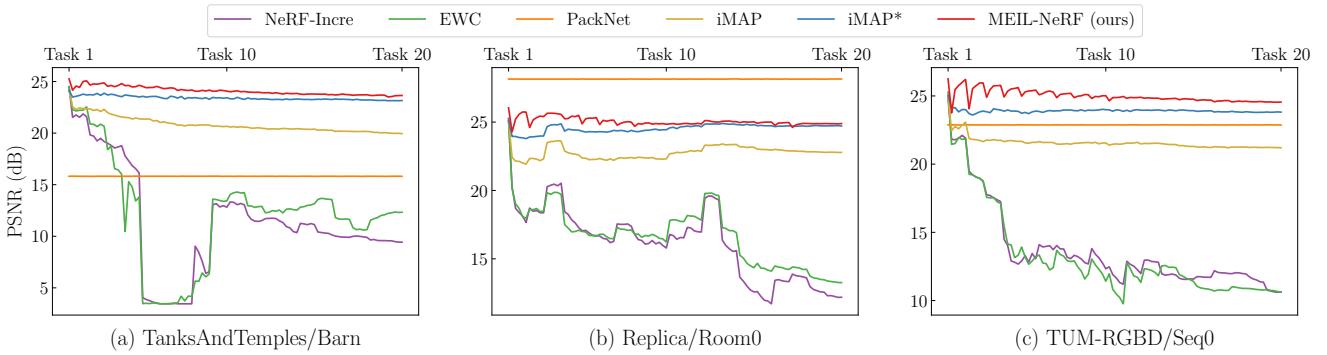
We describe the experiment settings in Section 5.1; dataset construction for incremental learning in Section 5.2; and report the experiment results and ablation studies in Section 5.3 and 5.4, respectively.

### 5.1. Experiment Settings

We set the vanilla NeRF under incremental scenario as a baseline and name it as *NeRF-Incre*. NeRF-Incre is incrementally trained with only current task data, making it susceptible to catastrophic forgetting and serving as lower bound. We also train the vanilla NeRF with the standard



**Figure 5. Evaluation of All Tasks.** The figure shows the evaluation of all tasks after training the last task in terms of PSNR. The baseline (NeRF-Incre) shows significant performance degradation in the initial tasks. EWC shows no improvement, and PackNet shows performance degradation as the number of available parameters decreases. For iMAP\* (similar performance to ours), the memory increases quickly. iMAP (similar memory to ours) shows low performance. Note that we randomly select one from each dataset and display MS-SSIM results in supple.



**Figure 6. Performance Degradation of the First Task.** The figure displays the performance degradation of the first task (Task 1) as the network continuously learns all the tasks. In the baseline (NeRF-Incre), the network rapidly forgets the first task. EWC is similar to the baseline, and PackNet shows no forgetting because the parameters of the first task are fixed. iMAP exhibit a stable performance after a slight drop in early stage, due to the lack of exemplar. In contrast, iMAP\* and our method significantly decelerate forgetting speed, and even our method slightly outperforms iMAP\*. The fluctuation in our method is due to the  $\lambda_p$  scheduling which is described in supple. We display MS-SSIM results in supple due to lack of space.

joint training, where all data is made available at all times. We label it as *NeRF-Joint* and regard it as upper bound.

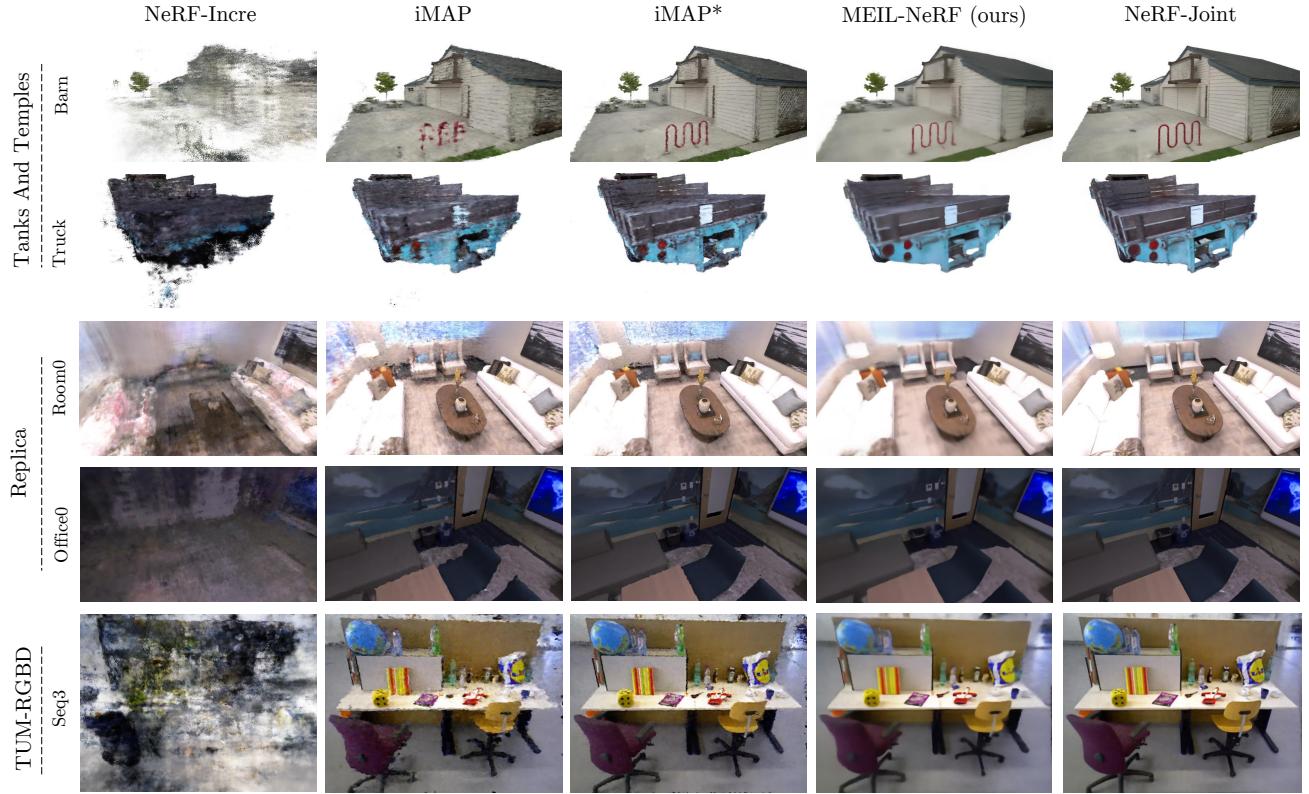
We also compare against the representative incremental learning algorithms (regularization, parameter isolation, and replay) on NeRF to observe how the algorithms developed for classification tasks generalize to NeRF under incremental scenarios. Selected as the representative of regularization, *Elastic Weight Consolidation* (EWC) [23] penalizes the changes in parameters that are important for past tasks. For parameter isolation, we adopt the idea of *PackNet* [32], which prunes less important parameters and retrain to make room for the next task. As for replay, we follow the general implementations of *iMAP* [50], which has tailored replay algorithm to SLAM. At the end of each task, we store random exemplars weighted by its loss, and repeat them while learning a new task. Since the performance of replay-based algorithms depends on the amount of memory reserved for each task, we vary the amount of memory size and implement two versions: iMAP that has a similar memory usage to our method and iMAP\* that has similar

performance to ours without restricting the memory usage.

We set the number of images  $\mathcal{N} = 5$  for each task by default and run  $3000 * \mathcal{N}$  iterations per task. We use  $m_c = 4096$  rays for learning the current task at every iteration. Our method and iMAP prevent forgetting by also training on  $m_p = m_c/2$  past rays, which are either generated (our method) or sampled from the memory buffer (iMAP).

## 5.2. Datasets

We construct datasets for incremental scenarios, based on *Tanks and Temples* [24], *Replica* [48], and *TUM-RGBD* [49]. To simulate the incremental scenarios, we rearrange the order of camera such that it moves sequentially and we select a portion of the dataset such that the previous images are not revisited. *Tanks and Temples* consists of photos taken while moving around a scene or object of interest. The dataset consists of scenes of various sizes ranging from a large scene (e.g., Barn) to a small object (e.g., Family). We rearrange the camera to turn in one direction, ending the sequence before it reaches the first scene.



**Figure 7. Qualitative Results.** We show the reconstructed images for the early task after training of all tasks in an incremental manner. The baseline (NeRF-Incre) severely suffers catastrophic forgetting and forgets the early-task scene information, whereas iMAP, iMAP\*, and our method preserves the scene of interest. iMAP and iMAP\* are our reproduction such that iMAP has similar memory usage to ours, while iMAP\* is produced to have similar performance to ours but without memory constraint (please refer to Figure 4). iMAP shows many artifacts due to the lack of samples. iMAP\* with more memory displays improved sharp appearance, but still has distortion artifacts. Despite using the similar memory usage to iMAP, our MEIL-NeRF shows more plausible and visually pleasing results than iMAP. Furthermore, the reconstruction quality of MEIL-NeRF is similar to iMAP\*, despite using substantially less memory than iMAP\*. Slight blur in the reconstruction by MEIL-NeRF may be due to using diverse generated samples (but with the same number of samples as iMAP\*) we obtain from RGN and NeRF. Please refer to the supplementary material to see more qualitative results on EWC and PackNet and other data sequences.

Replica dataset is a synthetic indoor dataset with accurate camera poses and rendered images. Since the camera moves smoothly, we sub-sample the images so that there is enough change in degrees between tasks. As with Tanks and Temples, we cut a part of the camera sequence, so the network does not review the past images. TUM-RGBD dataset is a dataset made for SLAM, which was taken with a hand-held camera or a camera mounted on a robot. Since images are taken while moving, most sequences have severely blurry pictures. For better evaluation of catastrophic forgetting, we select sequences with a low degree of blur.

### 5.3. Results

We evaluate the models with respect to Peak Signal-to-Noise Ratio (PSNR) and MultiScale Structural Similarity Index Measure (MS-SSIM). We show the average performance of all tasks on Table 1 and the performances of each task on three randomly selected scenes in Figure 5. Figure 6

also displays how the performance of the first task (Task 1) changes as the models learn new tasks over time to show catastrophic forgetting. As expected from NeRF-Incre, the performance on the initial task severely deteriorates due to catastrophic forgetting. EWC fails to reduce the adverse effects of catastrophic forgetting. In fact, EWC is shown to even hinder the learning of new tasks, decreasing the average performance over all tasks. PackNet performs worse on newer tasks since the number of parameters assigned to a new task reduces gradually. On the other hand, PackNet maintains the performance on the initial task since the parameters for the first task are fixed after pruning. Despite directly saving past tasks data, iMAP performs lower than our MEIL-NeRF, which uses similar amount of memory. With constant memory usage, our MEIL-NeRF effectively mitigates catastrophic forgetting and performs on par with iMAP\*, whose memory usage is substantially larger than our method, as shown in Figure 4. This demonstrates the

Table 2. Ablation for batch size of current task on TUM-RGBD/Seq0 ( $m_p = m_c/2$ )

PSNR / MS-SSIM	$m_c = 512$	$m_c = 1024$	$m_c = 2048$	$m_c = 4096$	$\Delta$
iMAP*	23.69 / 0.867	23.78 / 0.871	23.98 / 0.880	23.93 / 0.878	0.24 / 0.011
MEIL-NeRF	22.89 / 0.850	23.44 / 0.866	23.77 / 0.874	24.12 / 0.882	1.23 / 0.032
NeRF-Joint	25.31 / 0.909	25.63 / 0.914	25.93 / 0.919	26.10 / 0.922	0.79 / 0.013

efficiency and effectiveness of using NeRF as a memory storage and retrieving past information from NeRF itself.

We can gain more insight from observing the qualitative results shown in Figure 7. The figure shows the reconstruction of images for early tasks after incremental learning has finished (i.e., all tasks have been learned sequentially). The replay-based method iMAP\* shows relatively sharp images, but there are some distortion artifacts. Since only selected samples are stored and repeatedly used, iMAP\* overfits to specific rays, compromising multiview consistency. On the contrary, MEIL-NeRF shows smoother reconstruction without the artifacts. Our method has less artifacts since it can generate various samples by using RGN and NeRF. The reconstruction by MEIL-NeRF may have gotten a smoothing effect affected by accumulated errors from the iterative distillation process. Selecting important generated samples for training NeRF to obtain sharper and more accurate reconstruction can be one of the interesting future research directions.

## 5.4. Ablations

Table 2 shows how performance changes with the current task batch size  $m_c$ . Compared to iMAP\*, our method deteriorates relatively quickly as the batch size becomes smaller. The smaller batch size leads to underfitting of each task. Thus, the retrieved past task information from the network can become more inaccurate, degrading the performance.

We also perform an ablation study on loss function used for past tasks to justify our choice of Charbonnier loss function, as shown in Figure 8. L1 and Charbonnier loss exhibit higher performances than L2 loss, while  $\lambda_p$  scheduling provides further improvement. L1 loss and its variants (e.g., Charbonnier) known to facilitate the learning of networks to preserve edges and sharpness in reconstructed images, demonstrating even more performance improvement in MS-SSIM, as shown in the supplementary material.

Figure 9 reports the ablation experiments on ray generator network (RGN). The result shows that past color information cannot be retrieved from NeRF if random rays are used (even with 10 times more rays), underlining the importance of using desirable rays that point at the scene. To show the effectiveness of RGN in remembering past rays, we compare against using GT past rays. RGN remarkably provides similar performance to using GT past rays, underscoring the effectiveness of RGN in remembering past rays with constant memory usage.

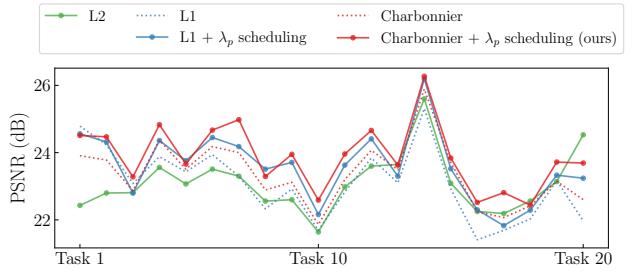


Figure 8. **Ablation for Loss.** We show TUM-RGBD/Seq0 results with L2, L1 and Charbonnier loss. The edge-preserving property of L1 and Charbonnier loss improve the performance especially in MS-SSIM.  $\lambda_p$  scheduling also helps improve performance by organizing a learning curriculum, from learning new tasks to remembering past information.

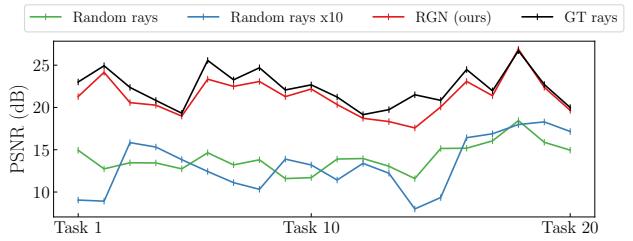


Figure 9. **Ablation for Ray Generator Network.** We show Tanks and Temples/Barn results with GT past rays, rays from RGN, and rays cast randomly. Exhibiting remarkably similar performance to using GT past rays, RGN demonstrates the capability of remembering past rays without increasing memory. Random rays still perform poorly even with using ten times more rays.

## 6. Limitation

Since MEIL-NeRF uses NeRF as a memory storage and query NeRF for past information, it has similar high-latency limitations as NeRF. NeRF has high-latency due to the procedure of sampling points and integrating them along camera rays. We believe that latency problems can be overcome through employing recent works [38, 39] that have attempted to improve the latency of NeRF.

## 7. Conclusion

In this work, we aim to further push NeRF towards practical scenarios, where we focus on incremental scenario. Under the formulated incremental scenario, previous incremental learning algorithms have failed to work for NeRF or demand for a large amount of memory. Thus, we propose a **Memory-Efficient Incremental Learning** algorithm for **NeRF (MEIL-NeRF)**. MEIL-NeRF manages to alleviate catastrophic forgetting with constant memory usage by treating the network itself as a mean of storing past information. We hope this work can inspire future research works on NeRF in incremental learning scenarios.

# Supplementary Material for MEIL-NeRF: Memory-Efficient Incremental Learning of Neural Radiance Fields

Jaeyoung Chung<sup>1</sup>

Kanggeon Lee<sup>1</sup>

Sungyong Baik<sup>2</sup>

Kyoung Mu Lee<sup>1</sup>

<sup>1</sup>Dept. of Electrical and Computer Engineering, ASRI, Seoul National University, Seoul, Korea

<sup>2</sup> Dept. of Data Science, Hanyang University, Seoul, Korea

robot0321@snu.ac.kr, dlrkdrjs97@snu.ac.kr, dsybaik@hanyang.ac.kr, kyoungmu@snu.ac.kr

In this supplementary document, we show the results and ablations omitted from the main paper due to lack of space and describe the details of comparison methods. Section S1 reports quantitative comparison results in terms of MS-SSIM; Section S2 exhibits qualitative results on EWC and PackNet and other data sequences; Section S3 includes the additional ablation studies for the proposed method; Section S4 elaborates on the details of the ray generator network (RGN); Finally, Section S5 provides implementation details and ablation studies on hyperparameters for other methods.

## S1. Quantitative Results

We show the MS-SSIM evaluation of all tasks after training is finished in Figure S2 and the performance degradation of the first task (Task 1) in Figure S3, in terms of MS-SSIM. They show similar trends to the PSNR evaluation in Figure 5 and Figure 6 of the main paper, respectively.

## S2. Qualitative Results

Figure S1 shows the qualitative results of EWC and PackNet omitted from Figure 7. EWC exhibits severe catastrophic forgetting, similar to NeRF-Incre. PackNet has high reconstruction quality in the early task like NeRF-Joint, since the parameters for the learned tasks are fixed after pruning and re-training. However, PackNet fails to provide high reconstruction quality in newer tasks, as discussed with additional qualitative results below. We also present other qualitative results in Figure S11, Figure S12, and Figure S13 displaying the images in some tasks after training is finished. In essence, the figures are the qualitative counterparts of Figure 5 and Figure S2. NeRF-Incre and EWC demonstrate severe catastrophic forgetting. In

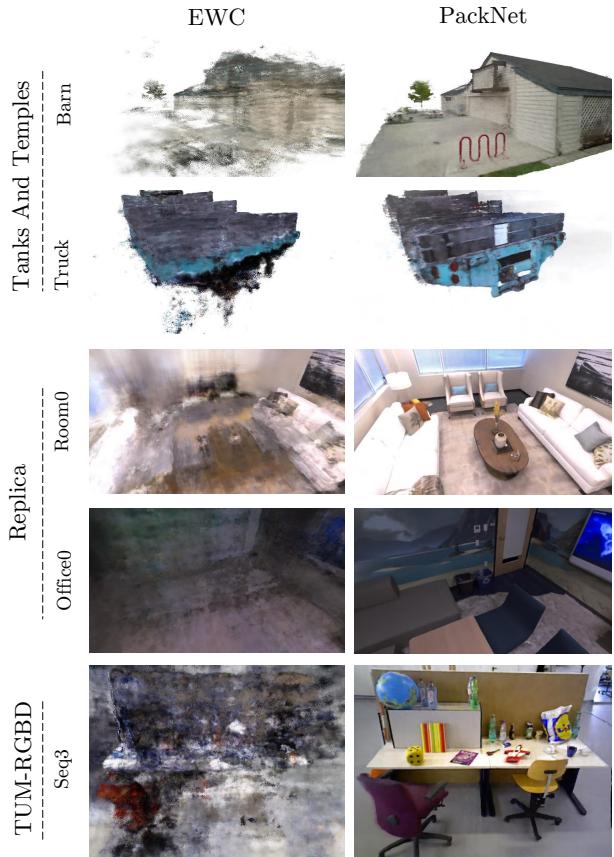


Figure S1. **Qualitative Results.** We visualize the images reconstructed by EWC and PackNet for the early task after the incremental learning of all tasks, omitted in Figure 7. While PackNet provides high quality in earlier tasks, it gives poor quality in newer tasks since very few parameters are reserved for learning new tasks, as shown in Figure S11 - S13.

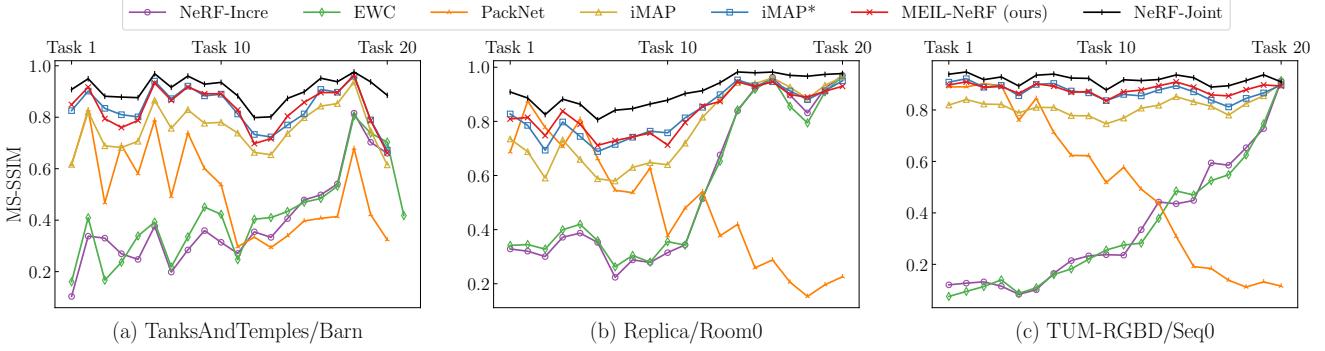


Figure S2. **Evaluation of All Tasks.** The figure shows the MS-SSIM evaluation averaged over all tasks after the incremental learning of all tasks, providing the MS-SSIM counterpart of Figure 5.

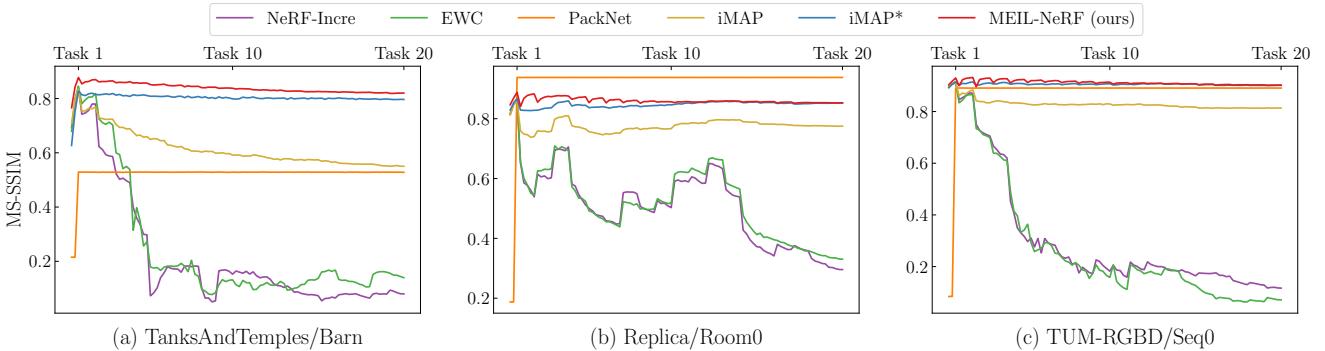


Figure S3. **Performance Degradation of the First Task.** The figure shows the MS-SSIM performance degradation (and thus catastrophic forgetting) of the first tasks (Task 1) after the incremental learning of all tasks, providing the MS-SSIM counterpart of Figure 6.

contrast, PackNet performs worse on newer tasks since the number of available parameters for learning new tasks gradually decreases. iMAP and iMAP\* prevent catastrophic forgetting to a great extent and provide good reconstruction quality across tasks; however, many blurs and distortions are present in the reconstructed images. Our method shows relatively smoothed images without noticeable artifacts.

Table S1. Ablation for batch size evaluated in Tanks and Temples/Barn

PSNR / MS-SSIM	$m_p = m_c/4$	$m_p = m_c/2$	$m_p = m_c$
$m_c = 4096$	21.92 / 0.821	22.14 / 0.836	22.30 / 0.835
$m_c = 1024$	21.72 / 0.805	21.30 / 0.801	21.82 / 0.812

### S3. Ablations for the proposed method

**Loss function used for past tasks.** In Figure S4, we conduct an ablation study on the loss function used for remembering past tasks in terms of MS-SSIM. L1 loss and its variants provide more notable performance improvement in MS-SSIM, compared to the PSNR results shown in Figure 8. Considering that MS-SSIM measures the structural similarity, the result corroborates the previous finding that

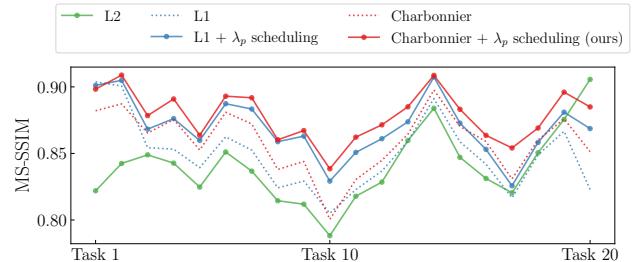
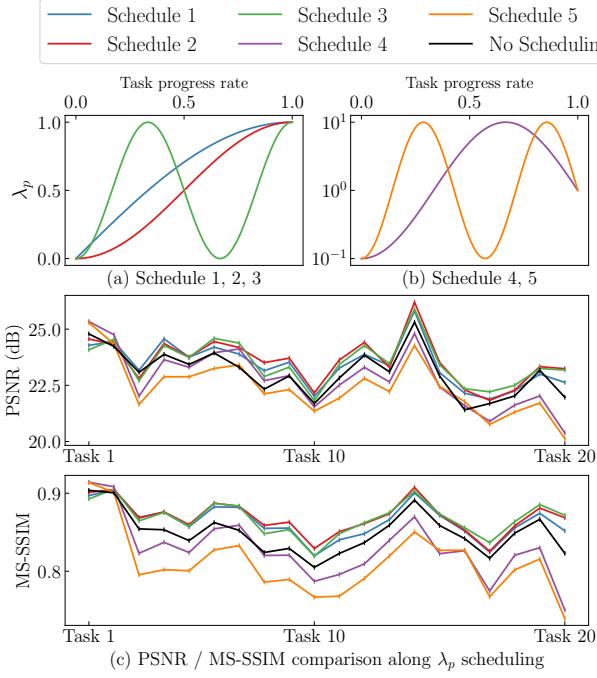


Figure S4. **Ablation for Loss.** This figure provides the MS-SSIM counterpart of Figure 8. We show the MS-SSIM evaluation on TUM-RGBD/Seq0 sequence after training our model with L2, L1, and Charbonnier loss to preserve the knowledge of past tasks. L1 loss and its variants (e.g., Charbonnier) demonstrate even more performance improvement in MS-SSIM. The result corroborates the previous finding that L1 loss and its variants lead to better edge-preserving capability.

L1 and Charbonnier loss provide better edge-preserving capability, reducing the errors from the iterative process of self-distillation.

**Hyperparameter  $\lambda_p$  scheduling.** A hyperparameter  $\lambda_p$  in Equation (4) controls a trade-off between learning a new



**Figure S5. Hyperparameter Scheduling.** We show the scheduling examples for the hyperparameter  $\lambda_p$  in (a) and (b). We present the performance of each schedule in (c), evaluated in TUM-RGBD/Seq0. We use schedule 2, showing slightly better results among them.

task and remembering past tasks. So we establish a learning curriculum for the hyperparameter during the learning of each task, which guides NeRF to focus on learning a new task first and then gradually shift the focus to remembering past tasks. We implement the learning curriculum by scheduling the hyperparameter  $\lambda_p$  according to the progress rate of learning in one task. Figure S5 shows the PSNR and MS-SSIM assessment of several designs of  $\lambda_p$  scheduling. For the task progress rate  $r$ , the schedule 1 ~ 5 are implemented as,

$$\lambda_p = \cos\left(\frac{\pi}{2}(1 - r)\right), \quad (\text{S1})$$

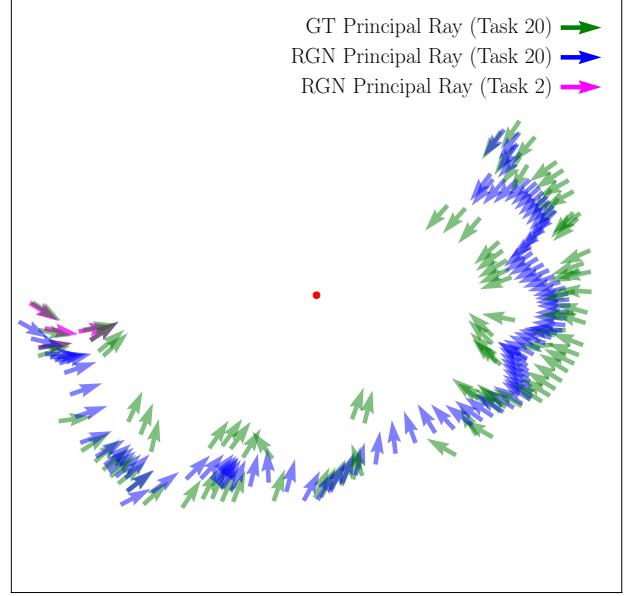
$$\lambda_p = \frac{1}{2} \cos(\pi(1 + r)), \quad (\text{S2})$$

$$\lambda_p = \frac{1}{2} \cos(\pi(1 + 3r)), \quad (\text{S3})$$

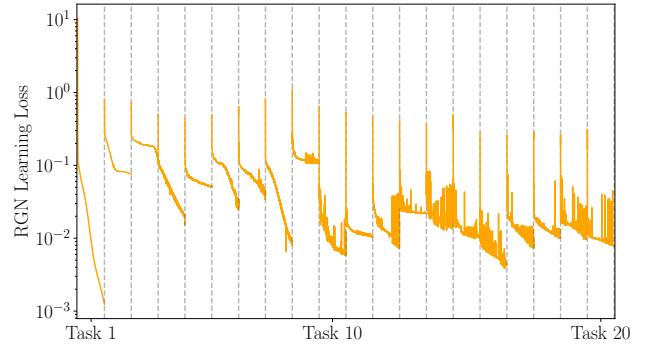
$$\lambda_p = 10^{\cos(\pi(1 + \frac{3}{2}r))}, \quad (\text{S4})$$

$$\lambda_p = 10^{\cos(\pi(1 + \frac{7}{2}r))}, \quad (\text{S5})$$

respectively. Schedule 1 ~ 3, which progressively climb from zero to one, generally perform better than Schedule 4 ~ 5. We select the second one (schedule 2), which demonstrates slightly better results among Schedule 1 ~ 3. **Batch size of past rays  $m_p$ .** Table S1 shows the change in the overall performance according to the batch size for past



**Figure S6. Visualization of GT rays and RGN generated rays.** We show the GT principal rays and RGN-generated principal rays in the Tanks and Temples/Barn dataset. The tail and direction of the arrow indicate the camera origin and direction, respectively. For convenience, we present the location of the scene with a red dot.



**Figure S7. RGN Training Loss.** We show how the average RGN training loss for each task changes as the number of tasks increases. Although RGN is a tiny network, it preserves the past ray information well by reaching a certain level of loss.

task  $m_p$ . In general, the performance tends to increase with  $m_p$ . Considering the trade-off between the performance and the computational efficiency, we choose  $m_p = m_c/2$ .

#### S4. Details of RGN

We illustrate the structure of RGN and the training/inference process in Figure S8. During training on each task, RGN  $f_T^{RGN}$  learns to map the past principal rays from the previous RGN network  $f_{T-1}^{RGN}$  and the current GT principal rays. Such self-distillation training scheme is simi-

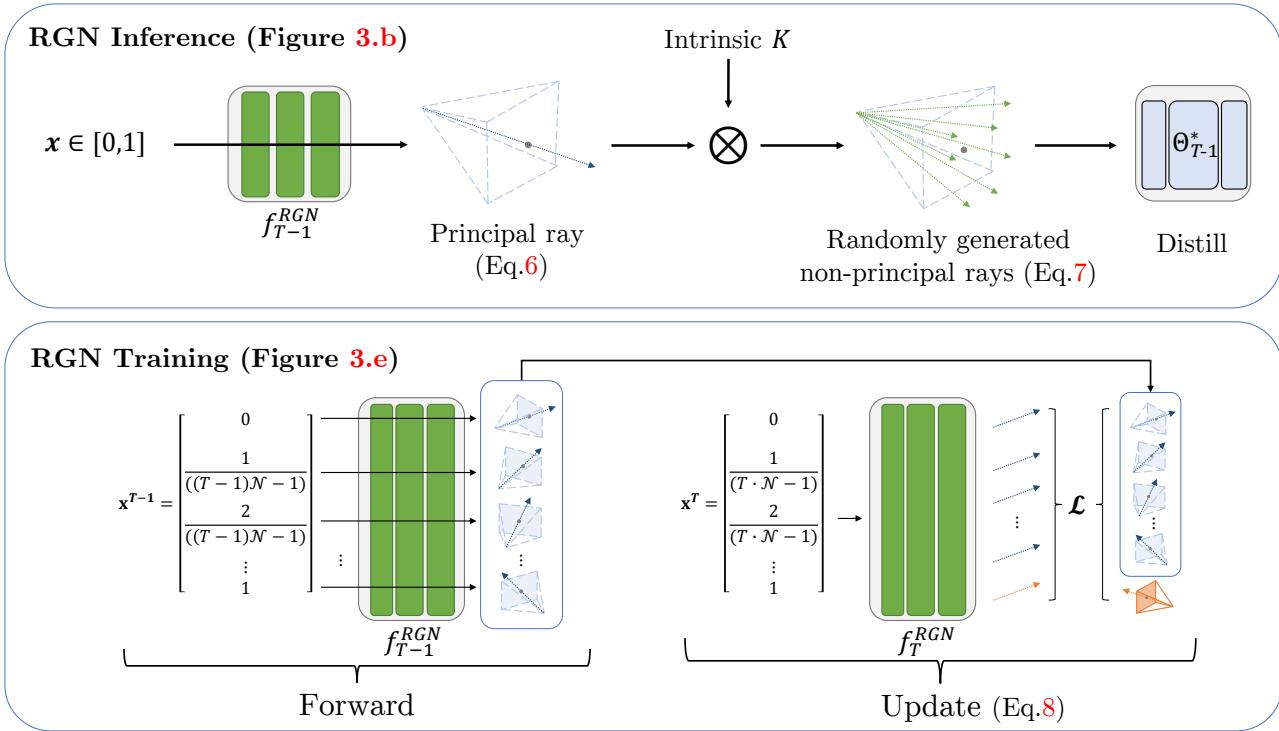


Figure S8. **Details of ray generator network.** We visualize the details of RGN, described in the main paper. The RGN inference (top) and training (bottom) blocks correspond to (b) and (e) in Figure 3 in the main paper, respectively.

lar to how we incrementally train NeRF. In inference time, we randomly select a number in  $[0, 1]$  and make a principal ray with RGN. Then, using the camera intrinsic  $K$ , we randomly generate non-principal rays around principal rays. As a result, we generate rays form a shape of cone emitted from the center of camera that covers the region of image plane. This design helps reduce the burden of learning on RGN, at a small cost of accuracy. Although we use a tiny network consisting of 3 layers, each with 16, 64, and 32 hidden units respectively, Figure S7 demonstrates that even such a small network can remember the past principal rays. This demonstrates the effectiveness of our RGN design.

The purpose of RGN is not to reproduce the rays of past tasks, but create the rays directed to the scene of interest near the past task cameras. Therefore, errors under certain level are allowed when reconstructing the past cameras. As shown in Figure S6, there are some accumulated errors in estimated camera poses. However, as shown in Figure 9, these generated and approximated rays are still effective in directing to the scene of interest near the past task cameras and thereby alleviating catastrophic forgetting.

## S5. Implementation details of other methods

In this section, we delineate the implementation details, especially hyperparameters, regarding other methods that

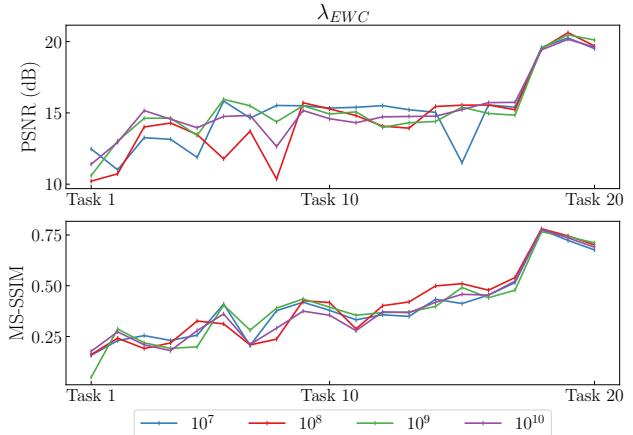


Figure S9. **Ablation for EWC.** We show the performance changes along the EWC weight, evaluated in Tanks and Temples/Barn

are used for comparisons. For fair comparisons, we choose the hyperparameters through ablation studies.

### S5.1. Details of EWC

We show the performance change in Figure S9 as we vary the weight of parameter regularization loss term for EWC [23]. Since the regularization loss is generally small, we multiply the loss by a large weight, as in the original

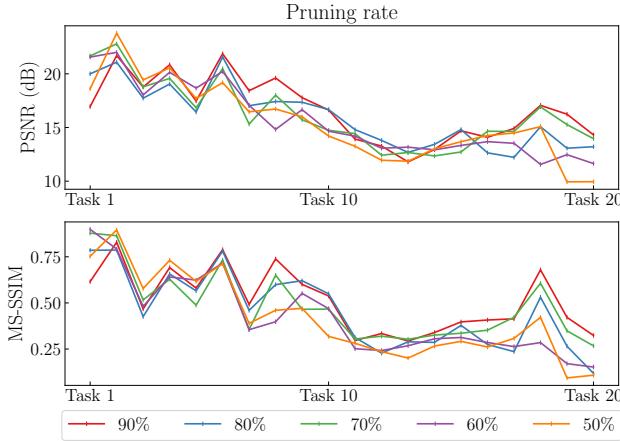


Figure S10. **Ablation for PackNet.** We show the performance changes along the pruning rate in Packnet, evaluated in Tanks and Temples/Barn

EWC paper. Regardless of weight values, the figure exhibits severe catastrophic forgetting in EWC. Throughout the experiments in this work, we use  $10^8$  for the weight, making the regularization loss to be approximately 10% of NeRF loss.

## S5.2. Details of PackNet

In PackNet [32], we repeat the process of training-pruning-retraining-freezing in order. Depending on a pruning rate in this process, the performance of PackNet varies, as shown in Figure S10. In the figure, the following trends could be observed. The higher the pruning rate, lower the performance on earlier tasks, but higher the performance on newer tasks. On the other hand, the lower the pruning rate, the higher performance on earlier tasks, but the lower performance on newer tasks. The observed trend is due to the fact that higher pruning rates lead to less parameters used for earlier tasks and more parameters reserved for newer tasks. We set the pruning rate to 0.5 to follow the settings of the original paper while clearly showing the characteristics of PackNet in NeRF.

Table S2. Ablation for the loss function used to train iMAP to remember past tasks. Evaluation is performed on TUM-RGBD/Seq0 data sequence.

<i>(iMAP)</i> Loss for past tasks	PSNR (dB)	MS-SSIM
L2 (iMAP*)	<b>23.85</b>	0.873
L1	23.34	0.876
L1 + $\lambda_p$ scheduling	23.70	<b>0.886</b>
Charbonnier	23.40	0.876
Charbonnier + $\lambda_p$ scheduling	23.70	0.884
MEIL-NeRF (Ours)	24.12	0.882

## S5.3. Details of iMAP

We perform an ablation study on a loss function used for remembering past tasks in the replay strategy, as in the proposed method (see Figure 8 and Figure S4). Table S2 shows the PSNR/MS-SSIM performance of iMAP trained with each loss function on TUM-RGBD/Seq0 data sequence. L1 and Charbonnier, which provide significant improvement in our method, do not have much effect in the replay strategy. Compared to the default L2, there is a slight drop in PSNR and slight improvement in MS-SSIM. As the difference is not significant, we follow the original implementation and use L2 loss function for remembering past tasks in iMAP.

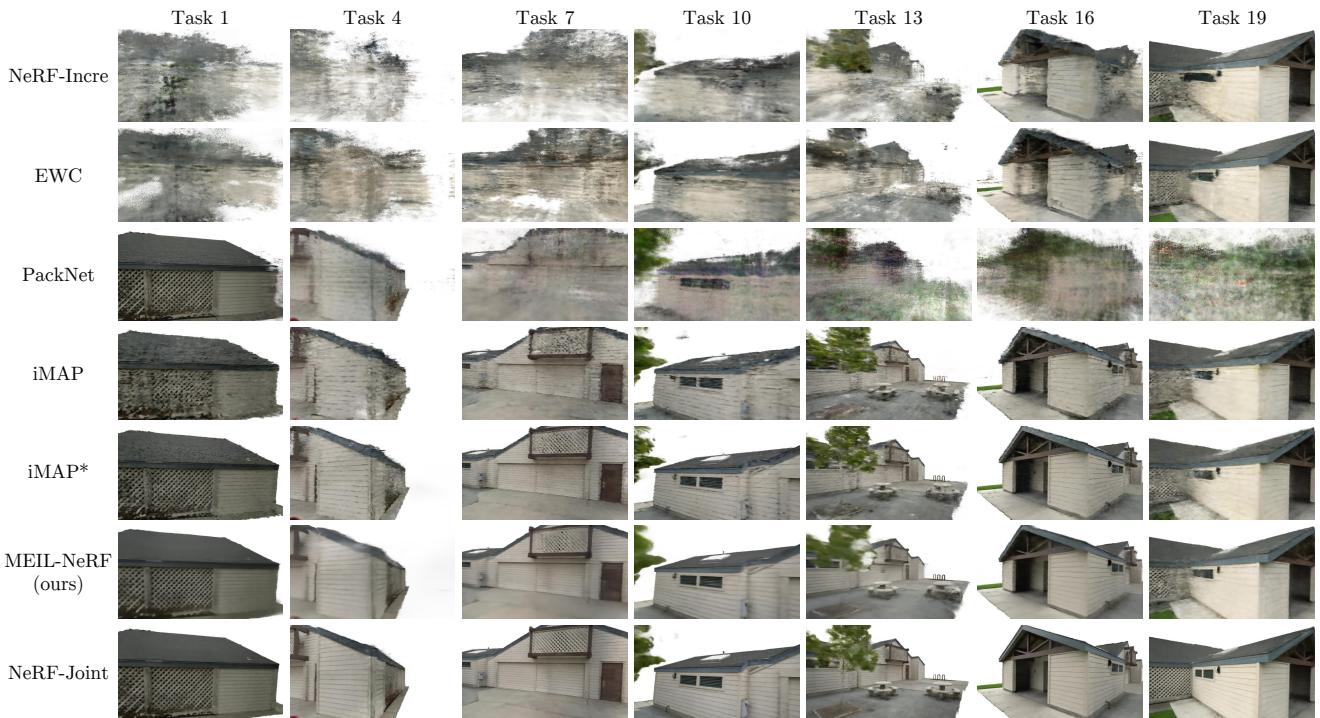


Figure S11. Qualitative Results in Tanks And Temples/Barn.

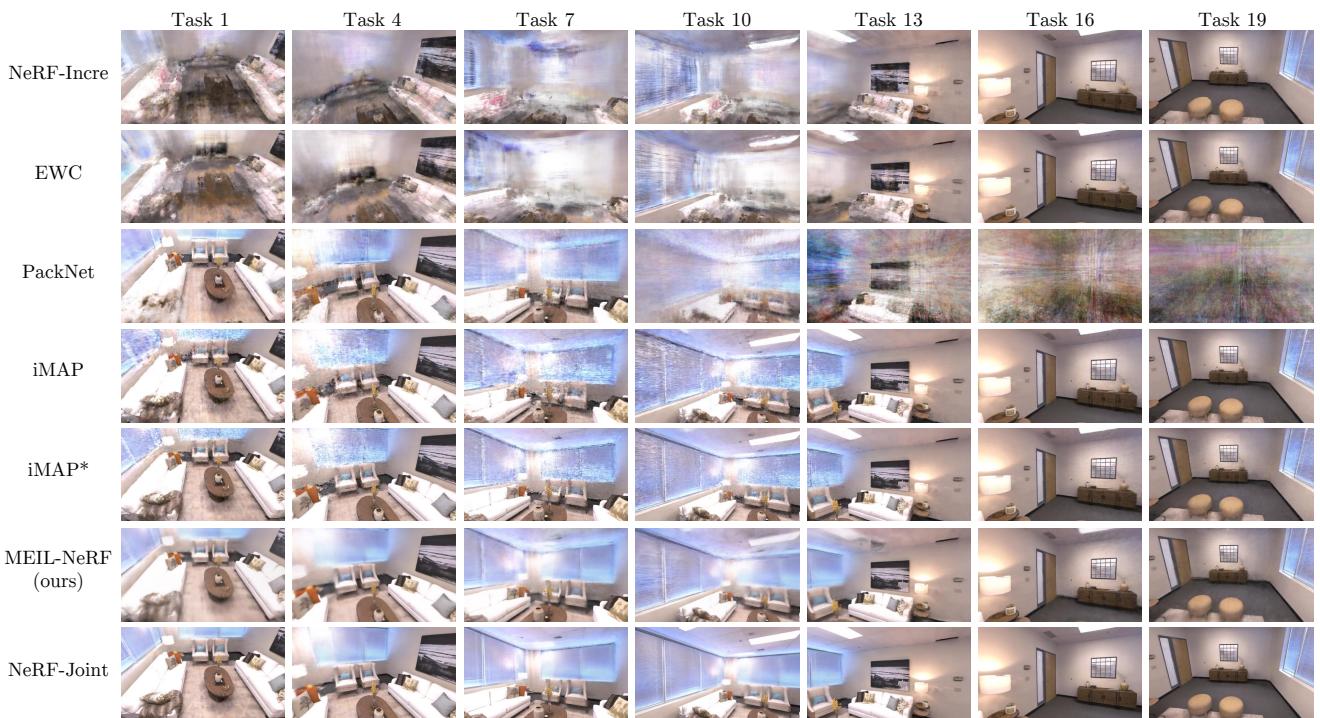


Figure S12. Qualitative Results in Replica/Room0.

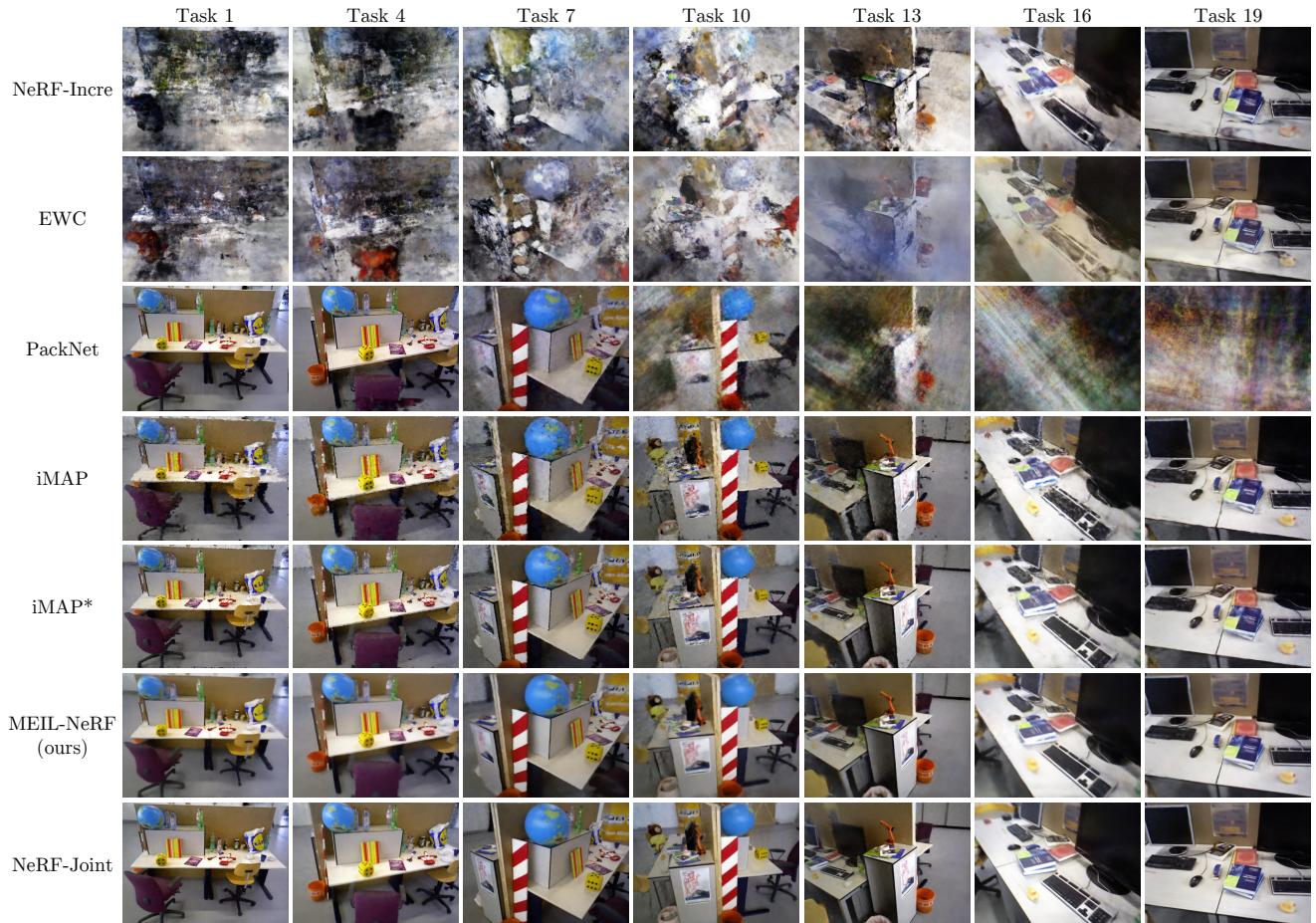


Figure S13. Qualitative Results in TUM-RGBD/Seq0.

## References

- [1] Rahaf Aljundi, Francesca Babiloni, Mohamed Elhoseiny, Marcus Rohrbach, and Tinne Tuytelaars. Memory aware synapses: Learning what (not) to forget. In *ECCV*, 2018. [3](#)
- [2] Tim Bailey and Hugh Durrant-Whyte. Simultaneous localization and mapping (slam): Part ii. *IEEE robotics & automation magazine*, 13(3):108–117, 2006. [1](#)
- [3] Jonathan T Barron, Ben Mildenhall, Matthew Tancik, Peter Hedman, Ricardo Martin-Brualla, and Pratul P Srinivasan. Mip-nerf: A multiscale representation for anti-aliasing neural radiance fields. In *ICCV*, 2021. [2](#)
- [4] Jonathan T Barron, Ben Mildenhall, Dor Verbin, Pratul P Srinivasan, and Peter Hedman. Mip-nerf 360: Unbounded anti-aliased neural radiance fields. In *CVPR*, 2022. [2](#)
- [5] Fabio Bruno, Stefano Bruno, Giovanna De Sensi, Maria-Laura Luchi, Stefania Mancuso, and Maurizio Muzzupappa. From 3d reconstruction to virtual reality: A complete methodology for digital archaeological exhibition. *Journal of Cultural Heritage (JCH)*, 11(1):42–49, 2010. [1](#)
- [6] Eric R Chan, Connor Z Lin, Matthew A Chan, Koki Nagano, Boxiao Pan, Shalini De Mello, Orazio Gallo, Leonidas J Guibas, Jonathan Tremblay, Sameh Khamis, et al. Efficient geometry-aware 3d generative adversarial networks. In *CVPR*, 2022. [2](#)
- [7] Eric R Chan, Marco Monteiro, Petr Kellnhofer, Jiajun Wu, and Gordon Wetzstein. pi-gan: Periodic implicit generative adversarial networks for 3d-aware image synthesis. In *CVPR*, 2021. [2](#)
- [8] Arslan Chaudhry, Puneet K. Dokania, Thalaiyasingam Ajanthan, and Philip H. S. Torr. Riemannian walk for incremental learning: Understanding forgetting and intransigence. In *ECCV*, 2018. [3](#)
- [9] Anpei Chen, Zexiang Xu, Fuqiang Zhao, Xiaoshuai Zhang, Fanbo Xiang, Jingyi Yu, and Hao Su. Mvsnerf: Fast generalizable radiance field reconstruction from multi-view stereo. In *ICCV*, 2021. [2](#)
- [10] Jianchuan Chen, Ying Zhang, Di Kang, Xuefei Zhe, Linchao Bao, Xu Jia, and Huchuan Lu. Animatable neural radiance fields from monocular rgb videos. *arXiv preprint arXiv:2106.13629*, 2021. [2](#)
- [11] Matthias De Lange, Rahaf Aljundi, Marc Masana, Sarah Parisot, Xu Jia, Aleš Leonardis, Gregory Slabaugh, and Tinne Tuytelaars. A continual learning survey: Defying forgetting in classification tasks. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 44(7):3366–3385, 2021. [2, 3](#)
- [12] Prithviraj Dhar, Rajat Vikram Singh, Kuan-Chuan Peng, Ziyan Wu, and Rama Chellappa. Learning without memorizing. In *CVPR*, 2019. [3](#)
- [13] Emilien Dupont, Adam Goliński, Milad Alizadeh, Yee Whye Teh, and Arnaud Doucet. Coin: Compression with implicit neural representations. *arXiv preprint arXiv:2103.03123*, 2021. [2](#)
- [14] Hugh Durrant-Whyte and Tim Bailey. Simultaneous localization and mapping: part i. *IEEE robotics & automation magazine*, 13(2):99–110, 2006. [1](#)
- [15] Robert M French. Catastrophic forgetting in connectionist networks. *Trends in cognitive sciences*, 3(4):128–135, 1999. [2, 3](#)
- [16] Guy Gafni, Justus Thies, Michael Zollhofer, and Matthias Nießner. Dynamic neural radiance fields for monocular 4d facial avatar reconstruction. In *CVPR*, 2021. [1](#)
- [17] Zekun Hao, Arun Mallya, Serge Belongie, and Ming-Yu Liu. Gancraft: Unsupervised 3d neural rendering of minecraft worlds. In *ICCV*, 2021. [2](#)
- [18] Peter Hedman, Pratul P Srinivasan, Ben Mildenhall, Jonathan T Barron, and Paul Debevec. Baking neural radiance fields for real-time view synthesis. In *ICCV*, 2021. [2](#)
- [19] Steven C. Y. Hung, Cheng-Hao Tu, Cheng-En Wu, Chien-Hung Chen, Yi-Ming Chan, and Chu-Song Chen. Compact-ing, picking and growing for unforgetting continual learning. In *NeurIPS*, 2019. [3](#)
- [20] Jeffrey Ichniowski, Yahav Avigal, Justin Kerr, and Ken Goldberg. Dex-nerf: Using a neural radiance field to grasp transparent objects. *arXiv preprint arXiv:2110.14217*, 2021. [2](#)
- [21] Joel Janai, Fatma Güney, Aseem Behl, Andreas Geiger, et al. Computer vision for autonomous vehicles: Problems, datasets and state of the art. *Foundations and Trends® in Computer Graphics and Vision*, 12(1–3):1–308, 2020. [1](#)
- [22] Petr Kellnhofer, Lars C Jebe, Andrew Jones, Ryan Spicer, Kari Pulli, and Gordon Wetzstein. Neural lumigraph rendering. In *CVPR*, 2021. [2](#)
- [23] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences (PNAS)*, 2017. [3, 5, 6, 12](#)
- [24] Arno Knapitsch, Jaesik Park, Qian-Yi Zhou, and Vladlen Koltun. Tanks and temples: Benchmarking large-scale scene reconstruction. *ACM Transactions on Graphics (TOG)*, 36(4), 2017. [6](#)
- [25] Kirios N Kutulakos and Steven M Seitz. A theory of shape by space carving. In *ICCV*, 1999. [4](#)
- [26] Zhizhong Li and Derek Hoiem. Learning without forgetting. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 2018. [3](#)
- [27] Chen-Hsuan Lin, Wei-Chiu Ma, Antonio Torralba, and Simon Lucey. Barf: Bundle-adjusting neural radiance fields. In *ICCV*, 2021. [2](#)
- [28] Lingjie Liu, Jiatao Gu, Kyaw Zaw Lin, Tat-Seng Chua, and Christian Theobalt. Neural sparse voxel fields. In *NeurIPS*, 2020. [2](#)
- [29] Stephen Lombardi, Tomas Simon, Gabriel Schwartz, Michael Zollhoefer, Yaser Sheikh, and Jason Saragih. Mixture of volumetric primitives for efficient neural rendering. *ACM Transactions on Graphics (TOG)*, 40(4):1–13, 2021. [2](#)
- [30] Xinzhu Ma, Zhihui Wang, Haojie Li, Pengbo Zhang, Wanli Ouyang, and Xin Fan. Accurate monocular 3d object detection via color-embedded 3d reconstruction for autonomous driving. In *ICCV*, 2019. [1](#)

- [31] Arun Mallya, Dillon Davis, and Svetlana Lazebnik. Piggy-back: Adapting a single network to multiple tasks by learning to mask weights. In *CVPR*, 2018. 3
- [32] Arun Mallya and Svetlana Lazebnik. Packnet: Adding multiple tasks to a single network by iterative pruning. In *CVPR*, 2018. 3, 5, 6, 13
- [33] Ricardo Martin-Brualla, Noha Radwan, Mehdi SM Sajjadi, Jonathan T Barron, Alexey Dosovitskiy, and Daniel Duckworth. Nerf in the wild: Neural radiance fields for unconstrained photo collections. In *CVPR*, 2021. 2
- [34] Nelson Max. Optical models for direct volume rendering. *IEEE Transactions on Visualization and Computer Graphics (TVCG)*, 1(2):99–108, 1995. 4
- [35] Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. Occupancy networks: Learning 3d reconstruction in function space. In *CVPR*, 2019. 2
- [36] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. In *ECCV*, 2020. 1, 2, 4, 5
- [37] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *arXiv preprint arXiv:2201.05989*, 2022. 2
- [38] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *ACM Transactions on Graphics (TOG)*, 41(4):102:1–102:15, July 2022. 8
- [39] Thomas Müller, Fabrice Rousselle, Jan Novák, and Alexander Keller. Real-time neural radiance caching for path tracing. *arXiv preprint arXiv:2106.12372*, 2021. 8
- [40] Raul Mur-Artal, Jose Maria Martinez Montiel, and Juan D Tardos. Orb-slam: a versatile and accurate monocular slam system. *IEEE transactions on robotics (T-RO)*, 31(5):1147–1163, 2015. 1
- [41] Michael Oechsle, Songyou Peng, and Andreas Geiger. Unisurf: Unifying neural implicit surfaces and radiance fields for multi-view reconstruction. In *ICCV*, 2021. 2
- [42] Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation. In *CVPR*, 2019. 2
- [43] Sida Peng, Junting Dong, Qianqian Wang, Shangzhan Zhang, Qing Shuai, Xiaowei Zhou, and Hujun Bao. Animatable neural radiance fields for modeling dynamic human bodies. In *ICCV*, 2021. 2
- [44] Christian Reiser, Songyou Peng, Yiyi Liao, and Andreas Geiger. Kilonerf: Speeding up neural radiance fields with thousands of tiny mlps. In *ICCV*, 2021. 2
- [45] Amir Rosenfeld and John K Tsotsos. Incremental learning through deep adaptation. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 42(3):651–663, 2018. 3
- [46] Antoni Rosinol, John J Leonard, and Luca Carlone. Nerf-slam: Real-time dense monocular slam with neural radiance fields. *arXiv preprint arXiv:2210.13641*, 2022. 3
- [47] Vincent Sitzmann, Julien Martel, Alexander Bergman, David Lindell, and Gordon Wetzstein. Implicit neural representations with periodic activation functions. In *NeurIPS*, 2020. 2
- [48] Julian Straub, Thomas Whelan, Lingni Ma, Yufan Chen, Erik Wijmans, Simon Green, Jakob J Engel, Raul Mur-Artal, Carl Ren, Shobhit Verma, et al. The replica dataset: A digital replica of indoor spaces. *arXiv preprint arXiv:1906.05797*, 2019. 6
- [49] Jürgen Sturm, Nikolas Engelhard, Felix Endres, Wolfram Burgard, and Daniel Cremers. A benchmark for the evaluation of rgb-d slam systems. In *IROS*, 2012. 6
- [50] Edgar Sucar, Shikun Liu, Joseph Ortiz, and Andrew J Davison. imap: Implicit mapping and positioning in real-time. In *ICCV*, 2021. 2, 3, 5, 6
- [51] Deqing Sun, Stefan Roth, and Michael J Black. Secrets of optical flow estimation and their principles. In *CVPR*, 2010. 4
- [52] Matthew Tancik, Ben Mildenhall, Terrance Wang, Divi Schmidt, Pratul P Srinivasan, Jonathan T Barron, and Ren Ng. Learned initializations for optimizing coordinate-based neural representations. In *CVPR*, 2021. 2
- [53] Matthew Tancik, Pratul Srinivasan, Ben Mildenhall, Sara Fridovich-Keil, Nithin Raghavan, Utkarsh Singhal, Ravi Ramamoorthi, Jonathan Barron, and Ren Ng. Fourier features let networks learn high frequency functions in low dimensional domains. In *NeurIPS*, 2020. 2
- [54] Dor Verbin, Peter Hedman, Ben Mildenhall, Todd Zickler, Jonathan T Barron, and Pratul P Srinivasan. Ref-nerf: Structured view-dependent appearance for neural radiance fields. In *CVPR*, 2022. 2
- [55] Qianqian Wang, Zhicheng Wang, Kyle Genova, Pratul P Srinivasan, Howard Zhou, Jonathan T Barron, Ricardo Martin-Brualla, Noah Snavely, and Thomas Funkhouser. Ibr-net: Learning multi-view image-based rendering. In *CVPR*, 2021. 2
- [56] Yiheng Xie, Towaki Takikawa, Shunsuke Saito, Or Litany, Shiqin Yan, Numair Khan, Federico Tombari, James Tompkin, Vincent Sitzmann, and Srinath Sridhar. Neural fields in visual computing and beyond. In *Computer Graphics Forum*, volume 41, pages 641–676. Wiley Online Library, 2022. 2
- [57] Gengshan Yang, Minh Vo, Natalia Neverova, Deva Ramanan, Andrea Vedaldi, and Hanbyul Joo. Banmo: Building animatable 3d neural models from many casual videos. In *CVPR*, 2022. 2
- [58] Alex Yu, Sara Fridovich-Keil, Matthew Tancik, Qinrong Chen, Benjamin Recht, and Angjoo Kanazawa. Plenoxtels: Radiance fields without neural networks. *arXiv preprint arXiv:2112.05131*, 2021. 2
- [59] Alex Yu, Vickie Ye, Matthew Tancik, and Angjoo Kanazawa. pixelnerf: Neural radiance fields from one or few images. In *CVPR*, 2021. 2
- [60] Friedemann Zenke, Ben Poole, and Surya Ganguli. Continual learning through synaptic intelligence. In *ICML*, 2017. 3
- [61] Linfeng Zhang, Jiebo Song, Anni Gao, Jingwei Chen, Chenglong Bao, and Kaisheng Ma. Be your own teacher: Im-

- prove the performance of convolutional neural networks via self distillation. In *ICCV*, 2019. [2](#), [4](#)
- [62] Zihan Zhu, Songyou Peng, Viktor Larsson, Weiwei Xu, Hujun Bao, Zhaopeng Cui, Martin R Oswald, and Marc Pollefeys. Nice-slam: Neural implicit scalable encoding for slam. In *CVPR*, 2022. [2](#), [3](#)