CLNeRF: Continual Learning Meets NeRF

Zhipeng Cai*
Intel Labs
zhipeng.cai@intel.com

Matthias Müller
Intel Labs
matthias.mueller.2@kaust.edu.sa

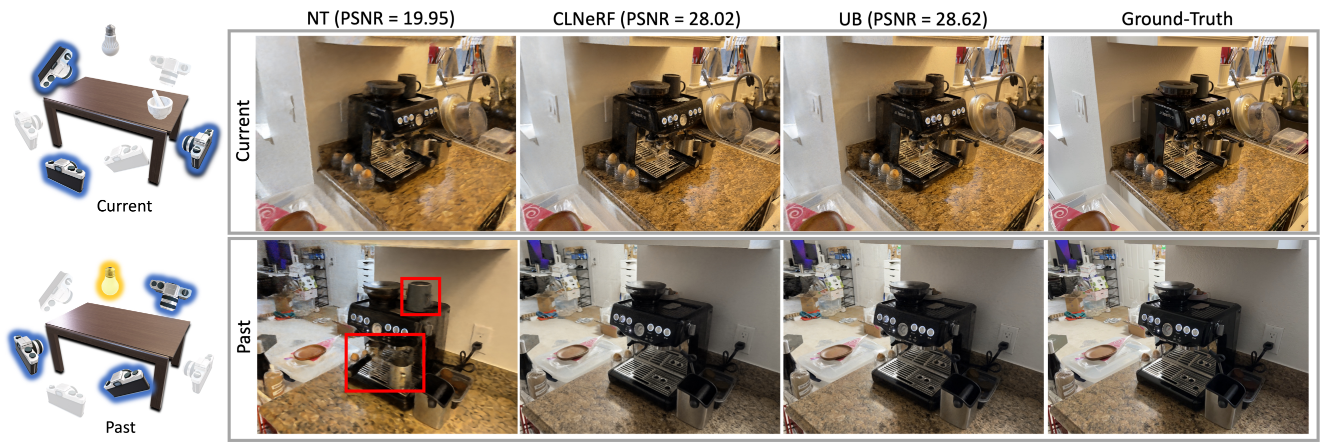


Figure 1. **Teaser.** This work studies continual learning for NeRFs. We propose a new benchmark – *World Across Time (WAT)* – to study practical scenarios where images of a scene arrive as a sequence of multiple scans with appearance and geometry changes, over an extended period of time. We also propose an effective system – *CLNeRF* – that can sequentially learn from these scans, without requiring stored historical images. The top and bottom rows show rendered novel views of the same scene for the current and past scans respectively. CLNeRF accurately renders both the current and the past scans, performing on-par with the upper bound model (UB) trained on all scans at once. Naively training (NT) on the sequence of scans overfits to the current scan, resulting in erroneous appearance (lightning) and geometry (extra cups marked by bounding boxes, which only exist in the current scan) for the past scan.

Abstract

Novel view synthesis aims to render unseen views given a set of calibrated images. In practical applications, the coverage, appearance or geometry of the scene may change over time, with new images continuously being captured. Efficiently incorporating such continuous change is an open challenge. Standard NeRF benchmarks only involve scene coverage expansion. To study other practical scene changes, we propose a new dataset, World Across Time (WAT), consisting of scenes that change in appearance and geometry over time. We also propose a simple yet effective method, CLNeRF, which introduces continual learning (CL) to Neural Radiance Fields (NeRFs). CLNeRF combines generative replay and the Instant Neural Graphics Primitives (NGP) architecture to effectively prevent catastrophic forgetting and efficiently update the model when new data arrives. We also add trainable appearance and geometry embeddings to NGP, allowing a single compact model to handle complex scene changes. Without the need to store historical images, CLNeRF trained sequentially over multiple scans of a changing scene performs on-par with the upper bound model trained on all scans at once. Compared to other CL baselines CLNeRF performs much better across standard benchmarks and WAT. The source code, a demo, and the WAT dataset are available at `https://github.com/IntelLabs/CLNeRF`.

1. Introduction

Neural Radiance Fields (NeRFs) have emerged as the preeminent method for novel view synthesis. Given images of a scene from multiple views, NeRFs can effectively interpolate between them. However, in practical applications (*e.g.*, city rendering [32]), the scene may change over time, resulting in a gradually revealed sequence of images with new scene coverage (new city blocks), appearance (lighting or weather) and geometry (new construction). Learning continually from such sequential data is an important problem.

Naive model re-training on all revealed data is expensive, millions of images may need to be stored for large scale systems [32]. Meanwhile, updating the model only on new data leads to catastrophic forgetting [22], *i.e.*, old scene geometry and appearances can no longer be recovered (see Fig. 1). Inspired by the continual learning literature for image classification [7], this work studies continual learning in the context of NeRFs to design a system that can learn from a sequence of scene scans without forgetting while requiring minimal storage.

Replay is one of the most effective continual learning algorithms; it trains models on a blend of new and historical data. *Experience replay* [5] explicitly stores a tiny portion of the historical data for replay, while *generative replay* [30] synthesizes replay data using a generative model (e.g., a GAN [9]) trained on historical data. Experience replay is more widely used in image classification, since generative models are often hard to train, perform poorly on high resolution images, and introduce new model parameters. In contrast, NeRFs excel at generating high-resolution images, making them ideal candidates for generative replay.

Motivated by this synergy between advanced NeRF models and generative replay, we propose *CLNeRF* which combines generative replay with Instant Neural Graphics Primitives (NGP) [24] to enable efficient model updates and to prevent forgetting *without the need to store historical images*. CLNeRF also introduces trainable appearance and geometry embeddings into NGP so that various scene changes can be handled by a single model. Unlike classification-based continual learning methods whose performance gap to the upper bound model is still non-negligible [30], the synergy between continual learning and advanced NeRF architectures allows CLNeRF to achieve a similar rendering quality as the upper bound model (see Fig. 1).

Contributions: (1) We study the problem of continual learning in the context of NeRFs. We present *World Across Time (WAT)*, a practical continual learning dataset for NeRFs that contains scenes with real-world appearance and geometry changes over time. (2) We propose *CLNeRF*, a simple yet effective continual learning system for NeRFs with minimal storage and memory requirements. Extensive experiments demonstrate the superiority of CLNeRF over other continual learning approaches on standard NeRF datasets and WAT.

2. Related Work

NeRF. Learning Neural Radiance Fields (NeRFs) is arguably the most popular technique for novel view synthesis (see [8] for a detailed survey). Vanilla NeRF [23] represents a scene implicitly using neural networks, specifically, multi-layer perceptrons (MLPs). These MLPs map a 3D location and a view direction to their corresponding color and opacity. An image of the scene is synthesized by casting camera rays into 3D space and performing volume rendering. Though effective at interpolating novel views, vanilla NeRF has several limitations, for example, the slow training/inference speed. This problem is addressed by using explicit scene representations [31, 24], or spatially-distributed small MLPs [26]. CLNeRF applies these advanced architectures to ensure efficient model updates during continual learning. Vanilla NeRF only considers static scenes; to handle varied lightning or weather conditions, trainable appearance embeddings are introduced [20, 32]. Transient objects in in-the-wild photos are handled by either introducing a transient MLP [20] or using segmentation masks [32]. CLNeRF adopts these techniques to allow a single model to handle complex scene changes. Concurrent to this work, Chung et al. [6] also study NeRFs in the context of continual learning. However, they only consider static scenes and the vanilla NeRF architecture. We consider scenes with changing appearance/geometry, and introduce a new dataset to study such scenarios. The proper combination of continual learning and more advanced architectures also makes CLNeRF simpler (no extra hyperparameters) and much more effective at mitigating forgetting.

Continual Learning. Continual learning aims to learn from a sequence of data with distribution shifts, without storing historical data (see [7] for a detailed survey). Naive training over non-IID data sequences suffers from catastrophic forgetting [12] and performs poorly on historical data. A popular line of work regularizes the training objective to prevent forgetting [12, 15]. However, since the regularization does not rely on historical data it is less effective in practice. Parameter isolation methods prevent forgetting by freezing (a subset of) neurons from previous tasks and use new neurons to learn later tasks [19, 29]. Though remembrance can be guaranteed [19], these methods have a limited capacity or grow the network significantly given a large number of tasks. Replay-based approaches use historical data to prevent forgetting. This historical data is either stored in a small replay buffer [5, 18], or synthesized by a generative model [30]. Generative replay [30], i.e., synthesizing historical data, is less effective for image classification, since the generative model introduces extra parameters, and performs poorly on high resolution images. In contrast, this work shows that advanced NeRF models and generative replay benefit from each other, since high quality replay data can be rendered without introducing new model parameters.

3. Method

3.1. Preliminaries

Before introducing CLNeRF, we first review the basics of NeRFs, and formulate the problem of continual learning.

NeRF. Given a set of images, NeRFs train a model parameterized by $\boldsymbol{\theta}$ that maps a 3D location $\mathbf{x} \in \mathbb{R}^3$ and a view direction $\mathbf{d} \in \mathcal{S}^3$ (a unit vector from the camera center to $\mathbf{x}$) to the corresponding color $c(\mathbf{x}, \mathbf{d}|\boldsymbol{\theta}) \in [0, 1]^3$ and opacity $\sigma(\mathbf{x}|\boldsymbol{\theta}) \in [0, 1]$. Given a target image view, we render the color for each pixel independently. For each pixel, we cast a

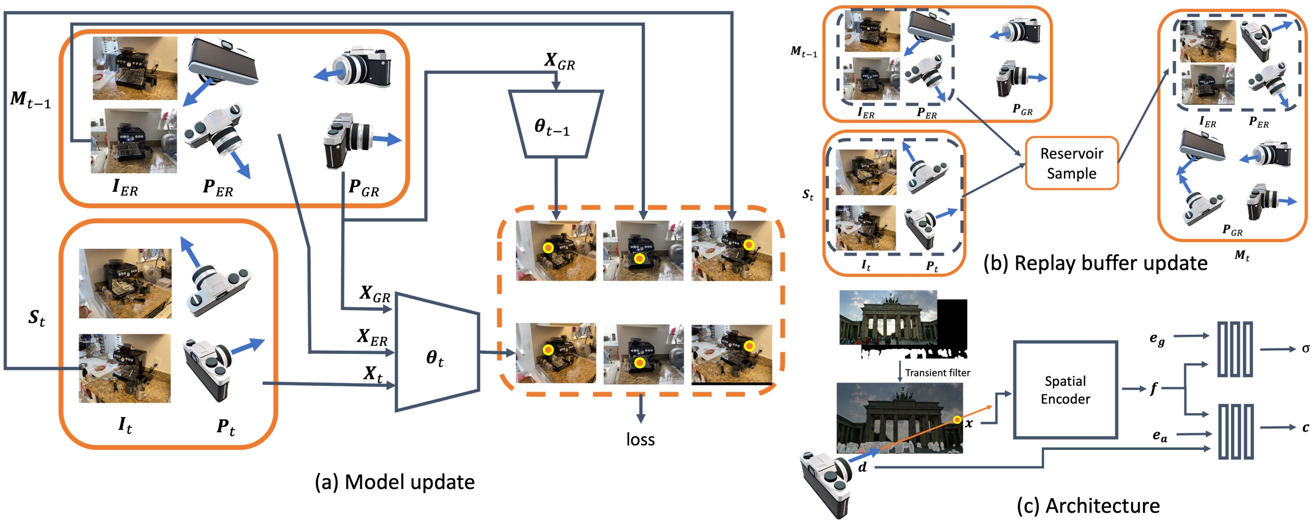


Figure 2. **System overview.** (a): At each time step t of continual NeRF, a new set of data $\mathbf{S}_t$ is generated. To update the model $\boldsymbol{\theta}_t$, we randomly generate in each training iteration a set of camera rays from camera parameters stored for experience replay ($\mathbf{P}_{\text{ER}}$), generative replay ($\mathbf{P}_{\text{GR}}$) and in the new data ($\mathbf{P}_t$). For rays from new data ($\mathcal{X}_t$) or experience replay ($\mathcal{X}_{\text{ER}}$), the corresponding image color is used for supervision. For rays from generative replay, *i.e.*, $\mathcal{X}_{\text{GR}}$, we use the latest deployed model $\boldsymbol{\theta}_{t-1}$ to generate pseudo-labels for supervision. After training $\boldsymbol{\theta}_t$, we replace the previously deployed model $\boldsymbol{\theta}_{t-1}$ with $\boldsymbol{\theta}_t$, and update the replay buffer $\mathbf{M}_t$. (b): To update the replay buffer $\mathbf{M}_t$ under optional experience replay, we perform reservoir sampling over image-camera-parameter pairs in $\mathbf{M}_{t-1}$ and $\mathbf{S}_t$, and add into $\mathbf{P}_{\text{GR}}$ camera parameters for all images not selected by reservoir sampling. (c) We use segmentation masks to filter transient objects, and apply appearance embeddings $\mathbf{e}_a$ and geometry embeddings $\mathbf{e}_g$ to the base architecture to handle scene changes at different time steps.

ray from the camera center $\mathbf{o} \in \mathbb{R}^3$ towards the pixel center, and sample a set of 3D points $\mathbf{X} = \{\mathbf{x}_i | \mathbf{x}_i = \mathbf{o} + \tau_i \mathbf{d}\}$ along the ray, where τ_i is the euclidean distance from the camera center to the sampled point. Then, we render the color $\hat{\mathbf{C}}(\mathbf{X})$ of the ray following the volume rendering [21] equation:

$$\hat{\mathbf{C}}(\mathbf{X}) = \sum_i w_i c(\mathbf{x}_i, \mathbf{d}|\boldsymbol{\theta}), \tag{1}$$

where $w_i = e^{-\sum_{j=1}^{i-1} \sigma(\mathbf{x}_j|\boldsymbol{\theta})(\tau_{j+1}-\tau_j)}(1 - e^{-\sigma(\mathbf{x}_i|\boldsymbol{\theta})(\tau_{i+1}-\tau_i)})$. Intuitively, Equation (1) computes the weighted sum of the colors on all sampled points. The weights w_i are computed based on the opacity and the distance to the camera center.

Continual NeRF. Throughout this paper, we refer to the continual learning problem for NeRFs as *continual NeRF*. At each time step t of continual NeRF:

1. A set of training images along with their camera parameters (intrinsics and extrinsics) $\mathbf{S}_t$ are generated.
2. The current model $\boldsymbol{\theta}_t$ and the replay buffer $\mathbf{M}_t$ (for storing historical data) are updated by:

$$\{\mathbf{M}_t, \boldsymbol{\theta}_t\} \leftarrow \text{update}(\mathbf{S}_t, \boldsymbol{\theta}_{t-1}, \mathbf{M}_{t-1}) \tag{2}$$

3. $\boldsymbol{\theta}_t$ is deployed for rendering novel views until $t+1$.

This process simulates the practical scenario where the model $\boldsymbol{\theta}_t$ is deployed continually. Once in a while, a set of new images arrives, potentially containing new views of the scene and changes in appearance or geometry. The goal is to update $\boldsymbol{\theta}_t$; ideally storage (to maintain historical data in $\mathbf{M}_t$) and memory (to deploy $\boldsymbol{\theta}_t$) requirements are small.

As shown in Fig. 2, CLNeRF addresses three major problems of continual NeRF: (1) effectively updating $\boldsymbol{\theta}_t$ using minimal storage, (2) updating $\mathbf{M}_t$ during optional experience replay, and (3) handling various scene changes with a single compact model. We provide further details on each of these components below.

3.2. Model Update

CLNeRF applies replay-based methods [5, 30] to prevent catastrophic forgetting. To enable applications with extreme storage limits, CLNeRF combines generative replay [30] with advanced NeRF architectures so that it is effective even when no historical image can be stored.

Fig. 2(a) depicts the model update process of CLNeRF at each time step t. The camera parameters of all historical images are stored in the replay buffer $\mathbf{M}_{t-1}$ for generative replay. A small number of images $\mathbf{I}_{\text{ER}}$ are optionally maintained when the storage is sufficient for experience replay [5]. At each training iteration of $\boldsymbol{\theta}_t$, CLNeRF generates a batch of camera rays $\mathcal{X} = \mathcal{X}_{\text{ER}} \bigcup \mathcal{X}_{\text{GR}} \bigcup \mathcal{X}_t$ uniformly from $\mathbf{P}_t \bigcup \mathbf{P}_{\text{ER}} \bigcup \mathbf{P}_{\text{GR}}$, where $\mathbf{P}_t$, $\mathbf{P}_{\text{GR}}$ and $\mathbf{P}_{\text{ER}}$ are respectively the camera parameters of new data $\mathbf{S}_t$, generative replay data and experience replay data. The training objective is:

$$\underset{\boldsymbol{\theta}_t}{\text{minimize}} \frac{\sum_{\mathbf{X} \in \mathcal{X}} \mathcal{L}_{\text{NeRF}}(C(\mathbf{X}), \hat{C}(\mathbf{X}|\boldsymbol{\theta}_t))}{|\mathcal{X}|}, \tag{3}$$

where $\mathcal{L}_{\text{NeRF}}$ is the loss for standard NeRF training, $C(\cdot)$ is the supervision signal from new data or replay, and $\hat{C}(\cdot|\boldsymbol{\theta}_t)$ is the color rendered by $\boldsymbol{\theta}_t$. For the rays $\mathbf{X} \in \mathcal{X}_{\text{GR}}$ sampled from $\mathbf{P}_{\text{GR}}$, we perform generative replay, *i.e.*, we set the supervision signal $C(\mathbf{X})$ as the image colors $\hat{C}(\mathbf{X}|\boldsymbol{\theta}_{t-1})$ generated by $\boldsymbol{\theta}_{t-1}$. For the other rays, $C(\mathbf{X})$ is the ground-truth image color. After the model update, we replace the previously deployed model $\boldsymbol{\theta}_{t-1}$ with $\boldsymbol{\theta}_t$ and update the replay buffer $\mathbf{M}_t$ (see Sec. 3.3 for more details). Only $\boldsymbol{\theta}_t$ and $\mathbf{M}_t$ are maintained until the next set of data $\mathbf{S}_{t+1}$ arrives.

Although all camera parameters are stored in $\mathbf{M}_{t-1}$, they only consume a small amount of storage, at most $N_{t-1}(d_{pose} + d_{int})$, where N_{t-1} is the number of historical images, and d_{pose} and d_{int} are the dimensions of camera poses and intrinsic parameters respectively; $d_{pose} = 6$ and $d_{int} \leq 5$ for common camera models [10]. d_{int} is shared if multiple images are captured by the same camera. As a concrete example, storing the parameters for 1000 samples each captured with a different camera requires roughly 45KB of storage in our experiment, much less than storing a single high resolution image. This guarantees the effectiveness of CLNeRF (see Sec. 5.1) even for applications with extreme storage limits.

We also emphasize the importance of random sampling. CLNeRF assigns *uniform* sampling weights between all views revealed so far. Some image-classification-based continual learning methods [5] and the cocurrent work for NeRFs [6] propose biased sampling strategies, where a fixed and large percentage ($\frac{1}{2}$ to $\frac{2}{3}$) of the rays are sampled from new data ($\mathbf{P}_t$). This strategy not only introduces new hyperparameters (*e.g.*, loss weights of old data or the proportion of rays from new data [6]), but also performs worse than uniform random sampling, as shown in Sec. 5.

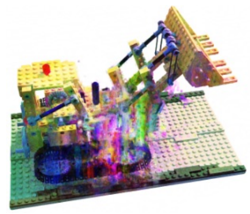
3.3. Replay Buffer Update

In the extreme case where no image can be stored for experience replay, we only store the camera parameters of historical data in $\mathbf{M}_t$ to make CLNeRF flexible and effective for practical systems with various storage sizes. When the storage is sufficient to maintain a subset of historical images for experience replay, we use a *reservoir buffer* [5]. Specifically, current data is added to $\mathbf{M}_t$ as long as the storage limit is not reached. Otherwise, as shown in Fig. 2 (b), given $\mathbf{M}_{t-1}$ capable of storing K images, we first generate for each image $\mathbf{I}_i \in \mathbf{S}_t$ a random integer $j_i \in \{1, 2, ..., N_i\}$, where N_i represents the order of $\mathbf{I}_i$ in the continual learning data sequence. If $j_i > K$, we do not add $\mathbf{I}_i$ into $\mathbf{M}_t$. Otherwise, we replace the j_i'th image in $\mathbf{M}_{t-1}$ with $\mathbf{I}_i$. Note that $\mathbf{M}_t$ stores all camera parameters regardless of if the corresponding image is stored or not.

We also experiment with the *prioritized replay buffer* [27], where $\mathbf{M}_t$ keeps images with the lowest rendering quality. Specifically, after updating $\boldsymbol{\theta}_t$, we iterate over all images in $\mathbf{M}_{t-1}$ and $\mathbf{S}_t$ and keep the K images with the lowest rendering PSNR [11] from $\boldsymbol{\theta}_t$. Though widely used in reinforcement learning, prioritized replay does not perform better than reservoir sampling in continual NeRF (see Sec. 5.2.2). A reservoir buffer is also simpler to implement and more efficient (no need to compare the rendering quality) to update; hence CLNeRF applies it by default.

3.4. Architecture

CLNeRF by default uses the Instant Neural Graphics Primitives (NGP) [24] architecture. This not only enables efficient model updates during continual NeRF, but also ensures the low overhead and effectiveness of generative replay. As shown in Sec. 5.2.3 using NGP as the backbone for CLNeRF results in better performance and efficiency compared to vanilla NeRF [23].



(a) Test view rendered by NGP with transient MLP.

(b) Test view rendered by NGP with masked transient objects.

(c) Training view rendered by NGP with transient MLP.

(d) GT training view.

Figure 3. **Result of NGP with transient MLPs.** Similar to [20], we add artificial transient objects and lightning changes to the *Lego* scene. NGP with transient MLPs overfits to the training views and fails to filter transient objects automatically.

A compact continual NeRF system should use a single model to incorporate scene changes, so that the model size does not increase significantly over time. We achieve this by adding trainable appearance and geometry embeddings to the base architecture (Fig. 2 (c)). Given a spatial location $\mathbf{x}$ and a viewing direction $\mathbf{d}$, we first encode $\mathbf{x}$ into a feature vector $\mathbf{f}$ (using the grid-based hash encoder for NGP, and an MLP for vanilla NeRF). Then, we generate the color and opacity respectively by $\mathbf{c} = D_c(\mathbf{f}, \mathbf{d}, \mathbf{e}_a)$ and $\sigma = D_\sigma(\mathbf{f}, \mathbf{e}_g)$, where D_c and D_σ are the color and opacity decoders (MLP for both NGP and vanilla NeRF); $\mathbf{e}_a$ is the trainable appearance embedding and $\mathbf{e}_g$ is the geometry embedding. Given a sequence of scans of the same scene, with appearance and geometry changes between different scans, we add one appearance embedding and one geometry embedding for each scan, i.e., for each time step t of continual NeRF. We set the dimension of appearance and geometry embeddings to 48 and 16 respectively, which ensures minimal model size increase during continual NeRF and is sufficient to encode complex real-world scene changes as shown in Sec. 5.

CLNeRF uses segmentation masks (from [3]) to filter transient objects. As shown in Fig. 3, we also explored using the transient MLP and robust training objectives [20] but found empirically that NGP is not compatible with this strategy – the non-transient network overfits to the transient objects and fails to filter them automatically. Note that we only remove transient/dynamic objects within *a single scan*, e.g., moving pedestrians. Scene changes between different scans, e.g., newly constructed buildings, are handled by geometry embeddings.

4. WAT: A Continual NeRF Benchmark

Most continual learning methods in the literature are evaluated on datasets synthesized from standard image classification benchmarks [7]. Although a similar strategy can be used on standard

Property \ Dataset	Synth-NeRF	NeRF++	NSVF	Phototourism	WAT
Real image	×	✓	✓	✓	✓
Appearance change	×	×	×	✓	✓
Geometry change	×	×	×	×	✓
Natural time stamp	×	×	×	×	✓
Indoor/Outdoor	Indoor	Outdoor	Both	Outdoor	Both
NO. Scenes	8	4	5	4	6
Images per scene	100	250-300	100	900-1700	200-400

Figure 4. **Images and properties of the proposed WAT dataset.** WAT contains both indoor and outdoor scenes (samples on top), which change in both appearance and geometry over different scans. Each column of the images shows the same scene scanned at two different times. The scans are naturally ordered according to real-world time during continual NeRF, simulating realistic applications. Compared to WAT, standard benchmarks (see Sec. 5 for details) either lack appearance or geometry change, or natural order of these changes, making continual NeRF less challenging.

NeRF benchmarks, it is not practical as it only considers static scenes with a gradually expanding rendering range. However, this does not model the real-world distribution shifts introduced by the change of time [4, 16], such as the change of scene appearance (*e.g.*, lighting and weather) and geometry (*e.g.*, new decoration of a room). To solve this problem, we propose *World Across Time (WAT)*, a new dataset and benchmark for practical continual NeRF.

As shown in Fig. 4, WAT consists of images captured from 6 different scenes (both indoor and outdoor). For each scene, we capture 5-10 videos at different real-world time to generate natural appearance or geometry changes across videos. We extract a subset of the video frames (200-400 images for each scene), and use colmap [28] to compute the camera parameters. For each scene, we hold out $\frac{1}{8}$ of the images for testing and use the remaining images for training. We order the images naturally based on the time that the corresponding videos were captured. At each time step t of continual NeRF, all images belonging to a new video are revealed to the model. Compared to standard NeRF datasets, WAT has diverse scene types, scene changes, and a realistic data order based on real-world time. As shown in Sec. 5.1, the natural time-based order makes WAT much more challenging than randomly dividing standard in-the-wild datasets into subsets (*e.g.*, as in the case of phototourism, which has similar appearance and pose distributions between different subsets). WAT enables us to study the importance of the model design for changing scenes. As shown in Sec. 5.1, methods designed only for static scenes perform poorly on WAT.

5. Experiments

In the experiments, we first compare CLNeRF against other continual learning approaches (Sec. 5.1). Then, we analyse different continual NeRF components in detail (Sec. 5.2). Although NGP is used by default in CLNeRF, we also experiment with vanilla NeRF to demonstrate the effect of architectures.

Implementation Details. Our implementation of the vanilla NeRF and NGP backbones is based on NeRFAcc [14] and NGP-PL [2] respectively. Following the base implementations, we allow 50K training iterations for vanilla NeRF and 20K for NGP whenever we train or update the model. We find that NGP produces "NaN" losses and exploding gradients when trained for too long , making it hard to initialize $\boldsymbol{\theta}_t$ with $\boldsymbol{\theta}_{t-1}$. Hence, we randomly initialize $\boldsymbol{\theta}_t$ and train the model from scratch at each time step, as done in GDumb [25]. Empirical results (Sec. 5.2.3) on vanilla NeRF show that initializing $\boldsymbol{\theta}_t$ with $\boldsymbol{\theta}_{t-1}$ can help continual NeRF, and we leave further investigation of this issue on NGP for future work. Unless otherwise stated, all hyperparameters strictly follow the base code. See Appendix A for further implementation details of different continual learning methods. Training one model with either NGP or vanilla NeRF backbone takes 5-20 minutes or 1-2 hours on a single RTX6000 GPU respectively.

Datasets. Besides WAT, we also evaluate methods on datasets derived from standard NeRF benchmarks. Specifically, we uniformly divide the training data of standard benchmarks into 10-20 subsets and reveal them sequentially during continual NeRF. For synthetic data, we use the dataset proposed in [23] (referred to as *Synth-NeRF*), resulting in 8 scenes, each with 10 time steps and 20 training images (with consecutive image IDs) per time step. For real-world data, we use two Tanks and Temples [13] subsets proposed in [17] (with background filter, referred to as *NSVF*) and [33] (without background filter, referred to as *NeRF++*). Both datasets are divided into multiple sub-trajectories with consecutive video frames. NeRF++ has 4 scenes, 10 time steps and 60-80 images per time step. For NSVF, we mimic the construction process of the concurrent work [6], and divide the first 100 training images of each scene into 20 subsets, resulting in 5 scenes, each with 20 time steps and 5 images per time step. Finally, we use 4 Phototourism scenes (Brandenburg Gate, Sacre Coeur, Trevi Fountain and Taj Mahal) along with the train-test split from [20]. Due to the lack of time stamps, we randomly divide each scene into 20 time steps and 42-86 images per time step (with consecutive image IDs).

Evaluation protocol. To evaluate a continual NeRF approach, we first train it sequentially over all time steps, then compute the mean PSNR/SSIM [11] on the held-out test images for all time steps.

5.1. Main Results

To evaluate CLNeRF, we compare it against: (1) *Naive Training (NT)*, where we train a model sequentially on new data without continual learning. NT represents the lower-bound performance that we can achieve on continual NeRF. (2) *Elastic Weight Consolidation (EWC)* [12], a widely-used regularization-based continual learning method. (3) *Experience Replay (ER)* [5], one of the most effective continual learning methods. (4) *MEIL-NeRF* [6], a concurrent work that also uses generative replay. For fair comparison, we use the ground-truth camera parameters to generate replay camera rays for MEIL-NeRF as done in CLNeRF, rather than using a small MLP to learn the rays of interests. This strategy makes the implementation simpler and performs better, as also demonstrated in the original paper. (5) The *upper bound model (UB)* trained on all data at once, representing the upper-bound performance of continual NeRF. For all methods that involve experience replay (ER and CLNeRF), we allow 10 images to be stored in the replay buffer to simulate the case of highly limited storage (see Sec. 5.2.2 for the

Method \ Dataset	Synth-NeRF	NeRF++	NSVF	WAT
NT (NeRF)	28.53/0.938	14.81/0.462	18.58/0.768	16.70/0.649
EWC (NeRF)	28.32/0.925	15.03/0.442	18.12/0.778	16.29/0.672
ER (NeRF)	30.29/0.948	17.35/0.542	26.51/0.902	21.36/0.709
MEIL-NeRF (NGP)	30.69/0.953	19.40/0.595	27.19/0.909	24.05/0.730
CLNeRF-noER (NGP)	31.96/0.956	20.31/0.632	29.32/**0.924**	25.44/0.762
CLNeRF (NGP)	**32.16/0.957**	**20.33/0.634**	**29.48**/0.923	**25.45/0.764**
UB (NGP)	32.94/0.959	20.34/0.648	30.28/0.931	25.85/0.767

Method \ Dataset	Phototourism
NT (NGP)	19.28/0.692
ER (NGP)	20.03/0.713
MEIL-NeRF (NGP)	22.35/0.746
CLNerf-noER (NGP)	22.67/0.751
CLNerf (NGP)	**22.88/0.752**
UB (NeRFW)	22.78/0.823
UB (NGP)	23.05/0.763

Table 1. **Main results**. The results are in the form of PSNR/SSIM [11], with the best performing method in bold. We label each method with the best performing architecture, *i.e.*, (vanilla) NeRF or NGP. CLNeRF performs the best among all continual NeRF approaches and across all datasets, even without storing any historical images (CLNeRF-noER). The performance gap between CLNeRF and the upper-bound model UB remains low for all datasets. We equip all competitors on WAT with trainable embeddings proposed in Sec. 3.4 for fairer comparison to CLNeRF. Without using the embeddings, the performance gap between these methods and CLNeRF increases significantly (NT: 16.17/0.625, EWC: 15.81/0.650, ER: 17.99/0.666, MEIL-NeRF: 20.92/0.720). On Phototourism, the performance difference across methods is much smaller due to the random division of time steps (making pose and appearance distributions similar over time). CLNerf still performs close to the upper-bound model. UB with NerfW performs worse than UB with NGP in terms of PSNR but better in terms of SSIM.

effect of the replay buffer size). For fair comparison, we choose the best-performing architecture for each method. See Sec. 5.2 for the effect of architectures, and Appendix D for NGP-only results.

As shown in Tab. 1, CLNeRF, even without storing historical data for experience replay (CLNeRF-noER), performs much better than other continual NeRF approaches across all datasets (see Appendix C for the results of individual scenes). With only 10 images (2.5%-10% of the complete dataset size) stored in the replay buffer, CLNeRF achieves comparable performance as UB, which requires storing all historical data. Although MEIL-NeRF also applies generative replay, the biased sampling strategy (towards new data) and the complex loss designed for vanilla NeRF are not suitable for advanced architectures like NGP. As a result, there is a significant performance gap compared to UB which is consistent with the results in the original paper. As shown later in Sec. 5.2.3, CLNeRF also performs better than MEIL-NeRF with a vanilla NeRF backbone. Interestingly, methods without generative replay (NT, EWC, ER) work better with vanilla NeRF. We analyze this phenomenon in detail in Sec. 5.2.3. For a fairer comparison on WAT, we use the trainable embeddings of CLNeRF also for the other methods in Tab. 1. The results without embeddings are reported in the caption; the gap to CLNeRF increases significantly.

We also apply CLNeRF to in-the-wild images from Phototourism. Since the NeRFAcc-based vanilla NeRF implementation does not perform well on Phototourism, and NeRFW [20] is slow ($>$ 1 week per time step with 8 GPUs) in continual NeRF, we instead report the performance using NGP for NT and ER, and the UB version of NeRFW based on the implementation of [1]. EWC cannot be applied to NGP since we need to perform re-initialization at each time step. We assign 1 appearance embedding to each image (rather than each time step) to handle per-image appearance change. As shown in Tab. 1 the NGP-based upper bound model performs better than NeRFW in terms of PSNR and worse in terms of SSIM. CLNeRF performs close to the upper bound model. Due to the lack of time stamps, we do not have a natural data order for Phototourism. This simplifies the problem since images from different (artificially created) time steps have similar pose and appearance distributions. As a result, the performance gap between different methods is much smaller compared to other datasets, even with a much larger number of images (800-1700 in Phototourism versus 100-400 in other datasets).

Fig. 5 shows qualitative results (see Appendix C for more). Each two rows show the novel views rendered for the current and past time steps. CLNeRF provides similar rendering quality as UB, with a much lower storage consumption. Without using continual learning, NT overfits to the current data, resulting in wrong geometry (the redundant lamp and the wrongly closed curtain in *Living room - past*), lightning and severe artifacts for past time steps. Due to the lack of historical data, EWC not only fails to recover past scenes, but also hinders the model from learning on new data. ER stores a small amount of historical data to prevent forgetting. However, the limited replay buffer size makes it hard to recover details in past time steps (see Sec. 5.2.2 for the effect of replay buffer sizes). The biased sampling strategy and complex loss design make MEIL-NeRF not only more complex (*e.g.*, extra hyperparameters), but also underfit on historical data. As a result, it loses detail from past time steps, even when equipped with the same trainable embeddings of CLNeRF.

These results show the importance of WAT on benchmarking continual NeRF under practical scene changes. They also show the superiority of CLNeRF over other baselines to robustly handle appearance and geometry changes.

5.2. Analysis

5.2.1 Ablation

This section analyzes the effectiveness of individual CLNeRF components. Specifically, we remove each component and report the performance drop. As shown in Tab. 2, the performance drops only slightly without experience replay (No ER). However, without generative replay (No GenRep), the performance dropps significantly. Note that NoGenRep is ER with NGP instead of vanilla NeRF. Hence, generative replay is more important in CLNeRF than experience replay to prevent forgetting. Without using NGP, *i.e.*, when applied to vanilla NeRF, CLNeRF also performs much worse,

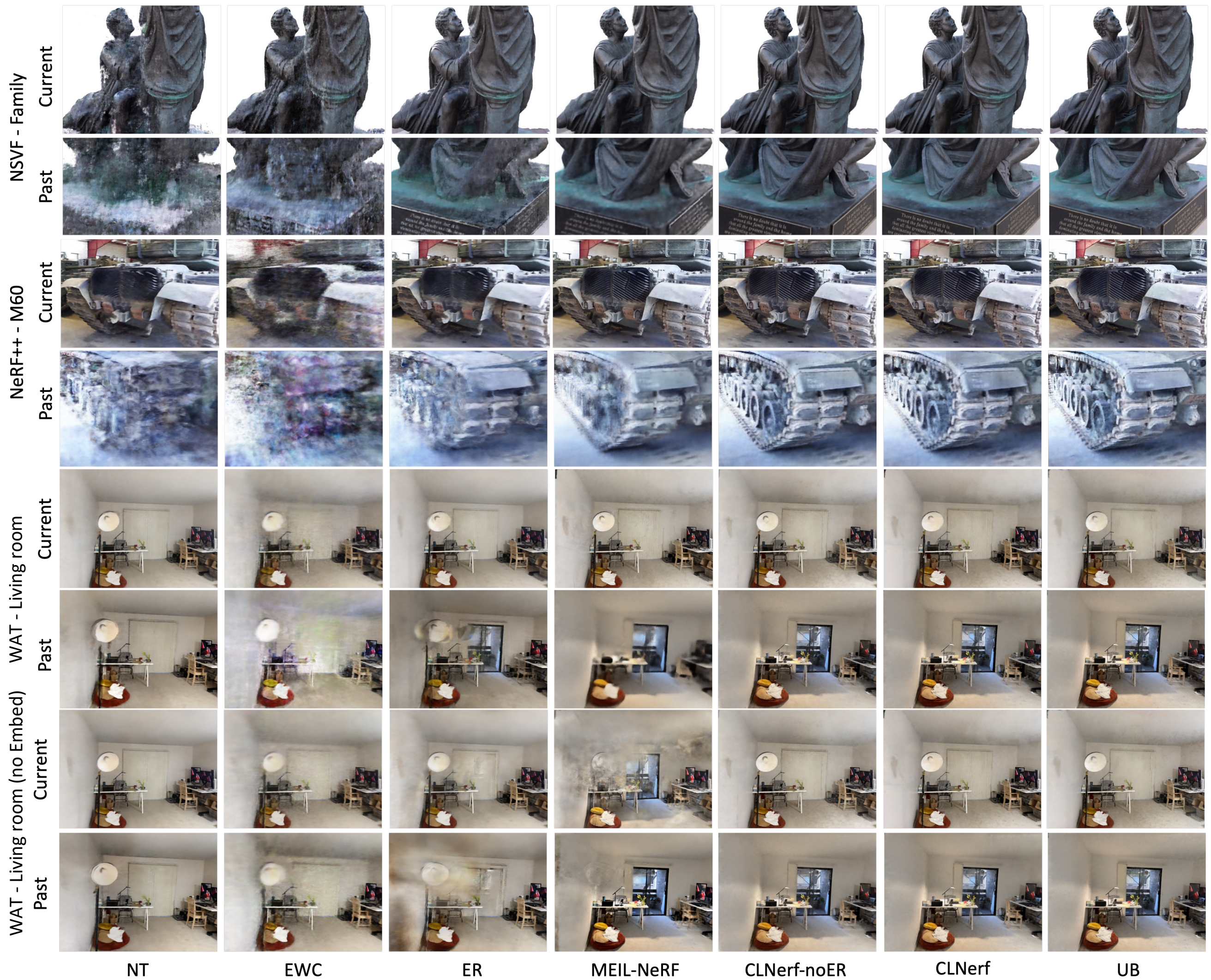


Figure 5. **Qualitative results.** Each two rows show the (zoom-in) test views of the current and past scans rendered by different methods. CLNeRF has a similar rendering quality as UB, even without storing any historical images. NT overfits to the new data, resulting in erroneous renderings for early scans. The regularization from EWC not only hinders the model from adapting to new data but also fails to recover the old scene appearance/geometry. Blur and artifacts appear on images rendered by ER and MEIL-NeRF, especially in early scans, due to the lack of enough replay data (ER), the biased sampling and loss function design (MEIL-NeRF). Without using the trainable embeddings proposed in CLNeRF (WAT - Living room (noEmbed)), other continual NeRF approaches perform much worse on WAT.

and sometimes (*e.g.*, on Synth-NeRF) worse than ER (NeRF) in Tab. 1. This result shows the importance of advanced architectures for guaranteeing the effectiveness of generative replay. Without the trainable embeddings (No Embed), CLNeRF cannot adapt well to changing appearance and geometry of WAT.

5.2.2 Effect of Replay Buffer

Replay Buffer Size. To mimic the case of highly limited storage, we only allow 10 historical images to be stored for experience replay in the main experiment. Here, we investigate the effect of replay buffer size. Specifically, we vary the replay buffer size of ER (with NGP) and CLNeRF in the pattern of $\{0, 10, 10\%, ..., 100\%\}$ and report the performance change. 0 and 10 (roughly $2.5\% - 5\%$ on WAT) are the number of stored images. The percentages are with respect to all images across time steps. As shown in Fig. 6, CLNeRF does not require any samples stored in the replay buffer to perform well but it also does not hurt performance. ER requires a large replay buffer size (80%) to perform on-par with CLNeRF. This interesting result shows that widely-used CL methods designed for image classification can be sub-optimal for other problems.

Replay Buffer Update Strategy. CLNeRF applies reservoir sampling to update the replay buffer at each time step. Here, we analyze the effect of different replay buffer update strategies. Specifically, we compare the performance of CLNeRF using reservoir sampling [5] and prioritized replay [27]. As shown in Tab. 3, changing the reservoir buffer to a prioritized replay buffer does not improve CLNeRF. Hence, a uniform coverage of the whole scene (changes) is sufficient for effective experience replay.

Method \ Dataset	Synth-NeRF	WAT
CLNeRF	**32.16/0.957**	**25.45/0.764**
No ER	31.96/0.957	25.44/0.762
No GenRep	27.35/0.919	18.52/0.634
No NGP	29.23/0.940	23.50/0.728
No Embed	N.A.	21.09/0.725

Table 2. **Ablation study**. The performance of CLNeRF drops slightly without ER, but significantly without generative replay (No GenRep) or the use of the NGP architecture. Without using the trainable embeddings (No Embed), CLNeRF performs much worse under the appearance and geometry changes of WAT. Synth-NeRF has only static scenes, hence no trainable embeddings are required.

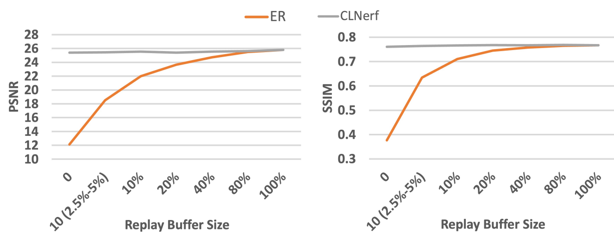


Figure 6. **Effect of replay buffer sizes (on WAT).** The performance of CLNeRF remains high across different replay buffer sizes. ER on NGP requires a replay buffer size of more than 80% of the dataset size to perform on-par with CLNeRF. Note that 80% corresponds to almost all historical data before the current time step, since all data from the current time step is always available. "10(2.5% − 5%)" means we allow 10 images to be stored in the replay buffer, which is roughly 2.5% − 5% of all images of a scene. "10%" means the replay buffer can store "10%" of all images from the same scene.

Method \ Dataset	Synth-NeRF	WAT
Reservoir	32.16/0.957	25.45/0.764
Prioritized	32.16/0.957	25.40/0.767

Table 3. **Effect of replay buffer update methods**. Prioritized sampling and reservoir sampling perform similarly. Due to the simplicity and efficiency, we use a reservoir buffer for CLNeRF.

Method \ Dataset	SynthNeRF	WAT
MEIL-NeRF (NeRF)	27.99/0.931	23.05/0.721
CLNeRF-noER (NeRF)	28.56/0.936	23.40/0.727
UB (NeRF)	31.52/0.948	24.01/0.740

Table 4. **CLNeRF vs. MEIL with vanilla NeRF**. CLNeRF also outperforms MEIL-NeRF with vanilla NeRF backbone, despite equipping MEIL-NeRF with the proposed trainable embeddings.

5.2.3 Effect of Architecture

To show the effectiveness of CLNeRF across architectures, we compare it against UB and MEIL-NeRF using vanilla NeRF as a backbone. We do not use experience replay for either CLNeRF or MEIL-NeRF, and we equip all methods with the trainable embeddings proposed in Sec. 3.4. As shown in Tab. 4, CLNeRF still performs slightly better than MEIL-NeRF, even though MEIL-NeRF was specifically designed based on vanilla NeRF. The performance gap between CLNeRF/MEIL-NeRF and UB on SynthNeRF is larger with vanilla NeRF than with NGP, highlighting the importance of advanced architectures for generative replay.

Method	NeRF	NeRF-Reinit	NGP
NT	28.53/0.938	25.01/0.900	21.66/0.858
ER	30.29/0.948	28.09/0.928	27.35/0.919
CLNeRF-noER	28.56/0.936	28.20/0.933	31.96/0.956
UB	31.52/0.948	NA	32.94/0.959

Table 5. **Effect of architectures to different methods (on Synth-NeRF)**. The performance gain of NT and ER using vanilla NeRF comes largely from the capability to initialize the current model $\boldsymbol{\theta}_t$ with the previous model $\boldsymbol{\theta}_{t-1}$. With re-initialization, vanilla NeRF still performs better than NGP on ER and NT. We conjecture that NGP overfits more to training data in the case of sparse views. Generative replay in CLNeRF allows NGP to overcome this issue, and perform better than vanilla NeRF.

As shown in Tab. 1, both ER and NT benefit more from vanilla NeRF. To reveal the underlying reason, we compare NT, ER, CLNeRF under 3 different training strategies: (1) Trained using vanilla NeRF without re-initialization (NeRF). (2) Trained using vanilla NeRF with re-initialization at each time step t (NeRF-Reinit). (3) Trained using NGP (NGP). As shown in Tab. 5, a large portion of the performance gap lies in the inheritance of model parameters from previous time steps, *i.e.*, not performing re-initialization for vanilla NeRF. When both are re-initialized, vanilla NeRF still performs slightly better than NGP for methods without generative replay. We conjecture that this is because NGP overfits more to the training data given sparse views (which is the case for NT and ER), and generalizes poorly on novel views. Performing generative replay allows NGP to overcome the sparse training view issue and exceed the performance of vanilla NeRF.

6. Conclusion

This work studies continual learning for NeRFs. We propose a new dataset – World Across Time (WAT) – containing natural scenes with appearance and geometry changes over time. We also propose CLNeRF, an effective continual learning system that performs close to the upper bound model trained on all data at once. CLNeRF uses generative replay and performs well even without storing any historical images. While our current experiments only cover scenes with hundreds of images, they are an important step toward deploying practical NeRFs in the real world. There are many interesting future research directions for CLNeRF. For example, solving the NaN loss problem of NGP to make model inheritance more effective during continual learning. Extending CLNeRF to the scale of Block-NeRF [32] is also an interesting future work.

References

[1] Nerfw pytorch lightning implementation. 'https://github.com/kwea123/nerf_pl/tree/nerfw'.

[2] NGP pytorch lightning. https://github.com/kwea123/ngp_pl

[3] pretrained deeplabv3 model. https://pytorch.org/vision/main/models/generated/torchvision.models.segmentation.deeplabv3_resnet101.html#torchvision.models.segmentation.DeepLabV3_ResNet101_Weights

[4] Zhipeng Cai, Ozan Sener, and Vladlen Koltun. Online continual learning with natural distribution shifts: An empirical study with visual data. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 8281–8290, 2021.

[5] Arslan Chaudhry, Marcus Rohrbach, Mohamed Elhoseiny, Thalaiyasingam Ajanthan, Puneet K Dokania, Philip HS Torr, and Marc'Aurelio Ranzato. On tiny episodic memories in continual learning. *arXiv preprint arXiv:1902.10486*, 2019.

[6] Jaeyoung Chung, Kanggeon Lee, Sungyong Baik, and Kyoung Mu Lee. Meil-nerf: Memory-efficient incremental learning of neural radiance fields. *arXiv preprint arXiv:2212.08328*, 2022.

[7] 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*, 44(7):3366–3385, 2021.

[8] Kyle Gao, Yina Gao, Hongjie He, Denning Lu, Linlin Xu, and Jonathan Li. Nerf: Neural radiance field in 3d vision, a comprehensive review. *arXiv preprint arXiv:2210.00379*, 2022.

[9] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *Communications of the ACM*, 63(11):139–144, 2020.

[10] Richard Hartley and Andrew Zisserman. *Multiple view geometry in computer vision*. Cambridge university press, 2003.

[11] Alain Hore and Djemel Ziou. Image quality metrics: Psnr vs. ssim. In *2010 20th international conference on pattern recognition*, pages 2366–2369. IEEE, 2010.

[12] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13):3521–3526, 2017.

[13] 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):1–13, 2017.

[14] Ruilong Li, Matthew Tancik, and Angjoo Kanazawa. Nerfacc: A general nerf acceleration toolbox. *arXiv preprint arXiv:2210.04847*, 2022.

[15] Zhizhong Li and Derek Hoiem. Learning without forgetting. *IEEE transactions on pattern analysis and machine intelligence*, 40(12):2935–2947, 2017.

[16] Zhiqiu Lin, Jia Shi, Deepak Pathak, and Deva Ramanan. The clear benchmark: Continual learning on real-world imagery. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*, 2021.

[17] Lingjie Liu, Jiatao Gu, Kyaw Zaw Lin, Tat-Seng Chua, and Christian Theobalt. Neural sparse voxel fields. *Advances in Neural Information Processing Systems*, 33:15651–15663, 2020.

[18] David Lopez-Paz and Marc'Aurelio Ranzato. Gradient episodic memory for continual learning. *Advances in neural information processing systems*, 30, 2017.

[19] Arun Mallya and Svetlana Lazebnik. Packnet: Adding multiple tasks to a single network by iterative pruning. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 7765–7773, 2018.

[20] 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 *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7210–7219, 2021.

[21] Nelson Max. Optical models for direct volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 1(2):99–108, 1995.

[22] Michael McCloskey and Neal J Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of learning and motivation*, volume 24, pages 109–165. Elsevier, 1989.

[23] 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. *Communications of the ACM*, 65(1):99–106, 2021.

[24] 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):1–15, 2022.

[25] Ameya Prabhu, Philip HS Torr, and Puneet K Dokania. Gdumb: A simple approach that questions our progress in continual learning. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part II 16*, pages 524–540. Springer, 2020.

[26] Christian Reiser, Songyou Peng, Yiyi Liao, and Andreas Geiger. Kilonerf: Speeding up neural radiance fields with thousands of tiny mlps. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 14335–14345, 2021.

[27] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

[28] Johannes Lutz Schönberger and Jan-Michael Frahm. Structure-from-motion revisited. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[29] Joan Serra, Didac Suris, Marius Miron, and Alexandros Karatzoglou. Overcoming catastrophic forgetting with hard atten-

tion to the task. In *International conference on machine learning*, pages 4548–4557. PMLR, 2018.

[30] Hanul Shin, Jung Kwon Lee, Jaehong Kim, and Jiwon Kim. Continual learning with deep generative replay. *Advances in neural information processing systems*, 30, 2017.

[31] Cheng Sun, Min Sun, and Hwann-Tzong Chen. Direct voxel grid optimization: Super-fast convergence for radiance fields reconstruction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5459–5469, 2022.

[32] Matthew Tancik, Vincent Casser, Xinchen Yan, Sabeek Pradhan, Ben Mildenhall, Pratul P Srinivasan, Jonathan T Barron, and Henrik Kretzschmar. Block-nerf: Scalable large scene neural view synthesis. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8248–8258, 2022.

[33] Kai Zhang, Gernot Riegler, Noah Snavely, and Vladlen Koltun. Nerf++: Analyzing and improving neural radiance fields. *arXiv preprint arXiv:2010.07492*, 2020.