# HollowNeRF: Pruning Hashgrid-Based NeRFs with Trainable Collision Mitigation

Xiufeng Xie, Riccardo Gherardi, Zhihong Pan, Stephen Huang
Oppo Mobile Telecommunications Corp.
2479 E Bayshore Rd, Palo Alto, CA, 94303, USA
xxie28@gmail.com

## Abstract

*Neural radiance fields (NeRF) have garnered significant attention, with recent works such as Instant-NGP accelerating NeRF training and evaluation through a combination of hashgrid-based positional encoding and neural networks. However, effectively leveraging the spatial sparsity of 3D scenes remains a challenge. To cull away unnecessary regions of the feature grid, existing solutions rely on prior knowledge of object shape or periodically estimate object shape during training by repeated model evaluations, which are costly and wasteful. To address this issue, we propose HollowNeRF, a novel compression solution for hashgrid-based NeRF which automatically sparsifies the feature grid during the training phase. Instead of directly compressing dense features, HollowNeRF trains a coarse 3D saliency mask that guides efficient feature pruning, and employs an alternating direction method of multipliers (ADMM) pruner to sparsify the 3D saliency mask during training. By exploiting the sparsity in the 3D scene to redistribute hash collisions, HollowNeRF improves rendering quality while using a fraction of the parameters of comparable state-of-the-art solutions, leading to a better cost-accuracy trade-off. Our method delivers comparable rendering quality to Instant-NGP, while utilizing just 31% of the parameters. In addition, our solution can achieve a PSNR accuracy gain of up to 1dB using only 56% of the parameters.*

## 1. Introduction

Neural Radiance Fields (NeRF [1, 15]) have gained widespread recognition across academia and industry due to their remarkable capability to generate photorealistic novel views of 3D scenes from a collection of 2D images. Inspired by the volumetric representation, NeRF models the scene as a continuous 5D plenoptic function, enabling the creation of high-fidelity renderings with accurate lighting and shading effects. This technique has found versatile applications in fields such as computer graphics, virtual and augmented reality, as well as robotics.

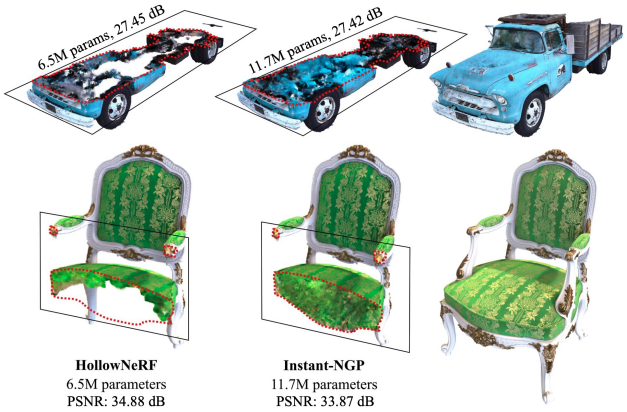



Figure 1: Cross-section views of rendered 3D scenes. The left column shows the hollow interior of HollowNeRF's rendering, while the middle column shows the solid interior of Instant-NGP's rendering. The right column displays HollowNeRF's rendering without slicing.

Training and evaluating NeRF models can be computationally expensive, and many recent works [20, 16, 4, 22, 18, 23] have focused on improving the efficiency of NeRF. Instant-NGP [16] is an established solution with state-of-the-art training speed. It employs a lightweight hashgrid for input encoding and a small multi-layer perceptron (MLP) to disambiguate hash collisions. However, the lightweight hashgrid unavoidably causes severe collisions at fine resolutions. These collisions are evenly scattered across the occupied voxels, resulting in a suboptimal accuracy. Another line of research focuses on accelerating NeRF rendering by only sampling near the surface of interest, but has to rely on prior knowledge of surface geometries (either from conventional algorithms such as shape-from-silhouette [13] or an MLP predicting the depth distribution along each camera ray [18]). However, a coarse surface estimation degrades

NeRF rendering quality, while a precise surface estimation adds too much complexity, defeating the purpose of acceleration.

We propose HollowNeRF, a novel NeRF compression method using trainable hash collision mitigation to improve rendering accuracy while consuming less parameters than existing NeRF methods. Built on a hash-based pipeline from Instant-NGP, HollowNeRF prioritizes the important features (of visible voxels) and prunes unnecessary features (of invisible voxels), leading to a redistribution of hash collision probability across the 3D volume. When two voxels direct to the same hash bucket, Instant-NGP shapes the shared feature in this bucket as a mixture of the desired features, which harms the accuracy. In contrast, HollowNeRF steers the shared feature to fit the more important voxel, and prunes the feature of the less important voxel toward $\mathbf{0}$, reducing interference to features sharing the same bucket. Specifically, when reading the feature of a certain voxel, HollowNeRF further scales the feature by a trainable *saliency weight* whose value captures the voxel's visibility. To reduce the cost, we divide the 3D space into coarse grid regions and assign a saliency weight to capture each region's visibility, forming a trainable 3D saliency grid. Unlike existing methods [13, 18] that require prior knowledge of the surface geometries, HollowNeRF learns to prioritize the important features by training the saliency grid, which converges to a "hollow" saliency distribution across the 3D volume. Figure 1 showcases this "hollow" rendering result.

The proposed design consists of three main components: a trainable 3D saliency grid to guide the compression of dense features (§3.1); a soft zero-skipping gate that enhance the MLP in the NeRF model to ensure a feature compressed to $\mathbf{0}$ translates to a zero density in the 3D space (§3.2); a pruner to further push unnecessary features to exact $\mathbf{0}$ instead of a small non-zero value by alternating direction method of multipliers or ADMM [3] (§3.3). Our experiments demonstrate that HollowNeRF achieves better accuracy (PSNR and LPIPS) than state-of-the-art methods while using significantly fewer parameters.

The key contributions of this work are:

- We propose a novel NeRF compression solution, HollowNeRF, that learns to prioritize features defining the visible surface and prune invisible internal features without prior knowledge of surface geometries.
- We use an ADMM-based optimization framework to prune unnecessary features during NeRF training and enhance the MLP with a soft zero-skipping gate to ensure that pruned features correctly map to zero density.
- We evaluate the performance of HollowNeRF on popular NeRF datasets and our solution demonstrates a significantly superior balance between cost and accuracy than state-of-the-art solutions.

## 2. Background and related work

Differentiable rendering [15, 7, 5, 22, 17] has emerged as a prominent alternative framework for novel view synthesis, alongside the conventional 3D rendering approaches. Following the seminal NeRF paper [15], the research community has matured the approach and enables it to handle arbitrarily large scenes [24, 29, 27, 2], complex reflections [8, 34, 21, 28, 14],and small training datasets with a limited number of views [10, 6, 19]. Recent works also substantially improve the efficiency of NeRF, reducing the run-time complexity at training [16, 22] and inference [16, 12, 18, 30, 32, 31, 9].

Comparatively fewer works have tackled the problem of maximizing visual fidelity for a given space complexity budget [16, 26], which is the focus of this paper. The original NeRF technique [15] encodes the entire 3D scene in the weights of an MLP that predicts opacity ($\sigma$) and color $(r, g, b)$ given a 3D position ($x$, $y$, $z$) and direction ($\theta$, $\phi$). While spatial complexity ( 438K parameters) is not the primary bottleneck, the substantial limitation of NeRF lies in the remarkably slow training and inference speeds. This is primarily attributed to the deep MLP with 8 hidden layers, each having a width of 256 neurons. It has since been shown [22] that differentiable rendering does not necessarily require neural networks. In [22], a scene is encoded as a neuron-free sparse voxel grid of opacities and RGB spherical harmonics. Without the MLP, training and inference run fast, at the cost of a significantly high spatial complexity.

In the continuum between the two extremes discussed above, there are techniques that harness the strengths of both ends and fare better in the balance between training speed and spatial complexity. Instant-NGP [16] is a representative example, which encodes features using a multi-resolution hashgrid and then feed the features into a lightweight MLP with only 2 hidden layers to decode the color and density; both the features in the hashgrid and the MLP are trained in conjunction. Our work extends the hash-based pipeline from Instant-NGP by achieving higher quality (PSNR & LPIPS) with less number of parameters. We achieve this goal by reclaiming the resources spent on empty, invisible, or internal regions. Instead of using an auxiliary MLP to predict space occupancy like DONeRF [18], we learn it through a trainable lightweight volumetric saliency grid.

Some existing literature has applied model compression techniques to NeRF: CC-NeRF [26] uses tensor decomposition to obtain a low-rank approximation of the learned network. Similarly, TensoRF [4] represents the volume as a 4D tensor and factorizes it into low-rank components. Unlike these compression works, HollowNeRF vets information based on its actual impact on the rendering accuracy, rather than raw entropy. The relative performance of these approaches are investigated in §4.4.

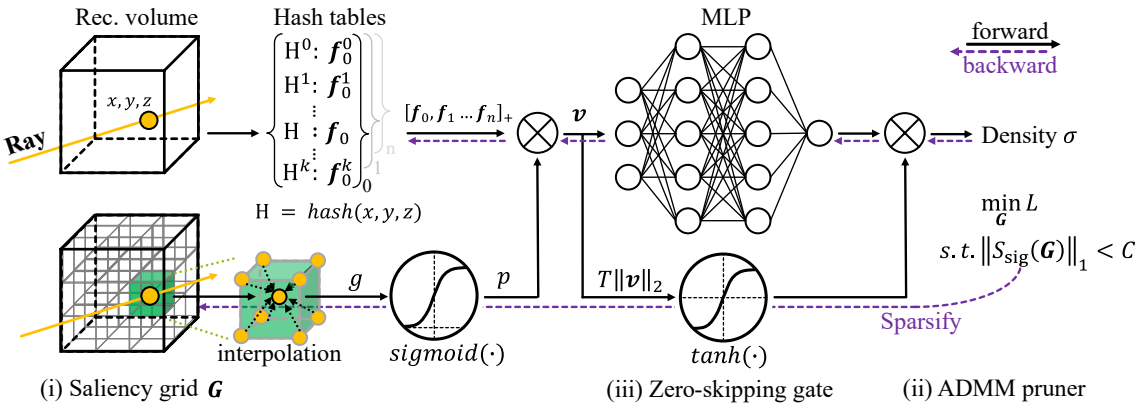


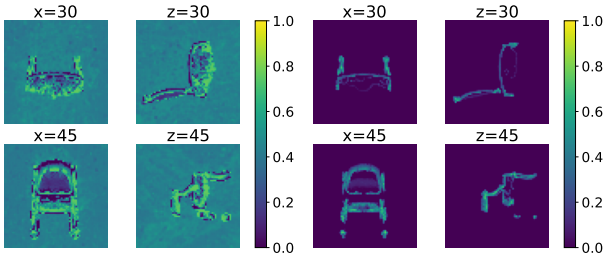

Figure 2: Overview of the HollowNeRF workflow.

## 3. HollowNeRF design

This section presents the system design of HollowNeRF, as outlined in Figure 2: *(i)* Given an input coordinate $\boldsymbol{x} = (x, y, z)$, we fetch the corresponding feature $\boldsymbol{f}$ from the multi-level hashgrid, following the Instant-NGP method [16]. *(ii)* We predict the saliency weight $p$ of position $\boldsymbol{x}$ by processing the information from a trainable 3D saliency grid $\mathcal{G}$ through trilinear interpolation and a sigmoid function (§3.1), and use $p$ to scale the feature $\boldsymbol{f}$ by $\boldsymbol{v} = p\boldsymbol{f}$. Figure 3a showcases a trained saliency grid $\mathcal{G}$, demonstrating that training $\mathcal{G}$ can mitigate hash collisions by suppressing the unnecessary features in empty or invisible regions. *(iii)* The weighted feature $\boldsymbol{v}$ is fed into an MLP, which decodes $\boldsymbol{v}$ to obtain the density $\sigma$ and color at position $\boldsymbol{x}$. To ensure that $\boldsymbol{v} = \mathbf{0}$ maps to $\sigma = 0$, we introduce a zero-skipping gate (§3.2). *(iv)* The density and color outputs of the MLP are used for volume rendering, and the resulting image is compared to the ground truth to obtain a loss function $L$. During training, an ADMM pruner (§3.3) sparsifies the saliency value distribution across the 3D space by enforcing a sparsity constraint when optimizing the loss $L$. Figure 3a shows that a large portion of the unnecessary features are not entirely eliminated by training the saliency grid, with saliency weights diminished but not reaching 0. Therefore, we introduce the ADMM pruner to explicitly prune saliency weights of the empty or invisible regions to 0, as shown in Figure 3b.

### 3.1. Trainable 3D saliency grid

In typical 3D scenes without large transparent objects, most regions of the space do not contribute to the final rendering as they are either empty or invisible from any view angle. To leverage this sparsity, methods such as Instant-NGP use a coarse-grained binary mask to capture space occupancy, which is periodically updated during training by evaluating the NeRF model at each voxel and checking if the density is above a threshold. This occupancy mask guides sampling in ray marching by skipping unoccupied regions. However, it still keeps unnecessary features in internal occluded regions that have non-zero densities but make no contributions to the visible surfaces from any view angle, as shown in Figure 1.




(a) Without ADMM pruner. (b) With ADMM pruner.

Figure 3: Slices of a $64 \times 64 \times 64$ saliency grid trained on the "Chair" scene from the NeRF synthetic dataset[15].

With this insight, we propose to reclaim the capacity spent on internal regions by learning the sparsity pattern through training, effectively making objects *hollow*. The benefits of this approach are twofold: it reduces space complexity by requiring fewer features, and improves accuracy by reducing hash collisions. However, precisely locating the invisible regions of objects can be challenging without any prior knowledge of the object's shape.

To addresses this challenge, our approach utilizes the hashgrid-based positional encoding from Instant-NGP [16] and extends it by introducing a trainable saliency grid $\mathcal{G}$ with a resolution of $T \times T \times T$. Specifically, HollowNeRF trains a 3D tensor data structure $\mathcal{G}$ alongside the NeRF model (feature hashgrid $\mathcal{H}$ and MLP weights $\mathcal{W}$), which gradually learns which regions to prioritize, such as a non-transparent object surface. We split the space coarsely as $T \times T \times T$ uniform grids and the 3D tensor $\mathcal{G}$ stores train-

able values representing the saliency of features near each grid voxel. Instead of letting multiple dense features in the same coarse grid share one value, we interpolate the values associated to the 8 voxels surrounding the input coordinate $\boldsymbol{x}$ to get a smoothed saliency distribution $\mathcal{P}$ over the 3D space. The trainable saliency grid represents in itself an opportunity to compress the NeRF model using differentiable model compression techniques, as discussed later in §3.3.

We initialize $\mathcal{G}$ as an all-ones tensor, where $T$ is typically much smaller than the resolution of the dense feature grid (In §4.2, we further investigate how different $T$ values affect the performance). Given a 3D coordinate $\boldsymbol{x} = (x, y, z)$, we fetch its corresponding feature vector $\boldsymbol{f}$ from the multi-level hashgrid [16]. We then locate the 8 voxels $(\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_8)$ surrounding $\boldsymbol{x}$ in the 3D saliency grid $\mathcal{G}$, and fetch its corresponding saliency value $g_i$. Specifically, the coordinate $\boldsymbol{x}_i = (x_i, y_i, z_i)$ are quantized to the values $(\hat{x}_i, \hat{y}_i, \hat{z}_i)$ between 0 and $T-1$, which are then used to index the saliency grid $\mathcal{G}$. This gives us the saliency value $g_i = \mathcal{G}(\hat{x}_i, \hat{y}_i, \hat{z}_i)$. After obtaining $g_i$ for each of the 8 saliency grid voxels surrounding $\boldsymbol{x}$, we calculate the *saliency weight* $p$ at coordinate $\boldsymbol{x}$ by trilinear interpolation followed by a sigmoid operator, so that $0 < p < 1$. The final weighted feature vector $\boldsymbol{v}$ is obtained by multiplying the saliency weight $p$ with the feature vector $\boldsymbol{f}$ fetched from the hashgrid, i.e., $\boldsymbol{v} = p\boldsymbol{f}$.

For mathematical analysis, we define the saliency distribution $\mathcal{P}$ as the collection of saliency weights $p$ at every possible coordinate $\boldsymbol{x}$, such that $p = \mathcal{P}(\boldsymbol{x})$. Our proposed method, HollowNeRF, learns the saliency distribution $\mathcal{P}$ by training the saliency grid $\mathcal{G}$ using gradient descent. It is worth noting that we inject $\mathcal{P}$ (and thus $\mathcal{G}$) to the training pipeline by feeding the weighted feature $\boldsymbol{v}$ instead of the raw feature $\boldsymbol{f}$ to the MLP for decoding; the rationale for this choice is that since the sigmoid function in the above workflow projects the saliency weights to the range $(0, 1)$, we can view the 3D saliency distribution as a probability distribution $\mathcal{P}(\boldsymbol{x})$ indicating the likelihood $p$ that a feature is non-zero at a given 3D position $\boldsymbol{x}$. There are only two possibilities for the feature at a given position: either the position is visible with probability $p = \mathcal{P}(\boldsymbol{x})$, and we retrieve the feature vector $\boldsymbol{f}$ from the multi-level hashgrid as in [16]; otherwise the position is empty or occluded with probability $1-p$, and the corresponding feature vector is an all-zero vector $\mathbf{0}$. Taking both cases into account, we can calculate the expectation $\boldsymbol{v}$ of the feature as:

$$\boldsymbol{v} = \mathcal{P}(\boldsymbol{x})\boldsymbol{f} + (1 - \mathcal{P}(\boldsymbol{x}))\,\mathbf{0} = p\boldsymbol{f}$$

Thus the weighted feature $\boldsymbol{v}$ fed to MLP decoder has a physical meaning, namely the expectation of the feature at $\boldsymbol{x}$ being either $\boldsymbol{f}$ fetched from feature hashgrid or $\mathbf{0}$. Then training the saliency grid is equivalent to learning the probability distribution $\mathcal{P}(\boldsymbol{x})$. This probability distribution represents the likelihood that a given voxel $\boldsymbol{x}$ contains a non-zero feature. A feature with a lower saliency weight $p$ is more likely to be $\mathbf{0}$ and will cause less interference to other features assigned to the same bucket in the feature hashgrid. When the saliency weight converges to 0, that feature is pruned with a probability of 1 and becomes a zero vector $\mathbf{0}$.

Training the saliency distribution can also be viewed as hash collision mitigation. Instant-NGP employs hash encoding and distributes hash collisions evenly across voxels with non-zero density. By learning to shuffle hash collisions across the space, HollowNeRF guarantees less interference for more important features. When the feature of a voxel is scaled by a higher saliency weight, it tends to accumulate more gradients during backward propagation, and the information at this voxel will dominate the trained feature value inside the hash bucket shared with other voxels. On the other hand, a feature scaled by zero saliency weight has a zero gradient, hence causes no harm to colliding features.

### 3.2. Soft zero-skipping gate

While HollowNeRF mitigates hash collisions by pruning unnecessary features to zero, the MLP used to decode these features into densities can break the sparsity in the 3D space: the MLP can be viewed as a function $\sigma = \mathcal{M}(\boldsymbol{v})$ mapping the feature vector $\boldsymbol{v}$ to the density $\sigma$, and typically $\mathcal{M}(\mathbf{0}) \neq 0$. Even though the saliency grid and ADMM pruner ensure most voxels direct to $\mathbf{0}$ features in the hashgrid, a typical MLP translates such a sparse feature domain to a non-sparse 3D space with non-zero densities scattered everywhere, which is evidently inaccurate. To enforce a 0 output density when the input feature is $\mathbf{0}$, we introduce a zero-skipping gate $g(\boldsymbol{v})$ to the MLP decoder:

$$\begin{aligned} \hat{\mathcal{M}}(\boldsymbol{v}) &= \hat{g}(\boldsymbol{v}) \cdot \mathcal{M}(\boldsymbol{v}) \\ \text{where} \quad \hat{g}(\boldsymbol{v}) &:= \tanh\left(\alpha \left\|\boldsymbol{v}\right\|_2\right) \end{aligned} \tag{1}$$

The gate $\hat{g}(\boldsymbol{v})$ is a differentiable approximation of the "hard" gate in Eq. (2) to enforce the desired property $\hat{\mathcal{M}}(\mathbf{0}) = 0$, and we use a constant $\alpha$ to control the steepness of the 0-to-1 transition.

$$g(\boldsymbol{v}) := \begin{cases} 0, & \text{if} \quad \left\|\boldsymbol{v}\right\|_2 = 0 \\ 1, & \text{otherwise} \end{cases} \tag{2}$$

The non-differentiable "hard" gate in Eq. (2) may cause abrupt density value changes during training. Also, the gradient $\frac{\partial g}{\partial p}$ of $g(\boldsymbol{v}) = g(p\boldsymbol{f})$ is 0 with $p \neq 0$, this 0 gradient prevents the ADMM pruner from suppressing a small saliency value $p$ to 0, resulting in insufficient feature pruning and little improvement in hash collision. Our experiments in §4.3 confirm that adding the "hard" gate $g(\boldsymbol{v})$ provides little gain over adding no gate.

Thus, we propose a "soft" zero-skipping gate $\hat{g}(\boldsymbol{v})$ in Eq. (1) that smoothly reduces the density output to zero for input features being pruned. As an approximation of $g(\boldsymbol{v})$,

the value of $\hat{g}(\boldsymbol{v})$ should smoothly transition from 0 to 1 for input $\boldsymbol{v}$ with small magnitudes $\|\boldsymbol{v}\|_2$, while otherwise staying close to 1 to avoid perturbing necessary information. The $\tanh$ function is a suitable choice. In our implementation we adjust the $\alpha$ value with a schedule, to gradually "harden" the soft gate as the training progresses. Specifically, for epochs below 1000, we set $\alpha$ to $10^4$ to ensure fast convergence. Afterward we increase $\alpha$ to $10^5$ to minimize the perturbation to the fine-tuning process.

### 3.3. ADMM pruner

Training the 3D saliency grid improves the accuracy by redistributing hash collisions, which, However, cannot fully eliminate unnecessary features, as shown in Figure 3a. To address this, we introduce an ADMM pruner that enforces sparsity in the saliency grid $\mathcal{G}$ during training.

We first motivate the use of ADMM pruner rigorously. Assuming a $K \times K \times K$ volume where each voxel $\boldsymbol{x}$ corresponds to a feature $\boldsymbol{f}$, and the ground truth feature $\boldsymbol{f}_1$ at voxel $\boldsymbol{x}_1 = (x_1, y_1, z_1)$ and $\boldsymbol{f}_2$ at a different voxel $\boldsymbol{x}_2 = (x_2, y_2, z_2)$ are typically different for a 3D scene. Instant-NGP encodes this $K \times K \times K$ feature grid into a small hashgrid $\mathcal{H}$ whose size $<< (K \times K \times K)$. When $\boldsymbol{f}_1$ and $\boldsymbol{f}_2$ collide with each other by directing to the same hash bucket, their values queried from the hashtgrid become identical $\boldsymbol{f}_1^* = \boldsymbol{f}_2^* = \boldsymbol{f}_H$, where $\boldsymbol{f}_H$ is the trained feature stored in the shared hash bucket. Since the ground truth features $\boldsymbol{f}_1 \neq \boldsymbol{f}_2$ while the queried features $\boldsymbol{f}_1^* = \boldsymbol{f}_2^*$, the queried values differ from the ground truth, causing compression artifacts. To solve this problem, HollowNeRF allows the colliding features to be different by introducing the scalar saliency weight $p$ to scale the queried features: $\boldsymbol{f}_1^* = p(\boldsymbol{x}_1)\boldsymbol{f}_H$ and $\boldsymbol{f}_2^* = p(\boldsymbol{x}_2)\boldsymbol{f}_H$. However, during training, the share feature $\boldsymbol{f}_H$ may not find a value that satisfies both equations, as they can be written as $\boldsymbol{f}_1^* = \frac{p(\boldsymbol{x}_1)}{p(\boldsymbol{x}_2)}\boldsymbol{f}_2^*$ which is not always true especially when $\boldsymbol{f}_1$ and $\boldsymbol{f}_2$ are non-zero. Then the queried features $\boldsymbol{f}_1^*$ and $\boldsymbol{f}_2^*$ may still cause compression artifacts (although less than instant-NGP). The ADMM pruner addresses this challenge by pruning the ground truth $\boldsymbol{f}_1$ or $\boldsymbol{f}_2$ to $\mathbf{0}$ when $\boldsymbol{x}_1$ or $\boldsymbol{x}_2$ is empty/invisible. For example, if voxel $\boldsymbol{x}_1$ is invisible from any view angle ($\boldsymbol{f}_1 = \mathbf{0}$), there exists a single $\boldsymbol{f}_H$ value to satisfy both $\boldsymbol{f}_1 = p(\boldsymbol{x}_1)\boldsymbol{f}_H$ and $\boldsymbol{f}_2 = p(\boldsymbol{x}_2)\boldsymbol{f}_H$, that is $\boldsymbol{f}_H = \boldsymbol{f}_2$, and there is no compression artifacts. The optimizer can converge to this optimal $\boldsymbol{f}_H$ value through gradient descent by learning that $p(\boldsymbol{x}_1)$ is 0. In summary, making the feature grid sparse reduces the compression artifacts when encoding features into a hashgrid.

In what follows, we describe two methods for achieving sparsity: a basic L1-regularization approach and our ADMM pruner. To sparsify the saliency grid $\mathcal{G}$, one simple approach is to add an L1 regularization term to the original MSE loss function $L$ typically used in NeRF training:

$$\min_{\mathcal{W},\mathcal{H},\mathcal{G}} \left(L + \lambda \| S_{\text{sig}}\left(\mathcal{G}\right) \|_1\right) \tag{3}$$

where $\lambda$ is a constant weight to control the amount of sparsity and compression, and $\mathcal{H}$ and $\mathcal{W}$ denote respectively the embeddings in the multi-level hashgrid and the weights in the MLP. We use the sigmoid operator $S_{\text{sig}}(\cdot)$ here because the forward pipeline (Figure 2) uses the saliency value after the sigmoid operator to weight the feature vector, and our ultimate goal is a sparse distribution of saliency value to prune unnecessary features. Note that $S_{\text{sig}}(\mathcal{G})$ here denotes applying sigmoid operator on each element $g$ of the 3D tensor $\mathcal{G}$, and the result is a 3D tensor with all non-negative elements. Therefore, the absolute operator in the L1 regularization term can be removed:

$$\|S_{\text{sig}}(\mathcal{G})\|_1 = \sum_{g \in \mathcal{G}} |S_{\text{sig}}(g)| = \sum_{g \in \mathcal{G}} S_{\text{sig}}(g)$$

Removing the absolute operator from the regularization term makes it differentiable and compatible with the training pipeline, eliminating the need to use more complex optimization algorithms such as proximal gradient descent to handle non-differentiable regularization terms.

Although the usage of L1 regularization is already a marked improved over the baseline (see §4.3), we find it is often difficult to select a single $\lambda$ hyperparameter to control sparsity and compression. During training when the 3D saliency grid $\mathcal{G}$ has already become sparse enough, the regularization term $\lambda\|S_{\text{sig}}(\mathcal{G})\|_1$ may still provide gradients that further prune $\mathcal{G}$. If $\lambda$ is set too high or if the algorithm is let to run for too long, visible features may be pruned resulting in performance degradation.

We use an ADMM pruner to overcome the limitations of simple L1 regularization. By introducing a configurable sparsity constraint $C$ such that $\|S_{\text{sig}}(\mathcal{G})\|_1 < C$, feature pruning becomes the constrained optimization problem:

$$\begin{aligned} \min_{\mathcal{M},\mathcal{H},\mathcal{G}} \quad & L\left(\mathcal{M},\mathcal{H},\mathcal{G}\right) \\ \text{s.t.} \quad & \|S_{\text{sig}}\left(\mathcal{G}\right)\|_1 < C \end{aligned}$$

We transform this constrained optimization problem into an equivalent unconstrained minimax problem in Problem (5) by using the augmented Lagrangian method which employs a trainable dual variable $\gamma$, also known as a Lagrange multiplier. In essence, $\gamma$ serves as a trainable alternative to the fixed hyperparameter $\lambda$ in Problem (3).

$$\begin{aligned} \min_{\mathcal{W},\mathcal{H},\mathcal{G}} \max_{\gamma \geq 0} \quad & L\left(\mathcal{W},\mathcal{H},\mathcal{G}\right) + \frac{\rho_\gamma}{2}\left[\|S_{\text{sig}}\left(\mathcal{G}\right)\|_1 - C\right]_+^2 \\ & + \gamma\left(\|S_{\text{sig}}\left(\mathcal{G}\right)\|_1 - C\right) \end{aligned} \tag{5}$$

| Method \ Hashgrid size / Param # | | $2^{14}$ | $2^{15}$ | $2^{16}$ | $2^{17}$ | $2^{18}$ | $2^{19}$ |
|---|---|---|---|---|---|---|---|
| Instant-NGP | | 0.50M | 0.94M | 1.77M | 3.34M | 6.22M | 11.49M |
| HollowNeRF | T=64 | 0.76M | 1.21M | 2.04M | 3.60M | 6.48M | 11.75M |
| | T=96 | 1.39M | 1.83M | 2.66M | 4.22M | 7.10M | 12.37M |
| | T=128 | 2.60M | 3.04M | 3.87M | 5.43M | 8.32M | 13.58M |

Table 1: The total number of model parameters, depending on the hashgrid size and saliency grid resolution $T$. This number counts the parameters in the hashgrid, MLP, and saliency grid. (both HollowNeRF and Instant-NGP employ a multi-resolution hashgrid with 16 levels. We use the maximum entries per level to denote the hashgrid size, following the notation in the Instant-NGP paper [16]. )

where $\rho_\gamma$ denotes the learning rate of dual variable $\gamma$, and $[\cdot]_+$ clamps the input to non-negative values.

At each iteration $t$, the optimizer first updates the coarse saliency grid $\mathcal{G}$, the feature hashgrid $\mathcal{H}$, and the MLP weights $\mathcal{W}$ to minimize the augmented loss function in Problem (5) using gradient descent, where the dual variable $\gamma$ is treated as a constant. $\gamma$ is then updated using gradient ascent with $\gamma^{(t+1)} = \left[\gamma^{(t)} + \rho_\gamma \left(\|S_{\text{sig}}\left(\mathcal{G}\right)\|_1 - C\right)\right]_+$

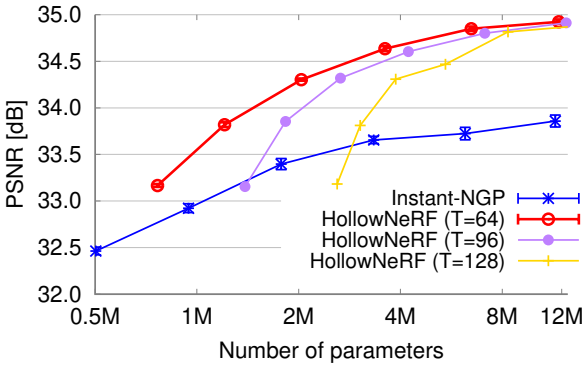
## 4. Evaluation

In this section, we first investigate the effect of different hyper-parameters, including hashgrid sizes, saliency grid sizes, and sparsity constraints, on the performance (§4.2). We then conduct an ablation study to validate each of HollowNeRF's design components (§4.3). Finally, we evaluate the performance of HollowNeRF on various 3D scenes and compare it with the state-of-the-art works (§4.4).

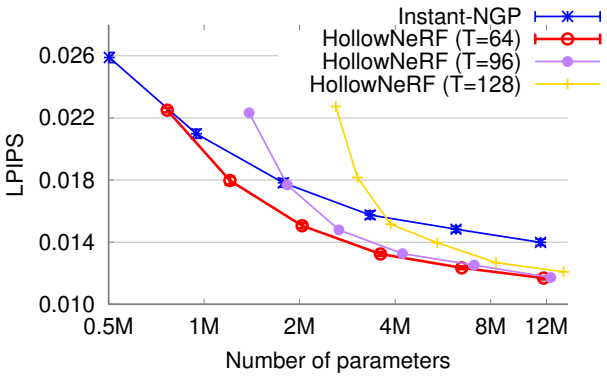
### 4.1. Experimental setup

We implement HollowNeRF on top of `torch-ngp` [25, 26], a PyTorch CUDA extension implementation of instant-NGP [16]. To ensure a fair comparison, we evaluated both HollowNeRF and instant-NGP using the same `torch-ngp` implementation. Both methods use a 16-level hashgrid with a feature dimension of 2, where the coarsest resolution of the feature grid is 16, and the finest resolution is 1024. Both methods employ a MLP decoder that has only two hidden layers with a width of 64 neurons, while HollowNeRF augments the MLP with a zero-skipping gate as described in §3.2.

We use the peak signal-to-noise ratio (PSNR) and the Learned Perceptual Image Patch Similarity (LPIPS) [33] with the default AlexNet backend as the performance metrics to assess the rendering quality. Each experiments was conducted on a single NVIDIA Tesla A100 GPU for both training and evaluation. The machine used was equipped with an AMD EPYC 7742 64-Core CPU, 1TB of memory, and 8 GPU cards in total.




(a) PSNR ↑ (higher is better).



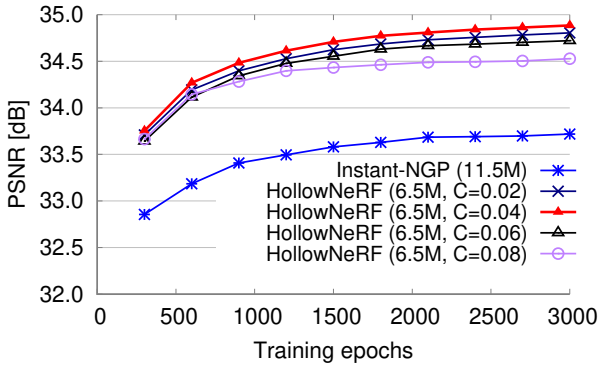

(b) LPIPS ↓ (lower is better).

Figure 4: Performance vs. saliency grid size $T$.

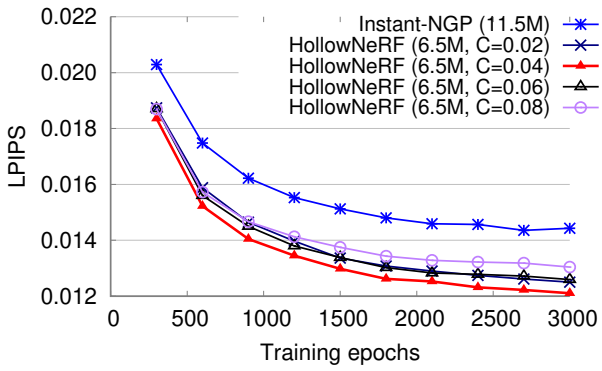
Unless otherwise specified, we train the model for $300,000$ steps using the Adam optimizer with an initial learning rate of $1 \times 10^{-2}$. Further details regarding the appropriate selection of HollowNeRF hyper-parameters, such as $T$ and $C$, are discussed in §4.2.

### 4.2. Investigating the cost-accuracy tradeoff

We first investigate the cost-accuracy tradeoff of HollowNeRF to determine the appropriate hyper-parameter config-

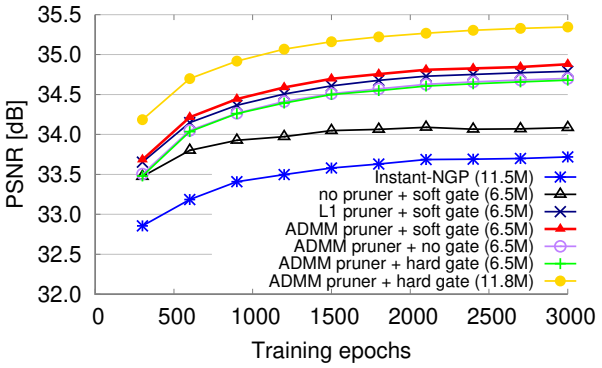



(a) PSNR ↑ (higher is better).

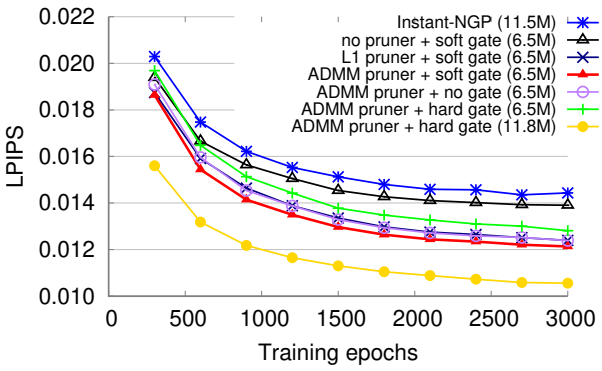



(b) LPIPS ↓ (lower is better).

Figure 5: Performance vs. sparsity constraint $C$.




(a) PSNR ↑ (higher is better).




(b) LPIPS ↓ (lower is better).

Figure 6: Ablation study for the design components.

uration for larger scale experiments in §4.4. We conducted experiments on the chair scene from the NeRF synthetic dataset, varying the hashgrid size from $2^{14}$ to $2^{19}$ and testing the saliency grid resolution of 64, 96, and 128 for each hashgrid size. Table 1 summarizes the cost of different configurations, including the number of parameters in the hashgrid, MLP, and saliency grid. We use a sparsity constraint of $C = 0.04$ in this experiment.

The results in Figure 4 indicate that increasing the saliency grid resolution $T$ from 64 to 96 or 128 does not improve accuracy much but results in a higher parameter count, which harms the cost-accuracy tradeoff. On the other hand, lowering $T$ below 64 causes unstable convergence. Consequently, we use a grid resolution of $T = 64$ for the subsequent experiments. Figure 4 further shows that HollowNeRF outperforms the baseline over a wide range of model sizes, achieving higher PSNR and lower LPIPS values for comparable parameter counts. The best overall model, HollowNeRF with $T = 64$, uses 6.48M parameters to attain a 34.85dB PSNR, as compared to Instant-NGP's 33.86dB using 11.49M. In other words, it achieves about 1dB higher PSNR than Instant-NGP, while utilizing only 56% of the parameters employed by Instant-NGP.

We also repeated the training and evaluation for each hashgrid size five times for HollowNeRF with $T = 64$ and instant-NGP, and plotted the standard deviations across the five tests as error bars in Figure 4. The results demonstrate that our approach is more stable than Instant-NGP in terms of convergence, with lower performance variation.

To investigate the impact of the choice of sparsity constraint $C$ on performance, we conducted additional experiments following the same setup as previous tests, while varying the $C$ values, and report the results in Figure 5. First, from Figure 5a, we observe that the $C = 0.04$ configuration yields the highest PSNR, in other words, HollowNeRF achieves the highest accuracy when 4% of the spatial voxels are non-zero. $C = 0.02$ exhibits slightly diminished accuracy, potentially due to excessive pruning in the 3D space, which compromises crucial information for scene reconstruction. Conversely, $C = 0.08$ shows the lowest accuracy across all tested HollowNeRF configurations, suggesting that keeping 8% non-zero voxels causes excessive hash collisions when packing them into the hashgrid, and the 3D scene to capture could be sparser. We have similar observation for LPIPS, as depicted in Figure 5b.

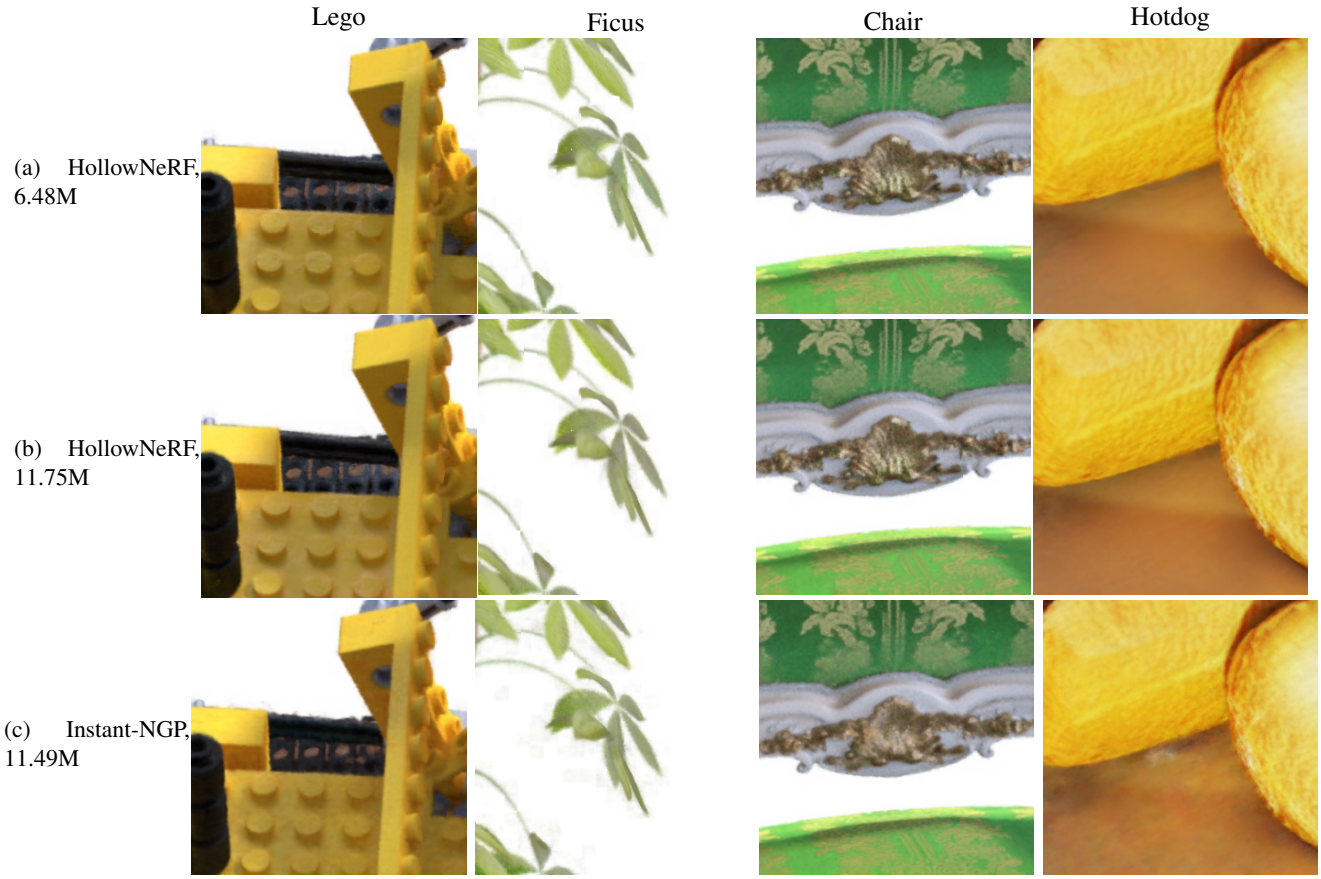



Figure 7: Qualitative comparison with instant-NGP over scenes from the NeRF synthetic dataset.

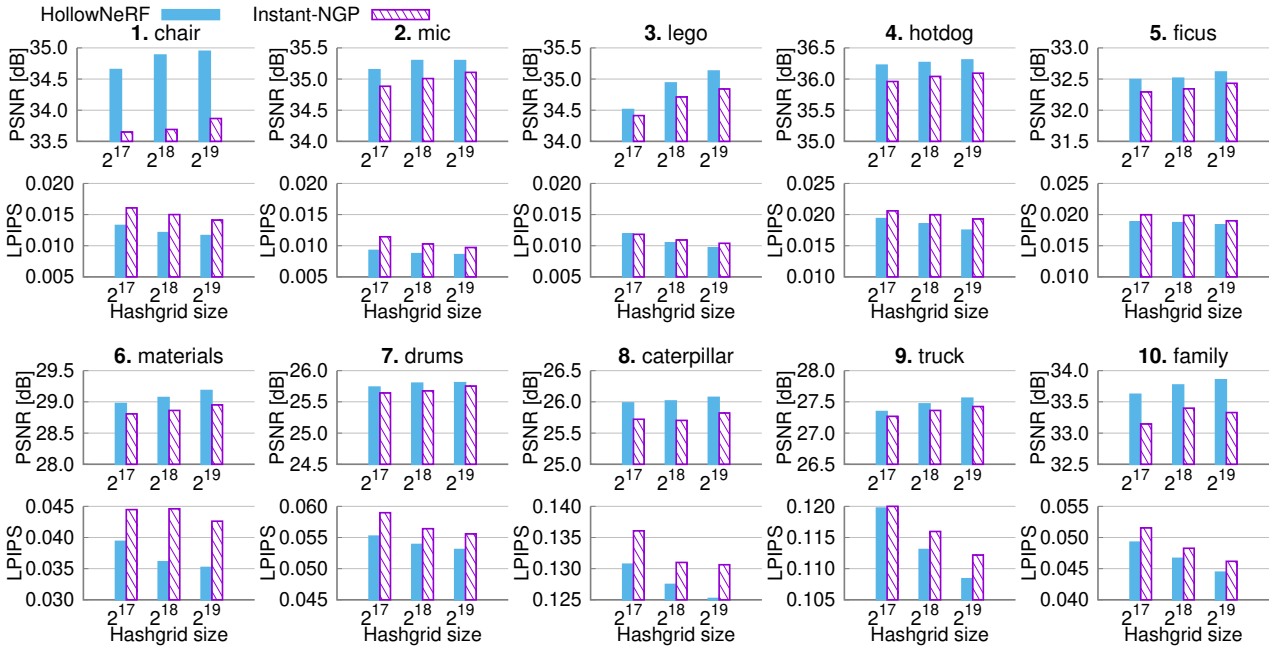



Figure 8: Comparison of PSNR(↑) and LPIPS (↓) performance between HollowNeRF and Instant-NGP on various scenes.

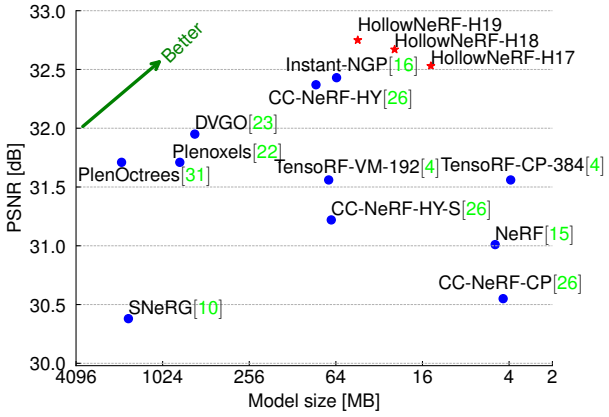



Figure 9: Comparison of HollowNeRF and recent methods in terms of the trade-off between PSNR and model size ("HollowNeRF-H19" means using a hashgrid size of $2^{19}$).

### 4.3. Ablation study

In this ablation study, we analyze the impact of each component in our solution by comparing the convergence speed and resulting performance of the configurations with and without each design component. We use a saliency grid resolution of 64 and a hashgrid size of $2^{18}$ (total 6.48M parameters) for all HollowNeRF configurations, while the baseline is instant-NGP with a hashgrid size of $2^{19}$ (total 11.49M parameters). To investigate how each component affects the convergence speed, we evaluate the performance on the test sets every 300 epochs during the total 3000 epochs.

The results presented in Figure 6 show that each design component presence gracefully increases the overall performance, without any catastrophic failure. Each of the HollowNeRF ablated configurations still outperforms instant-NGP, while employing less parameters.

### 4.4. Comparisons on standardized datasets

Finally, we evaluate the performance of HollowNeRF on 3D scenes from the NeRF synthetic dataset [15] and the more complicated Tanks and Temples dataset [11]. Based on the cost-accuracy tradeoff investigated in Sec. 4.2, we set HollowNeRF's hyper-parameters to a sparsity constraint of $C = 0.04$ and a saliency grid resolution of $T = 64$.

We first provide qualitative example views comparing HollowNeRF with Instant-NGP. The distinct advantages of HollowNeRF are evident through the visuals showcased in Figure 7. For example, from the lego scene in column 1, we observe that HollowNeRF with 6.48M parameters achieves notably superior rendering accuracy over Instant-NGP with 77% more parameters (11.49M). The advantage is particularly pronounced within the bulldozer track region. Furthermore, HollowNeRF with 11.75M parameters exhibits the highest accuracy, capturing fine details such as the tiny indentation on the edge of the black rod located at the left border of the view.

Then we show the PSNR and LPIPS performance of HollowNeRF and Instant-NGP on various 3D scenes (we sample 10 different scenes due to the page limit) in Figure 8. For each scene, we test three hashgrid sizes ($2^{17}$, $2^{18}$, $2^{19}$) for both methods. We can observe that HollowNeRF consistently outperforms Instant-NGP on all the 10 scenes tested, achieving better PSNR and LPIPS performance with fewer parameters when using the same hashgrid size. For seven out of ten tested scenes, the HollowNeRF with the smallest hashgrid size ($2^{17}$, for a total parameter count including MLP of 3.34M) achieves higher PSNR than the best Instant-NGP model (hashgrid size of $2^{19}$, corresponding to 11.49M parameters including the MLP).

Finally, we compare the average PSNR across the NeRF synthetic dataset [16] and the corresponding model size for HollowNeRF and several recent works. Figure 9 shows that HollowNeRF achieves the best cost-accuracy tradeoff among all tested methods. In particular, HollowNeRF with a hashgrid size of $2^{17}$ achieves an average PSNR of 32.53dB with only a mode size of 14.0MB.

## 5. Discussion

The compression gains of HollowNeRF require the 3D scene being sparse, *i.e.* mostly filled with either empty or occluded space, which holds for typical 3D scenes. HollowNeRF can still operate when this assumption does not hold, such as for scenes containing smoke, fire, clouds, but its performance will regress to the baseline without pruning.

The current version of HollowNeRF, like its predecessor Instant-NGP, faces challenges in modeling objects with reflective surfaces. In future works, we plan to extend the concept of HollowNeRF to the frameworks [8, 34, 21, 28, 14] with better support for reflective surfaces.

## 6. Conclusions

In this paper, we present HollowNeRF, a novel hashgrid-based NeRF technique that achieves superior rendering quality than state-of-the-art solutions like instant-NGP while using only a fraction of the parameters. HollowNeRF mitigates hash collisions by a simple feature pruning mechanism. Unlike existing methods that rely on explicit surface geometries for feature pruning, HollowNeRF learns a 3D saliency grid to guide feature compression during training, and employs an ADMM pruner to enforce a sparse feature domain. Our experiments demonstrate that HollowNeRF achieves a better balance between cost and accuracy than state-of-the-art solutions, making a solid step towards lightweight and ubiquitous NeRF applications.

## References


[1] 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 *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 5855–5864, October 2021. 1

[2] Jonathan T. Barron, Ben Mildenhall, Dor Verbin, Pratul P. Srinivasan, and Peter Hedman. Mip-nerf 360: Unbounded anti-aliased neural radiance fields. *CVPR*, 2022. 2

[3] Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato, Jonathan Eckstein, et al. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends® in Machine learning*, 3(1):1–122, 2011. 2

[4] Anpei Chen, Zexiang Xu, Andreas Geiger, Jingyi Yu, and Hao Su. Tensorf: Tensorial radiance fields. In *European Conference on Computer Vision (ECCV)*, 2022. 1, 2, 9

[5] Ronald Clark. Volumetric bundle adjustment for online photorealistic scene capture. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6124–6132, June 2022. 2

[6] Kangle Deng, Andrew Liu, Jun-Yan Zhu, and Deva Ramanan. Depth-supervised nerf: Fewer views and faster training for free. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 12882–12891, June 2022. 2

[7] Shubham Goel, Georgia Gkioxari, and Jitendra Malik. Differentiable stereopsis: Meshes from multiple views using differentiable rendering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8635–8644, June 2022. 2

[8] Yuan-Chen Guo, Di Kang, Linchao Bao, Yu He, and Song-Hai Zhang. Nerfren: Neural radiance fields with reflections. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 18409–18418, June 2022. 2, 9

[9] Peter Hedman, Pratul P Srinivasan, Ben Mildenhall, Jonathan T Barron, and Paul Debevec. Baking neural radiance fields for real-time view synthesis. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5875–5884, 2021. 2

[10] Ajay Jain, Matthew Tancik, and Pieter Abbeel. Putting nerf on a diet: Semantically consistent few-shot view synthesis. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 5885–5894, October 2021. 2, 9

[11] Arno Knapitsch, Jaesik Park, Qian-Yi Zhou, and Vladlen Koltun. Tanks and temples: Benchmarking large-scale scene reconstruction. *ACM Transactions on Graphics*, 36(4), 2017. 9

[12] Andreas Kurz, Thomas Neff, Zhaoyang Lv, Michael Zollhöfer, and Markus Steinberger. Adanerf: Adaptive sampling for real-time rendering of neural radiance fields. In *European Conference on Computer Vision (ECCV)*, 2022. 2

[13] H. Luo, A. Chen, Q. Zhang, B. Pang, M. Wu, L. Xu, and J. Yu. Convolutional neural opacity radiance fields. In *2021 IEEE International Conference on Computational Photography (ICCP)*, pages 1–12, Los Alamitos, CA, USA, may 2021. IEEE Computer Society. 1, 2

[14] Ben Mildenhall, Peter Hedman, Ricardo Martin-Brualla, Pratul P. Srinivasan, and Jonathan T. Barron. Nerf in the dark: High dynamic range view synthesis from noisy raw images. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 16190–16199, June 2022. 2, 9

[15] 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, 3, 9

[16] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *ACM Trans. Graph.*, 41(4):102:1–102:15, July 2022. 1, 2, 3, 4, 6, 9

[17] Jacob Munkberg, Jon Hasselgren, Tianchang Shen, Jun Gao, Wenzheng Chen, Alex Evans, Thomas Müller, and Sanja Fidler. Extracting triangular 3d models, materials, and lighting from images. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8280–8290, June 2022. 2

[18] Thomas Neff, Pascal Stadlbauer, Mathias Parger, Andreas Kurz, Joerg H. Mueller, Chakravarty R. Alla Chaitanya, Anton S. Kaplanyan, and Markus Steinberger. DONeRF: Towards Real-Time Rendering of Compact Neural Radiance Fields using Depth Oracle Networks. *Computer Graphics Forum*, 40(4), 2021. 1, 2

[19] Michael Niemeyer, Jonathan T. Barron, Ben Mildenhall, Mehdi S. M. Sajjadi, Andreas Geiger, and Noha Radwan. Regnerf: Regularizing neural radiance fields for view synthesis from sparse inputs. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5480–5490, June 2022. 2

[20] Christian Reiser, Songyou Peng, Yiyi Liao, and Andreas Geiger. Kilonerf: Speeding up neural radiance fields with thousands of tiny mlps. In *International Conference on Computer Vision (ICCV)*, 2021. 1

[21] Viktor Rudnev, Mohamed Elgharib, William Smith, Lingjie Liu, Vladislav Golyanik, and Christian Theobalt. Nerf for outdoor scene relighting. In *European Conference on Computer Vision (ECCV)*, 2022. 2, 9

[22] Sara Fridovich-Keil and Alex Yu, Matthew Tancik, Qinhong Chen, Benjamin Recht, and Angjoo Kanazawa. Plenoxels: Radiance fields without neural networks. In *CVPR*, 2022. 1, 2, 9

[23] Cheng Sun, Min Sun, and Hwann-Tzong Chen. Direct voxel grid optimization: Super-fast convergence for radiance fields reconstruction. In *CVPR*, 2022. 1, 9

[24] 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 (CVPR)*, pages 8248–8258, June 2022. 2

[25] Jiaxiang Tang. Torch-ngp: a pytorch implementation of instant-ngp, 2022. https://github.com/ashawkey/torch-ngp. 6

[26] Jiaxiang Tang, Xiaokang Chen, Jingbo Wang, and Gang Zeng. Compressible-composable neRF via rank-residual decomposition. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022. 2, 6, 9

[27] Haithem Turki, Deva Ramanan, and Mahadev Satyanarayanan. Mega-nerf: Scalable construction of large-scale nerfs for virtual fly-throughs. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 12922–12931, June 2022. 2

[28] 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 *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5491–5500, June 2022. 2, 9

[29] Yuanbo Xiangli, Linning Xu, Xingang Pan, Nanxuan Zhao, Anyi Rao, Christian Theobalt, Bo Dai, and Dahua Lin. Bungeenerf: Progressive neural radiance field for extreme multi-scale scene rendering. In *The European Conference on Computer Vision (ECCV)*, 2022. 2

[30] Qiangeng Xu, Zexiang Xu, Julien Philip, Sai Bi, Zhixin Shu, Kalyan Sunkavalli, and Ulrich Neumann. Point-nerf: Point-based neural radiance fields. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5438–5448, June 2022. 2

[31] Alex Yu, Ruilong Li, Matthew Tancik, Hao Li, Ren Ng, and Angjoo Kanazawa. Plenoctrees for real-time rendering of neural radiance fields. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5752–5761, 2021. 2, 9

[32] Jian Zhang, Jinchi Huang, Bowen Cai, Huan Fu, Mingming Gong, Chaohui Wang, Jiaming Wang, Hongchen Luo, Rongfei Jia, Binqiang Zhao, et al. Digging into radiance grid for real-time view synthesis with detail preservation. In *European Conference on Computer Vision*, pages 724–740. Springer, 2022. 2

[33] Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *CVPR*, 2018. 6

[34] Chengxuan Zhu, Renjie Wan, and Boxin Shi. Neural transmitted radiance fields. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022. 2, 9