The Thirty-Seventh AAAI Conference on Artificial Intelligence (AAAI-23)

TinyNeRF: Towards $100\times$ Compression of Voxel Radiance Fields

Tianli Zhao[1,2,3,4], **Jiayuan Chen**[3,4,5], **Cong Leng**[2,3,4], **Jian Cheng**[2,3,4*]

[1] School of Artificial Intelligence, University of Chinese Academy of Sciences. Beijing, China.
[2] Institute of Automation, Chinese Academy of Sciences, Beijing, China.
[3] AIRIA. Nanjing, China.
[4] Maicro.ai. Nanjing, China.
[5] Southeast University. Nanjing, China.

Abstract

Voxel grid representation of 3D scene properties has been widely used to improve the training or rendering speed of the Neural Radiance Fields (NeRF) while at the same time achieving high synthesis quality. However, these methods accelerate the original NeRF at the expense of extra storage demand, which hinders their applications in many scenarios. To solve this limitation, we present TinyNeRF, a three-stage pipeline: frequency domain transformation, pruning and quantization that work together to reduce the storage demand of the voxel grids with little to no effects on their speed and synthesis quality. Based on the prior knowledge of visual signals sparsity in the frequency domain, we convert the original voxel grids in the frequency domain via block-wise discrete cosine transformation (DCT). Next, we apply pruning and quantization to enforce the DCT coefficients to be sparse and low-bit. Our method can be optimized from scratch in an end-to-end manner, and can typically compress the original models by 2 orders of magnitude with minimal sacrifice on speed and synthesis quality.

1 Introduction

Synthesizing novel views of a 3D object from a sparse set of calibrated images is an appealing problem. It enhances customer experience for online product showcases and Street View maps (Tancik et al. 2022). Recently, Neural Radiance Fields (NeRF) (Mildenhall et al. 2020) has made a great breakthrough in this direction by representing the 3D radiance field implicitly with a coordinate neural network, yielding high quality of synthesized images, and the rendering quality has been further improved by many followup works (Zhang et al. 2020; Barron et al. 2021; Tancik et al. 2020; Sitzmann et al. 2020).

Despite its high synthesis quality, the original NeRF requires a large number of computations during both training and inference. It needs to query an MLP hundreds of times for rendering one single pixel, leading to its lengthy training time and inefficiency in novel view rendering. Thus, many follow up works have been done to accelerate its training (Liu et al. 2022; Kangle et al. 2022; Yu et al. 2022; Sun, Sun, and Chen 2022) or rendering process (Yu et al. 2022; Sun, Sun, and Chen 2022; Lindell, Martel, and Wetzstein 2021; Garbin et al. 2021; Liu et al. 2020; Yu et al. 2021; Reiser et al. 2021; Hedman et al. 2021; Wizadwongsa, Phongthawee, and Yenphraphai 2021), among which the voxel grid optimization based methods have achieved great success (Yu et al. 2022; Sun, Sun, and Chen 2022; Liu et al. 2020; Yu et al. 2021). These methods explicitly store volumetric 3D scene properties into voxel grids, which enables significant acceleration by removing empty voxels without scene contents and also yields great synthesis quality. For example, without the requirement of any generalizable pre-training or depth information, Plenoxels (Yu et al. 2022) and DVGO (Sun, Sun, and Chen 2022) can converge in less than 10 minutes on one single GPU, compared to days for the original NeRF (Mildenhall et al. 2020). However, their methods accelerate the training and rendering process at the cost of the large model size. For example, DVGO (Sun, Sun, and Chen 2022) needs to store two 3D grids with sizes 1×160^3 and 12×160^3, respectively, resulting in more than $200MB$ storage demand for one single scene, which is $40\times$ larger than the original NeRF model. This limitation hinders its applications in scenarios where online model transfer is frequently needed and storage resources are limited. Although the model size of these methods can be reduced by voxel pruning (Liu et al. 2020) and efficient data structures can be used to store the sparse voxels (Laine and Karras 2010; Lefebvre and Hoppe 2006; Niebner et al. 2013), the compression is still limited, and significant voxel pruning typically results in unacceptable quality degradation in rendering, as shown in Fig. 1.

Our goal in this paper is to reduce the storage demand for voxel grid optimization based NeRF methods, while at the same time maintaining their advantages in training/inference speed and synthesis quality. To this end, we present TinyNeRF. We take inspiration from the fact that most visual signals in the real-life are smooth in the spatial domain, so they should be sparse in the frequency domain. In other words, when decomposing these signals into different frequencies, most information will be centralized in the low-frequency region as shown in Fig. 4. Taking the advantage of this property, we compress the voxel grids in the frequency domain. Specifically, we first transform the voxel grid values in the frequency domain via block-wise discrete cosine transformation (DCT). After that, we apply pruning to the trans-

*The corresponding author.
Copyright © 2023, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

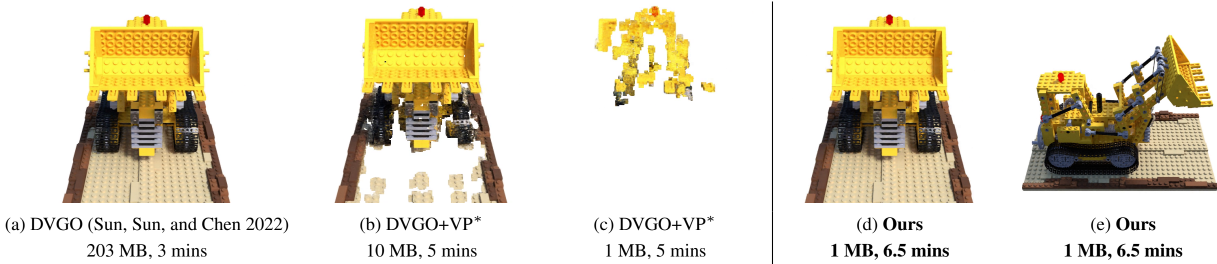

Figure 1: Comparison of model size and convergence time between our method and other variants, the training time is measured on one single NVIDIA A100 GPU. VP denotes voxel pruning (Liu et al. 2020) reimplemented based on DVGO, which removes low-density voxels. Significant voxel pruning (c) typically leads to unacceptable degradation in synthesizing quality. In contrast, our method (d), (e) can largely reduce the model size with minimal influence on synthesizing quality and training/inference speed.

formed grids by removing redundant DCT coefficients and retaining only the most informative coefficients. Next, we quantize the previously pruned coefficients by representing the non-zero floating-point values with low-bit integers multiplied by a single floating-point scalar. As a result, the required storage of the model is largely reduced. During inference, we recover the original grids with inverse discrete cosine transformation (IDCT) and apply the common volume rendering. We empirically show that the additional IDCT has only little effect on the rendering speed.

Our method can compress the voxel grids by more than $100\times$ with minimal sacrifice on rendering quality and speed. For example, we build our codes based on the recent state-of-the-art voxel grid based NeRF implementation DVGO (Sun, Sun, and Chen 2022)[1], the model size can be significantly reduced from $200MB$ to $2MB$, while the training time only grows from 3 minutes to 6.5 minutes on a single NVIDIA A100 GPU with only 0.2 degradation in PSNR and 0.003 degradation in SSIM on the synthetic nerf dataset.

2 Preliminaries

Neural radiance fields. To synthesize the image of a 3D object in novel views, NeRF represents the properties of the 3D scene implicitly into an MLP, which takes the location coordinate $\boldsymbol{p} \in \mathbb{R}^3$ and the viewing direction $\boldsymbol{d} \in \mathbb{R}^2$ as inputs, and outputs the corresponding density values $\sigma \in \mathbb{R}$ and colors $\boldsymbol{c} \in \mathbb{R}^3$:

$$\{\sigma(\boldsymbol{p}), \boldsymbol{\nu}(\boldsymbol{p})\} = f_{\boldsymbol{\theta}}(\boldsymbol{p}), \quad \boldsymbol{c}(\boldsymbol{p}, \boldsymbol{d}) = g_{\boldsymbol{\phi}}(\boldsymbol{\nu}(\boldsymbol{p}), \boldsymbol{d}) \tag{1}$$

where $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$ are the parameters of the MLP. The rendered pixel value $\hat{\boldsymbol{C}}(\boldsymbol{r})$ of the camera ray $\boldsymbol{r} = (\boldsymbol{p_0}, \boldsymbol{d})$ can be computed using the principles of the traditional volume rendering (Kajiya and Hersen 1984):

$$\hat{\boldsymbol{C}}(\boldsymbol{r}) = \int_0^{+\infty} T(t)\sigma(\boldsymbol{p}(t))\boldsymbol{c}(\boldsymbol{p}(t), \boldsymbol{d})dt \tag{2}$$

where $\boldsymbol{p_0}$ and $\boldsymbol{d}$ are the origin and the direction of the camera ray, respectively, $\boldsymbol{p}(t) = \boldsymbol{p}_0 + t\boldsymbol{d}^2$, and $T(t) = \exp\left[-\int_0^t \sigma(\boldsymbol{p}(s))ds\right]$ is the probability that a beam of light emitting from the origin $\boldsymbol{p_0}$ along the direction $\boldsymbol{d}$ hits the point $\boldsymbol{p}(t)$. Note that the volume density σ is restricted to be only dependent on the location $\boldsymbol{p}$ to encourage the scene representation to be multiview consistent, while the color $\boldsymbol{c}$ is dependent on both location $\boldsymbol{p}$ and direction $\boldsymbol{d}$ to model view-dependent scene. In practice, the intractable integral in Eq. (2) can be approximated by aggregating the densities and colors of densely sampled points along the ray (Max 1995; Mildenhall et al. 2020):

$$\hat{\boldsymbol{C}}(\boldsymbol{r}) \approx \sum_{i=1}^{N} T_i \alpha_i \boldsymbol{c_i} \tag{3}$$

where $\alpha_i = 1 - \exp(-\sigma_i \delta_i)$, $T_i = \prod_{j=1}^{i-1}(1 - \alpha_i)$. σ_i and $\boldsymbol{c}_i$ are the density and color of the ith sampled point, respectively, and δ_i is the distance between the ith sampled point and the next sampled point.

The rendering process of Eq. 3 is fully differentiable, so the parameters of the network can be optimized end-to-end

[1] https://github.com/sunset1995/DirectVoxGO

[2] In practice, the direction $\boldsymbol{d}$ is often represented by a unit 3-d vector

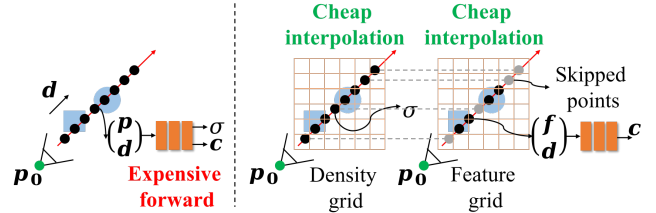

Figure 2: The comparison between the original NeRF (left) and the voxel grid optimization based NeRF (right). In the original NeRF, to render the pixel value of a camera ray $\boldsymbol{r} = (\boldsymbol{p_0}, \boldsymbol{d})$, the model must apply the expensive forward computation of an MLP on hundreds of sampled points along the ray to get their density values and colors. The voxel grid optimization-based methods explicitly store the position-dependent density values and scene feature embeddings into voxel grids. This design allows the model to skip the low-density points by efficiently querying the density grid with trilinear interpolation, leading to significant acceleration in training and rendering.

by minimizing the following loss function:

$$\mathcal{L} = \frac{1}{|\mathcal{R}|} \sum_{\boldsymbol{r} \in \mathcal{R}} \|\hat{\boldsymbol{C}}(\boldsymbol{r}) - \boldsymbol{C}(\boldsymbol{r})\|^2 \tag{4}$$

where $\mathcal{R}$ is the set of camera rays sampled in a mini batch, and $\boldsymbol{C}(\boldsymbol{r})$ is the true pixel value corresponding to the camera ray $\boldsymbol{r}$.

Voxel grid optimization. Despite its high rendering quality, the original NeRF requires heavy computation cost, because Eq. (3) typically needs to perform the forward of an MLP hundreds of times to render a single pixel. Voxel grid optimization based methods solve this problem by storing and optimizing position-dependent modalities of interest (*e.g.* density $\sigma(\boldsymbol{p})$, features $\boldsymbol{\nu}(\boldsymbol{p})$ in Eq. (1)) explicitly into voxel grid cells, and query their value at any position $\boldsymbol{p}$ via interpolation:

$$\sigma(\boldsymbol{p}) = a(\text{interp}(\boldsymbol{p}, \mathcal{V}^{(\sigma)})), \quad \boldsymbol{\nu}(\boldsymbol{p}) = \text{interp}(\boldsymbol{p}, \mathcal{V}^{(\nu)}) \tag{5}$$

where $\boldsymbol{p}$ is the queried location, and $\mathcal{V}^{(\sigma)} \in \mathbb{R}^{1 \times N_x \times N_y \times N_z}$ and $\mathcal{V}^{(\nu)} \in \mathbb{R}^{C \times N_x \times N_y \times N_z}$ are the voxel grids corresponding to the density and features, respectively. C is the dimension of the position-dependent feature, and N_x, N_y, N_z are the number of voxels along the three dimensions. $a(\cdot)$ is the density activation function such as shifted softplus (Sun, Sun, and Chen 2022). Note that the interpolation operation in Eq. (5) can be performed efficiently, which allows the model to directly skip low-density or occluded points during rendering:

$$\hat{\boldsymbol{C}}(\boldsymbol{r}) \approx \sum_{i \in \Omega(\boldsymbol{r})} T_i \alpha_i \boldsymbol{c}_i \tag{6}$$

where $\Omega(\boldsymbol{r}) = \{i : \alpha_i > \tau_1, T_i > \tau_2\}$. τ_1 and τ_2 are two user defined thresholds. $T_i, \alpha_i, \boldsymbol{c}_i$ are all the same as defined in Eqs. (1,3). In this way, the required number of MLP forward is largely reduced, resulting in significant acceleration. A graphical illustration of this process is shown in the right of Fig. 2.

Limitations of existing methods. The voxel grid optimization based methods accelerate the original NeRF by preventing the rendering of points without scene content. However, these methods achieve the acceleration at the expense of extra storage demand. The key components of these methods are the two pre-stored 3D grids which allow efficient querying via trilinear interpolation while requiring a large number of storage resources. Our main insight is that most visual signals in our real life are smooth in the spatial domain, which makes them tend to be sparse in the frequency domain. Thus, we transform the original voxel grids from spatial domain to the frequency domain with DCT and apply pruning and quantization to their DCT coefficients to reduce the storage demand. Although previous methods such as NSVF (Liu et al. 2020), Plenoxels (Yu et al. 2022) and DVGO (Sun, Sun, and Chen 2022) all prune empty voxels with no scene contents, while the pruning rate is still limited because of the large foreground-background ratio. Besides, the training time of NSVF is still too long because of the deep MLP in its scene representation.

3 Method

In this section, we introduce the proposed TinyNeRF. Before going into more details, we give a brief overview of our pipeline, which is illustrated in Fig. 3. We first transform the original voxel grid values to the frequency domain by expressing them with the combination of cosine basis wave functions oscillating at different frequencies via block-wise discrete cosine transformation (DCT) (the top left of the figure, Sec. 3.1). After the transformation, most of the signals information will be centralized in the low-frequency region. We next apply pruning (middle left at the top of the figure, Sec. 3.2) and quantization (middle right at the top of the figure, Sec. 3.3) to the transformed grids by removing less informative coefficients with lower magnitudes and representing the remained values with low-bit integers. In this way, we generate a rather sparse and low-bit model which can be stored in the disk or transferred online with little storage and network flow. The original grid values can be efficiently recovered from the compressed coefficients via block-wise inverse discrete cosine transformation (IDCT) to perform the following rendering (the bottom of the figure). The whole framework can be optimized in an end-to-end manner, and can be easily implemented with only little modification to the existing voxel grid optimization-based implementations.

3.1 Voxel Grids with Cosine Basis

Discrete Cosine Transformation. The first step of our method is to transform the original voxel grids to the frequency domain. Here we utilize the discrete cosine transformation (DCT) which is a Fourier-related transformation and has been widely used in image and video compression (Wallace 1992; Wiegand et al. 2003) because of its high degree of spectral compaction. For a 3D grid $\mathcal{V} \in \mathbb{R}^{C \times B_x \times B_y \times B_z}$, where C is the dimension of the feature stored in each voxel, and $B_x \times B_y \times B_z$ the shape of the grid. The discrete cosine transformation (DCT) and inverse discrete cosine transformation (IDCT) can be defined by:

$$\begin{aligned}
\bar{\mathcal{V}}_{c,i,j,k} &= \sum_{x,y,z} \frac{u_i u_j u_k}{\sqrt{B_x B_y B_z}} \mathcal{V}_{c,i,j,k} \times \\
&\quad \cos(\frac{\pi}{B_x} i(x+\frac{1}{2})) \cos(\frac{\pi}{B_y} j(y+\frac{1}{2})) \cos(\frac{\pi}{B_z} k(z+\frac{1}{2})) \\
\mathcal{V}_{c,x,y,z} &= \sum_{i,j,k} \frac{u_i u_j u_k}{\sqrt{B_x B_y B_z}} \bar{\mathcal{V}}_{c,i,j,k} \times \\
&\quad \cos(\frac{\pi}{B_x} i(x+\frac{1}{2})) \cos(\frac{\pi}{B_y} j(y+\frac{1}{2})) \cos(\frac{\pi}{B_z} k(z+\frac{1}{2}))
\end{aligned} \tag{7}$$

where x, y, z and i, j, k span from $[1,1,1]$ to $[B_x, B_y, B_z]$, and u_i is 1 at $i = 0$ and $\sqrt{2}$ otherwise. $\bar{\mathcal{V}} \in \mathbb{R}^{C \times B_x \times B_y \times B_z}$ is the transformed coefficients grid. In the remaining of this paper, we use the symbols with overbar to denote the DCT coefficients of the corresponding grids.

Signals Sparsity in the DCT Domain. In the view of signal decomposition, the transformation defined in Eq. (7) can be seen as factorizing the original grid $\mathcal{V}$ with cosine basis functions, where $\bar{\mathcal{V}}$ are the magnitudes corresponding to

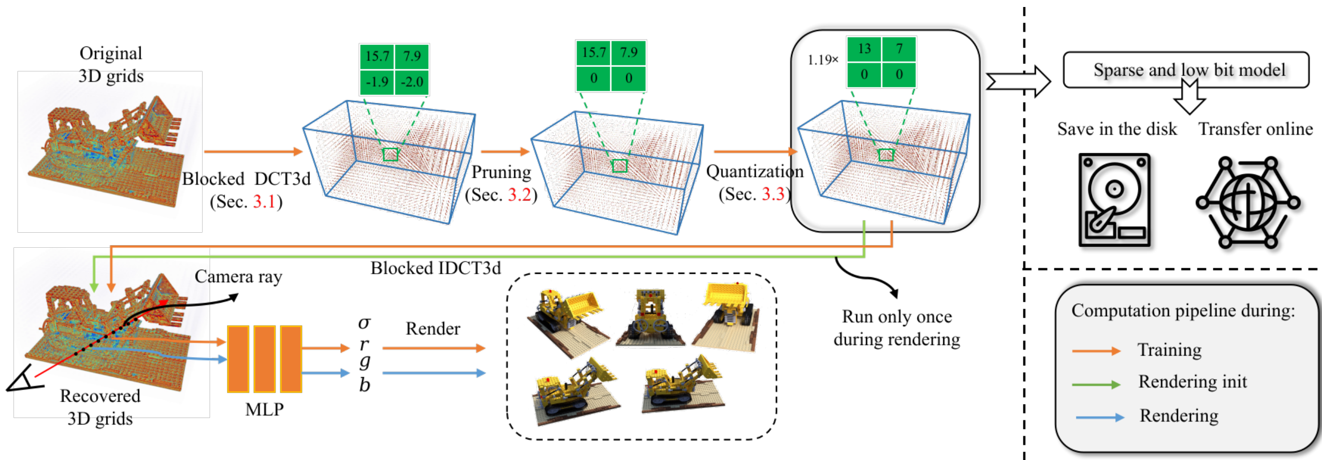

Figure 3: The overview framework of our method. Our method is inspired by the fact that most 3D scenes in our true life are smooth in the spatial field, so they tend to be sparse in the frequency field. We apply pruning and quantization to the DCT coefficients of the learnable scene properties stored in the 3D voxel grids, resulting in a rather sparse and low-bit model which requires little storage demand.

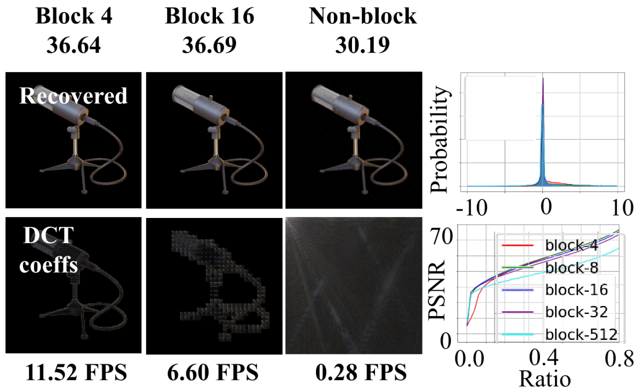

Figure 4: A toy example for the illustration of signals sparsity in the frequency domain. The running speed is measured on a single Intel Core i7-8700 CPU.

different frequencies. The high-frequency components are related to part of the original signals oscillating wildly in the spatial domain, such as edges. However, for most visual signals in the real life, edges are very sparse and nearby locations often share similar properties. Thus, after transformation, most information will be concentrated in the low-frequency region. As a result, the transformed coefficients $\bar{\mathcal{V}}$ will be rather sparse and can be roughly compressed with negligible information loss.

In Fig. 4 we further illustrate this with a toy example, where we randomly collect 100 images and report the distribution of their DCT coefficients. We find that most of the coefficients are centralized to zero after transformation. The left of the figure shows that the original image can be recovered with high quality even when 98% of the DCT coefficients are removed.

Block-wise DCT. In practice, during transformation, instead of applying DCT to the whole grid, we divide the voxels into blocks and apply block-wise DCT. Specifically, for a 3D grid $\mathcal{V} \in \mathbb{R}^{C \times N_x \times N_y \times N_z}$ we first divide the grid into $\frac{N_x}{B_x} \times \frac{N_y}{B_y} \times \frac{N_z}{B_z}$ blocks of sub-grids, each with size $C \times B_x \times B_y \times B_z$, where B_x, B_y, B_z are the block size along the three dimensions, and apply the DCT transformation to each of these blocks independently.

Several advantages motivate us to apply block-wise transformation. First, block-wise transformation requires less computation and thus is faster as shown in the left of Fig. 4. We show in the appendix that the number of required multiply-accumulate operations grows with the block-size. Second, block-wise transformation leads to less numerical errors as shown in the bottom right of the Fig. 4. Theoretically, the transformation defined in Eq. (7) is strictly accurate, while in practice, its computation suffers truncation errors because of the limited floating point precision in computers. As the block size grows, the required number of floating point operations grows, and so do the numerical errors. Third, the block-wise transformed coefficients are also concentrated around zero as shown in the top right of Fig. 4.

3.2 Pruning

After transformation, most of the signals information will be centralized on the low-frequency region as shown in the middle left at the top of Fig. 3. We then prune the transformed coefficients by enforcing the low-magnitude coefficients to be zero and only retaining the more informative ones. In other words, we sort the coefficients in descending order according to their absolute values and only retain the top K coefficients, where K is the expected number of retained coefficients.

We denote $\bar{\mathcal{V}}_P$ the pruned DCT coefficients. During the *pruning aware training*, we apply the volume rendering with the approximately recovered grid $\mathcal{V}_P = \text{IDCT}(\bar{\mathcal{V}}_P)$ and compute the gradients *w.r.t.* the learnable grid with straight

through estimator (STE) (Bengio, Leonard, and Courville 2013), which can be formulated by the following equation:

$$\frac{\partial \mathcal{L}}{\partial \mathcal{V}} \approx \frac{\partial \mathcal{L}}{\partial \mathcal{V}_P} \text{ where } \mathcal{V}_P = \text{IDCT}(\bar{\mathcal{V}}_P) \tag{8}$$

3.3 Quantization

After applying pruning aware training for several iterations, we quantize the pruned coefficients to further improve the compression by reducing the number of bits required to store each non-zero value. Specifically, let $\bar{\mathcal{V}}_P$ be the DCT coefficients after pruning, we quantize the retained values in $\bar{\mathcal{V}}_P$ by approximating them with low-bit integers multiplied by a single floating point scalar: $\bar{\mathcal{V}}_P \approx \alpha\bar{\mathcal{V}}_Q$, where $\alpha \in \mathbb{R}$, $\bar{\mathcal{V}}_Q$ is the quantized coefficients, in which each value is represented by a low bit integer. At the initialization of the quantization aware training, the values of α and $\bar{\mathcal{V}}_Q$ can be approximated by iterating the following two equations alternatively (Choukroun et al. 2019; Wang et al. 2019):

$$\bar{\mathcal{V}}_Q = \text{clip}([\frac{\bar{\mathcal{V}}_P}{\alpha}], Q_{min}, Q_{max}) \tag{9a}$$

$$\alpha = \frac{<\bar{\mathcal{V}}_P, \bar{\mathcal{V}}_Q>}{<\bar{\mathcal{V}}_Q, \bar{\mathcal{V}}_Q>} \tag{9b}$$

where $[\cdot]$ is the round operator, Q_{min} and Q_{max} are the minimum and maximum quantized values. For example, for b-bit quantization, they can be computed by: $Q_{min} = -2^{b-1}, Q_{max} = 2^{b-1} - 1$. $<\cdot,\cdot>$ denotes the inner product operator defined by summing up all the values in the element-wise multiplication of the two grids. Please refer to the Appendix for a detailed derivation of the above alternating optimization algorithm.

After the above initialization, we freeze the α and further refine the learnable voxels $\mathcal{V}$. We call this process the quantization aware training. Similarly, during the quantization aware training, we compute $\bar{\mathcal{V}}_Q$ with Eq. (9a), render the sampled camera rays with the recovered grid $\mathcal{V}_Q = \text{IDCT}(\alpha\bar{\mathcal{V}}_Q)$, and the gradients *w.r.t.* the original learnable voxel grids $\mathcal{V}$ are approximated with STE (Bengio, Leonard, and Courville 2013):

$$\frac{\partial \mathcal{L}}{\partial \mathcal{V}} \approx \frac{\partial \mathcal{L}}{\partial \mathcal{V}_Q}, \text{ where } \mathcal{V}_Q = \text{IDCT}(\alpha\bar{\mathcal{V}}_Q) \tag{10}$$

In this way, the whole framework can be optimized in an end-to-end manner.

4 Experimental Results

4.1 Experimental Setup

We implement our method in PyTorch with the block-wise DCT implemented in CUDA. The codes are built based on the recent state-of-the-art voxel grid based NeRF implementation DVGO (Sun, Sun, and Chen 2022)[3]. By default, the block size for the block-wise DCT is set to $4 \times 4 \times 4$ because we find it achieves good trade-off between compression, synthesizing quality and training speed. We keep all the hyper parameters the same with DVGO (Sun, Sun, and Chen 2022) for fair comparison. The grid resolutions for all the scenes are set to 160^3. The pruning and quantization are only enabled during the fine-stage training. We keep the total optimization iterations to be 20000 with 8192 camera rays per batch, where the pruning aware training is enabled after 5000 iterations of common training, and the quantization aware training is further enabled after 12000 iterations. The whole training process typically finishes in less than 10 minutes. The detailed pruning ratios and quantization bit widths for different target model sizes are shown in Tab. 1.

[3] https://github.com/sunset1995/DirectVoxGO

Model size	Density Ratio	Density Bit-width	Feature Ratio	Feature Bit-width
10MB	0.4	24	0.1	8
4MB	0.3	10	0.07	6
2MB	0.3	8	0.03	4
1.3MB	0.09	8	0.03	4
1MB	0.08	8	0.03	4

Table 1: Pruning ratios and quantization bit-widths for different model sizes.

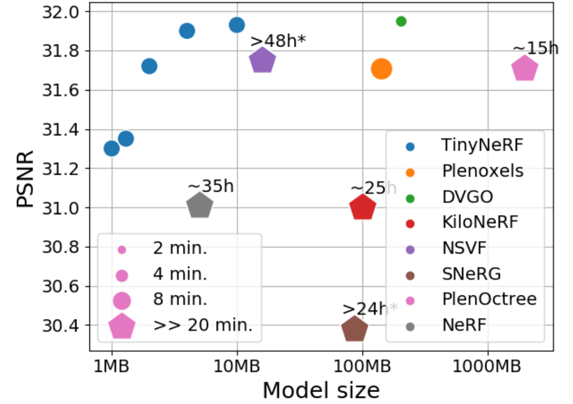

Figure 5: Comparison of model size & PSNR between TinyNeRF and state-of-the-art baseline NeRF methods. The size of each dot indicates the training time.

4.2 Comparisons

We evaluate our method on five inward-facing datasets, including Synthetic-NeRF (Mildenhall et al. 2020) which contains 8 objects with relistic images, Synthetic-NSVF (Liu et al. 2020) which contains 8 objects synthesized by NSVF, BlendedMVS (Yao et al. 2020) with realistic ambient lighting from real image blending, DeepVoxels (Sitzmann et al. 2019) with 4 Lambertian objects, and a real world data set Tanks&Temples (Knapitsch et al. 2017). Quantitative results are shown in Tab. 2, and the results on the NeRF Synthetic data set are plotted in Fig. 5. The actual training time of each methods are also shown in the figure. Comparing the dots in Fig. 5, our method achieves a better trade-off between model size and rendering quality. Comparing the sizes of the dots in the figure, our method has only little influence on the

Methods	Synthetic NeRF			Synthetic-NSVF			Blended MVS			T&T		
	PSNR↑	SSIM↑	#params (MB)↓	PSNR↑	SSIM↑	#params (MB)↓	PSNR↑	SSIM↑	#params (MB)↓	PSNR↑	SSIM↑	#params (MB)↓
Methods taking hours to days to train												
NeRF	31.01	0.947	5	30.81	0.952	5	24.15	0.828	5	25.78	0.864	5
PlenOctree	31.71	0.958	1976	-	-	-	-	-	-	27.99	0.917	1976
SNeRG	30.38	0.95	87	-	-	-	-	-	-	-	-	-
FastNeRF	29.97	0.941	-	-	-	-	-	-	-	-	-	-
AutoInt	25.55	0.91	-	-	-	-	-	-	-	-	-	-
NSVF	31.75	0.953	3.2∼16	35.18	0.979	3.2∼16	26.89	0.898	3.2∼16	28.48	0.901	3.2∼16
KiloNeRF	31.00	0.95	∼100	33.37	0.97	∼100	27.39	0.92	∼100	28.41	0.91	∼100
Methods taking minutes to train												
Plenoxels	31.71	0.958	143	-	-	-	-	-	-	20.40	0.696	143
DVGO	31.95	0.957	203	35.08	0.975	203	28.02	0.922	203	28.41	0.911	203
TinyNeRF(10MB)	31.93	0.956	10	34.95	0.975	10	28.15	0.922	10	28.33	0.911	10
TinyNeRF(4MB)	31.90	0.956	4	34.88	0.975	4	28.11	0.922	4	28.32	0.91	4
TinyNeRF(2MB)	31.72	0.954	2	34.62	0.968	2	28.02	0.919	2	28.19	0.908	2

Table 2: Comparisons on model size, training time and synthesizing quality with state-of-the-art NeRF methods.

training time, typically converging in less than 8 minutes. We refer the readers to the Appendix for detailed per-scene comparisons and the results on the DeepVoxels dataset.

4.3 Ablation Study

In this section, we study the influence of different components in our method, including compression in the frequency domain, utilizing block-wise transformation. We further study the sensitivity of our method to different choices of hyper parameters, including transformation block sizes, pruning ratios and quantization bit widths. To this end, we implement different variants of our method and compare them on the synthetic nerf data set. We report the main results in Fig. 6 and Tab. 3, and detailed quantitative results are given in the Appendix.

Pruning in the Spatial v.s. Frequency Domain. To study the influence of frequency domain compression, we implement a variant of TinyNeRF which applies pruning and quantization in the spatial domain, named TinyNeRF-S. Specifically, for the density grid, we prune the voxels with low density, and for the feature grid, we enforce the low-magnitude feature values to be zero. Besides, we also quantize the retained density and features with the same method described in Sec. 3.3. We keep all the other hyper parameters the same with our TinyNeRF. Results are shown in Fig. 6.

Our TinyNeRF (the red dots) consistently outperforms TinyNeRF-S by a large margin. For example, when compressing the original model to 2MB, TinyNeRF achieves 0.5 higher PSNR than TinyNeRF-S. The improvements are even near or more than 2db under the model size of 1MB and 1.3MB, respectively. In general, the advantages of frequency-domain compression are more obvious on smaller model sizes. These results support that it is better to compress the voxel grids in their frequency domain.

Influence of Block-wise DCT. Another key design of our method is to apply compression to the block-wise transformation of the original grids. To study the influence of block-wise DCT, we further implement a variant of TinyNeRF which performs DCT to the whole voxel grids while keeping all the other components the same with our method. The results are shown in Fig. 6 labeled by TinyNeRF-w/o block.

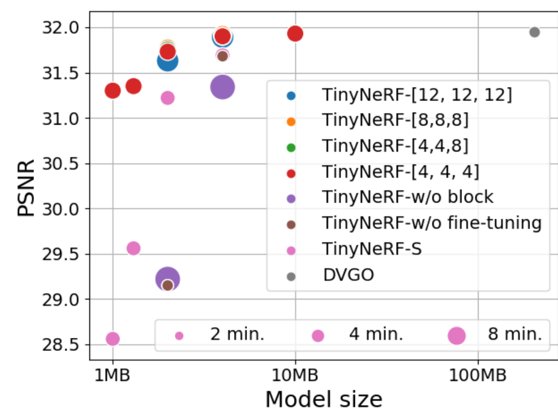

Figure 6: Comparison of model size & PSNR between TinyNeRF and its other variants. The size of each dot indicates the training time. Here, the text TinyNerf-$[x, y, z]$ means TinyNeRF with the DCT block size of $x \times y \times z$. TinyNeRF-w/o block means applying the DCT to the whole grids without any blocking. TinyNeRF-w/o fine-tuning means directly transforming, pruning and quantizing the grid coefficients without any retraining. TinyNeRF-S means pruning and quantizing the grid values in the spatial domain instead of frequency domain.

We see from the figure that (1) Without block-wise DCT, compression in the frequency domain performs even worse than compressing in the spatial domain. This result implies that naively applying the compression in the frequency domain doesn't result in satisfactory results. (2) With block-wise DCT, TinyNeRF achieves much better trade-off be-

Density grid	Ratio	0.3	0.15	0.3	0.15
	Bit width	8	16	8	16
Feature grid	Ratio	0.03	0.03	0.015	0.015
	Bit width	4	4	8	8
PSNR		31.72	31.59	31.59	31.54

Table 3: Ablation on the sensitivity to hyper parameters on the synthetic nerf data set. Ratio denotes the ratio of non-zero elements for pruning.

tween model size and synthesizing quality. (3) By comparing the sizes of the dots, we conclude that TinyNeRF with smaller block-size converge faster than TinyNeRF without block-wise DCT. All these results support that block-wise DCT requires fewer multiply-accumulate operations, and thus can be computed faster and suffers fewer truncation errors caused by the limited precision in computers.

Sensitivity to Hyper-parameters. There are some hyper parameters that may affect the effectiveness of our method, such as block sizes, pruning ratios, and quantization bit widths. To study their influences, we compare TinyNeRF with 1) different block sizes which is shown in Fig. 6, and 2) different pruning ratios and quantization bit widths which is shown in Tab. 3.

From Fig. 6, we see that the performance gaps between TinyNeRF with different DCT block-sizes are rather small. These results support that our method is not sensitive to the choice of DCT block size, as long as the block size is relatively small.

To study the sensitivity of our method to different configurations of pruning ratios and quantization bit widths, we adjust the pruning ratios and quantization bit widths of both the density grid and feature grid to compress the model to 2 MB. The detailed configurations and results on the synthetic nerf data set are shown in Tab. 3. We see from the table that TinyNeRF with different compression configurations perform very similarly, which indicates that our method is also not sensitive to different pruning ratios and quantization bit widths.

Compression without Fine-tuning. Interestingly, our method can also be used to compress a trained voxel grid based NeRF model *without any fine tuning*. Specifically, for a pre-trained NeRF model with voxel grids, we can directly transform the grids with block-wise DCT and apply pruning and quantization to the transformed coefficients without refining the compressed grids. Note that once the NeRF model is trained, this compression process can complete in seconds (most of the time is spared on the iterative algorithm to compute the scaling factor for quantization), thus the required time of this method is dominated by the pre-training process. To illustrate this, we compress the pre-trained DVGO (Sun, Sun, and Chen 2022) models on 8 scenes of the synthetic nerf data set, and the results are shown in Fig. 6 labled by "TinyNeRF-w/o fine tuning". The DVGO can converge in 3 minutes per scene, and original model size is 203 MB. We see from the figure that even without fine-tuning, our method can still compress the original model by $50\times$ with little sacrifice on PSNR ($31.95 \rightarrow 31.68$).

Timing Analysis. Our method requires additional computations such as DCT, IDCT, pruning, quantization, dequantization during both training and inference. A common concern about our method may be: How do these additional processes affect the training and inference time of the original models? To investigate this, we plot in the following figure the proportion of time spent on the additional computations during both training and inference. The time of all the processes are normalized to the total run time, and the detailed numerical results are shown in the Appendix.

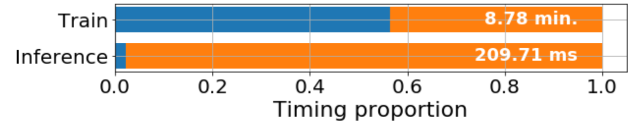

Figure 7: Timing analysis for TinyNeRF.

In the figure, the blue bars denote the run time of the additional computations, and the orange bars denote the run time of the original models. We see from the figure that the additional computations contribute very little to the inference time (5 ms / 209 ms). Interestingly, we find the during training, almost all of the additional run time are consumed in the pruning process (3.97 min. / 4.96 min.). Fortunately, this expensive operation is not required in the inference process. As a result, the additional run time of the inference time is far less than that of the training time.

5 Discussions and Limitations

In this paper, we present TinyNeRF, a method with three-step pipeline: frequency domain transformation, pruning and quantization that work together to reduce the disk storage demand for voxel radiance fields. Thanks to the priori of signals sparsity in the frequency domain, our method can typically compress the voxel grids by 2 orders of magnitude with little effect on synthesizing quality.

Although achieving high compression, there are still some limitations in our method that may potentially be very interesting future directions. First, our method needs to recover the original voxel grids during rendering. Thus, the *runtime* computation and memory requirements are not reduced. An interesting improvement is to avoid the runtime grid recovering to further reduce its memory demands during inference. Second, in this paper, we only show the usage of our method in voxel grid based NeRF. It is interesting to consider whether it is also applicable in grid-based solutions to other problems such as 3D surface reconstruction. Third, there are some other transformations that also lead to sparse coefficients. It is worth considering that are there any other choices of frequency domain transformations that are more suitable to voxel grids compression.

Acknowledgements

This work was supported in part by the National Key Research and Development Program of China under Grant

2021ZD0201504, National Natural Science Foundation of China under Grant 61972396.

References

Barron, J. T.; Mildenhall, B.; Tancik, M.; Hedman, P.; Martin-Brualla, R.; and Srinivasan, P. P. 2021. Mip-NeRF: A Multiscale Representation for Anti-Aliasing Neural Radiance Fields. In *International Conference on Computer Vision, (ICCV)*.

Bengio, Y.; Leonard, N.; and Courville, A. 2013. Estimating or Propagating Gradients Through Stochastic Neurons for Conditional Computation. *aeXiv:1308.3432*.

Choukroun, Y.; Eravchik, E.; Yang, F.; and Kisilev, P. 2019. Low-bit Quantization of Neural Networks for Efficient Inference. In *International Conference on Computer Vision, (ICCV)*.

Garbin, S. J.; Kowalski, M.; Johnson, M.; Shotton, J.; and Valentin, J. 2021. FastNeRF: High-Fidelity Neural Rendering at 200FPS. In *International Conference on Computer Vision, (ICCV)*.

Hedman, P.; Srinivasan, P. P.; Mildenhall, B.; Barron, J. T.; and Debevec, P. 2021. Baking Neural Radiance Fields for Real-Time View Synthesis. In *International Conference on Computer Vision, (ICCV)*.

Kajiya, J. T.; and Hersen, B. P. V. 1984. Ray Tracing Volume Densities. *Computer Graphics*, 18(3).

Kangle, D.; Andrew, L.; Jun-Yan, Z.; and Deva, R. 2022. Depth-supervised NeRF: Fewer Views and Faster Training for Free. In *IEEE Conference on Computer Vision and Pattern Recognition, (CVPR)*.

Knapitsch, A.; Park, J.; Zhou, Q.-Y.; and Koltun, V. 2017. Tanks and temples: Benchmarking large-scale scene resconstruction. *ACM Transactions on Graphics*, 36(4).

Laine, S.; and Karras, T. 2010. Efficient Sparse Voxel Octrees. In *ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games, (I3D)*.

Lefebvre, S.; and Hoppe, H. 2006. Perfect spatial hashing. *ACM Transactions on Graphics*, 25.

Lindell, D. B.; Martel, J. N.; and Wetzstein, G. 2021. AutoInt: Automatic Integration for Fast Neural Volume Rendering. In *IEEE Conference on Computer Vision and Pattern Recognition, (CVPR)*.

Liu, L.; Gu, J.; Lin, K. Z.; Chua, T.-S.; and Theobalt, C. 2020. Neural Sparse Voxel Fields. In *International Conference on Neural Information Processing Systems, (NeurIPS)*.

Liu, Y.; Peng, S.; Liu, L.; Wang, Q.; Wang, P.; Theoboalt, C.; Zhou, X.; and Wang, W. 2022. Neural Rays for Occlusion-aware Image-based Rendering. In *IEEE Conference on Computer Vision and Pattern Recognition, (CVPR)*.

Max, N. 1995. Optical models for direct volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 1.

Mildenhall, B.; Srinivasan, P. P.; Barron, J. T.; Ramamoorthi, R.; and Ng, R. 2020. Representing Scences as Neural Radiance Fields for View Synthesis. In *European Conference on Computer Vision, (ECCV)*.

Niebner, M.; Zollhofer, M.; Izadi, S.; and Stamminger, M. 2013. Readl-time 3D reconstruction at scale using voxel hashing. *ACM Transactions on Graphics*, 32(6).

Reiser, C.; Peng, S.; Liao, Y.; and Geiger, A. 2021. KiloNeRF: Speeding up Neural Radiance Fields with Thousands of Tiny MLPs. In *International Conference on Computer Vision, (ICCV)*.

Sitzmann, V.; Martel, J. N.; Bergman, A. W.; Lindell, D. B.; and Wetzstein, G. 2020. Implicit Neural Representation with Periodic Activation Functions. In *International Conference on Neural Information Processing Systems, (NeurIPS)*.

Sitzmann, V.; Thies, J.; Heide, F.; Niebner, M.; Wetzstein, G.; and Zollhofer, M. 2019. DeepVoxels: Learning Persistent 3D Feature Embeddings. In *IEEE Conference on Computer Vision and Pattern Recognition, (CVPR)*.

Sun, C.; Sun, M.; and Chen, H.-T. 2022. Direct Voxel Grid Optimization: Super-Fast Convergence for Radiance Fields Reconstruction. In *IEEE Conference on Computer Vision and Pattern Recognition, (CVPR)*.

Tancik, M.; Casser, V.; Yan, X.; Pradhan, S.; Mildenhall, B.; Srinivasan, P. P.; Barron, J. T.; and Kretzschmar, H. 2022. Block-NeRF: Scalable Large Scene Neural View Synthesis. *arXiv:2202.05263*.

Tancik, M.; Srinivasan, P. P.; Mildenhall, B.; Fridovich-Keil, S.; Raghavan, N.; Singhal, U.; Ramamoorthi, R.; Barron, J. T.; and Ng, R. 2020. Fourier Features Let Networks Learn High Frequency Functions in Low Dimensional Domains. In *International Conference on Neural Information Processing Systems, (NeurIPS)*.

Wallace, G. K. 1992. The JPEG Still Picture Compression Standard. *IEEE Transactions on Computer Eletronics*, 38(1).

Wang, P.; Hu, Q.; Zhang, Y.; Zhang, C.; Liu, Y.; and Cheng, J. 2019. Two-Step Quantization for Low-bit Neural Networks. In *IEEE Conference on Computer Vision and Pattern Recognition, (CVPR)*.

Wiegand, T.; Sullivan, G.; Bjontegaard, G.; and Luthra, A. 2003. Overview of the H.264/AVC Video Coding Standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 13.

Wizadwongsa, S.; Phongthawee, P.; and Yenphraphai, J. 2021. NeX: Real-time View Synthesis with Neural Basis Expansion. In *IEEE Conference on Computer Vision and Pattern Recognition, (CVPR)*.

Yao, Y.; Luo, Z.; Li, S.; Zhang, J.; Ren, Y.; Zhou, L.; Fang, T.; and Quan, L. 2020. BlendedMVS: A Large-Scale Dataset for Generalized Multi-view Stero Networks. In *IEEE Conference on Computer Vision and Pattern Recognition, (CVPR)*.

Yu, A.; Fridovich-Keil, S.; Tancik, M.; Chen, Q.; Recht, B.; and Kanazawa, A. 2022. Plenoxels: Radiance Fields without Neural Networks. In *IEEE Conference on Computer Vision and Pattern Recognition, (CVPR)*.

Yu, A.; Li, R.; Tancik, M.; Li, H.; Ng, R.; and Kanazawa, A. 2021. PlenOctrees for Real-time Rendering of Neural Radiance Fields. In *IEEE Conference on Computer Vision and Pattern Recognition, (CVPR)*.

Zhang, K.; Riegler, G.; Snavely, N.; and Koltun, V. 2020. NeRF++: Analyzing and Improving Neural Radiance Fields. *arXiv:2010.07492v2*.