

MIMO-NeRF: Fast Neural Rendering with Multi-input Multi-output Neural Radiance Fields

Takuhiro Kaneko
NTT Corporation

Abstract

Neural radiance fields (NeRFs) have shown impressive results for novel view synthesis. However, they depend on the repetitive use of a single-input single-output multilayer perceptron (SISO MLP) that maps 3D coordinates and view direction to the color and volume density in a sample-wise manner, which slows the rendering. We propose a multi-input multi-output NeRF (MIMO-NeRF) that reduces the number of MLPs running by replacing the SISO MLP with a MIMO MLP and conducting mappings in a group-wise manner. One notable challenge with this approach is that the color and volume density of each point can differ according to a choice of input coordinates in a group, which can lead to some notable ambiguity. We also propose a self-supervised learning method that regularizes the MIMO MLP with multiple fast reformulated MLPs to alleviate this ambiguity without using pretrained models. The results of a comprehensive experimental evaluation including comparative and ablation studies are presented to show that MIMO-NeRF obtains a good trade-off between speed and quality with a reasonable training time. We then demonstrate that MIMO-NeRF is compatible with and complementary to previous advancements in NeRFs by applying it to two representative fast NeRFs, i.e., a NeRF with sample reduction (DONeRF) and a NeRF with alternative representations (TensoRF).¹

1. Introduction

Images are two-dimensional (2D) projections of three-dimensional (3D) scenes. Solving the inverse problem, that is, learning 3D representations from 2D images and synthesizing novel views, is a fundamental concern in computer vision and graphics and has been extensively studied for various applications such as photo editing, content creation, virtual reality, and environmental understanding.

With the advent of implicit neural representations

¹The project page is available at <https://www.kecl.ntt.co.jp/people/kaneko.takuhiro/projects/mimo-nerf/>.

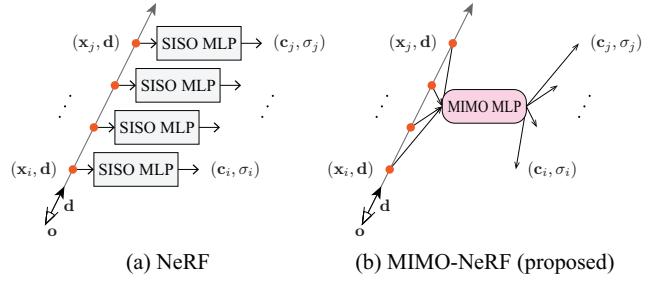


Figure 1. Comparison between NeRF and MIMO-NeRF (proposed). (a) A typical NeRF uses a SISO MLP that maps 3D coordinates and view direction to the color and volume density in a *sample-wise* manner. (b) In contrast, the proposed MIMO-NeRF uses a MIMO MLP that performs mappings in a *group-wise* manner. This change reduces the number of MLPs running and improves the rendering speed, but also requires addressing ambiguity in the color and volume density caused by the fact that these values are determined in a non-unique manner by a set of input coordinates that vary by viewpoint, grouping, and sampling. The main technical contribution of the present work is that of providing methods to mitigate this challenge. We demonstrate the impact of the proposed technique in Figure 2.

(e.g., [60, 47, 41, 30, 73, 75, 59]), substantial advancements have been made towards addressing this problem. Neural radiance fields (NeRFs) [41] have been noted as a successful approach. A NeRF represents a scene using a continuous function that maps 3D coordinates and view direction to the color and volume density and renders a pixel by integrating the outputs on a ray using volume rendering [36]. This formulation enables a NeRF to learn to synthesize geometrically consistent and high-fidelity novel views with only 2D supervision.

Despite this advantage, a typical NeRF suffers from slow rendering because it uses a *single-input single-output (SISO)* MLP that calculates the RGB color and volume density in a *sample-wise* manner (Figure 1(a)). Although this architecture ensures the independent representation of each point, which is useful, for example, for learning view-independent volume density, its computational cost increases in proportion to the number of samples for each

ray (e.g., on the order of hundreds). Several methods developed to address this issue can be roughly categorized into two approaches, including (1) *sample reduction* and (2) *alternative representations*.

A typical sample reduction strategy reduces the number of samples on a ray using a sampling network based on the depth [44] or density of a pretrained NeRF [50] or using a sampling network with an adaptive optimization mechanism [31, 14, 29]. These methods successfully accelerate the rendering process while retaining image quality adequately. However, most of these techniques still use a SISO MLP to predict the colors and volume densities of selected samples; therefore, they still need to run MLPs many times in proportion to the number of selected samples. This issue can be alleviated by reducing the number of selected samples, although this deteriorates the quality of the synthesized images accordingly.

As alternative representations, various sophisticated and faster representations such as 3D voxel grids [17, 22, 32, 70, 64], sparse voxel-based octrees [74, 15], multiplane images [69], tri-planes [7], vector-matrix decomposition [9], hashes [42], NeRF-specific structures [24], and space-wise MLPs [53, 54] have been devised. These representations contribute to achieving fast rendering while retaining image quality moderately well. However, after powerful features are extracted using alternative representations, SISO MLPs are still commonly used for the final prediction of color or volume density owing to their memory-efficient and continuous nature. Hence, part of the calculation cost still increases depending on the number of samples.

Consequently, owing to its compact, continuous (i.e., resolution-free), and independent (e.g., view-independent) nature, a SISO MLP is commonly used in various NeRFs. However, as mentioned above, the calculation cost increases with the number of samples. This is not preferable when considering the improvement in rendering speed. Possible simple solutions include, for example, a reduction of the number of samples or a reduction of the size of a model with a corresponding sacrifice of image quality. However, these solutions are not necessarily the best for handling the trade-off between quality and speed.² Alternatively, we propose a *multi-input multi-output NeRF (MIMO-NeRF)*, which is a novel variant of NeRF that represents a scene using a *MIMO* MLP that conducts mappings in a *group-wise* manner (Figure 1(b)). This modification enables a reduction in the number of MLPs running according to the number of grouped samples and consequently improves rendering speed.

However, in this approach, the uniqueness of the color and volume density of each point is not ensured because they are determined not only by the coordinates of the corresponding point but also by the coordinates of the other points in a group, which vary by viewpoint, grouping, and

²We discuss this trade-off in detail in Section 5.2.

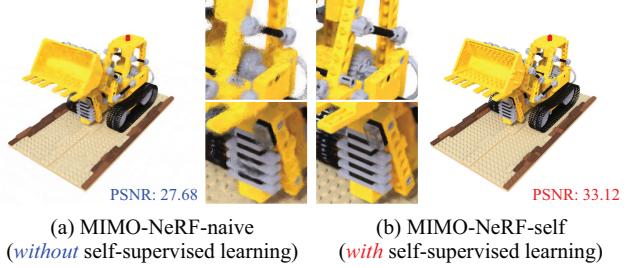


Figure 2. Challenge of training MIMO-NeRF and the impact of the proposed self-supervised learning. (a) MIMO-NeRF-naive suffers from ambiguity in the color and volume density of each point and deteriorates image quality. (b) We propose self-supervised learning to address this problem without relying on pretrained models. This mitigates fluctuation artifacts and improves image quality.

sampling. This leads to some *ambiguity* and causes fluctuation artifacts as shown in Figure 2(a). In particular, this ambiguity can be problematic when learning a 3D representation using only 2D supervision because obtaining direct supervision that can resolve the ambiguity is difficult. One possible solution is to train a standard (i.e., SISO) NeRF first and then distill the model onto the corresponding MIMO-NeRF. However, this increases training time because both student and teacher NeRFs must be trained. Alternatively, we have also developed a novel *self-supervised learning* approach in which we reformulate a MIMO MLP in several ways (in particular, we use *group shift* (Figure 3) and *variation reduction* (Figure 4)) and impose a consistent regularization so that the reformulated MIMO MLPs produce the same outputs. Because each reformulated MIMO MLP can render a pixel faster than the original SISO MLP, we can prevent a large sacrifice of training time even when using multiple reformulated MIMO MLPs by adequately adjusting the reformulation configuration. Figure 2(b) shows an example of the effects of this learning.

We investigated the benchmark performance of MIMO-NeRF by comparing it with possible alternatives (including the distillation of a pretrained NeRF, reduction of the number of samples, and reduction of the model size). We also performed ablation studies to examine the validity of each component of the proposed self-supervised learning method. We apply MiMO-NeRF to two representative fast NeRFs, including a NeRF with sample reduction (DONeRF [44]) and a NeRF with alternative representations (TensorRF [9]) to demonstrate that it is compatible with and complementary to previous advancements in NeRFs.

The main contributions of this study are summarized as follows.

- To speed up the rendering of NeRF, we propose *MIMO-NeRF*, which represents a scene using a *MIMO* MLP that maps the coordinates on a ray to the colors

and volume densities in a *group-wise* manner.

- We introduce novel *self-supervised learning* to mitigate the ambiguity in the color and volume density of each point and enable MIMO-NeRF to be trained without relying on pretrained models.
- We examined the effectiveness of MIMO-NeRF through a comprehensive experimental evaluation, and the results demonstrate the versatility of MIMO-NeRF in applications to two representative fast NeRFs. We also provide more detailed analyses and extended results in the Appendix A and on the project page.¹

2. Related work

Implicit neural representations. Implicit neural representations have attracted attention in 3D shape [48, 37, 11, 38, 55, 18, 3, 20] and 3D scene [25, 49, 6, 12, 58] reconstructions owing to their memory-efficient, continuous (i.e., resolution-free), and 3D-aware characteristics. Although research began with explicit 3D supervision, learning implicit 3D only from 2D supervision (i.e., inverse graphics) has also been achieved by incorporating differentiable rendering [60, 33, 47, 34, 41, 30, 73, 75, 59]. In this study, we focus on NeRFs as a representative example of the latter owing to their remarkable success in synthesizing geometrically consistent and high-quality novel view. However, applying our ideas to other implicit neural representations, such as those mentioned above, remains as an interesting direction for future research.

Advancements in NeRFs. Various extensions have been proposed since the emergence of NeRFs. For example, representative research topics include (1) improving image quality and enhancing applicable scenes (e.g., [76, 35, 52, 16, 71, 4, 5, 39, 66, 10]), (2) incorporation into other models, e.g., deep generative models, such as generative adversarial networks (GANs) [19] and diffusion probabilistic models [62, 23] (e.g., [56, 8, 46, 45, 21, 7, 13, 72, 26, 61, 51]), and (3) accelerating NeRFs for fast inference or fast training (e.g., [44, 50, 31, 14, 29, 17, 22, 32, 70, 64, 74, 15, 69, 7, 9, 42, 24, 53, 54]). The present work falls into the third category. However, our proposed approach is complementary to previous studies, including most of the above-mentioned works in all categories, because SISO MLPs have commonly been used as a partial or main network in previous studies and improving their rendering speed by replacing SISO MLPs with the proposed MIMO MLP is feasible. The results of the experimental evaluation validated this potential (Sections 5.4 and 5.5).

Acceleration of NeRFs. As discussed in Section 1, a typical NeRF is well known for its slow rendering because it uses a SISO MLP. Several approaches have been developed to address this issue. These can be roughly categorized into two approaches, including (1) *sample reduction* and

(2) *alternative representations*. (1) As described in Section 1, methods that reduce the number of samples using a sampling network can improve rendering speed by replacing a SISO MLP with the proposed MIMO MLP. We validated this statement during an experiment (Section 5.4) by applying our ideas to a representative NeRF in this category (DONeRF [44]). Another common approach in the first category is to render pixels using a light-field network [59, 2, 63, 67] instead of volume rendering [36]. This approach has been shown to achieve fast rendering by running only a single MLP for a given ray. However, owing to the lack of explicit geometry-aware representations driven by the use of volume densities, these methods suffer from limitations that are not faced by a standard NeRF in terms of restrictions on applicable scenes (e.g., toy datasets [59] and forward-facing datasets [2]), a requirement for high-capacity models (e.g., a deeper MLP [67] and a transformer [63]), and the need for extra modules (e.g., meta-learned priors [59], pretrained NeRFs [2, 67], or additional encoders [63]). (2) As explained in Section 1, alternative representations have the potential to accelerate the rendering speed by replacing the SISO MLP with the proposed MIMO MLP. We present an empirical investigation of this potential in Section 5.5 by incorporating MIMO-NeRF into TensoRF [9], a representative model in this category.

Learning of fast NeRFs. Knowledge distillation (or baking) methods are commonly used to train fast NeRFs. In these methods, a standard NeRF is first trained and then baked to faster representations [17, 22, 24, 53, 54]. However, this approach is disadvantageous in terms of training time because two separate models must be trained, i.e., teacher and student NeRFs. As an alternative, we consider a self-supervised learning approach in which we can train a model without a large increase in training time. We examine the performance differences between the proposed self-supervised learning scheme and a knowledge distillation scheme in Section 5.1.

3. Preliminaries: NeRF

We begin by explaining NeRFs as the basis for our model. As shown in Figure 1(a), a NeRF represents a point in a 3D space using a continuous SISO function f_{SISO} that maps the 3D position $\mathbf{x} \in \mathbb{R}^3$ and view direction $\mathbf{d} \in \mathbb{S}^2$ to the RGB color $\mathbf{c}(\mathbf{x}, \mathbf{d}) \in \mathbb{R}^3$ and volume density $\sigma(\mathbf{x}) \in \mathbb{R}^+$ in a sample-wise manner.

$$f_{\text{SISO}} : \mathbb{R}^3 \times \mathbb{S}^2 \rightarrow \mathbb{R}^3 \times \mathbb{R}^+, \quad (\mathbf{x}, \mathbf{d}) \mapsto (\mathbf{c}, \sigma). \quad (1)$$

Specifically, positional encoding [41, 65] is applied to \mathbf{x} and \mathbf{d} to represent the high-frequency details of an image. Subsequently, an MLP is applied to the encoded inputs to obtain \mathbf{c} and σ . For simplicity, we represent these series of processes in a unified manner as f_{SISO} .

A NeRF is based on ray tracing, in which a camera ray is defined as $\mathbf{r}(t) = \mathbf{o} + t\mathbf{d}$, where \mathbf{o} and \mathbf{d} respectively denote the origin and direction of the camera and t denotes a distance from the origin. A NeRF calculates the color of each pixel $\hat{\mathbf{C}}(\mathbf{r})$ by integrating the colors and volume densities on a ray $\mathbf{r}(t)$ within $t \in [t_n, t_f]$ using volume rendering [36]. In implementation, the calculation of the integral is intractable; therefore, a ray is discretized into N points; alternatively, the following discretized formulation can be used.

$$\hat{\mathbf{C}}(\mathbf{r}) = \sum_{i=1}^N T_i \alpha_i \mathbf{c}_i, \text{ where } T_i = \prod_{j=1}^{i-1} (1 - \alpha_j), \quad (2)$$

where the subscript i indicates that the variable corresponds to the i -th point on a ray, $\alpha_i = 1 - \exp(-\sigma_i \delta_i)$ is an alpha value, and $\delta_i = t_{i+1} - t_i$ is the distance between the i -th and $(i+1)$ -th points. f_{SISO} is optimized by minimizing the following pixel-wise loss.

$$\mathcal{L}_{\text{pixel}} = \|\hat{\mathbf{C}}(\mathbf{r}) - \mathbf{C}(\mathbf{r})\|_2^2, \quad (3)$$

where $\mathbf{C}(\mathbf{r})$ is the ground-truth color for a ray \mathbf{r} . In implementation, this loss is calculated for $\mathbf{r} \in \mathcal{R}$, where \mathcal{R} denotes a set of rays in each batch.

In practice, a NeRF uses coarse and fine networks. In the coarse network, a ray is discretized into N_c points using stratified sampling, whereas in the fine network, a ray is discretized into $N_c + N_f$ points using hierarchical sampling in which N_f additional points are sampled according to the output of the coarse network. The two networks are optimized by minimizing $\mathcal{L}_{\text{pixel}}$ for $\hat{\mathbf{C}}(\mathbf{r})$ predicted by each network. Hereafter, we omit a variable in parentheses (e.g., (\mathbf{r})) for simplicity.

4. MIMO-NeRF

4.1. MIMO formulation

There are several ways to group the input samples when constructing a MIMO MLP. For example, we can construct a general MLP that can accept any combination of samples in a 3D space, or we can construct a specific MLP that only accepts a group of nearby samples. In preliminary experiments (Appendix A.1), we found that the latter significantly outperformed the former because general models are more difficult to train than more specific models. Therefore, we adopted the latter in this study. In particular, we group neighboring samples on a ray as shown in Figure 1(b).³

More formally, given N samples on a ray, we group N_p samples from the sample nearest to the camera and create

³One possible alternative is to group near samples on different rays. However, in NeRFs, searching near points across different rays is not trivial because points are sampled unevenly via hierarchical sampling. Therefore, we simply group neighboring samples on the same ray in this study.

N/N_p groups. Subsequently, we apply a MIMO function f_{MIMO} to each group as follows.

$$f_{\text{MIMO}} : (\mathbb{R}^3)^{N_p} \times \mathbb{S}^2 \rightarrow (\mathbb{R}^3 \times \mathbb{R}^+)^{N_p}, \\ (\mathbf{x}_i, \dots, \mathbf{x}_j, \mathbf{d}) \mapsto (\mathbf{c}_i, \dots, \mathbf{c}_j, \sigma_i, \dots, \sigma_j), \quad (4)$$

where $i \in \{1, 1 + N_p, \dots, 1 + N - N_p\}$ and $j = i + N_p - 1$. Assuming that the grouped samples are lined on a ray, we use a single direction \mathbf{d} in the input of f_{MIMO} . In this formulation, the number of MLPs running to render a pixel (#Run) is equal to the number of groups and is calculated as # Run = N/N_p . Therefore, we can reduce the calculation cost, particularly that of # Run, by increasing N_p .

In inference, the only necessary modification to a NeRF is the simple replacement of f_{SISO} with f_{MIMO} and the same volume rendering (Equation 2) and sampling scheme (i.e., stratified and hierarchical sampling) can be used. With this modification strategy, MIMO-NeRF exhibits high compatibility and complementarity with previous NeRFs.

During training, $\mathcal{L}_{\text{pixel}}$ (Equation 3) can be used for $\hat{\mathbf{C}}$ obtained using f_{MIMO} . However, this loss does not necessarily suffice to address the ambiguity in the color and volume density of each point because this ambiguity occurs in a 3D space and the loss cannot regularize the 3D representations explicitly. Hence, we introduce self-supervised learning as discussed in subsequent sections.

4.2. MIMO reformulation

One possible solution to this ambiguity is to train a standard (i.e., SISO) NeRF first and then distill the model onto a corresponding MIMO-NeRF. However, this solution involves an increase in training time because it requires training not only a MIMO-NeRF but also a SISO NeRF. Consequently, this solution cannot entirely take advantage of the fast rendering of MIMO-NeRF in training.

Alternatively, we reformulate a MIMO MLP in multiple ways and impose a consistent regularization to produce the same RGB colors and alpha values. Because each reformulated MIMO MLP can render a pixel faster than the original SISO MLP, we can prevent a large increase in training time even when using multiple reformulated MIMO MLPs by adequately adjusting the reformulation configuration. When implementing this idea, the question arises as to how best to reformulate a MIMO MLP. To this end, we developed two methods, including (1) *group shift* and (2) *variation reduction*.

Group shift. We consider restricting this ambiguity by assessing each point in multiple ways using different groups and imposing consistency on the assessed results. More formally, we implement this by shifting groups and rewriting Equation 4 as follows.

$$f_{\text{MIMO}}^{\text{shift}} : (\mathbb{R}^3)^{N_p} \times \mathbb{S}^2 \rightarrow (\mathbb{R}^3 \times \mathbb{R}^+)^{N_p}, \\ (\mathbf{x}_{i'}, \dots, \mathbf{x}_{j'}, \mathbf{d}) \mapsto (\mathbf{c}_{i'}, \dots, \mathbf{c}_{j'}, \sigma_{i'}, \dots, \sigma_{j'}), \quad (5)$$

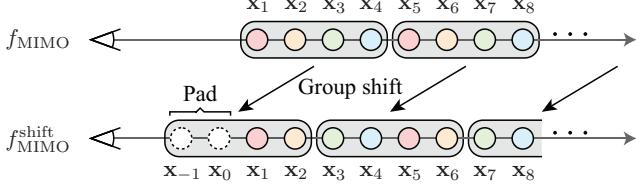


Figure 3. Example of the group shift when $N_p = 4$ and $s = 2$. We shift each group by s toward the camera after padding s samples before the front sample. This procedure enables assessing each sample in multiple ways using different groups.

where $i' \in \{k, k + N_p, \dots, k + N\}$ and $j' = i' + N_p - 1$. Here, k is the head index of the first group, which is shifted by $s \in \{1, \dots, N_p - 1\}$ toward the camera. Hence, $k \in \{2 - N_p, \dots, 0\}$. In practice, it is randomly sampled during training. More strictly, we add padding before this process to represent a sample with an index exceeding the original index, i.e., $i' < 1$ or $j' > N$. For clarity, we present an example in which $N_p = 4$ and $s = 2$ in Figure 3.

Variation reduction. In the original MIMO formulation, the abovementioned ambiguity is caused by N_p different input samples. To mitigate this, we consider reducing the variation in the input by replacing Equation 4 with the following.

$$f_{\text{MIMO}}^{\text{reduce}} : (\mathbb{R}^3)^{N_p} \times \mathbb{S}^2 \rightarrow (\mathbb{R}^3 \times \mathbb{R}^+)^{N_p}, \\ ([\mathbf{x}_{i''}]^R, \dots, [\mathbf{x}_{j''}]^R, \mathbf{d}) \mapsto (\mathbf{c}_{i'_1}, \dots, \mathbf{c}_{i'_R}, \dots, \mathbf{c}_{j'_1}, \dots, \mathbf{c}_{j'_R}, \\ \sigma_{i'_1}, \dots, \sigma_{i'_R}, \dots, \sigma_{j'_1}, \dots, \sigma_{j'_R}), \quad (6)$$

where $[.]^R$ denotes the operation of repeating the given variable R times, $i'' \in \{1, 1 + \frac{N_p}{R}, \dots, 1 + N - \frac{N_p}{R}\}$, and $j'' = i'' + \frac{N_p}{R} - 1$. After applying $f_{\text{MIMO}}^{\text{reduce}}$, we average $(\mathbf{c}_{k'_1}, \dots, \mathbf{c}_{k'_R})$ and $(\sigma_{k'_1}, \dots, \sigma_{k'_R})$ to obtain $\mathbf{c}_{k''}$ and $\sigma_{k''}$, where $k'' \in \{i'', \dots, j''\}$. In this formulation, the mentioned ambiguity is reduced by decreasing input variation from N_p to $\frac{N_p}{R}$. Conversely, the number of MLPs running increases by factor of R , i.e., # Run = $N/N_p \times R$; therefore, we need to select R carefully in practice to avoid a large increase in training time. For clarity, we present an example case in which $N_p = 4$ and $R = 2$ in Figure 4.

4.3. MIMO objective

Through the above processes, we obtain M reformulated MIMO MLPs in which we use different s for each MIMO MLP and set R such that the total number of # Run is not larger than that of the original SISO MLP. Hereafter, we use the superscript $m \in \{1, \dots, M\}$ to denote the variable corresponding to the m -th reformulated MIMO MLP, e.g., $\hat{\mathbf{C}}^m$ and R^m . We train these MLPs using two loss functions, including pixel-wise and 3D consistency losses.

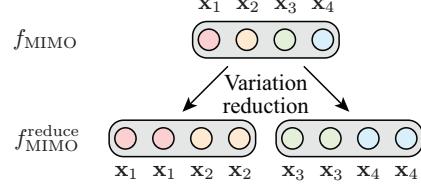


Figure 4. Example of variation reduction when $N_p = 4$ and $R = 2$. Samples with the same color correspond to the same coordinate. In $f_{\text{MIMO}}^{\text{reduce}}$, we reduce the number of unique samples in a group from $N_p (= 4)$ to $\frac{N_p}{R} (= 2)$ by repeating each sample $R (= 2)$ times to reduce the variation in a group.

Pixel-wise loss. We apply the pixel-wise loss (Equation 3) to each $\hat{\mathbf{C}}^m$ as follows.

$$\mathcal{L}_{\text{pixel}}^{\text{MIMO}} = \sum_{m=1}^M \|\hat{\mathbf{C}}^m - \mathbf{C}\|_2^2. \quad (7)$$

3D consistency loss. The pixel-wise loss provides supervision in a 2D space; however, it cannot impose an explicit regularization in a 3D space. Hence, we introduce a 3D consistency loss that encourages the reformulated MIMO MLPs to produce the same colors and alpha values in the 3D space. The 3D consistency loss consists of a color 3D consistency loss $\mathcal{L}_{\text{3D}}^{\text{color}}$ and an alpha value 3D consistency loss $\mathcal{L}_{\text{3D}}^{\text{alpha}}$ as follows.

$$\mathcal{L}_{\text{3D}}^{\text{color}} = \sum_{m_1=1}^{M-1} \sum_{m_2=m_1+1}^M \frac{1}{N} \sum_{i=1}^N [\mu_{m_1}^{m_2} \|\mathbf{c}_i^{m_1} - \text{sg}(\mathbf{c}_i^{m_2})\|_2^2 \\ + \mu_{m_2}^{m_1} \|\text{sg}(\mathbf{c}_i^{m_1}) - \mathbf{c}_i^{m_2}\|_2^2], \quad (8)$$

$$\mathcal{L}_{\text{3D}}^{\text{alpha}} = \sum_{m_1=1}^{M-1} \sum_{m_2=m_1+1}^M \frac{1}{N} \sum_{i=1}^N [\mu_{m_1}^{m_2} \|\alpha_i^{m_1} - \text{sg}(\alpha_i^{m_2})\|_2^2 \\ + \mu_{m_2}^{m_1} \|\text{sg}(\alpha_i^{m_1}) - \alpha_i^{m_2}\|_2^2]. \quad (9)$$

where ‘‘sg’’ indicate a stop-gradient operation. The 3D consistency loss \mathcal{L}_{3D} is calculated by $\mathcal{L}_{\text{3D}} = \mathcal{L}_{\text{3D}}^{\text{color}} + \mathcal{L}_{\text{3D}}^{\text{alpha}}$. We define $\mu_{m_i}^{m_j}$ as $\mu_{m_i}^{m_j} = \frac{\sqrt{R^{m_j}}}{\sqrt{R^{\max}} \sqrt{R^{m_i}}}$, where R^{\max} is the maximum of R^m in $m \in \{1, \dots, M\}$. We use this asymmetric weight on the assumption that \mathbf{c}_i^m and α_i^m with greater R^m have lower ambiguity and are more reliable. Hence, the effect of \mathcal{L}_{3D} is reduced. We empirically investigated the importance of this effect through an ablation study as described in Section 5.3.

Full objective. The full objective is defined as follows.

$$\mathcal{L}_{\text{MIMO}} = \mathcal{L}_{\text{pixel}}^{\text{MIMO}} + \lambda \mathcal{L}_{\text{3D}}, \quad (10)$$

where λ is a hyperparameter that balances the pixel-wise loss and 3D consistency loss.

5. Experiments

We conducted five experiments to investigate the effectiveness of MIMO-NeRF. In the first three experiments, we conducted a comprehensive study, including an investigation of benchmark performance (Section 5.1), an investigation of the trade-off between speed and quality (Section 5.2), and ablation studies (Section 5.3). In the remaining two experiments, we examined the versatility of MIMO-NeRF by applying it to two representative fast NeRFs, including a NeRF with sample reduction, i.e., DONeRF [44] (Section 5.4), and a NeRF with alternative representations, i.e., TensoRF [9] (Section 5.5). The main results of these experiments are provided here, and detailed analyses are presented with extended results in Appendix A. The implementation details are presented in Appendix B.

5.1. Investigation of benchmark performance

We investigated the benchmark performance of MIMO-NeRF by applying our ideas to the original *NeRF* [41]. In particular, we examined three variants of MIMO-NeRF, including *MIMO-NeRF-naive*, which simply replaced f_{SISO} (Equation 1) with f_{MIMO} (Equation 4) and was trained with a standard pixel-wise loss (Equation 3).⁴ *MIMO-NeRF-distill*, which is a student model distilled from a pretrained standard (i.e., SISO) NeRF. During training, we used a 3D consistency loss (Equations 8 and 9) that was adjusted for knowledge distillation, in addition to a standard pixel-wise loss (Equation 3). *MIMO-NeRF-self*, which is MIMO-NeRF that adopted the proposed self-supervised learning. We examined the performance of these models when N_p was varied within $\{2, 4, 8\}$.

Datasets. We investigated the benchmark performance on two commonly-used datasets. (1) *Blender dataset* [41] includes eight scenes, each of which consists of 360° views of complex objects at a resolution of 800 × 800 pixels. We used 100 and 200 views for training and testing, respectively. (2) *Local Light Field Fusion (LLFF) dataset* [40, 41], which consists of eight complex real-world scenes, each of which includes 20–62 forward-facing views at 1008 × 756 pixels. One-eighth of the images were used for testing, and the remainder were used for training. Where not otherwise specified, we used half-sized images following the default settings of a widely-used source code for NeRF⁵ to better investigate the various configurations.

Implementation. For a fair comparison, we implemented all the models with a commonly-used source code for NeRF⁵ and trained the models using the default settings provided in the code. The number of samples was set as

⁴For simplicity and a fair comparison, we only increased the input and output of the original SISO MLP and retained the other parameters (e.g., depth and width). Hence, the increase in model size was relatively small.

⁵<https://github.com/yenchenlin/nerf-pytorch>

$N_c = 64$ and $N_f = 128$ for the Blender dataset and $N_c = 64$ and $N_f = 64$ for the LLFF dataset. For MIMO-NeRF-self, we used two formulations with $R^1 = R^2 = 1$ when $N_p = 2$, two formulations with $R^1 = 1$ and $R^2 = 2$ when $N_p = 4$, and three formulations with $R^1 = 1$, $R^2 = 2$, and $R^3 = 4$ when $N_p = 8$. MIMO-NeRF-self was trained individually depending on N_p . An investigation of different reformulation methods is presented in Appendix A.2. Group shifts were applied to all cases. We set λ to 1 and 0.4 for the Blender and LLFF datasets, respectively. The effect of λ is analyzed in Appendix A.3. The implementation details are presented in Appendix B.1.

Evaluation metrics. Following the original *NeRF* study [41], we used the peak signal-to-noise ratio (*PSNR*), structural similarity index (*SSIM*) [68], and learned perceptual image patch similarity (*LPIPS*) [77] to quantitatively evaluate the image quality. To assess the calculation cost of inference and training, we report inference time (*I-time*) measured with an NVIDIA GeForce RTX 3080 Ti GPU and training time (*T-time*) measured with an NVIDIA A100-SXM4-80GB GPU.⁶ We also provide $\# \text{Run} \left(= \frac{N_c + (N_c + N_f)}{N_p}\right)$ and the number of parameters ($\# \text{Params}$) as supplementary information. $\# \text{Params}$ increases in MIMO-NeRF mainly because the total dimension of the positional embeddings is increased by N_p times according to the increase in the inputs.

Results. From Table 1, the following is observed.

Image quality. MIMO-NeRF-self outperformed not only MIMO-NeRF-naive but also MIMO-NeRF-distill in most cases in terms of PSNR, SSIM, and LPIPS. We conjecture that this occurred because the joint optimization of the teacher and student networks in MIMO-NeRF-self was more effective for training than the student-only optimization in MIMO-NeRF-distill. Even MIMO-NeRF-self suffered from a trade-off between speed and quality with increasing N_p ; however, MIMO-NeRF-self performed better than or comparably to the original *NeRF* when $N_p = 2$. This may have occurred because the advantage of accumulating neighboring information and the disadvantage of handling the ambiguity were antagonistic in this case.

Inference speed. All MIMO-NeRFs with the same inference procedure showed inference times improved by a factor of 1.84–5.78 with increasing N_p .

Training speed. MIMO-NeRF-naive achieved the fastest training because it used only a single MIMO formulation during training. MIMO-NeRF-self required more training time because it uses multiple reformulated MIMO MLPs; however, each calculation cost is low. Therefore, it did

⁶For simplicity and a fair comparison, we measured the calculation time using a standard PyTorch implementation⁵ for all the models. Optimizing the implementation for faster rendering (e.g., using custom CUDA kernels) would be interesting for future research.

Model	N_p	Blender						LLFF							
		PSNR↑	SSIM↑	LPIPS↓	# Run↓	I-time↓ (s)	T-time↓ (h)	# Params (M)	PSNR↑	SSIM↑	LPIPS↓	# Run↓	I-time↓ (s)	T-time↓ (h)	# Params (M)
NeRF	1	31.04	0.951	0.055	256	9.60	4.70	1.19	27.72	0.871	0.150	192	8.38	3.39	1.19
MIMO-NeRF-naive	2	30.18	0.944	0.065	128	5.15	3.09	1.26	27.31	0.860	0.167	96	4.55	2.12	1.26
MIMO-NeRF-distill		30.76	0.949	0.058	128	5.15	9.46	1.26	27.50	0.863	0.169	96	4.55	6.81	1.26
MIMO-NeRF-self	31.26	0.953	0.054	128	5.15	5.36	1.26	27.70	0.870	0.155	96	4.55	3.97	1.26	
MIMO-NeRF-naive	4	28.62	0.927	0.091	64	2.79	2.02	1.39	26.29	0.824	0.218	48	2.46	1.57	1.39
MIMO-NeRF-distill		30.22	0.946	0.065	64	2.79	8.42	1.39	27.37	0.861	0.172	48	2.46	6.25	1.39
MIMO-NeRF-self		30.94	0.950	0.058	64	2.79	4.68	1.39	27.51	0.865	0.162	48	2.46	3.44	1.39
MIMO-NeRF-naive	8	26.34	0.895	0.133	32	1.66	1.66	1.65	25.10	0.774	0.284	24	1.45	1.24	1.65
MIMO-NeRF-distill		29.39	0.937	0.075	32	1.66	8.07	1.65	27.01	0.851	0.184	24	1.45	5.91	1.65
MIMO-NeRF-self		30.40	0.945	0.065	32	1.66	5.86	1.65	26.97	0.851	0.180	24	1.45	4.43	1.65

Table 1. Benchmark performance of MIMO-NeRFs. MIMO-NeRF-self outperformed MIMO-NeRF-naive and MIMO-NeRF-distill in terms of PSNR, SSIM, and LPIPS in most cases with shorter training time than MIMO-NeRF-distill. All MIMO-NeRFs outperformed the original NeRF in terms of inference time owing to the reduction of # Run.

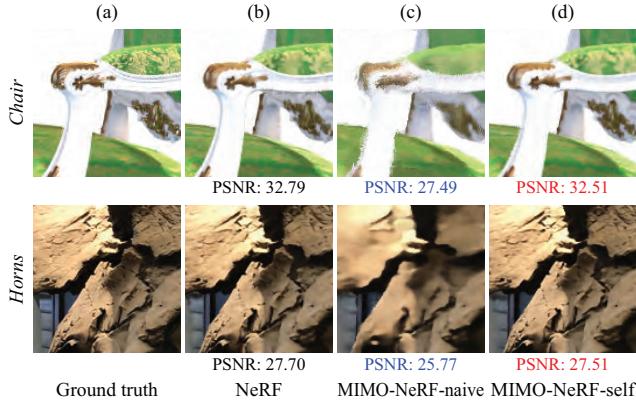


Figure 5. Qualitative comparison between NeRF and MIMO-NeRFs with $N_p = 8$. The models were trained using full-sized images. MIMO-NeRF-naive (c) produced some artifacts owing to the ambiguity in the color and volume density of each point. MIMO-NeRF-self (d) was useful for addressing this issue, and its results were close to those of NeRF (b), while inference time were improved by a factor of 5.8.

not suffer from a large increase in training time compared with MIMO-NeRF-distill, which requires training not only a MIMO-NeRF but also a SISO NeRF.

Summary. From these results, we found that when $N_p = 2$, MIMO-NeRF-self improved the inference speed of NeRF without compromising image quality, and when N_p was larger, there was a trade-off between speed and quality. We examine the validity of this trade-off in Section 5.2.

Qualitative results. Figure 5 shows a qualitative comparison between NeRF, MIMO-NeRF-naive, and MIMO-NeRF-self. In this experiment, we used full-sized images to train the models and set N_p to 8 for MIMO-NeRFs. MIMO-NeRF-naive produced some artifacts, whereas MIMO-NeRF-self adequately addressed this issue. The additional results are provided in Appendix A.6.

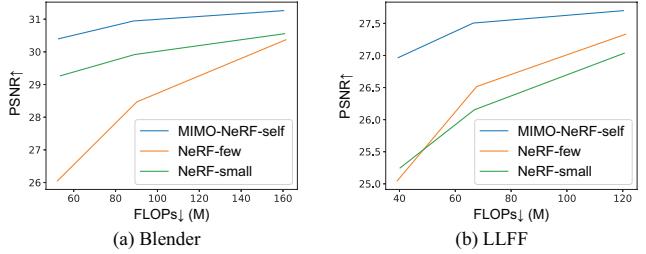


Figure 6. Relationship between FLOPs and PSNR. Higher values indicate better the image quality. Faster speeds are shown to the left. MIMO-NeRF-self achieves the best trade-off between speed and quality.

5.2. Investigation of speed-quality trade-off

We compared MIMO-NeRF with possible alternatives to investigate whether it achieved a good trade-off between speed and quality. In particular, we focused on methods that are general and applicable to various NeRFs, similar to MIMO-NeRF, and examined two variants, including *NeRF-few*, which reduced the number of samples on a ray, and *NeRF-small*, which reduced the number of features in the hidden layers. We adjusted the parameters such that their FLOPs were comparable to those of MIMO-NeRF.⁷

Results. The relationship between the FLOPs and PSNR is plotted in Figure 6. We found that MIMO-NeRF-self obtained a better trade-off between speed and quality than NeRF-few and NeRF-small. We provide other relationships (e.g., relationships between FLOPs/inference time and PSNR/SSIM/LPIPS) in Appendix A.4.⁸

⁷We tuned the models based on FLOPs because the performance of different methods for improving speed in terms of inference time may vary depending on the calculation tools used, such as GPU processor hardware. As a reference, we provide the relationship between the inference time and image quality in Appendix A.4.

⁸During training, the calculation cost of MIMO-NeRF-self was larger than those of NeRF-small and NeRF-few because multiple reformulated MIMO MLPs were used. To confirm this effect, we examined the performance of NeRF-few and NeRF-small with increasing batch sizes such that the calculation costs became almost the same as that of MIMO-NeRF.

N_p	GS	CL	AW	Blender			LLFF		
				PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓
2	\checkmark	\checkmark	\checkmark	30.17	0.943	0.067	27.21	0.856	0.170
				30.54	0.945	0.065	27.48	0.865	0.161
				31.26	0.953	0.054	27.70	0.870	0.155
4	\checkmark	\checkmark	\checkmark	30.84	0.949	0.058	27.39	0.862	0.166
				29.48	0.936	0.077	26.46	0.832	0.206
				30.87	0.949	0.060	27.44	0.864	0.164
				30.94	0.950	0.058	27.51	0.865	0.162
8	\checkmark	\checkmark	\checkmark	29.81	0.941	0.069	26.97	0.851	0.179
				27.39	0.907	0.116	24.29	0.734	0.332
				30.16	0.942	0.071	26.69	0.843	0.192
				30.40	0.945	0.065	26.97	0.851	0.180

Table 2. Results of ablation studies. Check marks in GS, CL, and AW indicate the use of group shift, 3D consistency loss, and asymmetric weights. In $N_p = 2$, asymmetric weights were not used in the full model because $R^1 = R^2 = 1$. Hence, it was not ablated.

5.3. Ablation studies

We also conducted ablation studies to better understand the performance of each element of the proposed self-supervised learning method. Specifically, we investigated the importance of group shift, 3D consistency loss, and asymmetric weights. When asymmetric weights were ablated, $\mu_{m_i}^{m_j}$ was set to 1.

Results. The results are listed in Table 2. We found that the full model achieved the best performance in most cases. The results validate the importance of each technique.

5.4. Application to DONeRF

We incorporated MIMO-NeRF into DONeRF [44], a representative NeRF with sample reduction, to demonstrate that MIMO-NeRF can complement existing fast NeRFs. DONeRF uses a sampling network called a depth oracle network to select samples and calculates the colors and volume densities of the selected samples using a shading network. It handles the trade-off between speed and image quality by adjusting the number of selected samples (N_s). We examined whether MIMO-NeRF could be used as an alternative to handle this trade-off.

Dataset. We evaluated the performance using the *DONeRF dataset* [44] comprising six synthetic indoor and outdoor scenes. Each scene included 300 forward-facing views at a resolution of 800×800 pixels. 70%, 10%, and 20% of the images were used for training, validation, and testing, respectively.

Implementation. We implemented the models according to the source code of DONeRF⁹ and trained them using the default settings. We applied MIMO-NeRF with $N_p = 4$ to *DONeRF-16* (i.e., DONeRF with $N_s = 16$). In the self-supervised learning process, we used two formulations with

self. We found that MIMO-NeRF-self achieved a better trade-off between speed and quality than the other variants. Detailed results are provided in Appendix A.4.

⁹<https://github.com/facebookresearch/DONeRF>

Model	PSNR↑	FLIP↓	# Run↓	I-time↓	T-time↓	# Params (M)
				(s)	(h)	
DONeRF-16	33.06	0.061	17	0.429	3.79	0.94
DONeRF-4	31.21	0.070	5	0.140	3.23	0.94
MIMO-DONeRF-16/4-naive	32.30	0.063	5	0.155	3.26	0.99
MIMO-DONeRF-16/4-self	32.72	0.061	5	0.155	3.56	0.99

Table 3. Comparison of quantitative scores between DONeRFs and MIMO-DONeRFs. MIMO-DONeRF-16/4-naive outperformed DONeRF-4 in terms of PSNR and FLIP with a small increase in I-time and T-time. The performance of MIMO-DONeRF-16/4-self was close to DONeRF-16 in terms of PSNR and FLIP with shorter I-time and T-time.

$R^1 = R^2 = 1$ and group shifts. This model is referred to as *MIMO-DONeRF-16/4*. We set λ to 0.001. As a baseline, we examined DONeRF-4, which had the same # Run as MIMO-DONeRF-16/4. The implementation details are presented in Appendix B.2.

Evaluation metrics. Following the study on DONeRF [44], we assessed the image quality using the *PSNR* and *FLIP* [1]. In addition, we used the *# Run*, *I-time*, *T-time*, and *# Params* described in Section 5.1. In DONeRF, # Run was calculated as $1 + \frac{N_s}{N_p}$.

Results. The results are summarized in Table 3. It is observed that MIMO-DONeRF-16/4-naive outperformed DONeRF-4 in terms of PSNR and FLIP with a small increase in I-time and T-time. MIMO-DONeRF-16/4-self enhanced the image quality with an increase in T-time, and its image quality approached that of DONeRF-16 in terms of PSNR and FLIP with faster inference and training. These results suggest that the increase in N_p (i.e., the replacement of the SISO MLP by the MIMO MLP) can be used as an alternative to the reduction in N_s (the number of selected samples) to obtain a better trade-off between speed and quality. A detailed analysis is presented in Appendix A.8.

5.5. Application to TensoRF

A NeRF with alternative representations is another representative fast approach. To demonstrate that MIMO-NeRF is also compatible with this model, we applied it to TensoRF [9], a representative model in this category. TensoRF uses a vector-matrix decomposition to calculate the volume densities and color features and applies a SISO MLP to the color features to decode the RGB colors. The corresponding ambiguity was relatively limited because the volume densities were extracted using an explicit representation. Hence, we simply replaced the SISO MLP with a MIMO MLP without modifying the training process while prioritizing training speed. These models are denoted as *MIMO-TensoRF- N_p* , where N_p was varied among {2, 4, 8}.

Datasets. We examined the performance of our approach on the *Blender* [41] and *LLFF* [40] datasets described in Section 5.1. Full-size images were used in this experiment.

Model	PSNR↑	SSIM↑	# Run↓	I-time↓ (s)	T-time↓ (m)	# Params (M)
TensoRF	33.23	0.963	9.95	1.25	11.50	18.8
MIMO-TensoRF-2	33.26	0.963	4.76	1.18	10.89	18.8
MIMO-TensoRF-4	32.98	0.961	2.40	1.15	10.67	18.8
MIMO-TensoRF-8	32.37	0.956	1.27	1.14	10.57	18.9

(a) Blender

Model	PSNR↑	SSIM↑	# Run↓	I-time↓ (s)	T-time↓ (m)	# Params (M)
TensoRF	26.73	0.837	126.73	6.64	23.41	46.8
MIMO-TensoRF-2	26.72	0.837	62.14	6.18	21.63	46.8
MIMO-TensoRF-4	26.72	0.836	30.16	5.76	21.15	46.8
MIMO-TensoRF-8	26.64	0.835	14.52	5.52	20.68	46.9

(b) LLFF

Table 4. Comparison of quantitative scores between TensoRF and MIMO-TensoRF. MIMO-TensoRF improved I-time and T-time while retaining PSNR and SSIM when $N_p \leq 2$ and $N_p \leq 4$ on the Blender and LLFF datasets, respectively.

Implementation. We implemented the models based on the official source code of TensoRF¹⁰ and trained all the models using the same default settings for a fair comparison. The implementation details are presented in Appendix B.3.

Evaluation metrics. Following the study on TensoRF [9], we measured the image quality using PSNR and SSIM [68]. In addition, we used the # Run, I-time, T-time, and # Params described in Section 5.1. In TensoRF, # Run is determined adaptively for each pixel. Therefore, we report the average.

Results. The results are presented in Table 4. It is observed that MIMO-TensoRF improved I-time and T-time with similar image quality when N_p was set within an adequate range (in particular, $N_p \leq 2$ on the Blender dataset and $N_p \leq 4$ on the LLFF dataset). These results suggest that MIMO-TensoRF can strengthen the inference and training speed of TensoRF without negative effects by adequately selecting N_p . A detailed analysis is presented in Appendix A.9.

6. Discussion

The results of these experiments in various situations demonstrate that MIMO-NeRF achieved a good trade-off between speed and quality. However, we also found that the quality degradation became significant with increasing N_p . One possible reason for this is that we did not modify the baseline network except for its input and output and did not increase the capacity of the models. It might be natural to implement a model of larger capacity to handle larger combinations of inputs and outputs. We did not adopt this strategy to ensure a fair comparison. However, searching for the best configurations considering the number of samples, the number of groups (the proposed new searching area), and the size of the model remain as a practically imperative and promising direction for further research.

¹⁰<https://github.com/apchenstu/TensoRF>

7. Conclusion

In this study, we have proposed MIMO-NeRF to improve the rendering speed of NeRF. Our core idea is that of replacing the SISO MLP used in standard NeRFs with a MIMO-MLP. We have developed a novel self-supervised learning method to address the ambiguity in the color and volume density of each point without relying on pretrained models. The results of an experimental evaluation have shown that MIMO-NeRF achieves a good trade-off between speed and quality with a reasonable training time. Although we have demonstrated the versatility of MIMO-NeRF by applying it to various NeRFs, many implicit neural representations aside from NeRFs also partially or primarily use SISO MLPs. We expect our ideas to be utilized with a few modifications to speed up the execution of such models.

References

- [1] Pontus Andersson, Jim Nilsson, Tomas Akenine-Möller, Magnus Oskarsson, Kalle Åström, and Mark D. Fairchild. FLIP: A difference evaluator for alternating images. *Proc. ACM Comput. Graph. Interact. Tech.*, 3(2), 2020. 8, 32
- [2] Benjamin Attal, Jia-Bin Huang, Michael Zollhoefer, Johannes Kopf, and Changil Kim. Learning neural light fields with ray-space embedding networks. In *CVPR*, 2022. 3
- [3] Matan Atzmon and Yaron Lipman. SAL: Sign agnostic learning of shapes from raw data. In *CVPR*, 2020. 3
- [4] Jonathan T. Barron, Ben Mildenhall, Matthew Tancik, Peter Hedman, Ricardo Martin-Brualla, and Pratul P. Srinivasan. Mip-NeRF: A multiscale representation for anti-aliasing neural radiance fields. In *ICCV*, 2021. 3
- [5] Jonathan T. Barron, Ben Mildenhall, Dor Verbin, Pratul P. Srinivasan, and Peter Hedman. Mip-NeRF 360: Unbounded anti-aliased neural radiance fields. In *CVPR*, 2022. 3
- [6] Rohan Chabra, Jan E. Lenssen, Eddy Ilg, Tanner Schmidt, Julian Straub, Steven Lovegrove, and Richard Newcombe. Deep local shapes: Learning local SDF priors for detailed 3D reconstruction. In *ECCV*, 2020. 3
- [7] Eric R. Chan, Connor Z. Lin, Matthew A. Chan, Koki Nagano, Boxiao Pan, Shalini De Mello, Orazio Gallo, Leonidas J. Guibas, Jonathan Tremblay, Sameh Khamis, Tero Karras, and Gordon Wetzstein. Efficient geometry-aware 3D generative adversarial networks. In *CVPR*, 2022. 2, 3
- [8] Eric R. Chan, Marco Monteiro, Petr Kellnhofer, Jiajun Wu, and Gordon Wetzstein. pi-GAN: Periodic implicit generative adversarial networks for 3D-aware image synthesis. In *CVPR*, 2021. 3
- [9] Anpei Chen, Zexiang Xu, Andreas Geiger, Jingyi Yu, and Hao Su. TensoRF: Tensorial radiance fields. In *ECCV*, 2022. 2, 3, 6, 8, 9, 24, 25, 26, 27, 33
- [10] Xingyu Chen, Qi Zhang, Xiaoyu Li, Yue Chen, Ying Feng, Xuan Wang, and Jue Wang. Hallucinated neural radiance fields in the wild. In *CVPR*, 2022. 3
- [11] Zhiqin Chen and Hao Zhang. Learning implicit fields for generative shape modeling. In *CVPR*, 2019. 3

- [12] Julian Chibane, Aymen Mir, and Gerard Pons-Moll. Neural unsigned distance fields for implicit function learning. In *NeurIPS*, 2020. 3
- [13] Yu Deng, Jiaolong Yang, Jianfeng Xiang, and Xin Tong. GRAM: Generative radiance manifolds for 3D-aware image generation. In *CVPR*, 2022. 3
- [14] Jiemin Fang, Lingxi Xie, Xinggang Wang, Xiaopeng Zhang, Wenyu Liu, and Qi Tian. NeuSample: Neural sample field for efficient view synthesis. *arXiv preprint arXiv:2111.15552*, 2021. 2, 3
- [15] Sara Fridovich-Keil, Alex Yu, Matthew Tancik, Qinhong Chen, Benjamin Recht, and Angjoo Kanazawa. Plenoxels: Radiance fields without neural networks. In *CVPR*, 2022. 2, 3
- [16] Guy Gafni, Justus Thies, Michael Zollhofer, and Matthias Nießner. Dynamic neural radiance fields for monocular 4D facial avatar reconstruction. In *CVPR*, 2021. 3
- [17] Stephan J. Garbin, Marek Kowalski, Matthew Johnson, Jamie Shotton, and Julien Valentin. FastNeRF: High-fidelity neural rendering at 200FPS. In *ICCV*, 2021. 2, 3
- [18] Kyle Genova, Forrester Cole, Daniel Vlasic, Aaron Sarna, William T. Freeman, and Thomas Funkhouser. Learning shape templates with structured implicit functions. In *ICCV*, 2019. 3
- [19] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *NIPS*, 2014. 3
- [20] Amos Gropp, Lior Yariv, Niv Haim, Matan Atzmon, and Yaron Lipman. Implicit geometric regularization for learning shapes. In *ICML*, 2020. 3
- [21] Jitao Gu, Lingjie Liu, Peng Wang, and Christian Theobalt. StyleNeRF: A style-based 3D-aware generator for high-resolution image synthesis. In *ICLR*, 2022. 3
- [22] Peter Hedman, Pratul P. Srinivasan, Ben Mildenhall, Jonathan T. Barron, and Paul Debevec. Baking neural radiance fields for real-time view synthesis. In *ICCV*, 2021. 2, 3
- [23] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. In *NeurIPS*, 2020. 3
- [24] Tao Hu, Shu Liu, Yilun Chen, Tiancheng Shen, and Jiaya Jia. EfficientNeRF: Efficient neural radiance fields. In *CVPR*, 2022. 2, 3
- [25] Chiyu Max Jiang, Avneesh Sud, Ameesh Makadia, Jingwei Huang, Matthias Nießner, and Thomas Funkhouser. Local implicit grid representations for 3D scenes. In *CVPR*, 2020. 3
- [26] Takahiro Kaneko. AR-NeRF: Unsupervised learning of depth and defocus effects from natural images with aperture rendering neural radiance fields. In *CVPR*, 2022. 3
- [27] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015. 31, 32, 33
- [28] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. ImageNet classification with deep convolutional neural networks. In *NIPS*, 2012. 33
- [29] Andreas Kurz, Thomas Neff, ZhaoYang Lv, Michael Zollhöfer, and Markus Steinberger. AdaNeRF: Adaptive sampling for real-time rendering of neural radiance fields. In *ECCV*, 2022. 2, 3
- [30] Chen-Hsuan Lin, Chaoyang Wang, and Simon Lucey. SDF-SRN: Learning signed distance 3D object reconstruction from static images. In *NeurIPS*, 2020. 1, 3
- [31] David B. Lindell, Julien N. P. Martel, and Gordon Wetzstein. AutoInt: Automatic integration for fast neural volume rendering. In *CVPR*, 2021. 2, 3, 20
- [32] Lingjie Liu, Jitao Gu, Kyaw Zaw Lin, Tat-Seng Chua, and Christian Theobalt. Neural sparse voxel fields. In *NeurIPS*, 2020. 2, 3
- [33] Shichen Liu, Shunsuke Saito, Weikai Chen, and Hao Li. Learning to infer implicit surfaces without 3D supervision. In *NeurIPS*, 2019. 3
- [34] Shaohui Liu, Yinda Zhang, Songyou Peng, Boxin Shi, Marc Pollefeys, and Zhaopeng Cui. DIST: Rendering deep implicit signed distance function with differentiable sphere tracing. In *CVPR*, 2020. 3
- [35] Ricardo Martin-Brualla, Noha Radwan, Mehdi S. M. Sajjadi, Jonathan T. Barron, Alexey Dosovitskiy, and Daniel Duckworth. NeRF in the wild: Neural radiance fields for unconstrained photo collections. In *CVPR*, 2021. 3
- [36] Nelson Max. Optical models for direct volume rendering. *IEEE Trans. Vis. Comput. Graph.*, 1(2), 1995. 1, 3, 4
- [37] Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. Occupancy networks: Learning 3D reconstruction in function space. In *CVPR*, 2019. 3
- [38] Mateusz Michalkiewicz, Jhony K. Pontes, Dominic Jack, Mahsa Baktashmotagh, and Anders Eriksson. Implicit surface representations as layers in neural networks. In *ICCV*, 2019. 3
- [39] 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 *CVPR*, 2022. 3
- [40] Ben Mildenhall, Pratul P. Srinivasan, Rodrigo Ortiz-Cayon, Nima Khademi Kalantari, Ravi Ramamoorthi, Ren Ng, and Abhishek Kar. Local light field fusion: Practical view synthesis with prescriptive sampling guidelines. *ACM Trans. Graph.*, 38(4), 2019. 6, 8, 30
- [41] 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, 3, 6, 8, 17, 30, 31, 32, 33
- [42] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *ACM Trans. Graph.*, 41(4), 2022. 2, 3
- [43] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted Boltzmann machines. In *ICML*, 2010. 30, 32, 33
- [44] Thomas Neff, Pascal Stadlbauer, Mathias Parger, Andreas Kurz, Joerg H. Mueller, Chakravarty R. Alla Chaitanya, Anton Kaplanyan, and Markus Steinberger. DONeRF: Towards real-time rendering of compact neural radiance fields using depth oracle networks. *Comput. Graph. Forum*, 40(4), 2021. 2, 3, 6, 8, 22, 32

- [45] Michael Niemeyer and Andreas Geiger. CAMPARI: Camera-aware decomposed generative neural radiance fields. In *3DV*, 2021. 3
- [46] Michael Niemeyer and Andreas Geiger. GIRAFFE: Representing scenes as compositional generative neural feature fields. In *CVPR*, 2021. 3
- [47] Michael Niemeyer, Lars Mescheder, Michael Oechsle, and Andreas Geiger. Differentiable volumetric rendering: Learning implicit 3D representations without 3D supervision. In *CVPR*, 2020. 1, 3
- [48] Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. DeepSDF: Learning continuous signed distance functions for shape representation. In *CVPR*, 2019. 3
- [49] Songyou Peng, Michael Niemeyer, Lars Mescheder, Marc Pollefeys, and Andreas Geiger. Convolutional occupancy networks. In *ECCV*, 2020. 3
- [50] Martin Piala and Ronald Clark. TermiNeRF: Ray termination prediction for efficient neural rendering. In *3DV*, 2021. 2, 3
- [51] Ben Poole, Ajay Jain, Jonathan T. Barron, and Ben Mildenhall. DreamFusion: Text-to-3D using 2D diffusion. In *ICLR*, 2023. 3
- [52] Albert Pumarola, Enric Corona, Gerard Pons-Moll, and Francesc Moreno-Noguer. D-NeRF: Neural radiance fields for dynamic scenes. In *CVPR*, 2021. 3
- [53] Daniel Rebain, Wei Jiang, Soroosh Yazdani, Ke Li, Kwang Moo Yi, and Andrea Tagliasacchi. DeRF: Decomposed radiance fields. In *CVPR*, 2021. 2, 3
- [54] Christian Reiser, Songyou Peng, Yiyi Liao, and Andreas Geiger. KiloNeRF: Speeding up neural radiance fields with thousands of tiny MLPs. In *ICCV*, 2021. 2, 3
- [55] Shunsuke Saito, Zeng Huang, Ryota Natsume, Shigeo Morishima, Angjoo Kanazawa, and Hao Li. PIFu: Pixel-aligned implicit function for high-resolution clothed human digitization. In *ICCV*, 2019. 3
- [56] Katja Schwarz, Yiyi Liao, Michael Niemeyer, and Andreas Geiger. GRAF: Generative radiance fields for 3D-aware image synthesis. In *NeurIPS*, 2020. 3
- [57] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015. 31, 33
- [58] Vincent Sitzmann, Julien Martel, Alexander Bergman, David Lindell, and Gordon Wetzstein. Implicit neural representations with periodic activation functions. In *NeurIPS*, 2020. 3
- [59] Vincent Sitzmann, Semon Rezhikov, Bill Freeman, Josh Tenenbaum, and Fredo Durand. Light field networks: Neural scene representations with single-evaluation rendering. In *NeurIPS*, 2021. 1, 3
- [60] Vincent Sitzmann, Michael Zollhöfer, and Gordon Wetzstein. Scene representation networks: Continuous 3D-structure-aware neural scene representations. In *NeurIPS*, 2019. 1, 3
- [61] Ivan Skorokhodov, Sergey Tulyakov, Yiqun Wang, and Peter Wonka. EpiGRAF: Rethinking training of 3D GANs. In *NeurIPS*, 2022. 3
- [62] Yang Song and Stefano Ermon. Generative modeling by estimating gradients of the data distribution. In *NeurIPS*, 2019. 3
- [63] Mohammed Suhail, Carlos Esteves, Leonid Sigal, and Ameesh Makadia. Light field neural rendering. In *CVPR*, 2022. 3
- [64] Cheng Sun, Min Sun, and Hwann-Tzong Chen. Direct voxel grid optimization: Super-fast convergence for radiance fields reconstruction. In *CVPR*, 2022. 2, 3
- [65] Matthew Tancik, Pratul P. Srinivasan, Ben Mildenhall, Sara Fridovich-Keil, Nithin Raghavan, Utkarsh Singhal, Ravi Ramamoorthi, Jonathan T. Barron, and Ren Ng. Fourier features let networks learn high frequency functions in low dimensional domains. In *NeurIPS*, 2020. 3, 30, 32, 33
- [66] Dor Verbin, Peter Hedman, Ben Mildenhall, Todd Zickler, Jonathan T. Barron, and Pratul P. Srinivasan. Ref-NeRF: Structured view-dependent appearance for neural radiance fields. In *CVPR*, 2022. 3
- [67] Huan Wang, Jian Ren, Zeng Huang, Kyle Olszewski, Menglei Chai, Yun Fu, and Sergey Tulyakov. R2L: Distilling neural radiance field to neural light field for efficient novel view synthesis. In *ECCV*, 2022. 3
- [68] Zhou Wang, Alan C. Bovik, Hamid R. Sheikh, and Eero P. Simoncelli. Image quality assessment: From error visibility to structural similarity. *IEEE Trans. Image Process.*, 13(4), 2004. 6, 9, 31
- [69] Suttisak Wizadwongsu, Pakkapon Phongthawee, Jiraphon Yenphraphai, and Supasorn Suwajanakorn. NeX: Real-time view synthesis with neural basis expansion. In *CVPR*, 2021. 2, 3
- [70] Liwen Wu, Jae Yong Lee, Anand Bhattad, Yu-Xiong Wang, and David Forsyth. DIVeR: Real-time and accurate neural radiance fields with deterministic integration for volume rendering. In *CVPR*, 2022. 2, 3
- [71] Wenqi Xian, Jia-Bin Huang, Johannes Kopf, and Changil Kim. Space-time neural irradiance fields for free-viewpoint video. In *CVPR*, 2021. 3
- [72] Yang Xue, Yuheng Li, Krishna Kumar Singh, and Yong Jae Lee. GIRAFFE HD: A high-resolution 3D-aware generative model. In *CVPR*, 2022. 3
- [73] Lior Yariv, Yoni Kasten, Dror Moran, Meirav Galun, Matan Atzmon, Basri Ronen, and Yaron Lipman. Multiview neural surface reconstruction by disentangling geometry and appearance. In *NeurIPS*, 2020. 1, 3
- [74] Alex Yu, Ruilong Li, Matthew Tancik, Hao Li, Ren Ng, and Angjoo Kanazawa. PlenOctrees for real-time rendering of neural radiance fields. In *ICCV*, 2021. 2, 3
- [75] Alex Yu, Vickie Ye, Matthew Tancik, and Angjoo Kanazawa. PixelNeRF: Neural radiance fields from one or few images. In *CVPR*, 2021. 1, 3
- [76] Kai Zhang, Gernot Riegler, Noah Snavely, and Vladlen Koltun. NeRF++: Analyzing and improving neural radiance fields. *arXiv preprint arXiv:2010.07492*, 2020. 3
- [77] 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, 31

A. Further analyses

In this appendix, the following analyses are presented:

- Appendix A.1: Effect of grouping methods
- Appendix A.2: Effect of reformulation methods
- Appendix A.3: Effect of hyperparameter
- Appendix A.4: Detailed analysis of speed-quality trade-off
- Appendix A.5: Effectiveness when increasing N
- Appendix A.6: Effectiveness for full-sized images
- Appendix A.7: Comparison with AutoInt
- Appendix A.8: Detailed analysis of application to DONeRF
- Appendix A.9: Detailed analysis of application to TensoRF

A.1. Effect of grouping methods

As discussed in Section 4.1, several methods exist for grouping the input samples when constructing a MIMO MLP. For example, when focusing on a method for grouping samples on a ray,¹¹ two opposite methods could be considered: (1) Construction of a *general* MIMO MLP that can accept any combination of samples in a ray. (2) Construction of a *specific* MIMO MLP that accepts only a group of nearby samples. This study adopts the latter method, assuming that learning a general model is more difficult than learning a specific one. This appendix examined their difference in performance to verify this statement. More precisely, we compared *MIMO-NeRF-naive*, which grouped neighboring samples on a ray, with *MIMO-NeRF-random*, which randomly grouped samples on a ray. To focus on the comparison of the grouping methods, we did not use an advanced training scheme such as self-supervised learning.

Results. Table 5 summarizes the results. We only present the image quality scores, that is, PSNR, SSIM, and LPIPS, because the difference in the grouping methods did not affect the other scores, that is, # Run, I-time, T-time, and # Params. As can be observed, MIMO-NeRF-naive outperforms MIMO-NeRF-random in all cases. These results indicated that the construction of a specific MIMO MLP was better in our experimental settings. We note that there is a possibility that a general MIMO-MLP can achieve comparable performance when using a larger-capacity model. However, in this case, the rendering speed slows down. Therefore, such a model is beyond the scope of this study.

¹¹We focused on grouping methods that can be conducted per ray for two reasons: (1) In typical NeRF training, rendering is performed for ran-

Model	N_p	Blender			LLFF		
		PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓
MIMO-NeRF-naive	2	30.18	0.944	0.065	27.31	0.860	0.167
		28.48	0.920	0.102	24.90	0.766	0.294
MIMO-NeRF-naive	4	28.62	0.927	0.091	26.29	0.824	0.218
		25.40	0.871	0.167	22.73	0.634	0.424
MIMO-NeRF-naive	8	26.34	0.895	0.133	25.10	0.774	0.284
		23.17	0.836	0.207	21.46	0.563	0.476

Table 5. Effect of grouping methods. MIMO-NeRF-naive, which groups neighboring samples on a ray, outperforms MIMO-NeRF-random, which groups samples on a ray randomly, in all the cases.

A.2. Effect of reformulation methods

In Section 5.1, for $N_p = 2^L$ ($L > 1$), we used L reformulated MIMO MLPs with

$$R^1 = 1, \dots, R^L = 2^{L-1}. \quad (11)$$

In this case, the total number of MLPs running is calculated as

$$\sum_{m=1}^L \frac{N}{N_p} R^m = N \left(\frac{1}{2^L} + \dots + \frac{1}{2} \right) = N \left(1 - \frac{1}{2^L} \right) < N. \quad (12)$$

Therefore, we can prevent a large increase in the training time compared with the original (i.e., SISO) MLP, in which the number of MLPs running is N .¹² We denote MIMO-NeRF with this reformulation method as *MIMO-NeRF-self-R1*. For further analysis, this appendix investigates other reformulation methods. In particular, when $N_p = 2$, the use of two reformulated MIMO MLPs with $R^1 = 1$ and $R^2 = 1$ is the only effective option in which the total number of MLPs running does not exceed N . Therefore, we investigated different reformulation methods for $N_p > 2$. Specifically, five reformulation methods were examined.

MIMO-NeRF-self-R2: This variant uses two reformulated MIMO MLPs with

$$R^1 = 1, R^2 = 1. \quad (13)$$

In this case, the total number of MLPs running is calculated as

$$\sum_{m=1}^2 \frac{N}{N_p} R^m = N \frac{2}{N_p} < N \text{ when } N_p > 2. \quad (14)$$

domly sampled rays. Therefore, a batch does not necessarily include near rays. (2) In NeRF, searching for near points across different rays is not trivial because points are sampled unevenly via hierarchical sampling.

¹²More strictly, when a group shift is conducted, padding is performed. In this case, the number of group shifts is added to the number of MLPs running in Equation 12. Note that the number of group shifts is equal to or smaller than the number of reformulated MIMO MLPs. Therefore, the effect was small.

Model	N_p	R	Blender				LLFF			
			PSNR↑	SSIM↑	LPIPS↓	T-time↓ (h)	PSNR↑	SSIM↑	LPIPS↓	T-time↓ (h)
NeRF	1	—	31.04	0.951	0.055	4.70	27.72	0.871	0.150	3.39
MIMO-NeRF-naive	—	—	30.18	0.944	0.065	3.09	27.31	0.860	0.167	2.12
MIMO-NeRF-distill	2	—	30.76	0.949	0.058	9.46	27.50	0.863	0.169	6.81
MIMO-NeRF-self	$R^1 = 1, R^2 = 1$	—	31.26	0.953	0.054	5.36	27.70	0.870	0.155	3.97
MIMO-NeRF-naive	—	—	28.62	0.927	0.091	2.02	26.29	0.824	0.218	1.57
MIMO-NeRF-distill	—	—	30.22	0.946	0.065	8.42	27.37	0.861	0.172	6.25
MIMO-NeRF-self-R1	4	$R^1 = 1, R^2 = 2$	30.94	0.950	0.058	4.68	27.51	0.865	0.162	3.44
MIMO-NeRF-self-R2	—	$R^1 = 1, R^2 = 1$	30.95	0.950	0.060	3.65	27.35	0.861	0.169	2.70
MIMO-NeRF-self-R3	—	$R^1 = 1, R^2 = 1, R^3 = 1$	30.89	0.949	0.060	5.05	27.27	0.860	0.171	3.78
MIMO-NeRF-naive	—	—	26.34	0.895	0.133	1.66	25.10	0.774	0.284	1.24
MIMO-NeRF-distill	—	—	29.39	0.937	0.075	8.07	27.01	0.851	0.184	5.91
MIMO-NeRF-self-R1	—	$R^1 = 1, R^2 = 2, R^3 = 4$	30.40	0.945	0.065	5.86	26.97	0.851	0.180	4.43
MIMO-NeRF-self-R2	8	$R^1 = 1, R^2 = 1$	30.02	0.940	0.076	2.61	26.52	0.833	0.207	2.13
MIMO-NeRF-self-R3	—	$R^1 = 1, \dots, R^7 = 1$	29.88	0.937	0.080	7.75	25.66	0.797	0.243	5.97
MIMO-NeRF-self-R4	—	$R^1 = 1, R^2 = 2$	29.86	0.939	0.077	3.33	26.61	0.836	0.205	2.41
MIMO-NeRF-self-R5	—	$R^1 = 1, R^2 = 4$	29.81	0.939	0.076	4.38	26.61	0.838	0.202	3.37
MIMO-NeRF-self-R6	—	$R^1 = 1, R^2 = 1, R^3 = 1$	30.11	0.941	0.074	3.76	26.41	0.830	0.208	2.73

Table 6. Effect of reformulation methods. We examined the PSNR, SSIM, LPIPS, and T-time scores when changing the reformulation methods. MIMO-NeRF-self-R1, which is used in the main experiments, achieves the best or comparable performance in terms of PSNR, SSIM, and LPIPS. Other variants are outperformed by it in most cases; however, some of them, e.g., MIMO-NeRF-self-R2 with $N_p = 4$ on the Blender and LLFF datasets and MIMO-NeRF-self-R2/R6 with $N_p = 8$ on the Blender dataset, achieve a performance comparable with that of MIMO-NeRF-distill while achieving faster training than the original NeRF.

MIMO-NeRF-self-R3: This variant employs $N_p - 1$ reformulated MIMO MLPs with

$$R^1 = 1, \dots, R^{N_p-1} = 1. \quad (15)$$

In this case, the total number of MLPs running is calculated as

$$\sum_{m=1}^{N_p-1} \frac{N}{N_p} R^m = N \frac{N_p - 1}{N_p} < N. \quad (16)$$

MIMO-NeRF-self-R4: This variant adopts two reformulated MIMO MLPs with

$$R^1 = 1, R^2 = 2. \quad (17)$$

In this case, the total number of MLPs running is calculated as

$$\sum_{m=1}^2 \frac{N}{N_p} R^m = N \frac{3}{N_p} < N \text{ when } N_p > 3. \quad (18)$$

This method is the same as MIMO-NeRF-self-R1 when $N_p = 4$.

MIMO-NeRF-self-R5: This variant uses two reformulated MIMO MLPs with

$$R^1 = 1, R^2 = \frac{N_p}{2}. \quad (19)$$

In this case, the total number of MLPs running is calculated as

$$\sum_{m=1}^2 \frac{N}{N_p} R^m = N \left(\frac{1}{2} + \frac{1}{N_p} \right) < N \text{ when } N_p > 2. \quad (20)$$

This method is the same as MIMO-NeRF-self-R1 when $N_p = 4$.

MIMO-NeRF-self-R6: This variant uses three reformulated MIMO MLPs with

$$R^1 = 1, R^2 = 1, R^3 = 1. \quad (21)$$

In this case, the total number of MLPs running is calculated as

$$\sum_{m=1}^3 \frac{N}{N_p} R^m = N \frac{3}{N_p} < N \text{ when } N_p > 3. \quad (22)$$

This method is identical to MIMO-NeRF-self-R3 when $N_p = 4$.

Results. Table 6 summarizes the results. Our findings were as follows:

MIMO-NeRF-self-R2 vs. MIMO-NeRF-self-R3 vs. MIMO-NeRF-self-R6. For these variants, the same variation reduction methods (i.e., $R^m = 1$) are used, whereas the numbers of reformulated MIMO MLPs (i.e., M) are different. We found that too many reformulated MIMO MLPs (i.e., MIMO-NeRF-self-R3) did not necessarily achieve the best performance. A possible reason for this is that an excessive number of constraints causes statistical averaging and deteriorates the image quality. As M increased, the training time increased. Therefore, the results suggest that the use of the MIMO-NeRF with a moderate value of M is preferable.

MIMO-NeRF-self-R2 vs. MIMO-NeRF-self-R4 vs. MIMO-NeRF-self-R5. In these variants, the number of reformulated MIMO-MLPs is the same (i.e., $M = 2$), whereas different reduction methods are used. We observed different

tendencies in the results of the Blender dataset and those for the LLFF dataset. In the Blender dataset, PSNR, SSIM, and LPIPS improved as R^2 decreased, whereas, in the LLFF dataset, they improved as R^2 increased. Although not the same, similar tendencies exist between MIMO-NeRF-self-R1 and MIMO-NeRF-self-R2 when $N_p = 4$. The results indicate that the variation reduction is more effective for the LLFF dataset, which includes forward-facing views, than for the Blender dataset, which contains 360° views when $M = 2$. However, it is noteworthy that MIMO-NeRF-self-R1 outperformed MIMO-NeRF-self-R6 on both datasets. These results indicate that variation reduction is effective for both datasets when M is sufficiently large. Delving deeper into these differences will be an interesting topic for future research.

MIMO-NeRF-self-R1 vs. the others. MIMO-NeRF-self-R1 achieved the best or comparable performance in terms of the image quality metrics, that is, PSNR, SSIM, and LPIPS, in all cases. We note that some other variants, such as MIMO-NeRF-self-R2 with $N_p = 4$ on the Blender and LLFF datasets and MIMO-NeRF-self-R2/R6 with $N_p = 8$ on the Blender dataset, are worse than MIMO-NeRF-self-R1 in terms of all or some of the image quality metrics, but are comparable with MIMO-NeRF-distill while having a shorter training time than NeRF. The results suggest the possibility of obtaining a reasonable quality and fast-inference model with a shorter training time by tuning the reformulation configurations.

A.3. Effect of hyperparameter

In the experiments presented in Sections 5.1–5.3, we set hyperparameter λ to 1 and 0.4 for the Blender and LLFF datasets, respectively. To analyze the effect of this hyperparameter, we examined the quantitative scores when varying λ within $\{0.4, 1\}$.

Results. Table 7 presents the results. We present only the image quality scores because the modification of λ does not affect the other scores, i.e., # Run, I-time, T-time, and # Params. As can be observed, MIMO-NeRF is sensitive to λ , and in all cases, it achieved the best performance when using the values utilized in the experiments presented in Sections 5.1–5.3 (i.e., $\lambda = 1$ for the Blender dataset and $\lambda = 0.4$ for the LLFF dataset). However, the difference is relatively small, and the scores in the worst case are still comparable to those of MIMO-NeRF-distill (Table 6). Therefore, we consider that this sensitivity is within an allowable range if $\lambda \in [0.4, 1]$.

A.4. Detailed analysis of speed-quality trade-off

In Section 5.2, we present the relationship between FLOPs and PSNR as a method to demonstrate the trade-off between speed and quality. For a detailed analysis, this

N_p	λ	Blender			LLFF		
		PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓
2	0.4	31.20	0.952	0.054	27.70	0.870	0.155
	1.0	31.26	0.953	0.054	27.56	0.866	0.160
4	0.4	30.93	0.950	0.058	27.51	0.865	0.162
	1.0	30.94	0.950	0.058	27.40	0.863	0.165
8	0.4	30.22	0.944	0.067	26.97	0.851	0.180
	1.0	30.40	0.945	0.065	26.83	0.848	0.184

Table 7. Effect of hyperparameter λ . MIMO-NeRF is sensitive to λ ; however, the difference is relatively small, and the scores in the worst case are still comparable to those of MIMO-NeRF-distill (Table 6).

appendix provides other relationships, including those between FLOPs/inference time and PSNR/SSIM/LPIPS.

Comparison models. In Section 5.2, we compared *MIMO-NeRF-self* with two possible alternatives: *NeRF-few*, which reduced the number of samples on a ray, and *NeRF-small*, which reduced the number of features in the hidden layers. In particular, we adjusted the parameters so that their FLOPs in inference were comparable to those of MIMO-NeRF-self. We describe the details of these models in Appendix B.1.2. As discussed in the footnote,⁸ an unignorable difference between MIMO-NeRF-self, MIMO-NeRF-few, and MIMO-NeRF-small is the difference in the calculation cost during training. Because MIMO-NeRF-self uses multiple reformulated MIMO MLPs during training, the calculation cost is higher than that of NeRF-small and NeRF-few. To confirm this effect, we examined the performance of NeRF-few and NeRF-small when increasing the batch size such that the calculation cost became almost the same as that of MIMO-NeRF-self. These variants are referred to as *NeRF-small+* and *NeRF-few+*. Furthermore, to confirm whether the proposed self-supervised learning was more effective than a simple increase in the batch size, we examined *MIMO-NeRF-naive+*, where we increased the batch size, similar to NeRF-small+ and NeRF-few+. More precisely, when $N_p = 2$, we used two reformulated MIMO MLPs with $R^1 = 1$ and $R^2 = 1$ for MIMO-NeRF-self. In this case, # Run was twice that of MIMO-NeRF-naive.¹³ Therefore, we increased the batch size twice for NeRF-few+ and NeRF-small+. Similarly, when compared to MIMO-NeRF-self with $N_p = 4$, we increased the batch size three times, and when compared to MIMO-NeRF-self with $N_p = 8$, we increased the batch size seven times.

Results. Figure 7 presents the relationship between FLOPs/inference time and PSNR/SSIM/LPIPS. We can observe that in most cases, MIMO-NeRF-self achieves a better trade-off between speed and quality in terms of every relationship than not only MIMO-NeRF-naive, NeRF-few,

¹³More strictly, # Run increases more when a group shift is conducted because padding is performed. In this case, the number of group shifts, which is equal to or smaller than the number of reformulated MIMO MLPs, is added to # Run. However, this was relatively small compared to the number of samples. Therefore, we ignore its effect here.

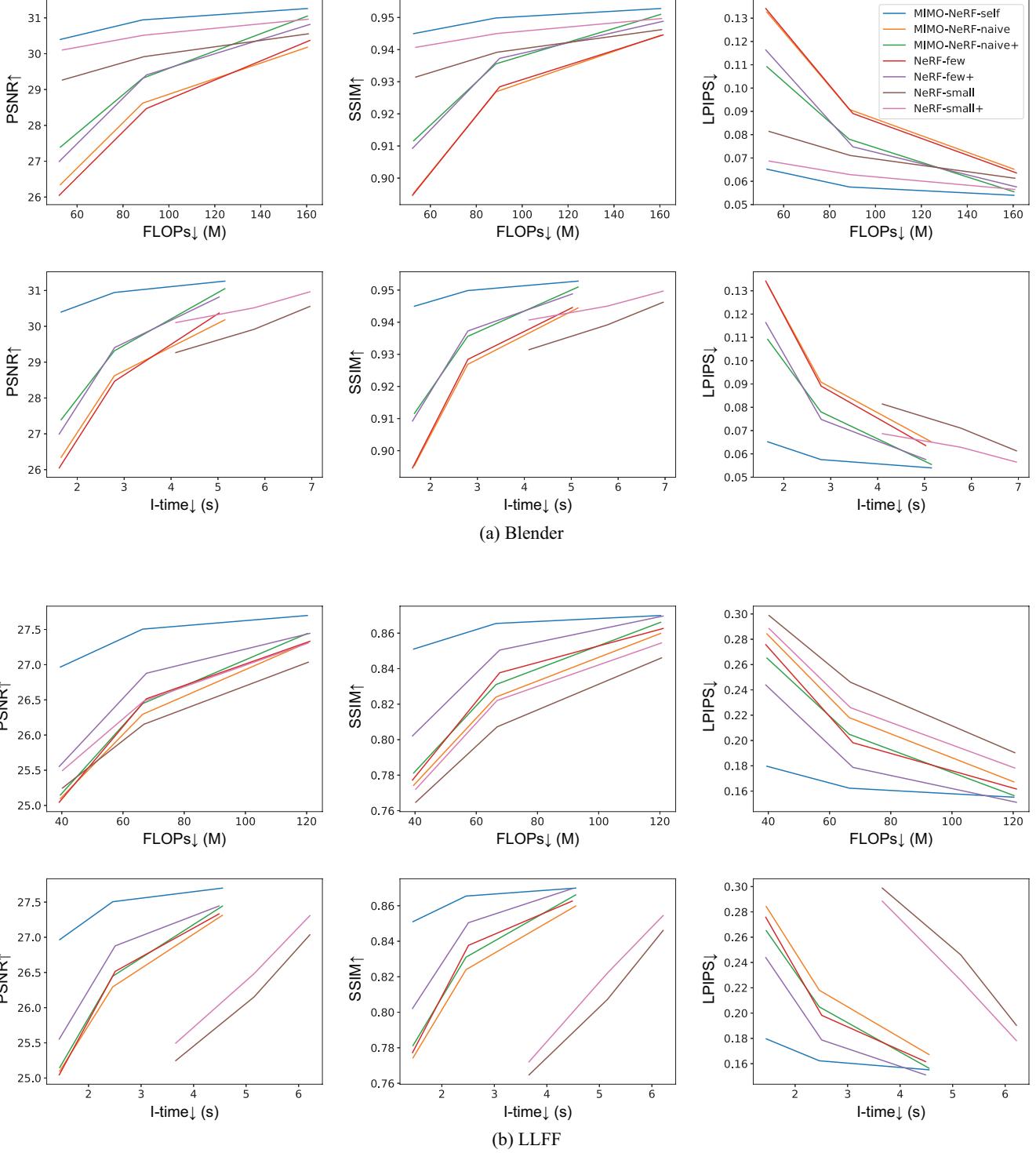


Figure 7. Relationships between FLOPs/inference time and PSNR/SSIM/LPIPS. The legend is provided in the upper right figure. In the “FLOPs” axis, the more to the left, the lower the calculation cost. In the “I-time” axis, the more to the left, the faster the inference. In the “PSNR” and “SSIM” axis, the more to the upper side, the better the image quality. In the “LPIPS” axis, the more to the lower side, the better the image quality. MIMO-NeRF-self (blue line) achieved the best trade-off between speed and quality in almost all cases.

Model	N	N_p	PSNR↑	SSIM↑	LPIPS↓	FLOPs (M)
NeRF	256	1	31.04	0.951	0.055	303.82
MIMO-NeRF-self ($N_p = 2$)	360	2	31.59	0.955	0.050	300.63
MIMO-NeRF-self ($N_p = 4$)	648	4	31.65	0.956	0.049	298.99

Table 8. Effectiveness when increasing N . We compared NeRF and MIMO-NeRF-self when the FLOPs are almost the same. We evaluated the models on the Blender dataset. All the scores become better as N and N_p increase.

and NeRF-small, which are presented in Sections 5.1 and 5.2, but also MIMO-NeRF-naive+, NeRF-few+, and NeRF-small+, which are trained under better conditions. These results strengthen our statement in the main text, that is, MIMO-NeRF-self achieves a better trade-off between speed and quality than the possible alternatives.

A.5. Effectiveness when increasing N

In the main experiments, we investigated the performance of MIMO-NeRF when the number of samples (i.e., N) is fixed. An interesting question is how MIMO-NeRF works well when increasing N within the range in which its FLOPs are comparable to those of the original NeRF. We conducted an additional experiment to answer this question.

Results. Table 8 presents the results. The models were evaluated using the Blender dataset. It can be seen that MIMO-NeRF-self outperforms NeRF in terms of all metrics, and all scores improve as N and N_p increase. The results indicate that tuning not only N but also N_p is important for obtaining the best performance under the same computational budget.

A.6. Effectiveness for full-sized images

In Sections 5.1–5.3, half-sized images are used to better investigate the various configurations. This appendix examines the effectiveness of MIMO-NeRF for full-sized images to verify whether the same conclusion holds independently of the image size. In particular, we investigate the benchmark performance for full-sized images using a protocol similar to that described in Section 5.1.

Quantitative results. Table 9 summarizes the results for all the metrics (i.e., PSNR, SSIM, LPIPS, # Run, I-time, T-time, and # Params). Table 10 lists PSNR, SSIM, and LPIPS for each scene. Similar to the analysis conducted in Section 5.1, we analyze the results from three perspectives:

Image quality. Similar to the results for half-sized images, MIMO-NeRF-self outperformed MIMO-NeRF-self-naive but also MIMO-NeRF-self-distill in most cases in terms of PSNR, SSIM, and LPIPS. Even MIMO-NeRF-self suffers from a trade-off between speed and quality as N_p increases; however, MIMO-NeRF-self is comparable to the original NeRF when $N_p = 2$.

Inference speed. Similar to the results for half-sized images, all MIMO-NeRFs improved the inference time by 1.83–5.77 times as N_p increased.

Training speed. Similar to the results for half-sized images, MIMO-NeRF-naive achieved the fastest training because it used only a single MIMO formulation during training. MIMO-NeRF-self increases the training time owing to the introduction of multiple reformulated MIMO MLPs; however, each calculation cost is lower than that of a SISO MLP in the original NeRF. Therefore, it does not suffer from a large increase in training time compared with MIMO-NeRF-distill, which requires the training of two networks, that is, a SISO-NeRF and a MIMO-NeRF.

Summary. From these results, we found that when $N_p = 2$, MIMO-NeRF-self improves the inference speed of NeRF while retaining the image quality, and when N_p is larger, MIMO-NeRF-self suffers from a trade-off between speed and quality; however, it achieves better image quality with a shorter training time than MIMO-NeRF-distill. These tendencies are the same as those for the half-sized images.

Qualitative results. Figures 8 and 9 present the qualitative results for the Blender and LLFF datasets, respectively. Examples of the synthesized videos are provided on the [project page](#).¹

Model	N_p	Blender							LLFF						
		PSNR↑	SSIM↑	LPIPS↓	# Run↓	I-time↓ (s)	T-time↓ (h)	# Params (M)	PSNR↑	SSIM↑	LPIPS↓	# Run↓	I-time↓ (s)	T-time↓ (h)	# Params (M)
NeRF [41]	1	30.94	0.946	0.070	256	38.22	12.54	1.19	26.45	0.811	0.249	256	45.46	16.22	1.19
MIMO-NeRF-naive		29.36	0.932	0.091	128	20.67	8.61	1.26	26.00	0.796	0.269	128	24.82	8.67	1.26
MIMO-NeRF-distill 2		30.55	0.943	0.077	128	20.67	25.27	1.26	26.21	0.799	0.278	128	24.82	30.79	1.26
MIMO-NeRF-self		31.01	0.947	0.071	128	20.67	14.13	1.26	26.46	0.812	0.253	128	24.82	17.51	1.26
MIMO-NeRF-naive		27.72	0.914	0.114	64	11.17	5.95	1.39	25.09	0.758	0.320	64	13.47	4.87	1.39
MIMO-NeRF-distill 4		30.01	0.939	0.083	64	11.17	22.58	1.39	26.14	0.798	0.279	64	13.47	26.97	1.39
MIMO-NeRF-self		30.66	0.944	0.075	64	11.17	12.37	1.39	26.35	0.809	0.258	64	13.47	14.52	1.39
MIMO-NeRF-naive		25.78	0.889	0.145	32	6.62	5.08	1.65	24.15	0.716	0.376	32	8.01	3.25	1.65
MIMO-NeRF-distill 8		28.85	0.929	0.095	32	6.62	21.74	1.65	25.91	0.793	0.285	32	8.01	25.34	1.65
MIMO-NeRF-self		29.92	0.938	0.084	32	6.62	14.95	1.65	25.99	0.800	0.270	32	8.01	19.16	1.65
NeRF [41]	1	31.01	0.947	0.081	—	—	—	—	26.50	0.811	0.250	—	—	—	—

Table 9. Benchmark performance of MIMO-NeRFs for full-sized images. The scores for the model with citation [41] are taken from another report [41]. We provide them as references. The other scores were calculated in our environment. We implemented all the models based on the commonly-used source code of NeRF. See Appendix B.1 for the implementation details. The PSNR, SSIM, and LPIPS for each scene are provided in Table 10.

Model	N_p	Blender							PSNR↑										
		Chair	Drums	Ficus	Hotdog	Lego	Materials	Mic	Ship	Avg.	Fern	Flower	Fortress	Horns	Laves	Orchids	Room	T-Rex	Avg.
NeRF	1	32.82	25.04	30.10	36.28	32.60	29.63	32.77	28.32	30.94	24.99	27.57	31.16	27.33	20.96	20.35	32.57	26.64	26.45
MIMO-NeRF-naive		31.82	24.42	25.59	35.72	30.86	27.13	31.50	27.88	29.36	24.76	27.50	30.75	26.68	20.88	20.27	31.62	25.49	26.00
MIMO-NeRF-distill 2		32.35	25.10	29.78	35.37	31.74	29.43	32.74	27.86	30.55	24.97	27.39	30.59	26.87	20.89	20.52	32.17	26.29	26.21
MIMO-NeRF-self		32.92	25.17	29.49	36.10	32.60	30.06	33.18	28.53	31.01	25.05	27.42	31.24	27.24	21.00	20.49	32.52	26.72	26.46
MIMO-NeRF-naive		29.21	23.06	25.39	34.20	28.33	26.18	28.93	26.48	27.72	24.31	27.01	29.98	25.41	20.11	19.81	29.96	24.11	25.09
MIMO-NeRF-distill 4		32.07	24.75	27.85	35.20	31.20	29.26	32.20	27.53	30.01	24.87	27.40	30.60	26.79	20.86	20.44	32.06	26.07	26.14
MIMO-NeRF-self		32.84	24.82	28.44	36.18	32.29	29.87	32.66	28.19	30.66	24.95	27.52	31.24	27.25	20.98	20.47	32.37	26.03	26.35
MIMO-NeRF-naive		27.17	21.33	23.38	32.01	25.32	25.16	27.25	24.60	25.78	23.06	26.06	28.76	24.51	19.85	18.79	29.11	23.10	24.15
MIMO-NeRF-distill 8		31.40	23.61	25.38	34.78	29.48	28.90	30.58	26.70	28.85	24.54	27.32	30.56	26.54	20.80	20.23	31.71	25.55	25.91
MIMO-NeRF-self		32.37	24.18	26.91	35.64	31.17	29.96	31.59	27.58	29.92	24.58	27.57	31.13	26.66	20.82	20.23	31.84	25.12	25.99
NeRF [41]	1	33.00	25.01	30.13	36.18	32.54	29.62	32.91	28.65	31.01	25.17	27.40	31.16	27.45	20.92	20.36	32.70	26.80	26.50

Model	N_p	Blender							SSIM↑										
		Chair	Drums	Ficus	Hotdog	Lego	Materials	Mic	Ship	Avg.	Fern	Flower	Fortress	Horns	Laves	Orchids	Room	T-Rex	Avg.
NeRF [41]	1	0.966	0.924	0.962	0.975	0.962	0.949	0.980	0.852	0.946	0.790	0.832	0.881	0.826	0.690	0.644	0.951	0.878	0.811
MIMO-NeRF-naive		0.957	0.914	0.918	0.973	0.949	0.921	0.973	0.847	0.932	0.781	0.825	0.863	0.799	0.684	0.625	0.944	0.847	0.796
MIMO-NeRF-distill 2		0.962	0.926	0.960	0.969	0.954	0.949	0.980	0.844	0.943	0.783	0.818	0.855	0.802	0.680	0.640	0.947	0.869	0.799
MIMO-NeRF-self		0.967	0.925	0.958	0.975	0.962	0.954	0.982	0.853	0.947	0.791	0.827	0.882	0.822	0.695	0.646	0.950	0.881	0.812
MIMO-NeRF-naive		0.927	0.891	0.919	0.964	0.920	0.913	0.956	0.822	0.914	0.758	0.801	0.824	0.742	0.630	0.589	0.923	0.801	0.758
MIMO-NeRF-distill 4		0.959	0.920	0.947	0.968	0.950	0.947	0.977	0.840	0.939	0.780	0.819	0.857	0.803	0.678	0.636	0.946	0.866	0.798
MIMO-NeRF-self		0.967	0.921	0.949	0.975	0.960	0.953	0.979	0.850	0.944	0.787	0.830	0.882	0.823	0.693	0.644	0.948	0.869	0.809
MIMO-NeRF-naive		0.900	0.858	0.891	0.951	0.876	0.898	0.946	0.794	0.889	0.701	0.760	0.771	0.700	0.610	0.528	0.907	0.754	0.716
MIMO-NeRF-distill 8		0.953	0.905	0.923	0.966	0.940	0.944	0.972	0.830	0.929	0.772	0.818	0.856	0.797	0.677	0.625	0.943	0.853	0.793
MIMO-NeRF-self		0.963	0.911	0.934	0.973	0.952	0.953	0.975	0.842	0.938	0.773	0.828	0.879	0.809	0.686	0.629	0.944	0.848	0.800
NeRF [41]	1	0.967	0.925	0.964	0.974	0.961	0.949	0.980	0.856	0.947	0.792	0.827	0.881	0.828	0.690	0.641	0.948	0.880	0.811

Model	N_p	Blender							LPIPS↓										
		Chair	Drums	Ficus	Hotdog	Lego	Materials	Mic	Ship	Avg.	Fern	Flower	Fortress	Horns	Laves	Orchids	Room	T-Rex	Avg.
NeRF [41]	1	0.046	0.091	0.044	0.121	0.050	0.063	0.028	0.206	0.081	0.280	0.219	0.171	0.268	0.316	0.321	0.178	0.249	0.250
MIMO-NeRF-naive		0.057	0.107	0.108	0.047	0.067	0.103	0.035	0.205	0.091	0.294	0.222	0.201	0.303	0.319	0.338	0.188	0.284	0.269
MIMO-NeRF-distill 2		0.052	0.091	0.049	0.057	0.060	0.062	0.024	0.220	0.077	0.311	0.247	0.225	0.318	0.327	0.333	0.189	0.274	0.278
MIMO-NeRF-self		0.045	0.091	0.055	0.046	0.050	0.057	0.023	0.203	0.071	0.283	0.225	0.175	0.284	0.311	0.319	0.176	0.252	0.253
MIMO-NeRF-naive		0.088	0.141	0.104	0.062	0.110	0.107	0.063	0.239	0.114	0.321	0.255	0.272	0.377	0.374	0.386	0.236	0.336	0.320
MIMO-NeRF-distill 4		0.054	0.100	0.069	0.058	0.066	0.064	0.027	0.223	0.083	0.311	0.245	0.222	0.317	0.329	0.339	0.192	0.276	0.279
MIMO-NeRF-self		0.045	0.098	0.069	0.046	0.052	0.059	0.026	0.205	0.075	0.288	0.221	0.176	0.284	0.314	0.325	0.182	0.272	0.258
MIMO-NeRF-naive		0.112	0.181	0.134	0.096	0.163	0.123	0.079	0.273	0.145	0.386	0.323	0.348	0.427	0.402	0.445	0.281	0.396	0.376
MIMO-NeRF-distill 8		0.060	0.123	0.095	0.061	0.080	0.069	0.037	0.236	0.095	0.318	0.245	0.222	0.322	0.334	0.352	0.201	0.289	0.285
MIMO-NeRF-self		0.048	0.114	0.089	0.050	0.066	0.061	0.033	0.215	0.084	0.303	0.227	0.179	0.301	0.323	0.345	0.190	0.292	0.270
NeRF [41]	1	0.046	0.091	0.044	0.121	0.050	0.063	0.028	0.206	0.081	0.280	0.219	0.171	0.268	0.316	0.321	0.178	0.249	0.250

Table 10. Comparison of PSNR, SSIM, and LPIPS for each scene on the Blender and LLFF datasets with full-sized images. The scores for the model with citation [41] are taken from another report [41]. We provide them as references. The other scores were calculated in our environment. We implemented all the models based on the commonly-used source code of NeRF. See Appendix B.1 for the implementation details. The scores for the other metrics are provided in Table 9.

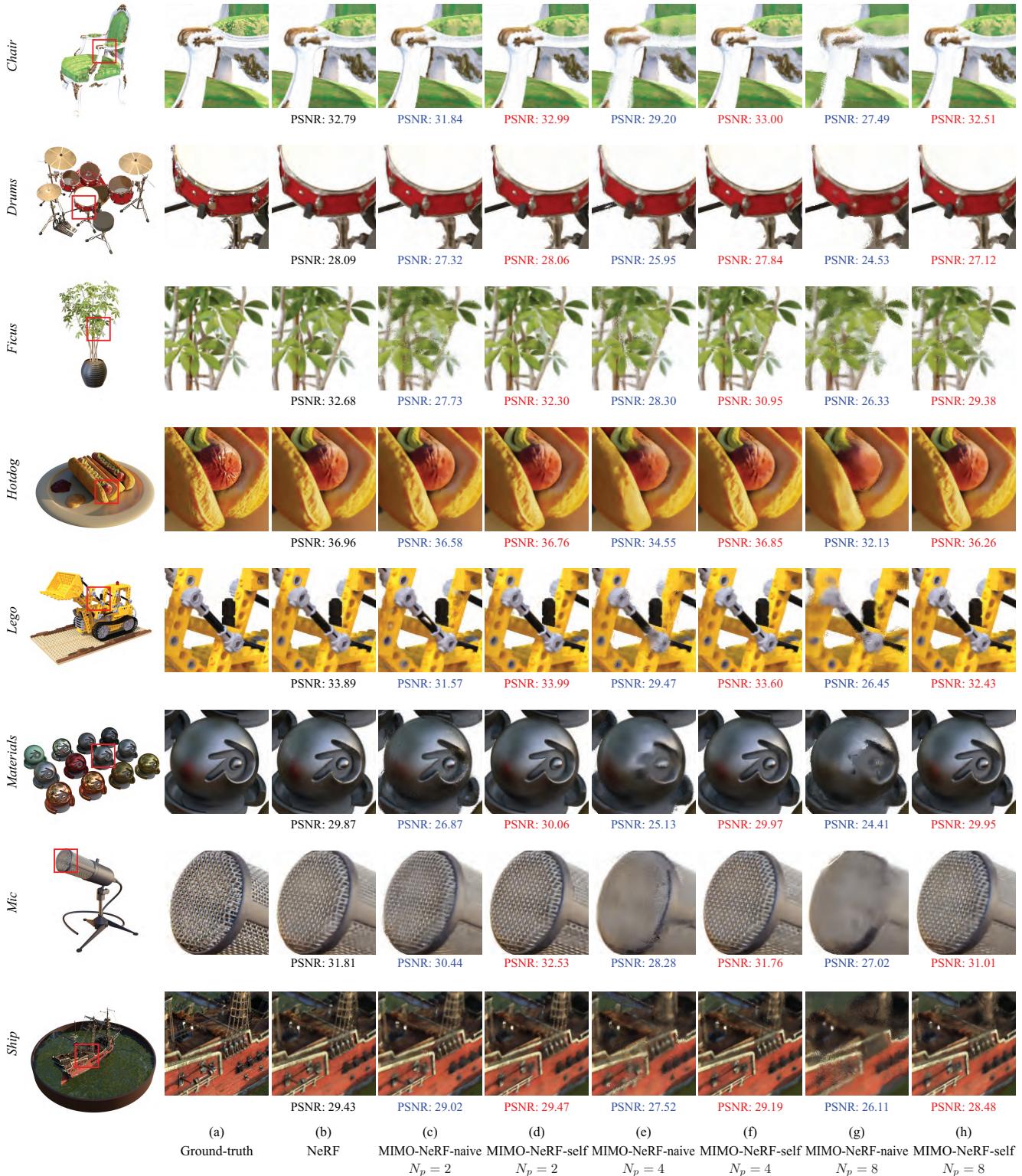


Figure 8. Qualitative comparison between NeRF, MIMO-NeRF-naive, and MIMO-NeRF-self on the Blender dataset. This figure is an extension of Figure 5. We report PSNR for the displayed view. The average scores for all views are presented in Table 10. As shown in (c), (e), and (g), the deterioration of image quality becomes obvious in MIMO-NeRF-naive as N_p increases. In contrast, MIMO-NeRF-self is resistant to this deterioration, as shown in (d), (f), and (h).

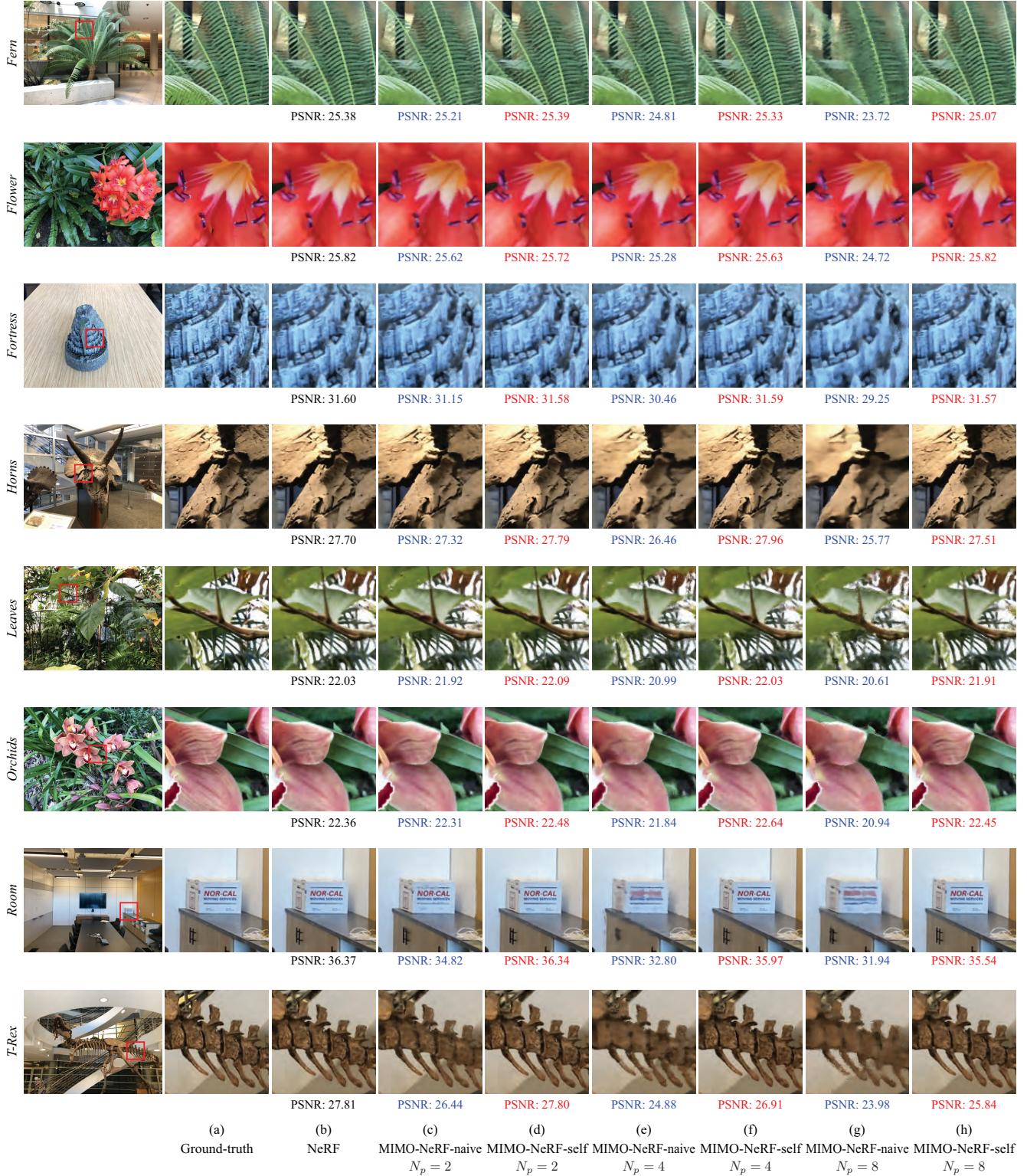


Figure 9. Qualitative comparison between NeRF, MIMO-NeRF-naive, and MIMO-NeRF-self on the LLFF dataset. This figure is an extension of Figure 5. We report PSNR for the displayed view. The average scores for all views are listed in Table 10. As shown in (c), (e), and (g), the deterioration of image quality becomes obvious in MIMO-NeRF-naive as N_p increases. In contrast, MIMO-NeRF-self is robust against this deterioration, as shown in (d), (f), and (h).

A.7. Comparison with AutoInt

To further clarify the utility of MIMO-NeRF, we compared it with AutoInt [31], which reduces the number of MLPs running (# Run) using an integral network that calculates the colors and volume densities *per segment* instead of *per point*. In particular, we investigated the difference in performance between MIMO-NeRF-self and AutoInt when # Run was the same.

Results. Table 11 summarizes these results. The model was evaluated using the Blender dataset (full-size images). It can be observed that MIMO-NeRF-self outperformed AutoInt in most cases. Another important difference is that AutoInt requires the use of a specific and complex grad network during training, whereas MIMO-NeRF can be trained using a standard network such as that implemented using PyTorch.

Model	# Run↓	PSNR↑	SSIM↑	LPIPS↓
AutoInt ($N = 32$) [31]	32	26.83	0.926	0.151
MIMO-NeRF-self ($N_p = 8$)	32	29.92	0.938	0.084
AutoInt ($N = 16$) [31]	16	26.04	0.916	0.167
MIMO-NeRF-self ($N_p = 16$)	16	28.69	0.925	0.099
AutoInt ($N = 8$) [31]	8	25.55	0.911	0.170
MIMO-NeRF-self ($N_p = 32$)	8	27.19	0.908	0.118

Table 11. Comparison of AutoInt and MIMO-NeRF-self. We compared AutoInt and MIMO-NeRF-self when # Run was the same. We evaluated the models on the Blender dataset (full-sized images). The scores for AutoInt are taken from the AutoInt paper [31]. In most cases, MIMO-NeRF-self outperforms AutoInt.

A.8. Detailed analysis of application to DONeRF

In Section 5.4, we compared MIMO-DONeRF-16/4-naive and MIMO-DONeRF-16/4-self with *DONeRF-16*, in which the number of selected samples (N_s) is the same as that of MIMO-DONeRF-16/4-naive and MIMO-DONeRF-16/4-self (i.e., $N_s = 16$), and *DONeRF-4*, in which the number of MLPs running (# Run) is the same as that of MIMO-DONeRF-16/4-naive and MIMO-DONeRF-16/4-self (i.e., # Run = 5). For further analysis, this appendix provides a comparison with *DONeRF-11*, in which the training time (T-time) is almost the same as that of MIMO-DONeRF-16/4-self, and *DONeRF-5*, in which the inference time (I-time) is close to (more strictly, slightly longer than) that of MIMO-DONeRF-16/4-naive and MIMO-DONeRF-16/4-self. We evaluated the models using the same metrics as those described in Section 5.4.

Quantitative results. Table 12 summarizes the results for all metrics. Table 13 lists the PSNR and FLIP for each scene. Our findings are as follows:

MIMO-DONeRF-16/4-naive vs. DONeRF-5 (close I-time). We found that MIMO-DONeRF-16/4-naive outperformed or was comparable to DONeRF-5 in terms of PSNR and FLIP for all scenes. MIMO-DONeRF-16/4-naive also slightly outperformed DONeRF-5 in terms of the I-time and T-time. Therefore, MIMO-DONeRF-16/4-naive does not have any disadvantages compared to MIMO-DONeRF-5.

MIMO-DONeRF-16/4-self vs. DONeRF-11 (close T-time). MIMO-DONeRF-16/4-self and DONeRF-11 were comparable in terms of average PSNR and FLIP, and whether they were better or worse depended on the view and metrics. Although T-time was almost the same between these two models, MIMO-DONeRF-16/4 outperforms DONeRF-11 significantly in terms of I-time (approximately half of it). Overall, MIMO-DONeRF-16/4-self is better than DONeRF-11 in terms of significantly better I-time.

Summary. Even when considering the models in which I-time is close to that of MIMO-DONeRFs and T-time is close to that of MIMO-DONeRF-self, the results indicate that MIMO-DONeRFs have advantages. As discussed in Section 5.4, the results suggest that an increase in N_p (i.e., the replacement of the SISO MLP by the MIMO MLP) can be used as a better alternative to a reduction in N_s (the number of selected samples) when seeking a better trade-off between speed and quality.

Qualitative results. Figure 10 shows the qualitative results. Examples of the synthesized videos are provided on the [project page](#).¹

Model	N_s	N_p	PSNR↑	FLIP↓	# Run↓	I-time↓ (s)	T-time↓ (h)	# Params (M)
DONeRF-4	4	1	31.21	0.070	5	0.140	3.23	0.94
DONeRF-5	5	1	31.65	0.067	6	0.164	3.29	0.94
DONeRF-11	11	1	32.76	0.063	12	0.304	3.57	0.94
DONeRF-16	16	1	33.06	0.061	17	0.429	3.79	0.94
MIMO-DONeRF-16/4-naive	16	4	32.30	0.063	5	0.155	3.26	0.99
MIMO-DONeRF-16/4-self	16	4	32.72	0.061	5	0.155	3.56	0.99
DONeRF-4 [44]	4	1	31.14	0.071	—	—	—	—
DONeRF-16 [44]	16	1	33.03	0.062	—	—	—	—

Table 12. Comparison of quantitative scores between DONeRFs and MIMO-DONeRFs. The scores for the model with citation [44] are taken from another report [44]. We provide them as references. The other scores were calculated in our environment. We implemented all the models based on the official DONeRF source code. See Appendix B.2 for the implementation details. This table is an extended version of Table 3. In addition to the scores provided in Table 3, this table provides the scores for DONeRF-5, in which I-time is close to those of MIMO-DONeRF-16/4-naive and MIMO-DONeRF-16/4-self, and DONeRF-11, in which T-time is close to that of MIMO-DONeRF-16/4-self. The PSNR and FLIP for each scene are presented in Table 13.

Model	N_s	N_p	PSNR↑						Avg.
			Barbershop	Bulldozer	Classroom	Forest	Pavillon	San Miguel	
DONeRF-4	4	1	30.76	33.29	34.03	30.90	31.02	27.24	31.21
DONeRF-5	5	1	31.14	34.33	34.52	31.12	31.22	27.58	31.65
DONeRF-11	11	1	31.90	36.37	35.90	31.80	31.60	28.98	32.76
DONeRF-16	16	1	32.13	36.87	36.15	31.79	31.71	29.71	33.06
MIMO-DONeRF-16/4-naive	16	4	31.60	35.14	35.19	31.61	32.50	27.77	32.30
MIMO-DONeRF-16/4-self	16	4	32.11	35.57	35.65	31.76	32.80	28.41	32.72
DONeRF-4 [44]	4	1	30.84	33.46	33.43	30.63	31.07	27.41	31.14
DONeRF-16 [44]	16	1	32.15	36.98	36.27	31.32	31.79	29.67	33.03

Model	N_s	N_p	FLIP↓						Avg.
			Barbershop	Bulldozer	Classroom	Forest	Pavillon	San Miguel	
DONeRF-4	4	1	0.064	0.044	0.053	0.075	0.099	0.083	0.070
DONeRF-5	5	1	0.064	0.040	0.051	0.074	0.097	0.078	0.067
DONeRF-11	11	1	0.059	0.033	0.047	0.071	0.096	0.070	0.063
DONeRF-16	16	1	0.058	0.032	0.047	0.072	0.095	0.065	0.061
MIMO-DONeRF-16/4-naive	16	4	0.059	0.036	0.049	0.071	0.087	0.078	0.063
MIMO-DONeRF-16/4-self	16	4	0.056	0.035	0.045	0.072	0.086	0.072	0.061
DONeRF-4 [44]	4	1	0.065	0.048	0.058	0.077	0.098	0.080	0.071
DONeRF-16 [44]	16	1	0.059	0.036	0.045	0.074	0.094	0.065	0.062

Table 13. Comparison of PSNR and FLIP for each scene between DONeRFs and MIMO-DONeRFs. The scores for the model with citation [44] are taken from another report [44]. We provide them as references. The other scores were calculated in our environment. We implemented all the models based on the official DONeRF source code. See Appendix B.2 for the implementation details. The scores for the other metrics are listed in Table 12.

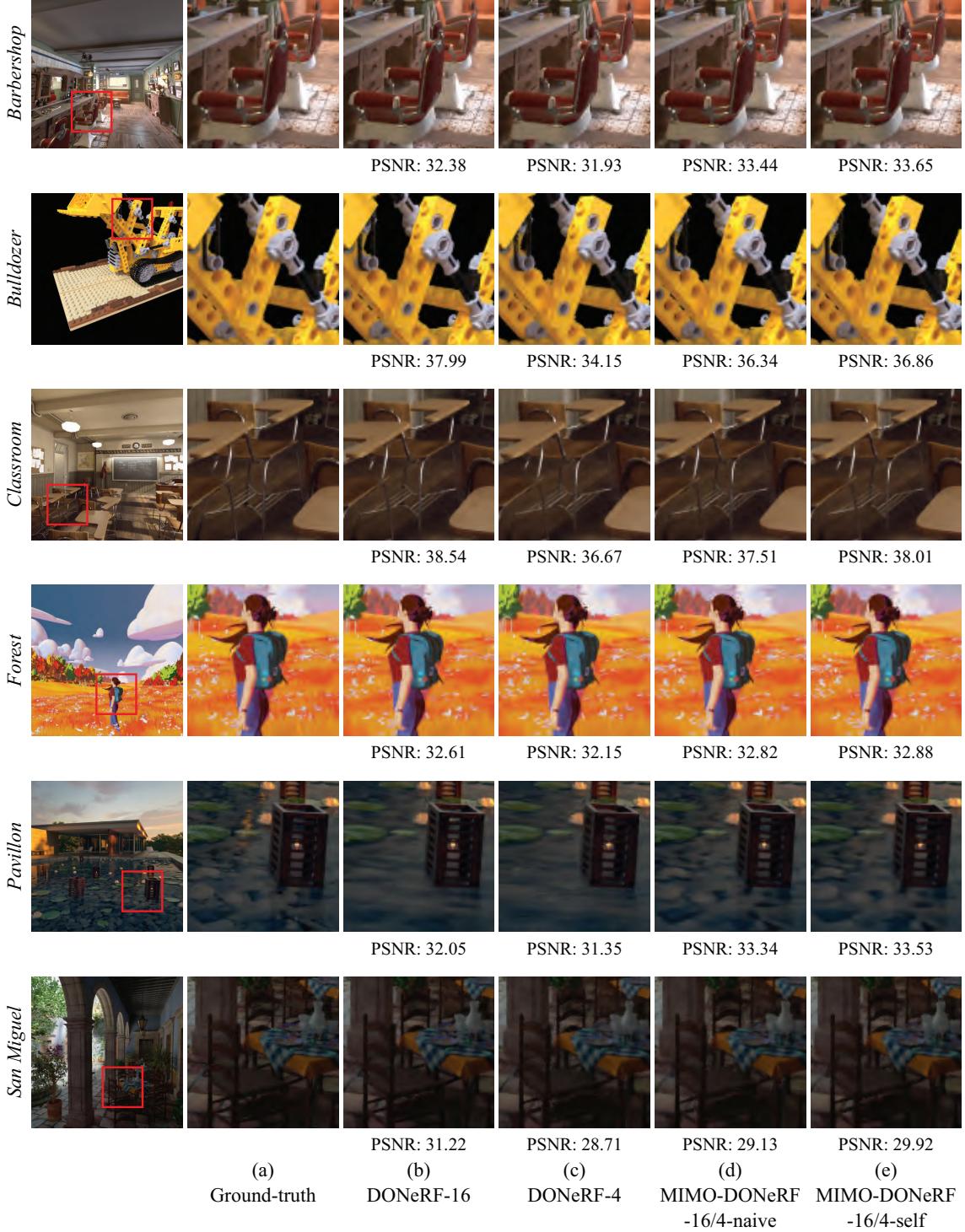


Figure 10. Qualitative comparison between DONeRF-16, DONeRF-4, MIMO-DONeRF-16/4-naive, and MIMO-DONeRF-16/4-self. Best viewed zoomed in. We report PSNR for the displayed view. The average scores for all views are given in Table 13. DONeRF-4 (c) sometimes yields artifacts, e.g., for the belt in the “Bulldozer” scene or for the hair in the “Forest” scene. DONeRF-16 (b), MIMO-DONeRF-16/4-naive (d), and MIMO-DONeRF-16/4-self (e) mitigate this defect by increasing the number of samples. It should be noted that DONeRF-16 increases the inference time approximately three times, while MIMO-DONeRF-16/4-naive and MIMO-DONeRF-16/4-self only increase the inference time 1.1 times. Another interesting finding is that MIMO-DONeRF-16/4-naive (d) and MIMO-DONeRF-16/4-self (e) succeed in representing lotus leaves in the “Pavillon” scene, whereas DONeRF-16 (b) and DONeRF-4 (c) fail to do so. The possible reason is that MIMO-NeRFs can accumulate neighbor information using grouped samples, and this provides a positive effect.

A.9. Detailed analysis of application to TensoRF

In Section 5.5, we used the variants of TensoRF that achieved the best image quality as baselines. Specifically, we used *TensoRF-VM-192-30k* ($R_\sigma = 16$, $R_c = 48$, and the iteration of 30k) for the Blender dataset and used *TensoRF-VM-96* ($R_{\sigma,1} = R_{\sigma,2} = 4$, $R_{\sigma,3} = 16$, $R_{c,1} = R_{c,2} = 12$, and $R_{c,3} = 48$) for the LLFF dataset. As models with faster training but lower quality, a previous study [9] also presented *TensoRF-VM-48* ($R_\sigma = R_c = 8$, and the iteration of 30k) and *TensoRF-VM-192-15k* ($R_\sigma = 16$, $R_c = 48$, and the iteration of 15k) for the Blender dataset, and *TensoRF-VM-48* ($R_{\sigma,1} = R_{\sigma,2} = 4$, $R_{\sigma,3} = 16$, $R_{c,1} = R_{c,2} = 4$, and $R_{c,3} = 16$) for the LLFF dataset. This appendix examines whether MIMO-NeRF is also effective for these models. Based on the observation that MIMO-TensoRF-VM-192-30k retains image quality when $N_p \leq 2$ on the Blender dataset (Section 5.5), we examined MIMO-TensoRF-VM-48 and MIMO-TensoRF-VM-192-15k with $N_p \in \{2, 4\}$ on the Blender dataset. Similarly, based on the observation that MIMO-TensoRF-VM-96 can achieve comparable image quality when $N_p \leq 4$ on the LLFF dataset (Section 5.5), we examined MIMO-TensoRF-VM-96 with $N_p \in \{2, 4, 8\}$ on the LLFF dataset. We evaluated the models using VGG_{VGG} and VGG_{Alex} , in addition to the metrics described in Section 5.5.

Quantitative results. Table 14 lists the results for all the metrics. Tables 15 and 16 summarize the PSNR, SSIM, LPIPS_{VGG}, and LPIPS_{Alex} scores for each scene in the Blender and LLFF datasets, respectively. We observed the same tendencies as those described in Section 5.5. On the Blender dataset, MIMO-TensoRFs can improve the I-time and T-time of the original TensoRFs with similar image quality when $N_p \leq 2$. On the LLFF dataset, MIMO-TensoRFs can improve the I-time and T-time of the original TensoRFs with similar image quality when $N_p \leq 4$. These results suggest that MIMO-TensoRF can strengthen the inference/training speed of TensoRF without deteriorating the image quality by adequately selecting N_p .

Qualitative results. Figures 11 and 12 present the qualitative results for the Blender and LLFF datasets, respectively. Examples of the synthesized videos are provided on the [project page](#).¹

Model	N_p	Blender							
		PSNR↑	SSIM↑	LPIPS _{VGG}	LPIPS _{Alex}	# Run↓	I-time↓ (s)	T-time↓ (m)	# Params (M)
TensoRF-VM-48	1	32.45	0.957	0.056	0.032	10.24	1.16	9.45	4.7
MIMO-TensoRF-VM-48-2	2	32.49	0.957	0.056	0.032	4.95	1.09	8.95	4.7
MIMO-TensoRF-VM-48-4	4	32.25	0.955	0.060	0.034	2.49	1.06	8.85	4.8
TensoRF-VM-192-15k	1	32.74	0.961	0.051	0.030	10.11	1.27	5.64	18.9
MIMO-TensoRF-VM-192-15k-2	2	32.79	0.961	0.051	0.030	4.78	1.19	5.36	18.9
MIMO-TensoRF-VM-192-15k-4	4	32.55	0.958	0.055	0.032	2.40	1.16	5.22	18.9
TensoRF-VM-192-30k	1	33.23	0.963	0.047	0.026	9.95	1.25	11.50	18.8
MIMO-TensoRF-VM-192-30k-2	2	33.26	0.963	0.047	0.026	4.76	1.18	10.89	18.8
MIMO-TensoRF-VM-192-30k-4	4	32.98	0.961	0.051	0.028	2.40	1.15	10.67	18.8
MIMO-TensoRF-VM-192-30k-8	8	32.37	0.956	0.058	0.033	1.27	1.14	10.57	18.9
TensoRF-VM-48 [9]	1	32.39	0.957	0.057	0.032	—	—	—	—
TensoRF-VM-192-15k [9]	1	32.52	0.959	0.053	0.032	—	—	—	—
TensoRF-VM-192-30k [9]	1	33.14	0.963	0.047	0.027	—	—	—	—
Model	N_p	LLFF							
		PSNR↑	SSIM↑	LPIPS _{VGG}	LPIPS _{Alex}	# Run↓	I-time↓ (s)	T-time↓ (m)	# Params (M)
TensoRF-VM-48	1	26.48	0.832	0.213	0.125	120.78	6.14	19.83	23.4
MIMO-TensoRF-VM-48-2	2	26.51	0.833	0.211	0.124	58.42	5.70	18.21	23.4
MIMO-TensoRF-VM-48-4	4	26.50	0.832	0.211	0.124	28.11	5.24	17.20	23.4
MIMO-TensoRF-VM-48-8	8	26.41	0.830	0.215	0.126	13.58	5.03	16.86	23.5
TensoRF-VM-96	1	26.73	0.837	0.201	0.115	126.73	6.64	23.41	46.8
MIMO-TensoRF-VM-96-2	2	26.72	0.837	0.201	0.115	62.14	6.18	21.63	46.8
MIMO-TensoRF-VM-96-4	4	26.72	0.836	0.202	0.115	30.16	5.76	21.15	46.8
MIMO-TensoRF-VM-96-8	8	26.64	0.835	0.204	0.116	14.52	5.52	20.68	46.9
TensoRF-VM-48 [9]	1	26.51	0.832	0.217	0.135	—	—	—	—
TensoRF-VM-96 [9]	1	26.73	0.839	0.204	0.124	—	—	—	—

Table 14. Comparison of quantitative scores between TensoRFs and MIMO-TensoRFs. The scores for the model with citation [9] are taken from another report [9]. We provide them as references. The other scores were calculated in our environment. We implemented all the models based on the official TensoRF source code. See Appendix B.3 for the implementation details. This table is an extended version of Table 4. The PSNR, SSIM, LPIPS_{VGG}, and LPIPS_{Alex} scores for each scene are presented in Tables 15 and 16.

Blender											
Model	N_p	PSNR↑									Avg.
		Chair	Drums	Ficus	Hotdog	Lego	Materials	Mic	Ship		
TensoRF-VM-48	1	34.71	25.57	33.44	36.88	35.69	29.38	33.83	30.07	32.45	
MIMO-TensoRF-VM-48-2	2	34.85	25.56	33.34	37.00	35.84	29.43	33.91	30.00	32.49	
MIMO-TensoRF-VM-48-4	4	34.61	25.24	33.05	36.95	35.61	29.26	33.52	29.75	32.25	
TensoRF-VM-192-15k	1	35.11	25.80	33.73	37.04	36.00	29.80	34.31	30.10	32.74	
MIMO-TensoRF-VM-192-15k-2	2	35.32	25.65	33.87	37.22	36.06	29.77	34.35	30.07	32.79	
MIMO-TensoRF-VM-192-15k-4	4	35.06	25.36	33.67	37.10	35.77	29.54	34.11	29.80	32.55	
TensoRF-VM-192-30k	1	35.79	25.96	34.14	37.50	36.62	30.10	34.98	30.72	33.23	
MIMO-TensoRF-VM-192-30k-2	2	35.91	25.96	34.28	37.64	36.61	30.11	34.97	30.62	33.26	
MIMO-TensoRF-VM-192-30k-4	4	35.60	25.56	34.04	37.51	36.42	29.80	34.60	30.28	32.98	
MIMO-TensoRF-VM-192-30k-8	8	35.04	24.85	33.24	37.04	35.92	29.13	33.87	29.87	32.37	
TensoRF-VM-48 [9]	1	34.68	25.58	33.37	36.81	35.51	29.45	33.59	30.12	32.39	
TensoRF-VM-192-15k [9]	1	34.95	25.63	33.46	36.85	35.78	29.78	33.69	30.04	32.52	
TensoRF-VM-192-30k [9]	1	35.76	26.01	33.99	37.41	36.46	30.12	34.61	30.77	33.14	
SSIM↑											
Model	N_p	Chair	Drums	Ficus	Hotdog	Lego	Materials	Mic	Ship	Avg.	
		0.980	0.930	0.979	0.979	0.979	0.942	0.985	0.883	0.957	
TensoRF-VM-48	1	0.981	0.930	0.979	0.980	0.980	0.942	0.985	0.881	0.957	
MIMO-TensoRF-VM-48-2	2	0.980	0.925	0.977	0.980	0.979	0.940	0.983	0.876	0.955	
MIMO-TensoRF-VM-48-4	4	0.982	0.935	0.981	0.981	0.982	0.950	0.987	0.887	0.961	
TensoRF-VM-192-15k	1	0.982	0.934	0.982	0.982	0.982	0.949	0.987	0.886	0.961	
MIMO-TensoRF-VM-192-15k-2	2	0.983	0.934	0.982	0.982	0.982	0.949	0.987	0.893	0.963	
MIMO-TensoRF-VM-192-15k-4	4	0.982	0.929	0.981	0.981	0.981	0.946	0.986	0.880	0.958	
TensoRF-VM-192-30k	1	0.985	0.937	0.983	0.983	0.983	0.952	0.989	0.894	0.963	
MIMO-TensoRF-VM-192-30k-2	2	0.985	0.937	0.983	0.983	0.984	0.952	0.988	0.893	0.963	
MIMO-TensoRF-VM-192-30k-4	4	0.984	0.931	0.982	0.983	0.983	0.949	0.987	0.886	0.961	
MIMO-TensoRF-VM-192-30k-8	8	0.982	0.922	0.978	0.980	0.981	0.942	0.984	0.877	0.956	
TensoRF-VM-48 [9]	1	0.980	0.929	0.979	0.979	0.979	0.942	0.984	0.883	0.957	
TensoRF-VM-192-15k [9]	1	0.982	0.933	0.981	0.980	0.981	0.949	0.985	0.886	0.959	
TensoRF-VM-192-30k [9]	1	0.985	0.937	0.982	0.982	0.983	0.952	0.988	0.895	0.963	
LPIPS _{VGG} ↓											
Model	N_p	Chair	Drums	Ficus	Hotdog	Lego	Materials	Mic	Ship	Avg.	
		0.029	0.085	0.029	0.038	0.023	0.073	0.020	0.153	0.056	
TensoRF-VM-48	1	0.028	0.086	0.030	0.036	0.022	0.073	0.021	0.154	0.056	
MIMO-TensoRF-VM-48-2	2	0.029	0.092	0.035	0.038	0.024	0.077	0.025	0.159	0.060	
MIMO-TensoRF-VM-48-4	4	0.024	0.076	0.024	0.035	0.020	0.062	0.017	0.150	0.051	
TensoRF-VM-192-15k	1	0.023	0.077	0.024	0.034	0.020	0.063	0.017	0.150	0.051	
MIMO-TensoRF-VM-192-15k-2	2	0.025	0.085	0.028	0.036	0.021	0.069	0.021	0.154	0.055	
MIMO-TensoRF-VM-192-15k-4	4	0.021	0.071	0.022	0.031	0.018	0.058	0.014	0.139	0.047	
TensoRF-VM-192-30k	1	0.020	0.072	0.022	0.030	0.017	0.058	0.015	0.140	0.047	
MIMO-TensoRF-VM-192-30k-2	2	0.022	0.080	0.026	0.032	0.018	0.065	0.019	0.144	0.051	
MIMO-TensoRF-VM-192-30k-4	4	0.026	0.089	0.032	0.041	0.021	0.076	0.025	0.153	0.058	
MIMO-TensoRF-VM-192-30k-8	8	0.021	0.071	0.022	0.031	0.018	0.058	0.014	0.139	0.047	
TensoRF-VM-48 [9]	1	0.030	0.087	0.028	0.039	0.024	0.072	0.021	0.155	0.057	
TensoRF-VM-192-15k [9]	1	0.026	0.078	0.025	0.038	0.021	0.063	0.020	0.153	0.053	
TensoRF-VM-192-30k [9]	1	0.022	0.073	0.022	0.032	0.018	0.058	0.015	0.138	0.047	
LPIPS _{Alex} ↓											
Model	N_p	Chair	Drums	Ficus	Hotdog	Lego	Materials	Mic	Ship	Avg.	
		0.013	0.057	0.015	0.017	0.009	0.036	0.011	0.095	0.032	
TensoRF-VM-48	1	0.013	0.057	0.015	0.016	0.009	0.037	0.011	0.096	0.032	
MIMO-TensoRF-VM-48-2	2	0.013	0.062	0.017	0.017	0.009	0.041	0.013	0.101	0.034	
MIMO-TensoRF-VM-48-4	4	0.011	0.054	0.013	0.016	0.008	0.029	0.010	0.096	0.030	
TensoRF-VM-192-15k	1	0.011	0.055	0.013	0.015	0.008	0.030	0.010	0.097	0.030	
MIMO-TensoRF-VM-192-15k-2	2	0.011	0.056	0.014	0.013	0.008	0.034	0.011	0.098	0.032	
MIMO-TensoRF-VM-192-15k-4	4	0.012	0.060	0.015	0.016	0.008	0.034	0.011	0.098	0.032	
TensoRF-VM-192-30k	1	0.009	0.049	0.012	0.013	0.007	0.026	0.008	0.084	0.026	
MIMO-TensoRF-VM-192-30k-2	2	0.009	0.050	0.012	0.012	0.007	0.026	0.008	0.086	0.026	
MIMO-TensoRF-VM-192-30k-4	4	0.010	0.056	0.014	0.013	0.007	0.032	0.009	0.087	0.028	
MIMO-TensoRF-VM-192-30k-8	8	0.011	0.064	0.018	0.017	0.008	0.041	0.013	0.094	0.033	
TensoRF-VM-48 [9]	1	0.014	0.059	0.015	0.017	0.009	0.036	0.012	0.098	0.032	
TensoRF-VM-192-15k [9]	1	0.013	0.056	0.014	0.017	0.009	0.029	0.013	0.101	0.032	
TensoRF-VM-192-30k [9]	1	0.010	0.051	0.012	0.013	0.007	0.026	0.009	0.085	0.027	

Table 15. Comparison of PSNR, SSIM, LPIPS_{VGG}, and LPIPS_{Alex} for each scene on the Blender dataset between TensoRFs and MIMO-TensoRFs. The scores for the model with citation [9] are taken from another report [9]. We provide them as references. The other scores were calculated in our environment. We implemented all the models based on the official TensoRF source code. See Appendix B.3 for the implementation details. The scores for the other metrics are summarized in Table 14.

Model	N_p	PSNR↑								Avg.
		Fern	Flower	Fortress	Horns	Leaves	Orchids	Room	T-Rex	
TensoRF-VM-48	1	25.18	27.88	31.11	27.83	21.27	19.94	31.66	26.99	26.48
MIMO-TensoRF-VM-48-2	2	25.23	27.95	31.14	27.88	21.24	19.93	31.59	27.08	26.51
MIMO-TensoRF-VM-48-4	4	25.21	27.85	31.19	27.83	21.26	19.97	31.51	27.19	26.50
MIMO-TensoRF-VM-48-8	8	25.13	27.82	31.06	27.74	21.20	19.98	31.25	27.09	26.41
TensoRF-VM-96	1	25.00	28.29	31.47	28.35	21.09	19.81	32.22	27.63	26.73
MIMO-TensoRF-VM-96-2	2	25.14	28.36	31.43	28.38	21.00	19.86	32.17	27.40	26.72
MIMO-TensoRF-VM-96-4	4	25.16	28.21	31.48	28.29	21.10	19.89	32.18	27.47	26.72
MIMO-TensoRF-VM-96-8	8	25.12	28.08	31.27	28.22	21.08	19.98	31.87	27.54	26.64
TensoRF-VM-48 [9]	1	25.31	28.22	31.14	27.64	21.34	20.02	31.80	26.61	26.51
TensoRF-VM-96 [9]	1	25.27	28.60	31.36	28.14	21.30	19.87	32.35	26.97	26.73
Model	N_p	SSIM↑								Avg.
		Fern	Flower	Fortress	Horns	Leaves	Orchids	Room	T-Rex	
TensoRF-VM-48	1	0.806	0.854	0.889	0.865	0.745	0.651	0.946	0.898	0.832
MIMO-TensoRF-VM-48-2	2	0.808	0.855	0.891	0.868	0.744	0.651	0.946	0.899	0.833
MIMO-TensoRF-VM-48-4	4	0.809	0.852	0.891	0.865	0.745	0.650	0.945	0.901	0.832
MIMO-TensoRF-VM-48-8	8	0.807	0.850	0.891	0.866	0.739	0.649	0.939	0.899	0.830
TensoRF-VM-96	1	0.800	0.861	0.899	0.883	0.744	0.643	0.952	0.910	0.837
MIMO-TensoRF-VM-96-2	2	0.803	0.865	0.900	0.884	0.739	0.644	0.952	0.907	0.837
MIMO-TensoRF-VM-96-4	4	0.806	0.857	0.901	0.883	0.739	0.644	0.950	0.909	0.836
MIMO-TensoRF-VM-96-8	8	0.804	0.855	0.897	0.882	0.740	0.648	0.945	0.910	0.835
TensoRF-VM-48 [9]	1	0.816	0.859	0.889	0.859	0.746	0.655	0.946	0.890	0.832
TensoRF-VM-96 [9]	1	0.814	0.871	0.897	0.877	0.752	0.649	0.952	0.900	0.839
Model	N_p	LPIPS _{VGG} ↓								Avg.
		Fern	Flower	Fortress	Horns	Leaves	Orchids	Room	T-Rex	
TensoRF-VM-48	1	0.244	0.186	0.157	0.207	0.226	0.282	0.179	0.219	0.213
MIMO-TensoRF-VM-48-2	2	0.243	0.184	0.155	0.203	0.227	0.283	0.179	0.216	0.211
MIMO-TensoRF-VM-48-4	4	0.240	0.187	0.154	0.208	0.227	0.284	0.179	0.212	0.211
MIMO-TensoRF-VM-48-8	8	0.241	0.188	0.153	0.205	0.233	0.285	0.196	0.215	0.215
TensoRF-VM-96	1	0.249	0.172	0.142	0.180	0.220	0.281	0.162	0.201	0.201
MIMO-TensoRF-VM-96-2	2	0.245	0.168	0.141	0.179	0.227	0.283	0.161	0.205	0.201
MIMO-TensoRF-VM-96-4	4	0.241	0.176	0.139	0.181	0.226	0.284	0.167	0.199	0.202
MIMO-TensoRF-VM-96-8	8	0.240	0.179	0.141	0.182	0.228	0.282	0.179	0.201	0.204
TensoRF-VM-48 [9]	1	0.237	0.187	0.159	0.221	0.230	0.283	0.181	0.236	0.217
TensoRF-VM-96 [9]	1	0.237	0.169	0.148	0.196	0.217	0.278	0.167	0.221	0.204
Model	N_p	LPIPS _{Alex} ↓								Avg.
		Fern	Flower	Fortress	Horns	Leaves	Orchids	Room	T-Rex	
TensoRF-VM-48	1	0.156	0.113	0.078	0.125	0.155	0.195	0.088	0.091	0.125
MIMO-TensoRF-VM-48-2	2	0.155	0.112	0.075	0.120	0.156	0.198	0.089	0.088	0.124
MIMO-TensoRF-VM-48-4	4	0.152	0.114	0.075	0.126	0.156	0.197	0.089	0.086	0.124
MIMO-TensoRF-VM-48-8	8	0.153	0.113	0.075	0.119	0.160	0.197	0.102	0.086	0.126
TensoRF-VM-96	1	0.156	0.101	0.066	0.103	0.143	0.193	0.076	0.079	0.115
MIMO-TensoRF-VM-96-2	2	0.154	0.098	0.065	0.102	0.148	0.194	0.075	0.082	0.115
MIMO-TensoRF-VM-96-4	4	0.151	0.101	0.065	0.103	0.147	0.196	0.082	0.077	0.115
MIMO-TensoRF-VM-96-8	8	0.148	0.103	0.067	0.101	0.152	0.192	0.088	0.077	0.116
TensoRF-VM-48 [9]	1	0.161	0.121	0.084	0.146	0.167	0.204	0.093	0.108	0.135
TensoRF-VM-96 [9]	1	0.155	0.106	0.075	0.123	0.153	0.201	0.082	0.099	0.124

Table 16. Comparison of PSNR, SSIM, LPIPS_{VGG}, and LPIPS_{Alex} for each scene on the LLFF dataset between TensoRFs and MIMO-TensoRFs. The scores for the model with citation [9] are taken from another report [9]. We provide them as references. The other scores were calculated in our environment. We implemented all the models based on the official TensoRF source code. See Appendix B.3 for the implementation details. The scores for the other metrics are summarized in Table 14.

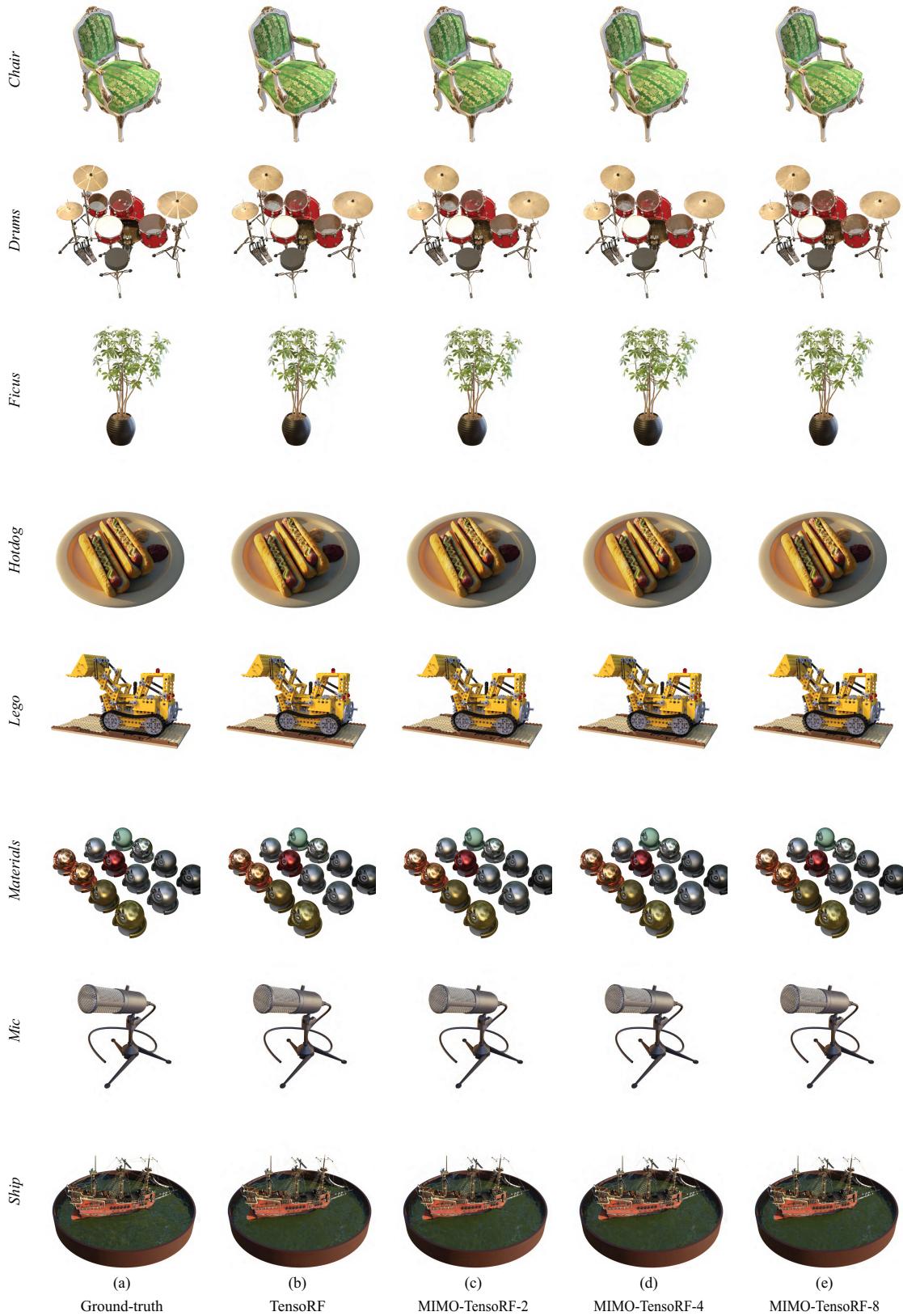


Figure 11. Qualitative comparison between TensoRF, MIMO-TensoRF-2, MIMO-TensoRF-4, and MIMO-TensoRF-8 on the Blender dataset. Best viewed zoomed in. TensoRF-VM-192-30k was used as a baseline, and MIMO-NeRF was incorporated into it.

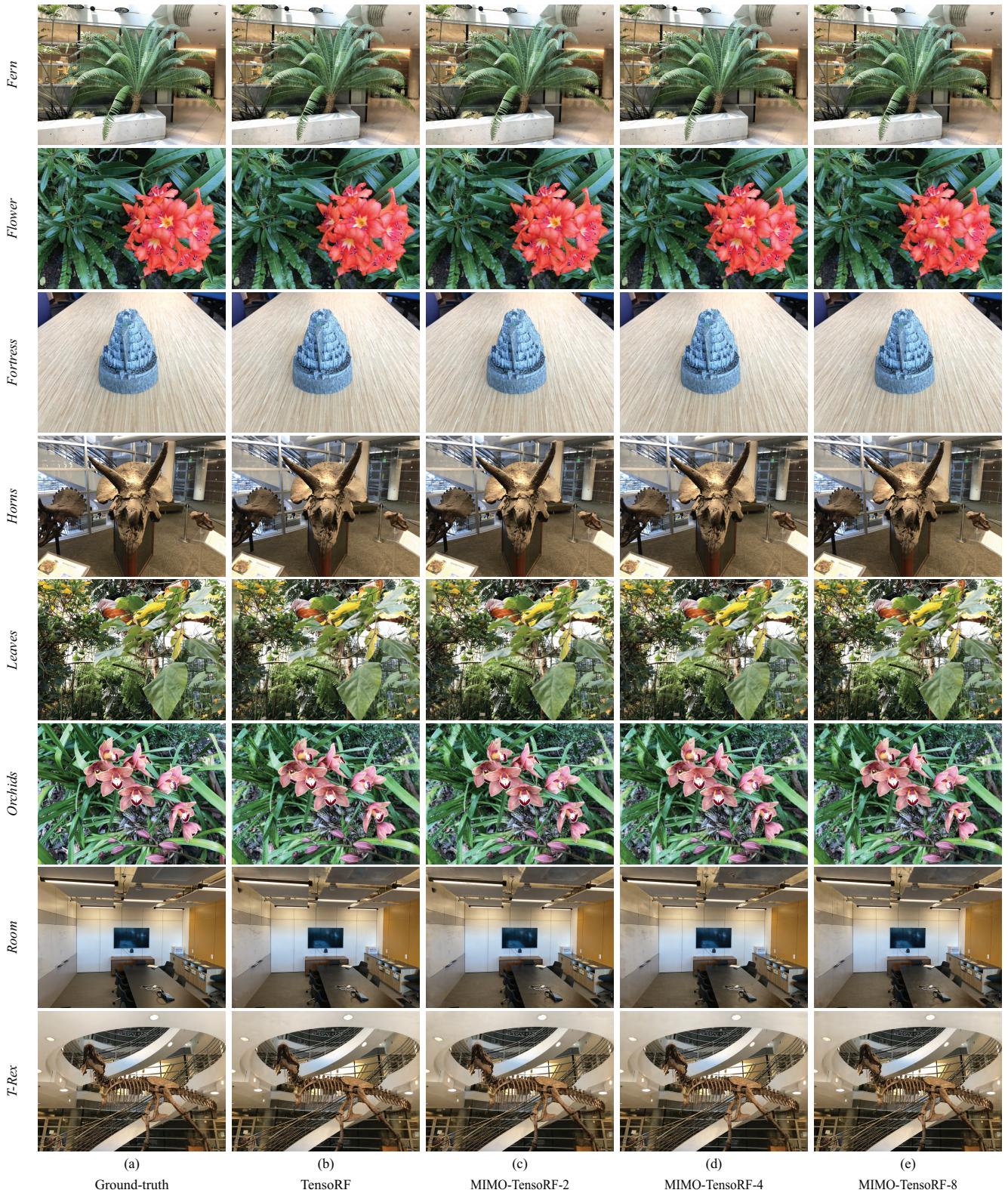


Figure 12. Qualitative comparison between TensoRF, MIMO-TensoRF-2, MIMO-TensoRF-4, and MIMO-TensoRF-8 on the LLFF dataset. Best viewed zoomed in. TensoRF-VM-96 was used as a baseline, and MIMO-NeRF was incorporated into it.

B. Implementation details

The following implementation details are provided in this appendix:

- Appendix B.1: Implementation details of NeRF (Sections 5.1–5.3 and Appendices A.1–A.7)
- Appendix B.2: Implementation details of DONeRF (Section 5.4 and Appendix A.8)
- Appendix B.3: Implementation details of TensoRF (Section 5.5 and Appendix A.9)

B.1. Implementation details of NeRF

B.1.1 Datasets

In the experiments discussed in Sections 5.1–5.3, we used two datasets commonly employed in previous studies on NeRFs. The detailed information is as follows:

Blender dataset [41]. The dataset included eight scenes: *Chair*, *Drums*, *Ficus*, *Hotdog*, *Lego*, *Materials*, *Mic*, and *Ship*. Each scene contained 360° views of complex objects at a resolution of 800 × 800 pixels. They were rendered using a Blender Cycles path tracer and exhibited complicated geometries and non-Lambertian materials. For the training and testing, 100 and 200 views were used, respectively. The data were downloaded from the NeRF authors’ website [41].¹⁴ The license information is provided on the website.

Local Light Field Fusion (LLFF) dataset [40]. Specifically, we used the dataset with addition obtained from [41]. The dataset consists of eight complex real-world scenes: *Fern*, *Flower*, *Fortress*, *Horns*, *Leaves*, *Orchids*, *Room*, and *T-Rex*. Each of these included 20–62 forward-facing views at a resolution of 1008 × 756 pixels. They were captured using a forward-facing handheld cell phone. One-eighth of the images were used for testing, and the rest were used for training. The data were downloaded from the NeRF authors’ website [41].¹⁴ The license information is provided on the website.

As mentioned in Section 5.3, we primarily used half-sized images following the default settings of an open-source NeRF code⁵ to better investigate the various configurations. We also used full-sized images for representative cases to confirm whether the effectiveness of MIMO-NeRF was independent of the image size. We discuss these cases in Appendix A.6.

B.1.2 Model configurations

NeRF. We implemented the baseline NeRF using the open-source code of NeRF.⁵ The model configuration of the base-

Model	Blender					LLFF				
	N_c	N_f	N_p	F	FLOPs (M)	N_c	N_f	N_p	F	FLOPs (M)
NeRF	64	128	1	256	303.82	64	64	1	256	227.87
MIMO-NeRF	64	128	2	256	160.33	64	64	2	256	120.25
NeRF-few	34	68	1	256	161.41	34	34	1	256	121.06
NeRF-small	64	128	1	184	160.72	64	64	1	184	120.54
MIMO-NeRF	64	128	4	256	88.59	64	64	4	256	66.44
NeRF-few	19	38	1	256	90.20	19	19	1	256	67.65
NeRF-small	64	128	1	135	89.09	64	64	1	135	66.82
MIMO-NeRF	64	128	8	256	52.72	64	64	8	256	39.54
NeRF-few	11	22	1	256	52.22	11	11	1	256	39.16
NeRF-small	64	128	1	103	53.62	64	64	1	103	40.22

Table 17. Comparison of the number of coarse samples (N_c), number of fine samples (N_f), number of grouped samples (N_p), number of features in a hidden layer (F), and FLOPs between NeRF, MIMO-NeRF, NeRF-few, and NeRF-small. The hyperparameters of NeRF-few and NeRF-small were adjusted such that their FLOPs became almost the same as that of MIMO-NeRF.

line NeRF followed the default settings provided in the code. Specifically, the input position $\mathbf{x} \in \mathbb{R}^3$ and view direction $\mathbf{d} \in \mathbb{S}^2$ were encoded to a 63-dimensional vector $\gamma(\mathbf{x})$ and 27-dimensional vector $\gamma(\mathbf{d})$, respectively, using positional encoding [41, 65]. Subsequently, the encoded position $\gamma(\mathbf{x})$ was applied to an 8-layer MLP with rectified unit (ReLU) activation [43], each layer of which had 256 hidden units. The MLP included a skip connection that incorporated $\gamma(\mathbf{x})$ into the fifth layer. The volume density $\sigma \in \mathbb{R}^+$ was calculated from the output of the MLP using a linear layer. At a different branch, the output of the MLP was converted using a linear layer with 256 hidden units, and the encoded direction $\gamma(\mathbf{d})$ was then concatenated into the converted result. After the concatenated vector was converted to a 128 vector using a 1-layer MLP with ReLU activation, it was used to calculate the RGB color $\mathbf{c} \in \mathbb{R}^3$ using an additional linear layer. We used the same network architecture for coarse and fine MLPs. For the half-sized images, the numbers of coarse and fine samples (i.e., N_c and N_f) were set to 64 and 128 for the Blender dataset and to 64 and 64 for the LLFF dataset, respectively. For the full-sized images, N_c and N_f were set to 64 and 128, respectively, for both datasets.

MIMO-NeRF. MIMO-NeRF has the same network architecture as the baseline NeRF, except for the inputs and outputs. Particularly, the above-mentioned network was modified to accept N_p inputs, that is, $(\mathbf{x}_i, \dots, \mathbf{x}_j)$, with view direction \mathbf{d} , and produce N_p outputs, that is, $(\mathbf{c}_i, \dots, \mathbf{c}_j)$ and $(\sigma_i, \dots, \sigma_j)$, where N_p was the number of grouped samples and $j = i + N_p - 1$. The other parameters, such as the dimensions of the hidden units, number of layers, type of activation function, N_c , and N_f , were the same as those in the baseline NeRF.

NeRF-few. In NeRF-few, which was used in the experiment described in Section 5.2, the number of samples (N_c and N_f) was adjusted such that its FLOPs became almost the

¹⁴https://drive.google.com/drive/folders/128yBriW1IG_3NJ5Rp7APSTZsJqdJdfc1

same as those of MIMO-NeRF. Detailed values are listed in Table 17.

NeRF-small. In NeRF-small, which was used in the experiment discussed in Section 5.2, the number of features in the hidden layers (F) was adjusted such that its FLOPs were almost the same as those of MIMO-NeRF. Detailed values are listed in Table 17.

B.1.3 Training settings

Half-sized images. For a fair comparison, we trained all the models using the same training settings except that in MIMO-NeRF, the NeRF loss function, i.e., $\mathcal{L}_{\text{pixel}}$ (Equation 3), was replaced with $\mathcal{L}_{\text{MIMO}} = \mathcal{L}_{\text{pixel}}^{\text{MIMO}} + \lambda \mathcal{L}_{\text{3D}}$ (Equation 10). Specifically, when we trained the models using half-sized images, we referred to the default settings provided in the open-source code of NeRF.⁵ More precisely, the models were trained for $200k$ iterations using the Adam optimizer [27] with an initial learning rate of 5×10^{-4} and momentum terms β_1 and β_2 of 0.9 and 0.999, respectively. The batch size was set to 1024 rays. For MIMO-NeRF, we set $\lambda = 1$ for the Blender dataset and $\lambda = 0.4$ for the LLFF dataset.

Full-size images. For full-size images, we trained the models according to the configurations provided in the official NeRF source code [41].¹⁵ For the Blender dataset, the models were trained for $500k$ iterations using the Adam optimizer [27] with an initial learning rate of 5×10^{-4} and momentum terms β_1 and β_2 of 0.9 and 0.999, respectively. The batch size was set to 1024 rays. For the LLFF dataset, the models were trained for $200k$ iterations using the Adam optimizer [27] with an initial learning rate of 5×10^{-4} , β_1 of 0.9, and β_2 of 0.999. The batch size was set to 4096 rays. For MIMO-NeRF, λ was set to 1 for the Blender dataset and 0.4 for the LLFF dataset.

B.1.4 Evaluation metrics

We used seven evaluation metrics to measure the performance of NeRF and MIMO-NeRF quantitatively: peak signal-to-noise ratio (*PSNR*), structural similarity index (*SSIM*) [68], learned perceptual image patch quality (*LPIPS*) [77], number of MLPs running (# *Run*), inference time (*I-time*), training time (*T-time*), and number of parameters (# *Params*). The PSNR, SSIM, and LPIPS were used as image quality metrics, following the original NeRF study [41]. I-time and T-time were used to measure the inference and training speeds, respectively. # Run and the # Params were provided as supplement information. The details of these metrics are as follows:

PSNR. PSNR is a metric that is widely used for assessing the signal quality and is calculated as $\text{PSNR} =$

$-10 \log_{10} \|\hat{\mathbf{I}} - \mathbf{I}\|_2^2$, where $\hat{\mathbf{I}}$ and \mathbf{I} denote the synthesized and ground-truth images, respectively, assuming that images are in $[0, 1]$. It measures the ratio between the maximum possible power of a signal and the power of the noise, which affects the signal quality. The larger the PSNR, the better the image quality.

SSIM. SSIM measures the structural similarity between two images and is commonly used to evaluate image quality. The larger the SSIM, the better the image quality.

LPIPS. LPIPS measures the distance between two images using the features of a pretrained DNN. The LPIPS has been demonstrated to have a better correlation with human perceptual judgment than the PSNR or SSIM [77]. We used the VGG network [57] as the pretrained DNN, following the NeRF study [41]. The smaller the LPIPS, the better the image quality.

Run. # Run indicates the number of MLPs running required for rendering a single pixel. In NeRF and MIMO-NeRF, it is calculated as $\frac{N_c}{N_p} + \frac{N_c + N_f}{N_p}$, where $\frac{N_c}{N_p}$ is the # Run for the MLP in the coarse strategy, and $\frac{N_c + N_f}{N_p}$ is the # Run for the MLP in the fine strategy. In the baseline NeRF, $N_p = 1$. The smaller the value of # Run, the faster the rendering speed when the speed for each run is the same.

I-time. The inference time was measured using a single NVIDIA GeForce RTX 3080 Ti Laptop GPU. The smaller the I-time, the faster the inference. For simplicity and a fair comparison, we measured the inference time using a standard PyTorch implementation.⁵ Optimizing the implementation for faster inference (e.g., using custom CUDA kernels) would be interesting for future research.

T-time. The training time was measured using a single NVIDIA A100-SXM4-80GB GPU. The smaller the T-time, the faster the training. Similar to I-time, for simplicity and a fair comparison, we measured the training time using a standard PyTorch implementation.⁵ Optimizing the implementation for faster training (e.g., using custom CUDA kernels) would be interesting for future research.

Params. # Params indicates the number of parameters of the MLPs, including one in the coarse strategy and the other in the fine strategy. As mentioned in Section 5.3, # Params increases in MIMO-NeRF mainly because the total dimension of the encoded position $\gamma(\mathbf{x})$ increased by N_p times according to the increase in the inputs, as described in Appendix B.1.2. It should be noted that MIMO-NeRF has the same network as the baseline NeRF except for the inputs and outputs; therefore, the # Params does not increase N_p times. For example, in the experiments discussed in Section 5.3, the # Params increased by 1.06, 1.17, and 1.39 times when N_p was 2, 4, and 8, respectively.

¹⁵<https://github.com/bmild/nerf>

B.2. Implementation details of DONeRF

B.2.1 Dataset

In the experiments discussed in Section 5.4, the models were evaluated using the *DONeRF dataset* introduced by DONeRF [44]. The detailed information is as follows:

DONeRF dataset [44]. The dataset included six synthetic indoor and outdoor scenes: *Barbershop*, *Bulldozer*, *Classroom*, *Forest*, *Pavillon*, and *San Miguel*. They exhibit fine and high-frequency details and a wide depth range. Each scene included 300 forward-facing views with 800×800 pixels each. They were rendered using the Blender Cycles path tracer. The poses were randomly sampled within the view cell, where the rotation was limited to 30° in pitch and 20° in yaw relative to the initial camera direction. For training, validation, and testing, 70%, 10%, and 20% of images were used, respectively. Following the original DONeRF study [44], the images were downsampled to 400×400 pixels to accelerate the training. We downloaded the data from the DONeRF authors’ website [44].¹⁶ The license information is provided on the website.

B.2.2 Model configurations

DONeRF. We implemented DONeRF using the source code provided by the authors [44].⁹ In particular, DONeRF was composed of two networks: a depth oracle network and a shading network.

Depth oracle network. The depth oracle network predicted the depth from the position $\mathbf{x} \in \mathbb{R}^3$ and view direction $\mathbf{d} \in \mathbb{S}^2$. In this network, positional encoding was not adopted for the inputs because it has been demonstrated that it does not improve performance [44]. After \mathbf{x} and \mathbf{d} were concatenated, they were converted to depth using an 8-layer MLP, where each layer had 256 hidden units and ReLU activation [43] except for the last output layer.

Shading network. The shading network predicted the RGB color $\mathbf{c} \in \mathbb{R}^3$ and the volume density $\sigma \in \mathbb{R}^+$ from \mathbf{x} and \mathbf{d} for the samples selected by the depth oracle network. It had the same network architecture as that of the depth oracle network except for the following two points: (1) positional encoding [41, 65] was applied to \mathbf{x} and \mathbf{d} to obtain a 63-dimensional vector $\gamma(\mathbf{x})$ and 27-dimensional vector $\gamma(\mathbf{d})$, respectively, and (2) only $\gamma(\mathbf{x})$ was used at the first layer and $\gamma(\mathbf{d})$ was concatenated to the feature vector before the last layer.

In *DONeRF- N_s* (e.g., DONeRF-16), the number of selected samples in the shading network was set to N_s (e.g., 16).

MIMO-DONeRF. We incorporated the MIMO-NeRF concept into the shading network because the depth oracle net-

work is already a fast network that runs only once for each ray. The difference between the shading network of DONeRF and that of MIMO-DONeRF is limited to the difference in the inputs and outputs. Specifically, the shading network of DONeRF was modified to accept N_p inputs, that is, $(\mathbf{x}_i, \dots, \mathbf{x}_j)$, with view direction \mathbf{d} , and generate N_p outputs, that is, $(\mathbf{c}_i, \dots, \mathbf{c}_j)$ and $(\sigma_i, \dots, \sigma_j)$, where N_p was the number of grouped samples and $j = i + N_p - 1$. The other parameters, such as the dimensions of the hidden units, number of layers, and type of activation function, were the same as those in DONeRF. In *MIMO-DONeRF- N_s/N_p* (e.g., MIMO-DONeRF-16/4), the number of samples selected by the depth oracle network was set to N_s (e.g., 16), and the number of grouped samples was set to N_p (e.g., 4).

B.2.3 Training settings

For a fair comparison, we trained DONeRF and MIMO-DONeRF using the same configurations, except that in MIMO-DONeRF, $\mathcal{L}_{\text{MIMO}}$ (Equation 10) was used as an alternative to the NeRF loss function, that is, $\mathcal{L}_{\text{pixel}}$ (Equation 3). Specifically, we trained them using the default settings provided in the official DONeRF source code.⁹ The depth oracle and shading networks were separately trained for $300k$ iterations using the Adam optimizer [27] with a learning rate of 5×10^{-4} and momentum terms β_1 and β_2 of 0.9 and 0.999, respectively. The batch size was set to 4096 rays. The hyperparameter for MIMO-DONeRF was set to $\lambda = 0.001$.

B.2.4 Evaluation metrics

We used six evaluation metrics to quantitatively investigate the performance of DONeRF and MIMO-DONeRF: *PSNR*, *FLIP* [1], *# Run*, *I-time*, *T-time*, and *# Params*. The PSNR and FLIP were used as image quality metrics, following the original DONeRF study [44]. I-time and T-time were used to assess inference and training speeds, respectively. *# Run* and *# Params* were provided as supplement information. The definitions of PSNR, I-time, T-time, and *# Params* are the same as those in Appendix B.1.4. Detailed information on the other two metrics (FLIP and *# Run*) is as follows:

FLIP. FLIP is a metric that evaluates the differences between rendered images and the corresponding ground-truth images. The effectiveness of the FLIP was demonstrated through a user study [1]. The smaller the FLIP, the better the image quality.

Run. In DONeRF and MIMO-DONeRF, *# Run* is calculated as $1 + \frac{N_s}{N_p}$, where N_s indicates the number of samples selected by the depth oracle network and used as inputs in the shading networks, and N_p indicates the number of grouped samples. In DONeRF, N_p is 1. In the above

¹⁶<https://repository.tugraz.at/records/jjs3x-4f133>

equation, the first term, 1, represents # Run for the depth oracle network, and the second term, $\frac{N_s}{N_p}$, represents # Run for the shading network. The smaller the value of # Run, the faster the rendering speed when the speed for each run is the same.

B.3. Implementation details of TensoRF

B.3.1 Datasets

In the experiments described in Section 5.5, we evaluated performance using the Blender and LLFF datasets. In particular, full-sized images were used. The details of these two datasets are presented in Appendix B.1.1.

B.3.2 Model configurations

TensoRF. TensoRF was implemented using the official source code provided by the authors [9].¹⁰ Particularly, in the experiments described in Section 5.5, we used two variants of TensoRF to achieve the best image quality: For the Blender dataset, we used *TensoRF-VM-192-30k*, which had 192 components with $R_\sigma = 16$ and $R_c = 48$. For the LLFF dataset, we used *TensoRF-VM-96*, which had 96 components with $R_{\sigma,1} = R_{\sigma,2} = 4$, $R_{\sigma,3} = 16$, $R_{c,1} = R_{c,2} = 12$, and $R_{c,3} = 48$. In the experiments described in Appendix A.9, we additionally used three variants of TensoRF: For the Blender dataset, we used *TensoRF-VM-48*, which had 48 components with $R_\sigma = R_c = 8$, and *TensoRF-VM-192-15k*, which had the same network architecture as that of TensorRF-VM-192-30k but the number of training iterations was halved. For the LLFF dataset, we use *TensoRF-VM-48*, which had 48 components with $R_{\sigma,1} = R_{\sigma,2} = 4$, $R_{\sigma,3} = 16$, $R_{c,1} = R_{c,2} = 4$, and $R_{c,3} = 16$. In all models, a two-layer MLP with 128-dimensional hidden layers and ReLU activation [43] was used as the RGB color decoding function. The MLP receives an embedding of the viewing direction and features extracted from the tensor factors. Embedding was performed using positional encoding [41, 65] with frequencies of two.

MIMO-TensoRF. We applied the MIMO-NeRF concept to the RGB color decoding function, that is, the two-layer MLP, and changed this MLP from a SISO MLP to a MIMO MLP. More precisely, we modified the MLP to receive N_p features, that is, $(\mathbf{f}_i, \dots, \mathbf{f}_j)$, with view direction \mathbf{d} , and produced N_p RGB colors, that is, $(\mathbf{c}_i, \dots, \mathbf{c}_j)$, where N_p was the number of grouped samples, and $j = i + N_p - 1$. Other parameters, such as the dimensions of the hidden units, number of layers, and type of activation function, were the same as those used in TensoRF. In the experiments, we varied $N_p \in \{2, 4, 8\}$ and denoted MIMO-TensoRF with N_p as *MIMO-TensoRF- N_p* .

B.3.3 Training settings

For a fair comparison, we trained TensoRF and MIMO-TensoRF using the same training settings. As discussed in Section 5.5, in MIMO-TensoRF, the correspondence ambiguity is relatively small because the volume density, σ , is calculated using an unambiguous explicit representation. Hence, we trained MIMO-TensoRF using standard TensoRF loss functions and did not use $\mathcal{L}_{\text{MIMO}}$ while prioritizing the training speed. Specifically, we trained TensoRF and MIMO-TensoRF using the default settings provided in the official TensoRF source code.¹⁰ The models were trained for T iterations using the Adam optimizer [27] with initial learning rates of 0.02 for tensor factors and 0.001 for the MLP decoder and momentum terms β_1 and β_2 of 0.9 and 0.99, respectively. For the Blender dataset, T was set to 30k except for TensoRF-VM-192-15k and MIMO-TensoRF-VM-192-15k, where T was set to 15k. For the LLFF dataset, T was set to 25k. The batch size was set to 4096 rays.

B.3.4 Evaluation metrics

We used eight evaluation metrics to assess the performance of TensoRF and MIMO-TensoRF quantitatively: *PSNR*, *SSIM*, *LPIPS_{VGG}*, *LPIPS_{Alex}*, # Run, I-time, T-time, and # Params. PSNR, SSIM, LPIPS_{VGG}, and LPIPS_{Alex} were used as image quality metrics following the original TensoRF study [9]. I-time and T-time were used as the inference and training speed metrics, respectively. # Run and # Params were provided as supplement information. The definitions of PSNR, SSIM, I-time, T-time, and # Params are the same as those in Appendix B.1.4. Detailed information on the other three metrics (LPIPS_{VGG}, LPIPS_{Alex}, and # Run) is as follows:

LPIPS_{VGG}. This metric is identical to the LPIPS metric described in Appendix B.1.4. LPIPS_{VGG} measures the distance between two images by using the feature space of the VGG network [57]. The smaller the LPIPS_{VGG}, the better the image quality.

LPIPS_{Alex}. LPIPS_{Alex} measures the distance between two images using the feature space of the Alex network [28]. The smaller the LPIPS_{Alex}, the better the image quality.

Run. In TensoRF and MIMO-TensoRF, only samples with weights (i.e., $T_i \alpha_i$ in Equation 2) greater than the threshold were provided to the RGB decoding function. Therefore, the number of samples, i.e., N , was adaptively determined per scene and per pixel. Consequently, # Run = $\frac{N}{N_p}$ was also adaptively determined for each scene and pixel. In our experiments, we report the values averaged over the scenes and pixels.