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

Takuhiro Kaneko
NTT Communication Science Laboratories, NTT Corporation

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,[1] 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.

[1]We focused on grouping methods that can be conducted per ray for two reasons: (1) In typical NeRF training, rendering is performed for randomly 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.

Model	N_p	Blender PSNR↑	Blender SSIM↑	Blender LPIPS↓	LLFF PSNR↑	LLFF SSIM↑	LLFF LPIPS↓
MIMO-NeRF-naive	2	30.18	0.944	0.065	27.31	0.860	0.167
MIMO-NeRF-random	2	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
MIMO-NeRF-random	4	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
MIMO-NeRF-random	8	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.

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 decreases. Therefore, such a model is beyond the scope of this study.

A.2. Effect of reformulation methods

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

$$R^1 = 1, \dots, R^L = 2^{L-1}. \tag{11}$$

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

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

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.[2] 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

$$R^1 = 1, R^2 = 1. \tag{13}$$

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

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

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

$$R^1 = 1, \ldots, R^{N_p - 1} = 1. \tag{15}$$

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

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

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

$$R^1 = 1, R^2 = 2. \tag{17}$$

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

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

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

$$R^1 = 1, R^2 = \frac{N_p}{2}. \tag{19}$$

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

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

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

$$R^1 = 1, R^2 = 1, R^3 = 1. \tag{21}$$

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

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

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

[2] More 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 PSNR↑	Blender SSIM↑	Blender LPIPS↓	Blender T-time↓ (h)	LLFF PSNR↑	LLFF SSIM↑	LLFF LPIPS↓	LLFF T-time↓ (h)
NeRF	1	–	31.04	0.951	0.055	4.70	27.72	0.871	0.150	3.39
MIMO-NeRF-naive	2	–	30.18	0.944	0.065	3.09	27.31	0.860	0.167	2.12
MIMO-NeRF-distill		–	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	4	–	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		$R^1 = 1, R^2 = 2$	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	8	–	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		$R^1 = 1, R^2 = 1$	30.02	0.940	0.076	2.61	26.52	0.833	0.207	2.13
MIMO-NeRF-self-R3		$R^1 = 1, \ldots, R^7 = 1$	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.

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 $\lambda \in [0.4, 1]$.

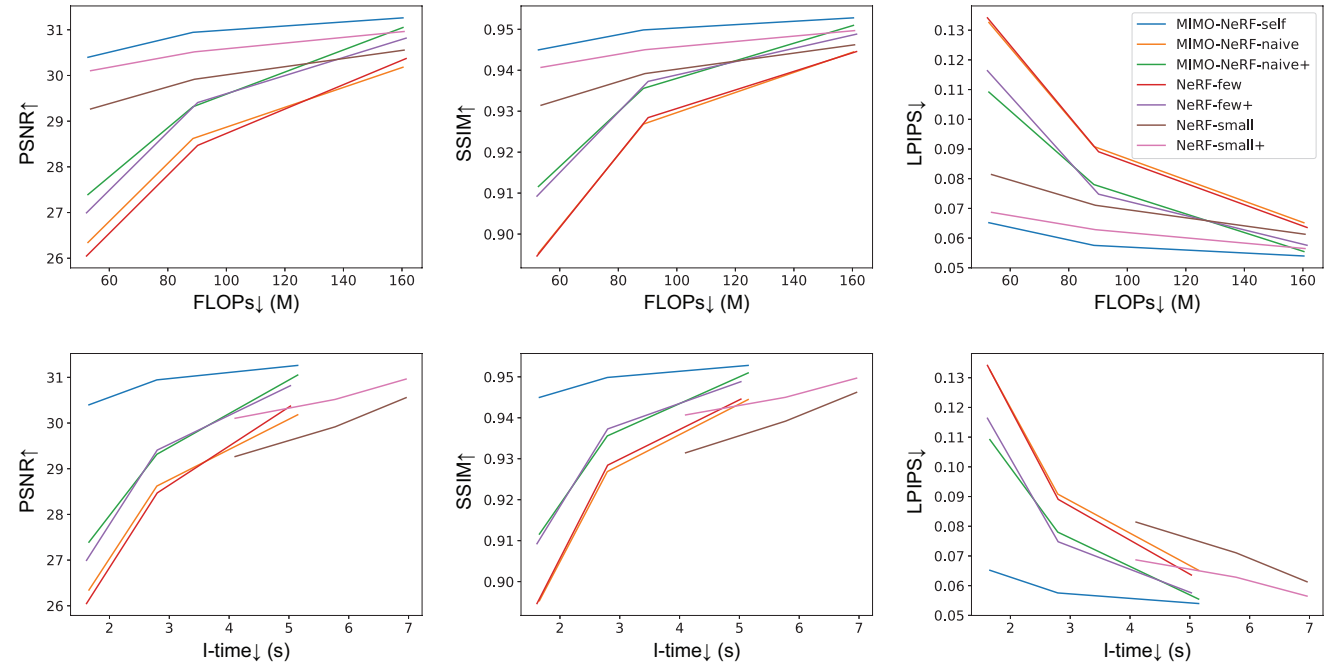
N_p	λ	Blender PSNR↑	Blender SSIM↑	Blender LPIPS↓	LLFF PSNR↑	LLFF SSIM↑	LLFF LPIPS↓
2	0.4	31.20	0.952	0.054	27.70	0.870	0.155
	1.0	31.26	0.953	0.054	27.56	0.866	0.160
4	0.4	30.93	0.950	0.058	27.51	0.865	0.162
	1.0	30.94	0.950	0.058	27.40	0.863	0.165
8	0.4	30.22	0.944	0.067	26.97	0.851	0.180
	1.0	30.40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

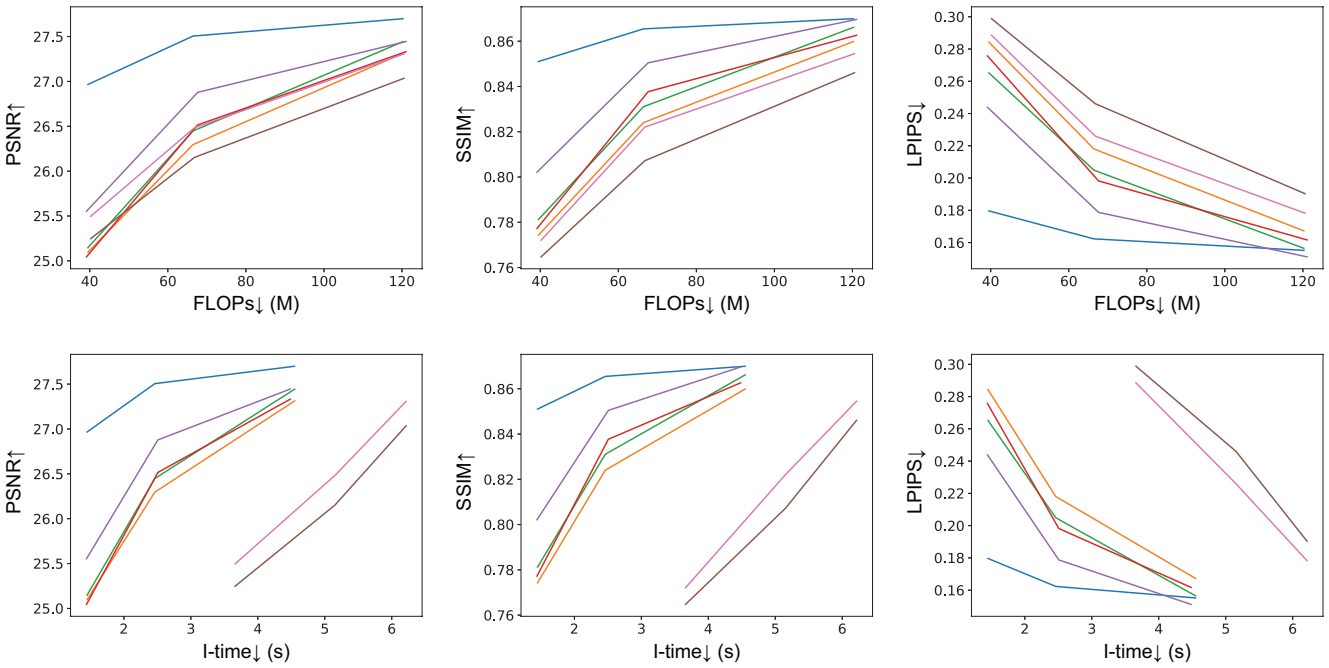
A.4. Detailed analysis of speed-quality trade-off

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 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 in Section 5.2, 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 increas-



(a) Blender



(b) LLFF

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.

ing 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.[3] 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, 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.

[3]More 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.

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.

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, SISO-NeRF and 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. The synthesized videos are provided in the directory of `videos/NeRF` in the Supplementary Material. In this directory, `NeRF.mp4` was generated by the original NeRF, `MIMO-NeRF-`N_p`-naive.mp4` was synthesized by MIMO-NeRF-naive, where the number of grouped samples was N_p, and `MIMO-NeRF-`N_p`-self.mp4` was synthesized by MIMO-NeRF-self, where the number of grouped samples was N_p.

Model	N_p	PSNR↑	SSIM↑	LPIPS↓	Blender # Run↓	Blender I-time↓ (s)	Blender T-time↓ (h)	Blender # Params (M)	PSNR↑	SSIM↑	LPIPS↓	LLFF # Run↓	LLFF I-time↓ (s)	LLFF T-time↓ (h)	LLFF # Params (M)
NeRF [6]	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 [6]	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 [6] are taken from another report [6]. 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.

PSNR↑

Model	N_p	Blender Chair	Drums	Ficus	Hotdog	Lego	Materials	Mic	Ship	Avg.	LLFF 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 [6]	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

SSIM↑

Model	N_p	Blender Chair	Drums	Ficus	Hotdog	Lego	Materials	Mic	Ship	Avg.	LLFF Fern	Flower	Fortress	Horns	Laves	Orchids	Room	T-Rex	Avg.
NeRF [6]	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 [6]	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

LPIPS↓

Model	N_p	Blender Chair	Drums	Ficus	Hotdog	Lego	Materials	Mic	Ship	Avg.	LLFF Fern	Flower	Fortress	Horns	Laves	Orchids	Room	T-Rex	Avg.
NeRF [6]	1	0.046	0.091	0.045	0.045	0.048	0.063	0.026	0.198	0.070	0.281	0.214	0.173	0.273	0.312	0.314	0.173	0.254	0.249
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 [6]	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 [6] are taken from another report [6]. 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).

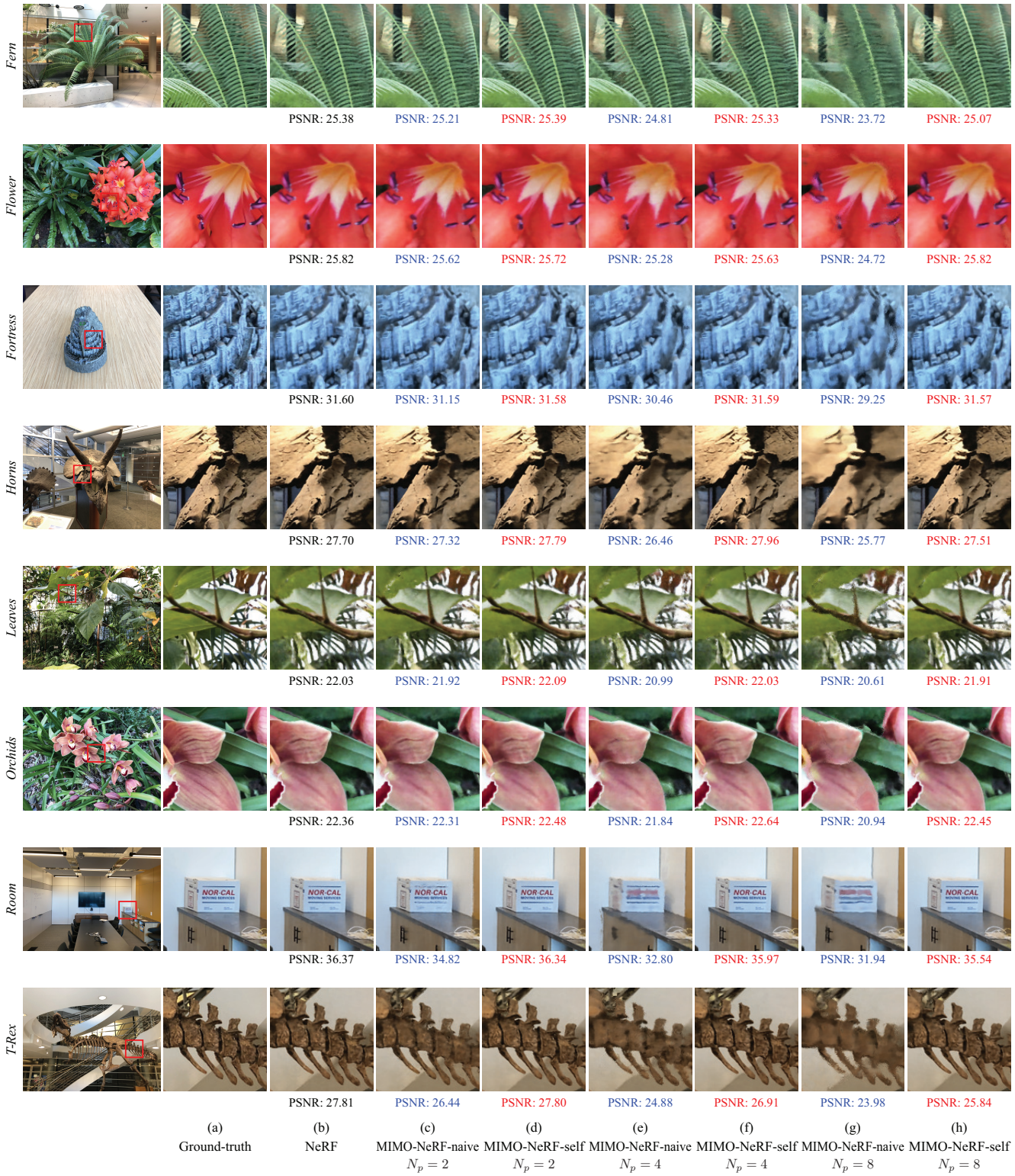


Figure 9. Qualitative comparison between NeRF, MIMO-NeRF-self, and MIMO-NeRF-self on the LLFF dataset. This figure is an extension of Figure 5. We report PSNR for the displayed view. The average scores for all views are listed in Table 10. As shown in (c), (e), and (g), the deterioration of image quality 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 [4], which reduces the number of MLPs running (# Run) using an integral network that calculates the colors and volume densities per segment. 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 ($N = 32$) [4]	32	26.83	0.926	0.151
MIMO-NeRF-self ($N_p = 8$)	32	29.92	0.938	0.084
AutoInt ($N = 16$) [4]	16	26.04	0.916	0.167
MIMO-NeRF-self ($N_p = 16$)	16	28.69	0.925	0.099
AutoInt ($N = 8$) [4]	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 [4]. 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 when seeking a better trade-off between speed and quality.

Qualitative results. Figure 10 shows the qualitative results. The synthesized videos are provided in the directory of `videos/DONeRF` in the Supplementary Material. In this directory, the `DONeRF-16.mp4` was synthesized using DONeRF-16, and `MIMO-DONeRF-16-4-self.mp4` was synthesized using MIMO-DONeRF-16/4-self.

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 [8]	4	1	31.14	0.071	–	–	–	–
DONeRF-16 [8]	16	1	33.03	0.062	–	–	–	–

Table 12. Comparison of quantitative scores between DONeRFs and MIMO-DONeRFs. The scores for the model with citation [8] are taken from another report [8]. 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	Barbershop	Bulldozer	Classroom	PSNR↑ Forest	Pavillon	San Miguel	Avg.
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 [8]	4	1	30.84	33.46	33.43	30.63	31.07	27.41	31.14
DONeRF-16 [8]	16	1	32.15	36.98	36.27	31.32	31.79	29.67	33.03

Model	N_s	N_p	Barbershop	Bulldozer	Classroom	FLIP↓ Forest	Pavillon	San Miguel	Avg.
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 [8]	4	1	0.065	0.048	0.058	0.077	0.098	0.080	0.071
DONeRF-16 [8]	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 [8] are taken from another report [8]. 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.

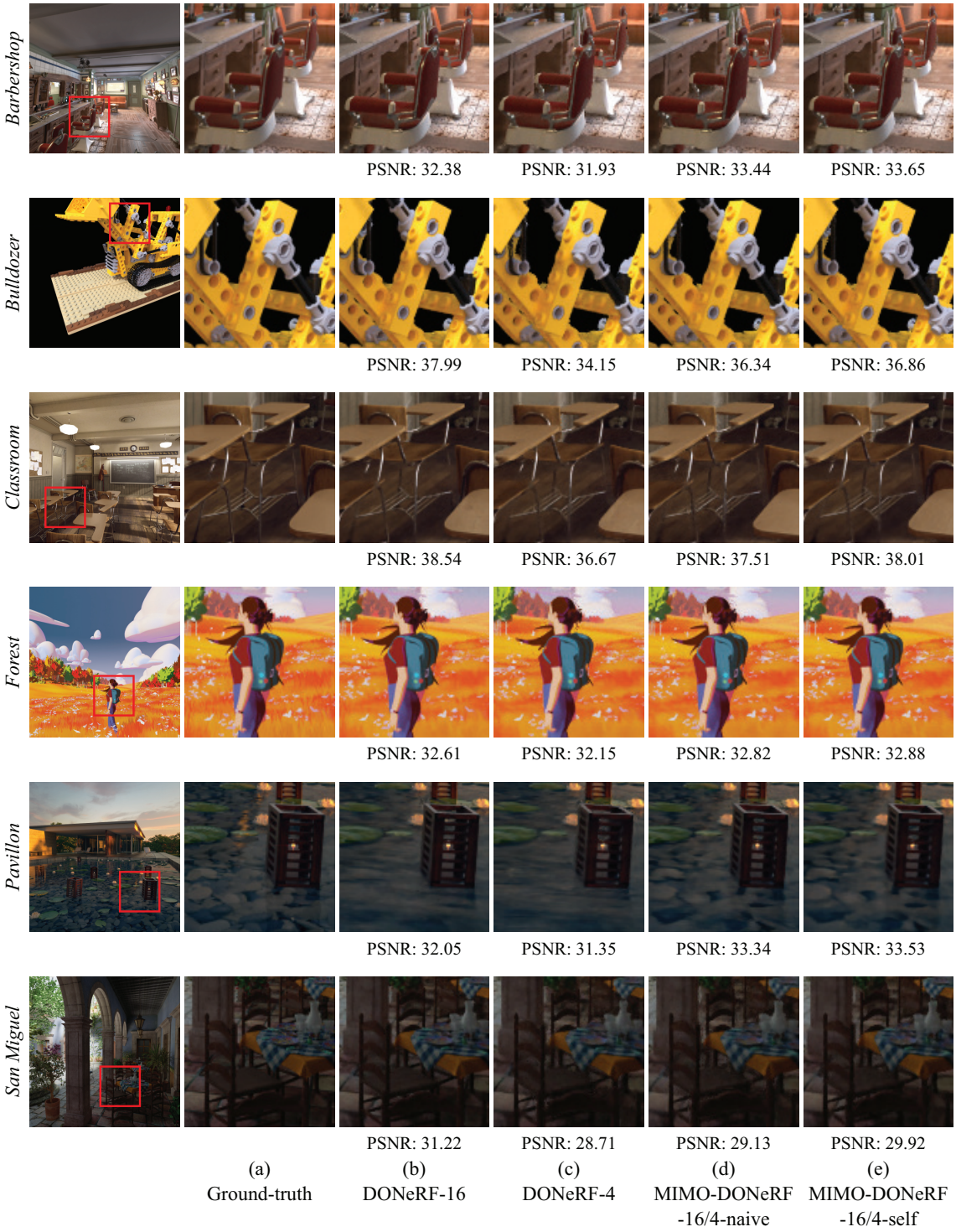


Figure 10. Qualitative comparison between DONeRF-16, DONeRF-4, MIMO-DONeRF-16/4-naive, and MIMO-DONeRF-16/4-self. Best viewed zoomed in. We report PSNR for the displayed view. The average scores for all views are given in Table 13. DONeRF-4 (c) sometimes yields artifacts, e.g., for the belt in the "Bulldozer" 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 [2] 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, $\text{LPIPS}_{\text{VGG}}$, and $\text{LPIPS}_{\text{Alex}}$ 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. The synthesized videos are provided in the directory of `videos/TensoRF` in the Supplementary Material. In this directory, `TensoRF.mp4` was synthesized using TensoRF-VM192-30k, and `MIMO-TensoRF-2.mp4` was synthesized using the MIMO-TensoRF-VM192-30k-2.

Model	N_p	Blender PSNR↑	SSIM↑	$\text{LPIPS}_{\text{VGG}}$	$\text{LPIPS}_{\text{Alex}}$	# 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 [2]	1	32.39	0.957	0.057	0.032	–	–	–	–
TensoRF-VM-192-15k [2]	1	32.52	0.959	0.053	0.032	–	–	–	–
TensoRF-VM-192-30k [2]	1	33.14	0.963	0.047	0.027	–	–	–	–

Model	N_p	LLFF PSNR↑	SSIM↑	$\text{LPIPS}_{\text{VGG}}$	$\text{LPIPS}_{\text{Alex}}$	# 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 [2]	1	26.51	0.832	0.217	0.135	–	–	–	–
TensoRF-VM-96 [2]	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 [2] are taken from another report [2]. 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, $\text{LPIPS}_{\text{VGG}}$, and $\text{LPIPS}_{\text{Alex}}$ scores for each scene are presented in Tables 15 and 16.

Blender

PSNR↑

Model	N_p	Chair	Drums	Ficus	Hotdog	Lego	Materials	Mic	Ship	Avg.
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 [2]	1	34.68	25.58	33.37	36.81	35.51	29.45	33.59	30.12	32.39
TensoRF-VM-192-15k [2]	1	34.95	25.63	33.46	36.85	35.78	29.78	33.69	30.04	32.52
TensoRF-VM-192-30k [2]	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.
TensoRF-VM-48	1	0.980	0.930	0.979	0.979	0.979	0.942	0.985	0.883	0.957
MIMO-TensoRF-VM-48-2	2	0.981	0.930	0.979	0.980	0.980	0.942	0.985	0.881	0.957
MIMO-TensoRF-VM-48-4	4	0.980	0.925	0.977	0.980	0.979	0.940	0.983	0.876	0.955
TensoRF-VM-192-15k	1	0.982	0.935	0.981	0.981	0.982	0.950	0.987	0.887	0.961
MIMO-TensoRF-VM-192-15k-2	2	0.983	0.934	0.982	0.982	0.982	0.949	0.987	0.886	0.961
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 [2]	1	0.980	0.929	0.979	0.979	0.979	0.942	0.984	0.883	0.957
TensoRF-VM-192-15k [2]	1	0.982	0.933	0.981	0.980	0.981	0.949	0.985	0.886	0.959
TensoRF-VM-192-30k [2]	1	0.985	0.937	0.982	0.982	0.983	0.952	0.988	0.895	0.963

$\text{LPIPS}_{\text{VGG}}$↓

Model	N_p	Chair	Drums	Ficus	Hotdog	Lego	Materials	Mic	Ship	Avg.
TensoRF-VM-48	1	0.029	0.085	0.029	0.038	0.023	0.073	0.020	0.153	0.056
MIMO-TensoRF-VM-48-2	2	0.028	0.086	0.030	0.036	0.022	0.073	0.021	0.154	0.056
MIMO-TensoRF-VM-48-4	4	0.029	0.092	0.035	0.038	0.024	0.077	0.025	0.159	0.060
TensoRF-VM-192-15k	1	0.024	0.076	0.024	0.035	0.020	0.062	0.017	0.150	0.051
MIMO-TensoRF-VM-192-15k-2	2	0.023	0.077	0.024	0.034	0.020	0.063	0.017	0.150	0.051
MIMO-TensoRF-VM-192-15k-4	4	0.025	0.085	0.028	0.036	0.021	0.069	0.021	0.154	0.055
TensoRF-VM-192-30k	1	0.021	0.071	0.022	0.031	0.018	0.058	0.014	0.139	0.047
MIMO-TensoRF-VM-192-30k-2	2	0.020	0.072	0.022	0.030	0.017	0.058	0.015	0.140	0.047
MIMO-TensoRF-VM-192-30k-4	4	0.022	0.080	0.026	0.032	0.018	0.065	0.019	0.144	0.051
MIMO-TensoRF-VM-192-30k-8	8	0.026	0.089	0.032	0.041	0.021	0.076	0.025	0.153	0.058
TensoRF-VM-48 [2]	1	0.030	0.087	0.028	0.039	0.024	0.072	0.021	0.155	0.057
TensoRF-VM-192-15k [2]	1	0.026	0.078	0.025	0.038	0.021	0.063	0.020	0.153	0.053
TensoRF-VM-192-30k [2]	1	0.022	0.073	0.022	0.032	0.018	0.058	0.015	0.138	0.047

$\text{LPIPS}_{\text{Alex}}$↓

Model	N_p	Chair	Drums	Ficus	Hotdog	Lego	Materials	Mic	Ship	Avg.
TensoRF-VM-48	1	0.013	0.057	0.015	0.017	0.009	0.036	0.011	0.095	0.032
MIMO-TensoRF-VM-48-2	2	0.013	0.057	0.015	0.016	0.009	0.037	0.011	0.096	0.032
MIMO-TensoRF-VM-48-4	4	0.013	0.062	0.017	0.017	0.009	0.041	0.013	0.101	0.034
TensoRF-VM-192-15k	1	0.011	0.054	0.013	0.016	0.008	0.029	0.010	0.096	0.030
MIMO-TensoRF-VM-192-15k-2	2	0.011	0.055	0.013	0.015	0.008	0.030	0.010	0.097	0.030
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 [2]	1	0.014	0.059	0.015	0.017	0.009	0.036	0.012	0.098	0.032
TensoRF-VM-192-15k [2]	1	0.013	0.056	0.014	0.017	0.009	0.029	0.013	0.101	0.032
TensoRF-VM-192-30k [2]	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, $\text{LPIPS}_{\text{VGG}}$, and $\text{LPIPS}_{\text{Alex}}$ for each scene on the Blender dataset between TensoRFs and MIMO-TensoRFs. The scores for the model with citation [2] are taken from another report [2]. 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.

PSNR↑

Model	N_p	Fern	Flower	Fortress	Horns	Leaves	Orchids	Room	T-Rex	Avg.
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 [2]	1	25.31	28.22	31.14	27.64	21.34	20.02	31.80	26.61	26.51
TensoRF-VM-96 [2]	1	25.27	28.60	31.36	28.14	21.30	19.87	32.35	26.97	26.73

SSIM↑

Model	N_p	Fern	Flower	Fortress	Horns	Leaves	Orchids	Room	T-Rex	Avg.
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 [2]	1	0.816	0.859	0.889	0.859	0.746	0.655	0.946	0.890	0.832
TensoRF-VM-96 [2]	1	0.814	0.871	0.897	0.877	0.752	0.649	0.952	0.900	0.839

$\text{LPIPS}_{\text{VGG}}$↓

Model	N_p	Fern	Flower	Fortress	Horns	Leaves	Orchids	Room	T-Rex	Avg.
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 [2]	1	0.237	0.187	0.159	0.221	0.230	0.283	0.181	0.236	0.217
TensoRF-VM-96 [2]	1	0.237	0.169	0.148	0.196	0.217	0.278	0.167	0.221	0.204

$\text{LPIPS}_{\text{Alex}}$↓

Model	N_p	Fern	Flower	Fortress	Horns	Leaves	Orchids	Room	T-Rex	Avg.
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 [2]	1	0.161	0.121	0.084	0.146	0.167	0.204	0.093	0.108	0.135
TensoRF-VM-96 [2]	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, $\text{LPIPS}_{\text{VGG}}$, and $\text{LPIPS}_{\text{Alex}}$ for each scene on the LLFF dataset between TensoRFs and MIMO-TensoRFs. The scores for the model with citation [2] are taken from another report [2]. 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.

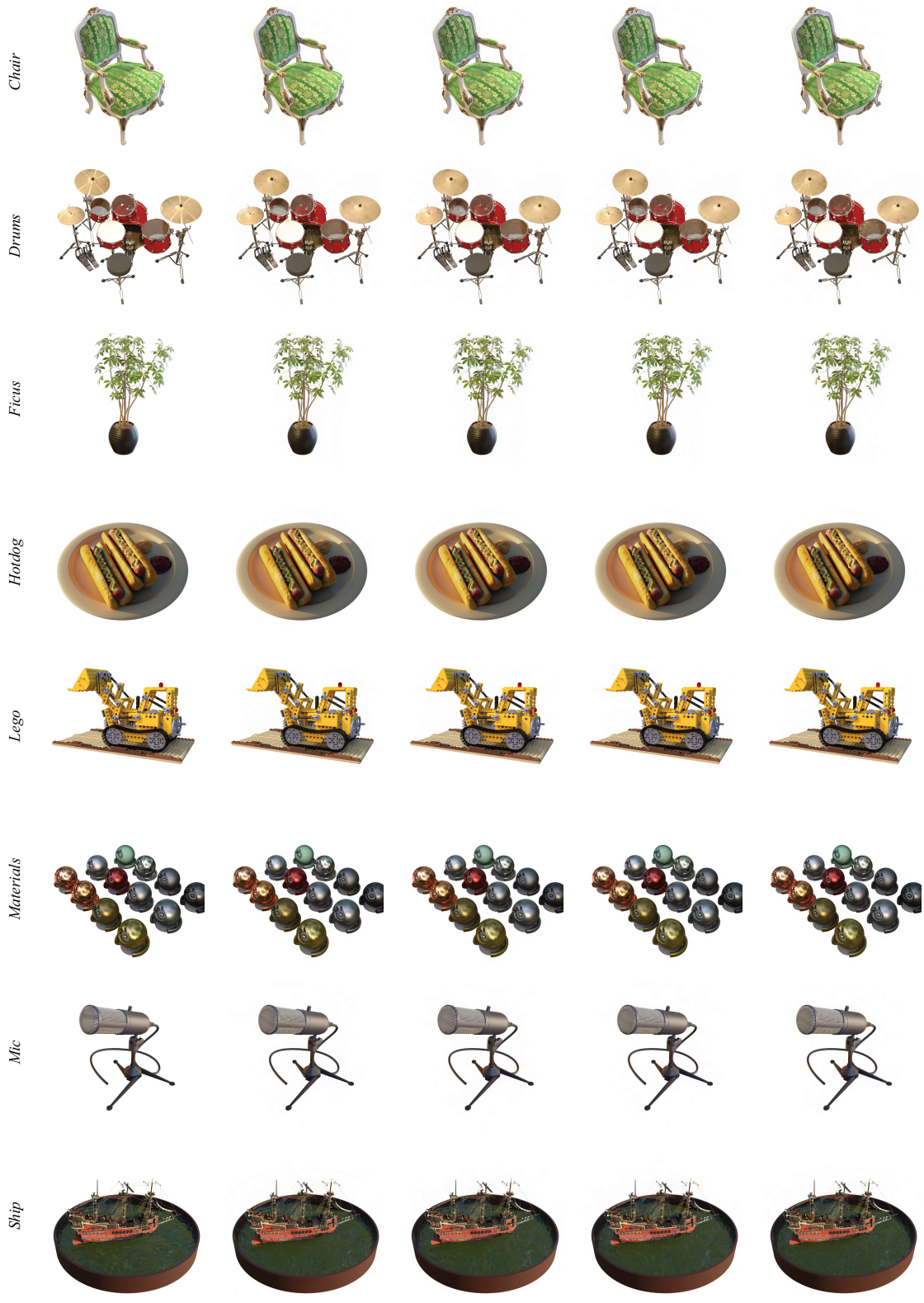


Figure 11. Qualitative comparison between TensoRF, MIMO-TensoRF-2, MIMO-TensoRF-4, and MIMO-TensoRF-8 on the Blender dataset. Best viewed zoomed in. TensoRF-VM-192-30k was used as a baseline, and MIMO-NeRF was incorporated into it.

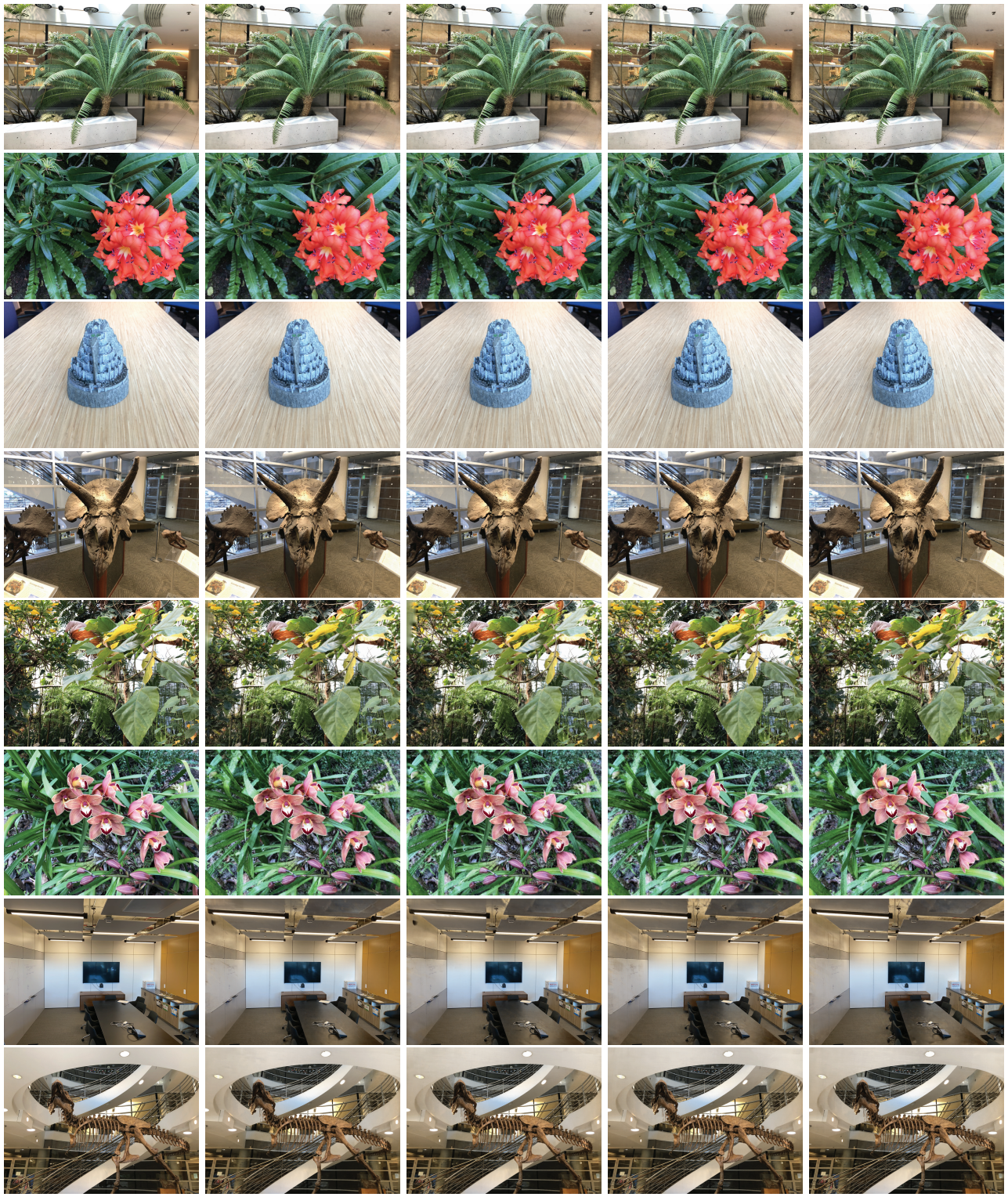


Figure 12. Qualitative comparison between TensoRF, MIMO-TensoRF-2, MIMO-TensoRF-4, and MIMO-TensoRF-8 on the LLFF dataset. Best viewed zoomed in. TensoRF-VM-96 was used as a baseline, and MIMO-NeRF was incorporated into it.

B. Implementation details

The following implementation details are provided in this appendix:

- Appendix B.1: Implementation details of NeRF (Sections 5.1–5.3)
- Appendix B.2: Implementation details of DONeRF (Section 5.4)
- Appendix B.3: Implementation details of TensoRF (Section 5.5)

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 [6]. 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 [6].[4] The license information is provided on the website.

Local Light Field Fusion (LLFF) dataset [5]. Specifically, we used the dataset with addition obtained from [6]. 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 [6].[4] 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 code[5] 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 baseline 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 [6, 10]. Subsequently, the encoded position $\gamma(\mathbf{x})$ was applied to an 8-layer MLP with rectified unit (ReLU) activation [7], 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.

Model	Blender N_c	Blender N_f	Blender N_p	Blender F	Blender FLOPs (M)	LLFF N_c	LLFF N_f	LLFF N_p	LLFF F	LLFF 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 parameters of NeRF-few and NeRF-small were adjusted such that their FLOPs became almost the same as that of MIMO-NeRF.

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, $(\mathbf{x}_i, \ldots, \mathbf{x}_j)$, with view direction $\mathbf{d}$, and produce N_p outputs, that is, $(\mathbf{c}_i, \ldots, \mathbf{c}_j)$ and $(\sigma_i, \ldots, \sigma_j)$, 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 same as those of MIMO-NeRF. Detailed values are listed in Table 17.

NeRF-small. In NeRF-small, which was used in the experi-

[4] https://drive.google.com/drive/folders/128yBriW1IG_3NJ5Rp7APSTZsJqdJdfc1

[5] https://github.com/yenchenlin/nerf-pytorch

ment 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 [3] 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 [6].[6] For the Blender dataset, the models were trained for $500k$ iterations using the Adam optimizer [3] 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 [3] 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*) [11], learned perceptual image patch quality (*LPIPS*) [12], 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 [6]. 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 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 [12]. We used the VGG network [9] as the pretrained DNN, following the NeRF study [6]. The smaller the LPIPS, the better the image quality.

 # Run. # Run indicates the number of MLPs running required for rendering a single pixel. NeRF and MIMO-NeRF are 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.

B.2. Implementation details of DONeRF

B.2.1 Datasets

In the experiments discussed in Section 5.4, the models were evaluated using the *DONeRF dataset* introduced by DONeRF [8]. The detailed information is as follows:

[6] https://github.com/bmild/nerf

DONeRF dataset [8]. 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 [8], the images were downsampled to 400×400 pixels to accelerate the training. We downloaded the data from the DONeRF authors' website [8].[7] 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 [8].[8] 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 [8]. 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 [7] 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 [6, 10] 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 network is already a fast single-input network. 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, $(\mathbf{x}_i, \ldots, \mathbf{x}_j)$, with view direction $\mathbf{d}$, and generate N_p outputs, that is, $(\mathbf{c}_i, \ldots, \mathbf{c}_j)$ and $(\sigma_i, \ldots, \sigma_j)$, 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).

[7] https://repository.tugraz.at/records/jjs3x-4f133

[8] https://github.com/facebookresearch/DONERF

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.[8] The depth oracle and shading networks were separately trained for $300k$ iterations using the Adam optimizer [3] 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 [8]. 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 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 [2].[9] 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 TensoRF-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 [7] 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 [6, 10] 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, $(\mathbf{f_i}, \ldots, \mathbf{f_j})$, with view direction $\mathbf{d}$, and produced N_p RGB colors, that is, $(\mathbf{c}_i, \ldots, \mathbf{c}_j)$, 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.[9] The models were trained for T iterations using the Adam optimizer [3] 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$_{\text{VGG}}$, and LPIPS$_{\text{Alex}}$ were used as image quality metrics following the original TensoRF study [2]. 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$_{\text{VGG}}$, LPIPS$_{\text{Alex}}$, and # Run) is as follows:

LPIPS$_{\text{VGG}}$. This metric is identical to the LPIPS metric described in Appendix B.1.4. LPIPS$_{\text{VGG}}$ measures the distance between two images by using the feature space of the VGG network [9]. The smaller the LPIPS$_{\text{VGG}}$, the better the image quality.

LPIPS$_{\text{Alex}}$. LPIPS$_{\text{Alex}}$ measures the distance between two images using the feature space of the Alex network [**?**]. The smaller the LPIPS$_{\text{Alex}}$, 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.

[9] https://github.com/apchenstu/TensoRF

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. 21

[2] Anpei Chen, Zexiang Xu, Andreas Geiger, Jingyi Yu, and Hao Su. TensoRF: Tensorial radiance fields. In *ECCV*, 2022. 13, 14, 15, 16, 22

[3] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015. 20, 21, 22

[4] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. ImageNet classification with deep convolutional neural networks. In *NIPS*, 2012. 22

[5] David B. Lindell, Julien N. P. Martel, and Gordon Wetzstein. AutoInt: Automatic integration for fast neural volume rendering. In *CVPR*, 2021. 9

[6] 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. 19

[7] 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. 6, 19, 20, 21, 22

[8] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted Boltzmann machines. In *ICML*, 2010. 19, 21, 22

[9] 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. 11, 20, 21

[10] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015. 20, 22

[11] 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. 19, 21, 22

[12] 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. 20

[13] 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. 20