Real-Time Neural Light Field on Mobile Devices

Junli Cao[1] Huan Wang[2] Pavlo Chemerys[1] Vladislav Shakhrai[1] Ju Hu[1]
Yun Fu[2] Denys Makoviichuk[1] Sergey Tulyakov[1] Jian Ren[1]
[1]Snap Inc. [2]Northeastern University

Abstract

Recent efforts in Neural Rendering Fields (NeRF) have shown impressive results on novel view synthesis by utilizing implicit neural representation to represent 3D scenes. Due to the process of volumetric rendering, the inference speed for NeRF is extremely slow, limiting the application scenarios of utilizing NeRF on resource-constrained hardware, such as mobile devices. Many works have been conducted to reduce the latency of running NeRF models. However, most of them still require high-end GPU for acceleration or extra storage memory, which is all unavailable on mobile devices. Another emerging direction utilizes the neural light field (NeLF) for speedup, as only one forward pass is performed on a ray to predict the pixel color. Nevertheless, to reach a similar rendering quality as NeRF, the network in NeLF is designed with intensive computation, which is not mobile-friendly. In this work, we propose an efficient network that runs in real-time on mobile devices for neural rendering. We follow the setting of NeLF to train our network. Unlike existing works, we introduce a novel network architecture that runs efficiently on mobile devices with low latency and small size, i.e., *saving* $15\times \sim 24\times$ *storage compared with MobileNeRF. Our model achieves high-resolution generation while maintaining real-time inference for both synthetic and real-world scenes on mobile devices,* e.g., 18.04*ms (iPhone 13) for rendering one* 1008×756 *image of real 3D scenes. Additionally, we achieve similar image quality as NeRF and better quality than MobileNeRF (PSNR* 26.15 vs. 25.91 *on the real-world forward-facing dataset)*[1].

1. Introduction

Remarkable progress seen in the domain of neural rendering [32] promises to democratize asset creation and rendering, where no mesh, texture, or material is required – only a neural network that learned a representation of an object or a scene from multi-view observations. The trained model can be queried at arbitrary viewpoints to generate novel views. To be made widely available, this exciting application requires such methods to run on resource-constrained devices, such as mobile phones, conforming to their limitations in computing, wireless connectivity, and hard drive capacity.

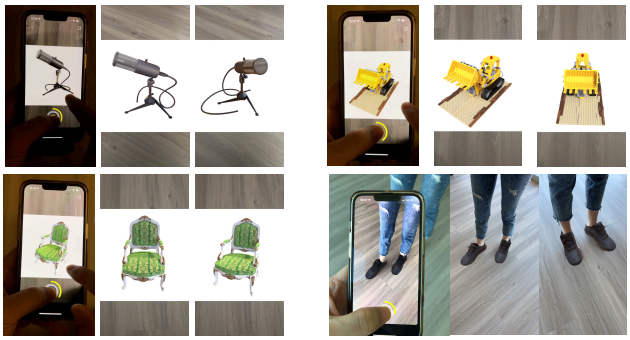

Figure 1. Examples of deploying our approach on mobile devices for real-time interaction with users. Due to the small model size (8.3MB) and fast inference speed (18 $\sim$ 26ms per image on iPhone 13), we can build neural rendering applications where users interact with 3D objects on their devices, enabling various applications such as virtual try-on. We use publicly available software to make the on-device application for visualization [1, 3].

Unfortunately, the impressive image quality and capabilities of NeRF [32] come with a price of slow rendering speed. To return the color of the queried pixel, hundreds of points need to be sampled along the ray that ends up in that pixel, which is then integrated to get the radiance. To enable real-time applications, many works have been proposed [12, 33, 36, 44], yet, they still require high-end GPUs for rendering and hence are not available for resource-constrained applications on mobile or edge devices. An attempt is made to trade rendering speed with storage in MobileNeRF [10]. While showing promising acceleration results, their method requires storage for texturing saving. For example, for a single real-world scene from the forward-facing dataset [32], MobileNeRF requires 201.5MB of storage. Clearly, downloading and storing tens, hundreds, or even thousands of such scenes in MobileNeRF

[1]More demo examples in our Webpage.

format on a device is prohibitively expensive.

A different approach is taken in Neural Light Fields (NeLF) that directly maps a ray to the RGB color of the pixel by performing only one forward pass per ray, resulting in faster rendering speed [5, 24, 27]. Training NeLF is challenging and hence requires increased network capacity. For example, Wang *et al.* [40] propose an 88-layer fully-connected network with residual connections to distill a pre-trained radiance model effectively. While their approach achieves better rendering results than vanilla NeRF at 30× speedup, running it on mobile devices is still not possible, as it takes three seconds to render one 200×200 image on iPhone 13 shown in our experiments.

In this work, we propose MobileR2L, a real-time neural rendering model built with mobile devices in mind. Our training follows a similar distillation procedure introduced in R2L [40]. Differently, instead of using an MLP, a backbone network used by most neural representations, we show that a well-designed *convolutional* network can achieve real-time speed with the rendering quality similar to MLP. In particular, we revisit the network design choices made in R2L and propose to use the 1×1 Conv layer in the backbone. A further challenge with running a NeRF or NeLF on mobile devices is an excessive requirement of RAM. For example, to render an 800×800 image, one needs to sample $640,000$ rays that need to be stored, causing out-of-memory issues. In 3D-aware generative models [9, 15], this issue is alleviated by rendering a radiance feature volume and upsampling it with a convolutional network to obtain a higher resolution. Inspired by this, we render a light-field volume that is upsampled to the required resolution. Our MobileR2L features several major advantages over existing works:

- MobileR2L achieves real-time inference speed on mobile devices (Tab. 3) with better rendering quality, *e.g.*, PSNR, than MobileNeRF on the synthetic and real-world datasets (Tab. 1).
- MobileR2L requires an order of magnitude less storage, reducing the model size to 8.3MB, which is $15.2\times \sim 24.3\times$ less than MobileNeRF.

Due to these contributions, MobileR2L can unlock wide adoption of neural rendering in real-world applications on mobile devices, such as a virtual try-on, where the real-time interaction between devices and users is achieved (Fig. 1).

2. Related Works

Neural Radiance Field (NeRF). NeRF [32] shows the possibility of representing a scene with a simple multi-layer perceptron (MLP) network. Going forward, many extensions follow up in improving rendering quality, *e.g.*, MipNeRF [6], MipNeRF 360 [7], and Ref-NeRF [39], rendering efficiency, *e.g.*, NSVF [28], Nex [42], FastNeRF [13], Baking [16], Plenoctree [44], KiloNeRF [36], DeRF [35], DoNeRF [34], Instant-NGP [33], and R2L [40], and training efficiency, *e.g.*, AutoInt [26] and Plenoxels [12].

Efficient NeRF Rendering. Since this paper falls into the category of improving *rendering efficiency* as we target *real-time* rendering on *mobile* devices, we single out the papers of this group and discuss them in length here. There are generally four groups. **(1)** The first group trades speed with space, *i.e.*, they precompute and cache scene representations and the rendering reduces to table lookup. Efficient data structure like sparse octree, *e.g.*, Plenoctree [44], is usually utilized to make the rendering even faster. **(2)** The second attempts reduce the number of sampled points along the camera ray during rendering as it is the root cause of prohibitively slow rendering speed. Fewer sampled points typically lead to performance degradation, so as compensation, they usually introduce extra information, such as depth, *e.g.*, DoNeRF [34], or mesh, *e.g.*, MobileNeRF [10], to maintain the visual quality. **(3)** The third group takes a "divide and conquer" strategy. DeRF [35] decomposes the scene spatially to Voronoi diagrams and learns each diagram with a small network. KiloNeRF [36] also employs a decomposition scheme. Differently, they decompose the scene into thousands of small regular grinds. Each is tackled with a small network. Such decomposition poses challenges to parallelism. Thus they utilize customized parallelism implementation to obtain speedup. **(4)** The fourth group is a newly surging one, represented by the recent works RSEN [5] and R2L [40]. They achieve rendering efficiency by representing the scene with *NeLF (neural light field*, see related works below) instead of NeRF. NeLF avoids the dense sampling on camera ray, leading to a much faster inference speed than NeRF. On the downside, NeLF is typically much harder to learn than NeRF. As a remedy, these works [40] typically integrate a pre-trained NeRF model as a teacher to synthesize extra data for *distillation* [8, 19]. Therefore, the resulting model is fast with a reasonably small representation size, *i.e.*, the model size.

Neural Light Field (NeLF). Light field is a different way of representing scenes. The idea has a long history in the computer graphics community, *e.g.*, Light fields [24] and Lumigraphs [14] cache plenty of images and enable real-time rendering at the cost of limited camera pose and excessive storage overhead. One of the most intriguing properties of NeLF is that rendering one image only requires one network forward, resulting in a significantly faster speed than NeRF-based methods. With the recent surge of the neural radiance field, some works attempt to revive the idea of the neural light field for efficient neural rendering. Sitzmann *et al.* [37] material the idea of using a neural network to model the scene, and the rendering process reduces to a single network forward. Despite the encouraging idea, their method has only been evaluated on scenes with simple

shapes, not matching the quality of NeRF on complex real-world scenes. Later, RSEN [5] and NeuLF [27] are introduced. RSEN divides the space into many voxel grids. Only in each grid, it is a NeLF, which needs alpha-composition (a step of NeRF) to render the final color, making their method *a mixture of NeLF and NeRF*. R2L [40] is a pure NeLF network that avoids the alpha-composition step in rendering, which is also one of the most relevant works to this paper. However, R2L is still not compact and fast enough for mobile devices. Based on our empirical study, an R2L model runs for around three seconds per frame on iPhone 13 even for low-resolution like 200×200. This paper is meant to push the NeRF-to-NeLF idea even further, making it able to perform *real-time* rendering on mobile devices.

We will mainly compare to MobileNeRF [10] in this paper as it is the *only* method, to our best knowledge, that can run on mobile devices with matching quality to NeRF [32].

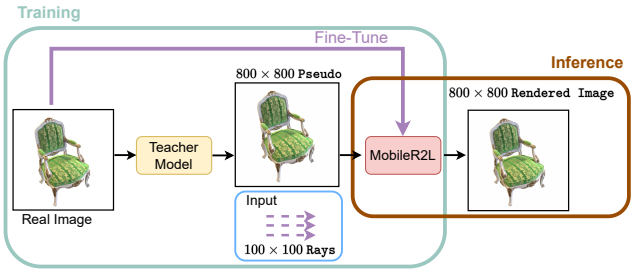

Figure 2. **Training and Inference Pipeline.** The training involves a teacher model to generate pseudo data, which is used to learn the MobileR2L. The teacher model, *e.g.*, NeRF, is trained on real images. Once we have the teacher model, we use it to generate pseudo images, *e.g.*, images with the resolution of 800×800, in addition to down-scaled rays, *e.g.*, rays with spatial size as 100×100, that share the same origin with the pseudo images to train the MobileR2L. After that, we use the real data to fine-tune MobileR2L. For inference, we directly forward the rays into the pre-trained MobileR2L to render images.

3. Methods

3.1. Prerequisites: NeRF and R2L

NeRF. NeRF [32] represents the scene implicitly with an MLP network F_Θ, which maps the 5D coordinate (spatial location (x, y, z) and view direction (θ, ϕ)) to a 1D volume density (opacity, denoted as σ here) and 3D radiance (denoted as c) such that $F_\Theta : \mathbb{R}^5 \mapsto \mathbb{R}^4$. Each pixel of an image is associated with a camera ray. To obtain the color of a pixel, the NeRF method samples many points along the camera ray and accumulates the radiance of all these points via *alpha compositing* [22, 30, 32]:

$$C(\mathbf{r}) = \sum_{i=1}^{N} T_i \cdot (1 - \exp(-\sigma_i \delta_i)) \cdot F_\Theta(\mathbf{r}(t_i), \mathbf{d}),$$
$$T_i = \exp(-\sum_{j=1}^{i-1} \sigma_j \delta_j), \tag{1}$$

where $\mathbf{r}$ means the camera ray; $\mathbf{r}(t_i) = \mathbf{o} + t_i\mathbf{d}$ represents the location of a point on the ray with origin $\mathbf{o}$ and direction $\mathbf{d}$; t_i is the Euclidean distance, *i.e.*, a scalar, of the point away from the origin; and $\delta_i = t_{i+1} - t_i$ refers to the distance between two adjacent sampled points. A stratified sampling approach is employed in NeRF [32] to sample the t_i in Eqn. 1. To enrich the input information, the position and direction coordinates are encoded by *positional encoding* [38], which maps a scalar ($\mathbb{R}$) to a higher dimensional space ($\mathbb{R}^{2L+1}$) through cosine and sine functions, where L (a predefined constant) stands for the frequency order (in the original NeRF [32], $L = 10$ for positional coordinates and $L = 4$ for direction coordinates).

The whole formulation and training of NeRF are straightforward. One critical problem preventing fast inference in NeRF is that the N, *i.e.*, the number of sampled points, in Eqn. 1 is pretty large (256 in the original NeRF paper due to their two-stage coarse-to-fine design). Therefore, the rendering computation for even a single pixel is prohibitively heavy. The solution proposed by R2L [40] is distilling the NeRF representation to NeLF.

R2L. Essentially, a NeLF function maps the oriented ray to RGB. To enrich the input information, R2L proposes a new ray representation – they also sample points along the ray just like NeRF [32] does; but differently, they *concatenate* the points to one vector, which is used as the ray representation and fed into a neural network to learn the RGB. Similar to NeRF, positional encoding [38] is also adopted in R2L to map each scalar coordinate to a high dimensional space. During training, the points are *randomly* (by a uniform distribution) sampled; during testing, the points are fixed.

The output of the R2L model is directly RGB, no density learned, and there is no extra alpha-compositing step, which makes R2L much faster than NeRF in rendering. One downside of the NeLF framework is, as shown in R2L [40], the NeLF representation is much harder to learn than NeRF; so as a remedy, R2L proposes an 88-layer deep ResMLP (residual MLP) architecture (much deeper than the network of NeRF) to serve as the mapping function.

R2L has two stages in training. In the first stage, they use a pre-trained NeRF model as a teacher to synthesize excessive `(origin, direction, RGB)` triplets as pseudo data; and then fed the pseudo data to train the deep ResMLP. This stage can make the R2L model achieve comparable performance to the teacher NeRF model. In the second stage, they finetune the R2L network from the last stage on the *original* data – this step can further boost the rendering quality as shown in the R2L work [40].

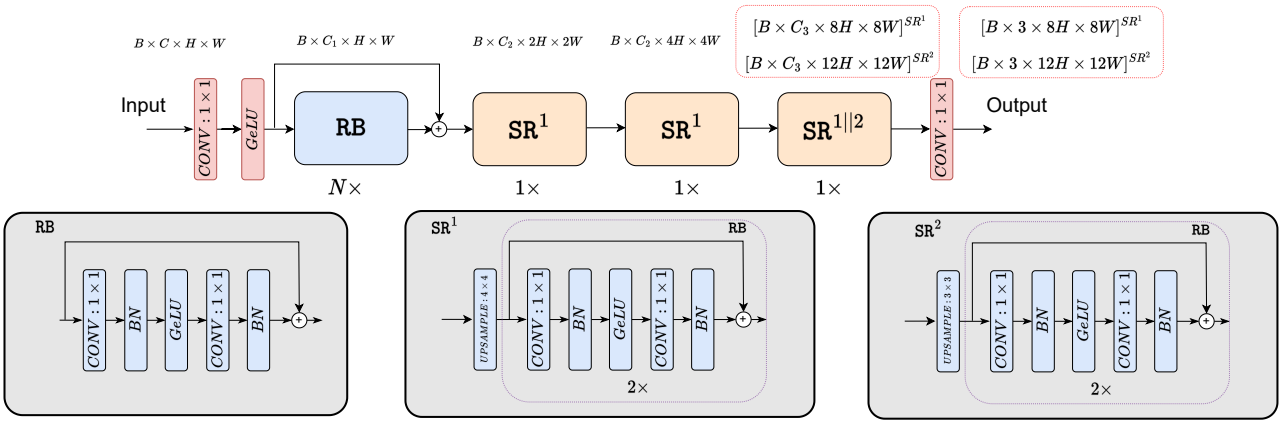

Figure 3. **Overview of Network.** The input tensor of MobileR2L has 4D shape: batch, channel, height, and width. The backbone includes residual blocks (RB) that is repeated 28 times ($N = 28$). Following the backbone, there are two types of super-resolution (SR) modules. The first SR module (SR^1) has kernerl size 4×4 in the Transpose CONV layer that doubles the input H, W to $2H, 2W$, whereas the second SR module (SR^2) has kernerl size 3×3, tripling the spatial size to $3H, 3W$. The configuration of $3 \times \text{SR}^1$ is used in the synthetic 360° dataset that upsamples the input $8\times$. For the real-world forward-facing dataset, we use the combination of $2 \times \text{SR}^1 + \text{SR}^2$ that upsamples the input $12\times$. Moreover, we use various output channels across RB and SR: $C_1 = 256$, $C_2 = 64$, and $C_3 = 16$.

3.2. MobileR2L

3.2.1 Overview

We follow the learning process of R2L to train our proposed MobileR2L, namely, using a pre-trained teacher model, such as NeRF [32], to generate pseudo data for the training of a lightweight neural network. To reduce the inference speed, we aim only to forward the network *once* when rendering an image. However, under the design of R2L, although one pixel only requires one network forward, directly feeding rays with large spatial size, *e.g.*, 800×800, into a network causes memory issues. Therefore, R2L forwards a partial of rays each time, increasing the speed overhead. To solve the problem, we introduce super-resolution modules, which upsample the low-resolution input, *e.g.*, 100×100, to a high-resolution image. Thus, we can obtain a high-resolution image with only one forward pass of the neural network during inference time. The training and inference pipeline is illustrated in Fig. 2, and we introduce more details for our network architecture in the following.

3.2.2 Network Architectures

The input rays can be represented as $\mathbf{x} \in \mathbb{R}^{B,6,H,W}$, where B denotes the batch size and H and W denote the spatial size. The ray origin and view directions are concatenated as the second dimension of $\mathbf{x}$. We then apply positional encoding $\gamma(\cdot)$ on $\mathbf{x}$ to map the ray origin and view directions into a higher dimension. Thus, we get the input of our neural network as $\gamma(\mathbf{x})$.

The network includes two main parts: an efficient backbone and Super-Resolution (SR) modules for high-resolution rendering, with the architecture provided in Fig. 3. Instead of using Fully Connected (FC) or linear layers for the network that is adopted by existing works [32, 40], we only apply convolution (CONV) layers in the backbone and super-resolution modules.

There are two main reasons for replacing FC with CONV layers. First, the CONV layer is better optimized by compilers than the FC layer [29]. Under the same number of parameters, the model with CONV 1×1 runs around 27% faster than the model with FC layers, as shown in Tab. 4. Second, suppose FC is used in the backbone, in that case, extra `Reshape` and `Permute` operations are required to modify the dimension of the output features from the FC to make the features compatible with the CONV layer in the super-resolution modules, as the FC and CONV calculate different tensor dimension. Such `Reshape` or `Permute` operation might not be hardware-friendly on some mobile devices [25]. With the CONV employed as the operator in the network, we then present more details for the backbone and SR modules.

Efficient Backbone. The architecture of the backbone follows the design of residual blocks from R2L [40]. Different from R2L, we adop the CONV layer instead of the FC layer in each residual block. The CONV layer has the size of kernel and stride as 1. Additionally, we use the normalization and activation functions in each residual block, which can improve the network performance without introducing latency overhead (experimental details in Tab. 4). The normalization and activation are chosen as batch normalization [20] and GeLU [17]. The backbone contains 60 CONV layers in total.

Super-Resolution Modules. To reduce the latency when

running the neural rendering on mobile devices, we aim to forward the neural network *once* to get the synthetic image. However, the exisitng network design of the neural light field requires large memory for rendering a high-resolution image, which surpasses the memory constrain the mobile devices. For example, rendering an image of 800×800 requires the prediction of $640,000$ rays. Forwarding these rays at once using the network from R2L [40] causes the out of memory issue even on the Nvidia Tesla A100 GPU (40G memory).

To reduce the memory and latency cost for high-resolution generation, instead of forwarding the number of rays that equals to the number of pixels, we only forward a partial of rays and learn all the pixels via super-resolution. Specifically, we propose to use the super-resolution modules following the efficient backbone to upsample the output to a high-resolution image. For example, to generate a 800×800 image, we forward a 4D tensor $\mathbf{x}$ with spatial size as 100×100 to the network and upsample the output from backbone three times. The SR module includes the stacking of two residual blocks. The first block includes three CONV layers with one as a 2D Transpose CONV layer and two CONV 1×1 layers; the second block includes two CONV 1×1 layers. After the SR modules, we apply another CONV layer followed by the Sigmoid activation to predict the final RGB color. We denote our model as D60-SR3 where it contains 60 CONV layers in the efficient backbone and 3 SR modules.

4. Experiments

Datasets. We conduct the comparisons mainly on two datasets: realistic synthetic 360° [32] and real-world forward-facing [31, 32]. The synthetic 360° dataset contains 8 path-traced scenes, with each scene including 100 images for training and 200 images for testing. Forward-facing contains 8 real-world scenes captured by cellphones, where the images in each scene vary from 20 to 60, and $1/8$ images are used for testing. We conduct our experiments (training and testing) on the resolution of 800×800 for synthetic 360° and 1008×756 ($4\times$ down-scaled from the original resolution) for forward-facing.

Implementation Details. We follow the training scheme of R2L [40], *i.e.*, using a teacher model to render pseudo images for the training of MobileR2L network. Specifically, we synthesize 10K pseudo images from the pre-trained teacher model [2] for each scene. We first train our MobileR2L on the generated pseudo data and then fine-tune it on the real data, as shown in Fig. 2. In all the experiments, we employ Adam [23] optimizer with an initial learning rate 5×10^{-4} that decays during the training. Our experiments are conducted on a cluster of Nvidia V100 and A100 GPUs with the batch size as 54 for the main results on the synthetic 360° and batch size as 36 for the forward-facing.

Table 1. **Quantitative Comparison** on Synthetic 360° and Forward-facing. Our method obtains better results on the three metrics than NeRF for the two datasets. Compared with MoibleNeRF and SNeRG, we achieve better results on most of the metrics.

	Synthetic 360°			Forward-facing		
	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓
NeRF [32]	31.01	0.947	0.081	26.50	0.811	0.250
NeRF-Pytorch [43]	30.92	0.991	0.045	26.26	0.965	0.153
SNeRG [16]	30.38	0.950	**0.050**	25.63	0.818	**0.183**
MoibleNeRF [10]	30.90	0.947	0.062	25.91	0.825	**0.183**
MobileR2L (Ours)	**31.34**	**0.993**	0.051	**26.15**	**0.966**	0.187
Our Teacher	33.09	0.961	0.052	26.85	0.827	0.226

Different from R2L, the spatial size of the input rays and the output rendered images in our approach are different. For each high-resolution image generated by the teacher model, we save the input rays corresponding to a lower-resolution image where the camera origins and directions are the same as the high-resolution one while the focal is down-scaled accordingly. Additionally, we do not sample the rays from different images as in R2L. Instead, the rays in each training sample share the same origin and reserve their spatial locations.

Considering the training data of the synthetic 360° and forward-facing datasets have different resolutions, the spatial size of the inputs for the two datasets are slightly different. Our network takes input with the spatial size of 100×100 for the synthetic 360° dataset and upsamples 8 times to render 800×800 RGB images. In contrast, the spatial size of 84×63 is used in the forward-facing dataset, and 1008×756 image ($12\times$ upsampling) is rendered. The kernel size and padding are adjusted in the last transpose CONV layer to achieve $8\times$ and $12\times$ upsampling with the 3 SR blocks.

4.1. Comparisons

Rendering Performance. To understand the image quality of various methods, we report three common metrics, including PSNR, SSIM [41], and LPIPS [45], on the realistic synthetic 360° and real forward-facing datasets, as demonstrated in Tab. 1. Compared with NeRF [32], our approach achieves better results on PSNR, SSIM, and LPIPS for the synthetic 360° dataset. On the forward-facing dataset, we obtain better SSIM and LPIPS than NeRF [32]. Similarly, our method achieves better results for all three metrics than MobileNeRF [10] on the synthetic 360° dataset and better PSNR and SSIM on the forward-facing dataset. Compared SNeRG [16], we obtain better PSNR and SSIM.

We also show the performance of the teacher model that used for training MobileR2L in Tab. 1 (Our Teacher). Note that there is still a performance gap between the student model (MobileR2L) and the teacher model. However, as we

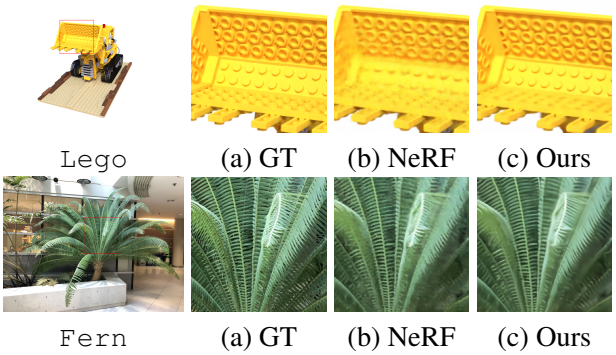

Figure 4. Visual comparison between our method and NeRF [32] (trained via NeRF-Pytorch [43]) on the synthetic 360° `Lego` (size: 800×800×3) and real-world forward-facing scene `Fern` (size: 1008×756×3). *Best viewed in color*. Please refer to our webpage for more visual comparison results.

Figure 5. Zoom-in comparisons. *Top row*: MobileNeRF [10]. Results are obtained from the code and demo released by the authors. *Bottom row*: MobileR2L. Our approach renders high-quality images even for zoom-in views.

show following (Tab. 5), a better teacher model can lead to a student model with higher performance. Compared with MoibleNeRF and SNeRG, our approach has the advantage that we can directly leverage the high-performing teacher models to help improve student training for different application scenarios. We further show the qualitative comparison results in Fig. 4. On the synthetic scene `Lego`, our MobileR2L outperforms NeRF clearly, delivering sharper and less distorted shapes and textures. On the real-world scene `Fern`, our result is less noisy, and the details, *e.g.*, the leaf tips, are sharper. Additionally, we provide the zoom-in comparison with MoibleNeRF [10] in Fig. 5. Our method achieves high-quality rendering for zoom-in view, which is especially important for 3D assets that users might perform zoom-in to look for more image details.

Disk Storage. One significant advantage of our method is that we do not require extra storage, even for complex scenes. As shown in Tab. 2, the storage of our approach is 8.3MB for both synthetic 360° and forward-facing datasets. The mesh-based methods like MobileNeRF demand more storage for real-world scenes due to saving more complex textures. As a result, our approach asks 24.3× less disk storage than MobileNeRF on forward-facing and 15.2× less on synthetic 360° dataset.

Table 2. **Analysis of Storage (MB)** required for different rendering methods. Our method has a clear advantage over existing works with much less storage required.

	Synthetic 360°			Forward-facing		
	MoibleNeRF [10]	SNeRG [16]	Ours	MobileNeRF [10]	SNeRG [16]	Ours
Disk storage	125.8	86.8	**8.3**	201.5	337.3	**8.3**

Table 3. **Analysis of Inference Speed.** Latency (ms) is obtained on iPhone with iOS 16. Follwing MoibleNeRF [10], we use the notation $\frac{M}{N}$ to indicate that M out of N scenes in the Forward-facing dataset that can not run on devices. Specifically, MobileNeRF can not render `Leaves` and `Orchids` in Forward-facing.

	Synthetic 360°		Forward-facing	
	MobileNeRF [10]	Ours	MobileNeRF [10]	Ours
iPhone 13	17.54	26.21	$27.15\frac{2}{8}$	18.04
iPhone 14	16.67	22.65	$20.98\frac{2}{8}$	16.48

Inference Speed. We profile and report the rendering speed of our proposed approach on iPhones (13, and 14, iOS 16) in Tab. 3. The models are compiled with CoreMLTools [11]. Our proposed method runs faster on real forward-facing scenes than the realistic synthetic 360° scenes. The latency discrepancy between the two datasets comes from the different input spatial sizes. MobileNeRF shows a lower latency than our models on the realistic synthetic 360° but higher on the real-world scenes. Both methods can run in real-time on devices. Note that MobileNeRF can not render two scenes, *i.e.*, `Leaves` and `Orchids`, due to memory issues, as they require complex texture to model the geometry. On the other hand, our approach is robust for different datasets.

Discussion. From the comparison of the rendering quality, disk storage, and inference speed, it can be seen that MobileR2L achieves overall better performance than MobileNeRF. More importantly, considering the usage of neural rendering on real-world applications, MobileR2L is more suitable as it requires much less storage, reducing the constraint for hardware and can render real-world scenes in real-time on mobile devices.

4.2. Ablation Analysis

Here we perform the ablation analysis to understand the design choices of the network. We use the scene of `Chair` from the synthetic 360° to conduct the analysis. All models

are trained for 20K iterations.

Options for Backbone. We study the two available operators for designing the backbone. MLP and 1×1 CONV layer are essentially equivalent operators and perform the same calculations, thus resulting in similar performance. However, we observe around 27% latency reduction on mobile devices (iPhone 13) when replacing the MLP layer with the 1×1 CONV layer. Specifically, as shown in Tab. 4, we design two networks, *i.e.*, MLP and CONV2D, with only residual blocks as in Fig. 3 but removing the activation, normalization, and super-resolution modules. We use the input size as 100×100 for the two models. Since the super-resolution modules are left out, we train the two networks for generating 100×100 images. As can be seen, the CONV2D model has a faster inference speed than MLP with similar performance. This is due to the CONV operation being a better-optimized operator than MLP. Additionally, due to the intrinsic design of our proposed MobileR2L, employing MLP layers requires two additional operators, *i.e.*, `Permute` and `Reshape`, before feeding to the super-resolution modules. `Permute` and `Reshape` might involve data movement that adds unnecessary overheads on some edge devices [25].

Analysis of Activation Function. R2L [40] and NeRF [32] use ReLU [4] activation to add the non-linearity to the models. In our proposed MobileR2L, we use GELU [18] instead. As shown in Tab. 4, by comparing the *CONV2D + ReLU* and *CONV2D + GeLU*, which are two networks trained with ReLU and GeLU activations, we notice that GeLU brings about 0.17 PSNR boost without any additional latency overhead. Similarly, we show that incorporating BatchNorm [21] layer into the ResBlock is also beneficial for better performance without introducing extra latency, as shown by *CONV2D + GeLU + BN* in Tab. 4. The three networks in the experiments are also trained to render 100×100 images.

Analysis on Input Dimension. We further analyze the optimal spatial resolution for the input tensor. Specifically, we benchmark the performance of three approaches, namely, 50×50 - *NeRF Teacher*, 100×100 - *NeRF Teacher*, and 200×200 - *NeRF Teacher* with the spatial size of input as the square of 50, 100, and 200 respectively. These models contain super-resolution modules to render 800×800 images and are trained with the NeRF [32] as a teacher model. Results are presented in Tab. 5. Using small input spatial size, *i.e.*, 50×50, achieves $2\times$ speedup than the model with a larger size, *i.e.*, 100×100, as less computation is required. However, the performance is also degraded with a 0.25 PSNR drop. Further increasing the spatial size to 200×200 makes the model unable to achieve real-time inference. Thus, we do not report the rendering performance, *e.g.*, PSNR, of the model.

Analysis of SR modules. We further show the necessity of using SR modules. We use the spatial size of 800×800 as the network input to render images with the same spatial size. We denote the setting as 800×800 - *w/o SR* in Tab. 5. Profiling the latency of such a network leads to compilation errors due to intensive memory usage. Thus, our proposed SR module is an effective approach for high-resolution synthesis without introducing much computation overhead.

Table 4. **Analysis of Network Design.** For all the comparisons, we use the input tensor with the spatial size as 100×100 and render the image with spatial size. The latency (ms) is measured on iPhone 13 (iOS16) with models compiled with CoreMLTools [11]

	PSNR↑	SSIM↑	LPIPS↓	Latency↓
MLP	19.13	0.9759	0.6630	19.57
CONV2D	19.16	0.9759	0.6301	14.30
CONV2D + ReLU	26.82	0.9949	0.0282	16.20
CONV2D + GeLU	26.99	0.9949	0.0730	17.00
CONV2D + GeLU + BN	**27.18**	**0.9954**	**0.0259**	17.00

Table 5. **Analysis of the spatial size of the input, usage of teacher model, and ray presentation**. Besides image quality metrics, we show the number of parameters for each model and the latency when running on iPhone 13.

	Params	PSNR↑	SSIM↑	LPIPS↓	Latency↓
50×50 - NeRF Teacher	3.9M	30.40	0.9965	0.0686	13.04
100×100 - NeRF Teacher	3.9M	30.65	0.9966	0.0668	26.21
200×200 - NeRF Teacher	3.9M	-	-	-	73.76
800×800 - w/o SR	3.9M	-	-	-	Error
100×100 - MipNeRF Teacher	3.9M	30.83	0.0997	0.0564	26.21
100×100 - MipNeRF Teacher, $K16$, $L10$	4.1M	30.90	0.9968	0.0583	31.05
100×100 - MipNeRF Teacher, $K16$, $L10$, D100	6.8M	31.37	0.9972	0.0470	44.52

Choice of Teacher Models. Since both R2L [40] and MobileR2L use a teacher model for generating pseudo data to train a lightweight network, we study whether a more powerful teacher model can improve performance. To conduct the experiments, we use MipNeRF [6] as a teacher model for the training MobileR2L, given MipNeRF shows better performance than NeRF on the synthetic $360°$ dataset. We denote the approach as 100×100 - *MipNeRF Teacher*. Through the comparison with 100×100 - *NeRF Teacher*, as in Tab. 5, we notice the quality of the rendered images is improved, *e.g.*, the PSNR is increased by 0.18, without the extra cost of latency. The comparison demonstrates that our approach has the potential to render higher-quality images when there are better teacher models are provided.

Analysis on Ray Representation. Here we analyze how the ray representation affects the latency-performance trade-off of MobileR2L. We follow the same ray representation paradigm as in R2L [40] and positional encoding as in NeRF [32]. Specifically, R2L sample K 3D points along the ray, and each point is mapped to a higher dimension by positional encoding with L positional coordinates. R2L sets $K = 16$ and $L = 10$, resulting in a vector with a dimension

of $1,008$. We apply the same setting to MobileR2L and denote the model as 100×100, $K16$, $L10$ - *MipNeRF Teacher* in Tab. 5. For our implementation in MobileR2L, we set $K = 8$ and $L = 6$ for the model, which has the dimension per ray as 312. The model is 100×100 - *MipNeRF Teacher*. By comparing the performance of the two models, we notice that larger K and L lead to negligible improvement over PSNR (0.07) while higher inference speed and bigger model size (more parameters). Therefore, we chose $K = 8$ and $L = 6$ as they bring more benefits for model size and latency, which are essential metrics when deploying neural rendering models on mobile devices.

Depth of the Backbone. Lastly, we show the effects of the backbone depth on the model performance. We use the depth as 60 for our backbone. By increasing the number of residual blocks in the backbone, *i.e.*, setting depth as 100, we obverse better model performance at the cost of higher latency, as shown by comparing the last two rows in Tab. 5. Using depth as 100 significantly increases the latency and the number of parameters, and the model fails to run in real time. Thus, we chose depth as 60, given the better trade-off between latency, model size, and performance.

4.3. Real-World Application

Here we demonstrate the usage of our technique for building a real-world application. Given the small size and faster inference speed of our model, we create a shoe try-on application that runs on mobile devices. Users can directly try the shoe rendered by MobileR2L using their devices, enabling real-time interaction.

The pipeline for building the application is illustrated in Fig. 6. We first use iPhone to capture 100 shoe images for training. The images are then segmented to remove the background. After that, we train a NeRF model [32] to generate pseudo data, which is later used for learning MobileR2L to render images with resolution as 1008×756. We apply foot tracking and overlay the rendered shoe on top of the user's feet. As can be seen from Fig. 6, our model is able to render high-quality images from various views for different users. The try-one usage proves the potential of leveraging neural rendering for building various applications such as Augment Reality.

5. Limitation and Conclusion

This work presents MobileR2L, the first neural network that renders images with similar quality as NeRF [32] while running in real-time on model devices. We perform extensive experiments to design an optimized network architecture that can be trained via data distillation to render high-resolution images, *e.g.*, 800×800. Additionally, since we do not require other information besides the neural network, MobileR2L dramatically saves the storage than other mesh-based methods like MoibleNeRF [10]. Furthermore, we prove that with our design, neural rendering can be used to build real-world applications, achieving real-time user interaction.

Figure 6. **Virtual Try-On Application.** From the collected images using a cellphone (a), we segment the foreground shoe (b) to train a MobileR2L model. We deploy the model on mobile devices, and users can directly try the shoe. The model renders images for novel views when users rotate the phone or change the foot positions (c).

Figure 7. Visual comparison on the real-world scene Room. Both our model and NeRF fail to synthesize the whiteboard writings on the upper-left of the cutout patch.

Although MobileR2L achieves promising inference speed with small model sizes, there are still two limitations of the current work that can be improved. First, we follow the training recipe of R2L [40], and R2L uses 10K pseudo images generated by the teacher model to train the student model. The number of training images is much more than the images used to train NeRF-based approaches, which require around 100 images, resulting in a longer training time than NeRF-based methods. Therefore, a future direction could be studying how to reduce the training costs for distillation-based works like R2L. Second, MobileR2L fails to generate some high-frequency details for the images. We show example images in Fig. 7. Using a larger model may alleviate the problem given more model capacity. Nevertheless, the inference speed is also increased for running the larger model on devices. Future efforts might help further optimize the network and training pipeline to boost the performance of the current model.

References

[1] Lens studio. https://ar.snap.com/en-US/lens-studio. 1

[2] Nerf-factory. https://github.com/kakaobrain/NeRF-Factory. 5

[3] Snap ml. https://docs.snap.com/lens-studio/references/guides/lens-features/machine-learning/ml-overview. 1

[4] Abien Fred Agarap. Deep learning using rectified linear units (relu), 2018. 7

[5] Benjamin Attal, Jia-Bin Huang, Michael Zollhöfer, Johannes Kopf, and Changil Kim. Learning neural light fields with ray-space embedding. In *CVPR*, 2022. 2, 3

[6] Jonathan T Barron, Ben Mildenhall, Matthew Tancik, Peter Hedman, Ricardo Martin-Brualla, and Pratul P Srinivasan. Mip-nerf: A multiscale representation for anti-aliasing neural radiance fields. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5855–5864, 2021. 2, 7

[7] Jonathan T Barron, Ben Mildenhall, Dor Verbin, Pratul P Srinivasan, and Peter Hedman. Mip-nerf 360: Unbounded anti-aliased neural radiance fields. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5470–5479, 2022. 2

[8] Cristian Buciluǎ, Rich Caruana, and Alexandru Niculescu-Mizil. Model compression. In *SIGKDD*, 2006. 2

[9] 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, et al. Efficient geometry-aware 3d generative adversarial networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16123–16133, 2022. 2

[10] Zhiqin Chen, Thomas Funkhouser, Peter Hedman, and Andrea Tagliasacchi. Mobilenerf: Exploiting the polygon rasterization pipeline for efficient neural field rendering on mobile architectures. *arXiv preprint arXiv:2208.00277*, 2022. 1, 2, 3, 5, 6, 8

[11] CoreMLTools. Use coremltools to convert models from third-party libraries to core ml., 2021. 6, 7, 11

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

[13] Stephan J Garbin, Marek Kowalski, Matthew Johnson, Jamie Shotton, and Julien Valentin. Fastnerf: High-fidelity neural rendering at 200fps. *arXiv preprint arXiv:2103.10380*, 2021. 2

[14] Steven J Gortler, Radek Grzeszczuk, Richard Szeliski, and Michael F Cohen. The lumigraph. In *Proceedings of the Annual Conference on Computer Graphics and Interactive Techniques*, 1996. 2

[15] Jiatao Gu, Lingjie Liu, Peng Wang, and Christian Theobalt. Stylenerf: A style-based 3d-aware generator for high-resolution image synthesis. *arXiv preprint arXiv:2110.08985*, 2021. 2

[16] Peter Hedman, Pratul P Srinivasan, Ben Mildenhall, Jonathan T Barron, and Paul Debevec. Baking neural radiance fields for real-time view synthesis. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5875–5884, 2021. 2, 5, 6

[17] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016. 4

[18] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus), 2016. 7

[19] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. In *NIPS Workshop*, 2014. 2

[20] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015. 4

[21] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015. 7

[22] James T Kajiya and Brian P Von Herzen. Ray tracing volume densities. *SIGGRAPH*, 18(3):165–174, 1984. 3

[23] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014. 5

[24] Marc Levoy and Pat Hanrahan. Light field rendering. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 31–42, 1996. 2

[25] Yanyu Li, Geng Yuan, Yang Wen, Eric Hu, Georgios Evangelidis, Sergey Tulyakov, Yanzhi Wang, and Jian Ren. Efficientformer: Vision transformers at mobilenet speed. *arXiv preprint arXiv:2206.01191*, 2022. 4, 7

[26] David B Lindell, Julien NP Martel, and Gordon Wetzstein. Autoint: Automatic integration for fast neural volume rendering. In *CVPR*, 2021. 2

[27] Celong Liu, Zhong Li, Junsong Yuan, and Yi Xu. Neulf: Efficient novel view synthesis with neural 4d light field. In *EGSR*, 2022. 2, 3

[28] Lingjie Liu, Jiatao Gu, Kyaw Zaw Lin, Tat-Seng Chua, and Christian Theobalt. Neural sparse voxel fields. In *NeurIPS*, 2020. 2

[29] Xingyu Liu, Jeff Pool, Song Han, and William J Dally. Efficient sparse-winograd convolutional neural networks. *arXiv preprint arXiv:1802.06367*, 2018. 4

[30] Nelson Max. Optical models for direct volume rendering. *TVCG*, 1(2):99–108, 1995. 3

[31] 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 Transactions on Graphics (TOG)*, 38(4):1–14, 2019. 5

[32] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. In *ECCV*, 2020. 1, 2, 3, 4, 5, 6, 7, 8, 11, 12

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

[34] Thomas Neff, Pascal Stadlbauer, Mathias Parger, Andreas Kurz, Joerg H. Mueller, Chakravarty R. Alla Chaitanya, Anton S. Kaplanyan, and Markus Steinberger. DONeRF: Towards Real-Time Rendering of Compact Neural Radiance

Fields using Depth Oracle Networks. *Comput. Graph. Forum*, 2021. 2

[35] Daniel Rebain, Wei Jiang, Soroosh Yazdani, Ke Li, Kwang Moo Yi, and Andrea Tagliasacchi. Derf: Decomposed radiance fields. In *CVPR*, 2021. 2

[36] Christian Reiser, Songyou Peng, Yiyi Liao, and Andreas Geiger. Kilonerf: Speeding up neural radiance fields with thousands of tiny mlps. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 14335–14345, 2021. 1, 2

[37] Vincent Sitzmann, Semon Rezchikov, William T Freeman, Joshua B Tenenbaum, and Fredo Durand. Light field networks: Neural scene representations with single-evaluation rendering. In *NeurIPS*, 2021. 2

[38] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NeurIPS*, 2017. 3

[39] 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. 2

[40] 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. 2, 3, 4, 5, 7, 8, 11

[41] Zhou Wang, A.C. Bovik, H.R. Sheikh, and E.P. Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4):600–612, 2004. 5

[42] Suttisak Wizadwongsa, Pakkapon Phongthawee, Jiraphon Yenphraphai, and Supasorn Suwajanakorn. Nex: Real-time view synthesis with neural basis expansion. In *CVPR*, 2021. 2

[43] Lin Yen-Chen. Nerf-pytorch. https://github.com/yenchenlin/nerf-pytorch/, 2020. 5, 6, 11, 12

[44] Alex Yu, Ruilong Li, Matthew Tancik, Hao Li, Ren Ng, and Angjoo Kanazawa. Plenoctrees for real-time rendering of neural radiance fields. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5752–5761, 2021. 1, 2

[45] Richard Zhang, Phillip Isola, Alexei A. Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric, 2018. 5

A. Additional Ablation Analysis

Besides the ablation study shown in Sec. 4.2, we provide more analysis of choices for deciding network architectures. The baseline network has the backbone with depth as 60 layers and 3 Residual Blocks (RB) in each Super-Resolution (SR) module. We use the data from `Chair` in the synthetic 360° dataset to train all models with 300K iterations on Nvidia V100 GPU with batch size as 8. The latency is profiled using iPhone 13 (iOS 16) with CoreML-Tools [11]. For the models that are too big to fit into the Nvidia V100 GPU, we only benchmark the latency instead of training the models. The conducted comparisons are introduced as follows:

- **Number of RB per SR module.** We verify the number of RB in each SR module. From Tab. 6, using 3 RB in each SR model, *i.e.*, 3RB per SR, achieves a better trade-off between network performance, *e.g.*, PSNR, and latency.
- **Number of output channels in SR.** We change the number of output channels of the RB in SR. We choose the design of 64 output channels in the first two SR modules and 16 channels in the last SR module, *i.e.*, C64-64-16 in Tab. 6, for it has a real-time inference speed on the mobile device while maintaining a good performance.
- **Width of backbone.** We adopt different width, *i.e.,* number of channels in the convolution layers, for the backbone. The network with the width as 256, *i.e.,* W256 in Tab. 6, gives us the better trade-off between latency and network performance.
- **Depth of backbone.** Lastly, we show how the depth, *i.e.,* number of layers, in the backbone affects the performance. We chose the depth as 60, *i.e.,* D60 in Tab. 6, due to the better PSNR and satisfied latency.

B. Per-Scene Quantitative Results

Here we provide detailed per-scene comparison results (PSNR, SSIM, and LPIPS) on the synthetic 360° (Tab. 7, Tab. 8, and Tab. 9) and forward-facing (Tab. 10, Tab. 11, and Tab. 12) datasets. Please note that as we implement our framework following NeRF-Pytorch [43] and R2L [40], we use the same evaluating pipeline for measuring the quantitative metrics. We also compare our results with the models trained by NeRF-Pytorch [43]. As can be seen from the comparisons, our approach (MobileR2L) achieves comparable or even better results than NeRF [32] and NeRF-Pytorch [43].

C. More Visual Comparisons

In Fig. 8, we present more visual comparison results on the synthetic 360° and forward-facing datasets. We note, (1) MobileR2L can produce the *correct* texture details that NeRF cannot, *e.g.*, on the scene `Mic`, NeRF almost loses the grid texture of the Mic while our MobileR2L manages to render it out; similarly, on the scene `Horn`, there is (at least) one button on the glass wall missed by NeRF while our MobileR2L does not. (2) When both MobileR2L and NeRF can render out the details, MobileR2L typically generates *clearer, sharper, and less noisy* results: on the scene `T-Rex`, it is obvious that our MobileR2L renders much less noisy railing; similar phenomenon can also be observed on the scene `Hotdog`, `Material`, and `Drum`.

Table 6. **Ablation analysis on network architectures.** We report the number of parameters (#Params), PSNR, SSIM, LPIPS, and Latency (ms, on iPhone 13) for each design choice.

	#Params	PSNR↑	SSIM↑	LPIPS↓	Latency↓
1RB per SR	3.9M	31.58	0.9973	0.0503	22.38
2RB per SR	3.9M	31.63	0.9973	0.0484	26.21
3RB per SR	3.9M	31.73	0.9973	0.0368	30.42
4RB per SR	4.0M	31.59	0.9972	0.0508	33.80
C16-16-16	3.8M	31.01	0.9969	0.0644	22.77
C32-32-32	3.9M	31.39	0.9972	0.0525	35.77
C64-64-16	3.9 M	31.63	0.9973	0.0484	26.21
C64-64-64	3.9M	-	-	-	59.31
W64	0.3M	28.83	0.9951	0.0896	13.22
W128	1.0M	30.23	0.9963	0.0699	17.91
W256	3.9M	31.63	0.9973	0.0484	26.21
W384	8.7M	32.28	0.9977	0.0359	39.08
W512	15.4M	-	-	-	53.47
D30	1.9M	30.86	0.9965	0.0609	18.72
D60	3.9M	31.63	0.9973	0.0484	26.21
D80	5.2M	31.60	0.9972	0.0499	31.49
D100	6.6M	-	-	-	36.55

Table 7. Per-scene PSNR↑ comparison on the Synthetic 360° dataset between NeRF [32], NeRF-Pytorch [43], and our approach.

Method	Chair	Drums	Ficus	Hotdog	Lego	Materials	Mic	Ship	Average
NeRF [32]	33.00	25.01	30.13	36.18	32.54	29.62	32.91	28.65	31.01
NeRF-Pytorch [43]	33.31	25.14	30.28	36.52	31.80	29.25	32.50	28.54	30.92
MobileR2L (Ours)	33.66	25.05	29.80	36.84	32.18	30.54	34.37	28.75	31.34

Table 8. Per-scene SSIM↑ comparison on the Synthetic 360° dataset between NeRF [32], NeRF-Pytorch [43], and our approach.

Method	Chair	Drums	Ficus	Hotdog	Lego	Materials	Mic	Ship	Average
NeRF [32]	0.967	0.925	0.964	0.974	0.961	0.949	0.980	0.856	0.947
NeRF-Pytorch [43]	0.998	0.985	0.996	0.998	0.991	0.989	0.996	0.980	0.991
MobileR2L (Ours)	0.998	0.986	0.996	0.998	0.992	0.992	0.997	0.982	0.993

Table 9. Per-scene LPIPS↓ comparison on the Synthetic 360° dataset between NeRF [32], NeRF-Pytorch [43], and our approach.

Method	Chair	Drums	Ficus	Hotdog	Lego	Materials	Mic	Ship	Average
NeRF [32]	0.046	0.091	0.044	0.121	0.050	0.063	0.028	0.206	0.081
NeRF-Pytorch [43]	0.025	0.066	0.023	0.022	0.029	0.035	0.021	0.144	0.045
MobileR2L (Ours)	0.027	0.083	0.025	0.026	0.043	0.029	0.012	0.162	0.051

Table 10. Per-scene PSNR↑ comparison on the Forward-facing dataset between NeRF [32], NeRF-Pytorch [43], and our approach.

Method	Room	Fern	Leaves	Fortress	Orchids	Flower	T-Rex	Horns	Average
NeRF [32]	32.70	25.17	20.92	31.16	20.36	27.40	26.80	27.45	26.50
NeRF-Pytorch [43]	32.10	24.80	20.50	31.20	20.45	27.50	26.48	27.05	26.26
MobileR2L (Ours)	32.09	24.39	20.52	30.81	20.06	27.61	26.71	27.01	26.15

Table 11. Per-scene SSIM↑ comparison on the Forward-facing dataset between NeRF [32], NeRF-Pytorch [43], and our approach.

Method	Room	Fern	Leaves	Fortress	Orchids	Flower	T-Rex	Horns	Average
NeRF [32]	0.948	0.792	0.690	0.881	0.641	0.827	0.880	0.828	0.811
NeRF-Pytorch [43]	0.989	0.976	0.921	0.995	0.920	0.968	0.972	0.983	0.965
MobileR2L (Ours)	0.995	0.973	0.923	0.995	0.916	0.971	0.973	0.982	0.966

Table 12. Per-scene LPIPS↓ comparison on the Forward-facing dataset between NeRF [32], NeRF-Pytorch [43], and our approach.

Method	Room	Fern	Leaves	Fortress	Orchids	Flower	T-Rex	Horns	Average
NeRF [32]	0.178	0.280	0.316	0.171	0.321	0.219	0.249	0.268	0.250
NeRF-Pytorch [43]	0.089	0.210	0.921	0.995	0.920	0.968	0.972	0.983	0.153
MobileR2L (Ours)	0.088	0.239	0.280	0.103	0.296	0.150	0.121	0.217	0.187

Figure 8. More visual comparisons between our method and NeRF [32] (trained via NeRF-Pytorch [43]) on the synthetic 360° (size: $800 \times 800 \times 3$) and real-world forward-facing scenes (size: $1008 \times 756 \times 3$). *Best viewed in color and zoomed in*.