Real-Time Neural Light Field on Mobile Devices

Junli Cao\textsuperscript{1}  Huan Wang\textsuperscript{2}  Pavlo Chemerys\textsuperscript{1}  Vladislav Shakhrai\textsuperscript{1}  Ju Hu\textsuperscript{1}
Yun Fu\textsuperscript{2}  Denys Makoviichuk\textsuperscript{1}  Sergey Tulyakov\textsuperscript{1}  Jian Ren\textsuperscript{1}
\textsuperscript{1}Snap Inc.  \textsuperscript{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× ~ 24× 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., 18.04ms (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)\textsuperscript{1}.

1. Introduction

Remarkable progress seen in the domain of neural rendering [33] promises to democratize asset creation and rendering, where no mesh, texture, or material is required – only a neural network that learns 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.

Unfortunately, the impressive image quality and capabilities of NeRF [33] 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, 34, 37, 45], 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 [33], MobileNeRF requires 201.5MB of storage. Clearly, downloading and storing tens, hundreds, or even thousands of such scenes in MobileNeRF
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, 25, 28, 41]. Training NeLF is challenging and hence requires increased network capacity. For example, Wang et al. [41] 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 [41]. 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, 20], 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× ~ 24.3× 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 [33] shows the possibility of representing a scene with a simple multilayer perceptron (MLP) network. Going forward, many extensions follow up in improving rendering quality (e.g., MipNeRF [6], MipNeRF 360 [7], and Ref-NeRF [40]), rendering efficiency (e.g., NSVF [29], Nex [43], AutoInt [27], FastNeRF [13], Baking [16], Plenocnet [45], KiloNeRF [37], DeRF [36], DoNeRF [35], R2L [41], and MobileNeRF [10]) and training efficiency (e.g., Plenoxels [12], and Instant-NGP [34]).

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., Plenocnet [45], 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 [35], or mesh, e.g., MobileNeRF [10], to maintain the visual quality. (3) The third group takes a “divide and conquer” strategy. DeRF [36] decomposes the scene spatially to Voronoi diagrams and learns each diagram with a small network. KiloNeRF [37] also employs a decomposition scheme. Differently, they decompose the scene into thousands of small regular grids. Each is addressed 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 [41]. They achieve rendering efficiency by representing the scene with NeLF (neural light field) instead of NeRF. NeLF avoids the dense sampling on camera ray, resulting in a much faster rendering speed than NeRF. On the downside, NeLF is typically much harder to learn than NeRF. As a remedy, these works (such as [41]) typically integrate a pre-trained NeRF model as a teacher to synthesize additional pseudo 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 [25] 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. [38] materialize 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 R2L [41] are introduced. RSEN divides the space into many voxel grids. Only in each grid, it is a NeLF, which needs alpha-composition to render the final color, making their method a mixture of NeLF and NeRF. R2L [41] 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 $200 \times 200$. 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 [33].

3. Methods
3.1. Prerequisites: NeRF and R2L.

NeRF. NeRF [33] represents the scene implicitly with an MLP network $F_{\phi}$, which maps the 5 coordinates (spatial location $(x, y, z)$ and view direction $(\theta, \phi)$) to a 1D volume density (opacity, denoted as $\sigma$ here) and 3D radiance (denoted as $c$) such that $F_{\phi}: \mathbb{R}^5 \rightarrow \mathbb{R}^4$. Each pixel of an image is associated with a camera ray. To predict the color $\hat{C}$ of a pixel, the NeRF method samples many points (denoted as $N$ below) along the camera ray and accumulates the radiance $c$ of all these points via alpha compositing [23, 31, 33]:

$$\hat{C}(r) = \sum_{i=1}^{N} T_i \cdot (1 - \exp(-\sigma_i)) \cdot c_i,$$

$$(c_i, \sigma_i) = F_{\phi}(r(t_i), d),$$

$$T_i = \exp(-\sum_{j=1}^{i-1} \sigma_j \delta_j),$$

where $r$ means the camera ray; $r(t_i) = o + t_i d$ represents the location of a point on the ray with origin $o$ and direction $d$; $t_i$ 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 [33] to sample the $t_i$ in Eqn. 1. To enrich the input information, the position and direction coordinates are encoded by positional encoding [39], 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 [33], $L = 10$ 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 [41] 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 [33] 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 [39] 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 [41], 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 first stage on...
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 \[33\], 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 x 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 x 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 \( x \in \mathbb{R}^{B,6,H,W} \), 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 \( x \). We then apply positional encoding \( \gamma(\cdot) \) on \( x \) to map the ray origin and view directions into a higher dimension. Thus, we get the input of our neural network as \( \gamma(x) \).

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 \[33, 41\], 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 \[30\]. Under the same number of parameters, the model with CONV \( 1 \times 1 \) 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 dimensions. However, such Reshape or Permute operation might not be hardware-friendly on some mobile devices \[26\]. 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 \[41\]. Different from R2L, we adopt the CONV layer instead of the FC layer in each residual block. The CONV layer has the kernel size 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 (see experimental details in Tab. 4). The normalization and activation are chosen as batch normaliza-
tion [21] 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 existing network design of the neural light field requires large memory for rendering a high-resolution image, which surpasses the memory constraint on mobile devices. For example, rendering an image of $800 \times 800$ requires the prediction of 640,000 rays. Forwarding these rays at once using the network from R2L [41] 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 $800 \times 800$ image, we forward a 4D tensor $x$ with spatial size as $100 \times 100$ to the network and upsample the output from backbone three times (*i.e.*, upsample by $2 \times$ each time). The SR module includes two stacked residual blocks. The first block includes three CONV layers with one as a 2D Transpose CONV layer and two CONV $1 \times 1$ layers; the second block includes two CONV $1 \times 1$ 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° [33] and real-world forward-facing [32, 33]. 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 $800 \times 800$ for synthetic 360° and $1008 \times 756$ (4× down-scaled from the original resolution) for forward-facing.

**Implementation Details.** We follow the training scheme of R2L [41], *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 [24] optimizer with an initial learning rate $5 \times 10^{-4}$ 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 on the forward-facing dataset.

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 length 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 $100 \times 100$ for the synthetic 360° dataset and upsamples by $8 \times$ to render $800 \times 800$ RGB images. In contrast, the spatial size of $84 \times 63$ is used in the forward-facing dataset, and $1008 \times 756$ image ($12 \times$ upsampling) is rendered. The kernel size and padding are adjusted in the last transposed 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: PSNR, SSIM [42], and LPIPS [46], on the realistic synthetic 360° and real forward-facing datasets, as demonstrated in Tab. 1. Compared with NeRF [33], 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 [33]. 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 to SNeRG [16], our method obtains better PSNR and SSIM.

We also show the performance of the teacher model used
achieves high-quality rendering for zoom-in view, which is in comparison with MoibleNeRF \cite{peng2023mobilenetfs}. Leaf tips, are sharper. Additionally, we provide the zoom-in scene and less distorted shapes and textures. On the real-world MobileR2L outperforms NeRF clearly, delivering sharper results in Fig. 4. Our method runs faster on real forward-facing scenes. MobileNeRF cannot render Leaves and Orchids in Forward-facing.

Figure 4. Visual comparison between our method and NeRF \cite{mildenhall2020nerf} (trained via NeRF-Pytorch \cite{weinberger2020nerfpytorch}) 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 4](image)

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.

<table>
<thead>
<tr>
<th>Disk Storage</th>
<th>Synthetic 360°</th>
<th>Forward-facing</th>
</tr>
</thead>
<tbody>
<tr>
<td>MobileNeRF \cite{peng2023mobilenetfs}</td>
<td>125.8</td>
<td>204.5</td>
</tr>
<tr>
<td>SNeRG \cite{yuan2022sne}</td>
<td>86.8</td>
<td>537.3</td>
</tr>
<tr>
<td>Ours</td>
<td>8.3</td>
<td>8.3</td>
</tr>
</tbody>
</table>

Table 3. Analysis of Inference Speed. Latency (ms) is obtained on iPhone with iOS 16. Following MobileNeRF \cite{peng2023mobilenetfs}, 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.

<table>
<thead>
<tr>
<th>System</th>
<th>Synthetic 360°</th>
<th>Forward-facing</th>
</tr>
</thead>
<tbody>
<tr>
<td>MobileNeRF \cite{peng2023mobilenetfs}</td>
<td>17.54</td>
<td>27.15</td>
</tr>
<tr>
<td>SNeRG \cite{yuan2022sne}</td>
<td>26.21</td>
<td>18.04</td>
</tr>
<tr>
<td>Ours</td>
<td>22.65</td>
<td>16.48</td>
</tr>
</tbody>
</table>

Figure 5. Zoom-in comparisons. Top row: MobileNeRF \cite{peng2023mobilenetfs}. 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.

![Figure 5](image)

in 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 show following (Tab. 5), a better teacher model can lead to a student model with higher performance. Compared with MobileNeRF and SNeRG, our approach has the advantage that we can directly leverage the high-performing teacher models to help improve student training in 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 MobileNeRF \cite{peng2023mobilenetfs} 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 takes 24.3× less disk storage than MobileNeRF on the forward-facing, and 15.2× less storage on the synthetic 360° dataset.

**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 \cite{coremltools}. 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 cannot render two scenes, i.e., Leaves and Orchids, due to memory issues, as they require complex textures to model the geometry. In contrast, our approach is robust for different scenes.

**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 200K iterations.

Options for Backbone. We study the two available operators for designing the backbone. MLP and $1 \times 1$ 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 $1 \times 1$ 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 $100 \times 100$ for the two models. Since the super-resolution modules are omitted, we train the two networks for generating $100 \times 100$ images. As can be seen, the CONV2D model has a faster inference speed than MLP with similar performance. This is due to the fact that CONV operation is better-optimized than MLP on mobile devices. Additionally, due to the intrinsic design of our proposed MobileR2L, employing MLP layers requires two additional operators, i.e., Permute and Reshape, before feeding the internal features to super-resolution modules, while Permute and Reshape involve data movement that adds unnecessary overheads on some edge devices [26].

Analysis of Activation Function. R2L [41] and NeRF [33] use ReLU [4] activation as non-linearity function. 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 Batch-Norm [22] 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 $100 \times 100$ 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, $50 \times 50$ - NeRF Teacher, $100 \times 100$ - NeRF Teacher, and $200 \times 200$ - 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 $800 \times 800$ images and are trained with the NeRF [33] as a teacher model. Results are presented in Tab. 5. The model using a small input spatial size, i.e., $50 \times 50$, achieves $2 \times$ speedup than the model with a larger size, i.e., $100 \times 100$, as less computation is required. However, the performance is also degraded by 0.25 PSNR. Further increasing the input spatial size to $200 \times 200$ makes the model unable to achieve real-time inference. Thus, we do not report the rendering performance of the models with $200 \times 200$ input size.

Analysis of SR modules. We further show the necessity of using SR modules. We use the spatial size of $800 \times 800$ as the network input to render images with the same spatial size. We denote the setting as $800 \times 800$ - 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 essential for high-resolution synthesis without introducing prohibitive computation overhead.

Choice of Teacher Models. Since both R2L [41] 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 the teacher model for the training MobileR2L, given MipNeRF shows better performance than NeRF on the synthetic 360° dataset. We denote the approach as $100 \times 100$ - MipNeRF Teacher. Through the comparison with $100 \times 100$ - 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 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 [41] and positional encoding as in NeRF [33]. Specifically, R2L sample $K$ 3D points along the ray to construct the ray representation, leaving us with the option to choose the number of points for the $K$ parameter. We perform the ablation analysis similar to the previous experiments and report the results in Tab. 5. Here we analyze the necessity of using SR modules when better teacher models are provided. To conduct the experiments, we use MipNeRF [6] as the teacher model for the training MobileR2L, given MipNeRF shows better performance than NeRF on the synthetic 360° dataset. We denote the approach as $100 \times 100$ - MipNeRF Teacher. Through the comparison with $100 \times 100$ - 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 better teacher models are provided.
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 $100 \times 100, K_{16}, L_{10} - \text{MipNeRF Teacher}$ 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 $100 \times 100 - \text{MipNeRF Teacher}$. By comparing the performance of the two models, we notice that larger $K$ and $L$ lead to negligible PSNR improvement (0.07), yet higher inference latency and bigger model size (more parameters). Therefore, we chose $K = 8$ and $L = 6$ for the consideration of model size and latency – they are the more important 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 observe 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 shoes try-on application that runs on mobile devices. Users can directly try on the shoes 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 [33] to generate pseudo data, which is later used for learning MobileR2L to render images in $1008 \times 756$ resolution. 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-on usage proves the potential of leveraging neural rendering for building various real-time interactive applications such as Augment Reality.

![Figure 6. Virtual Try-On Application.](image)

This work presents MobileR2L, the first neural light network that renders images with similar quality as NeRF [33] while running in real-time on mobile 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., $800 \times 800$. Additionally, since we do not require other information besides the neural network, MobileR2L dramatically saves the representation storage in stark contrast to other mesh-based methods like MobleNeRF [10]. Furthermore, we prove that with our design, neural rendering can be used to build real-world applications, achieving real-time user interaction.

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 [41], and R2L uses 10K pseudo images generated by the teacher NeRF model to train the student model. The number of training images is much more than the images used to train the teacher NeRF (which only requires around 100 images), resulting in a longer training time than NeRF-based methods. Therefore, a future direction could be reducing the training costs for distillation-based works like R2L and this work. Second, MobileR2L fails to generate some high-frequency details in the images. We show examples in Fig. 7. Using a larger model may alleviate this problem given a larger model capacity. Nevertheless, the inference latency will also increase accordingly on mobile devices. Future efforts may focus on optimizing the network and training pipeline to boost the performance of the current model.

![Figure 7. Visual comparison on the real-world scene Room.](image)
References


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


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


[33] 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

[34] 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


