
NeRF in detail: Learning to sample for view synthesis

Relja Arandjelović¹ Andrew Zisserman^{1,2}

¹DeepMind ²VGG, Dept. of Engineering Science, University of Oxford

Abstract

Neural radiance fields (NeRF) methods have demonstrated impressive novel view synthesis performance. The core approach is to render individual rays by querying a neural network at points sampled along the ray to obtain the density and colour of the sampled points, and integrating this information using the rendering equation. Since dense sampling is computationally prohibitive, a common solution is to perform coarse-to-fine sampling.

In this work we address a clear limitation of the vanilla coarse-to-fine approach – that it is based on a heuristic and not trained end-to-end for the task at hand. We introduce a differentiable module that learns to propose samples and their importance for the fine network, and consider and compare multiple alternatives for its neural architecture. Training the proposal module from scratch can be unstable due to lack of supervision, so an effective pre-training strategy is also put forward. The approach, named ‘NeRF in detail’ (NeRF-ID), achieves superior view synthesis quality over NeRF and the state-of-the-art on the synthetic Blender benchmark and on par or better performance on the real LLFF-NeRF scenes. Furthermore, by leveraging the predicted sample importance, a 25% saving in computation can be achieved without significantly sacrificing the rendering quality.

1 Introduction

We address the classic problem of view synthesis, where given multiple images of a scene taken with known cameras, the task is to faithfully generate novel views as seen by cameras arbitrarily placed in the scene.

In particular, we build on top of NeRF [37], the recent impressive approach that represents a scene with a neural network; where, for each pixel in an image, points are sampled along the ray connecting the camera centre and the pixel, the network is queried to produce color and density estimates, and this information is integrated to produce the pixel color. The samples are obtained through a heuristic hierarchical coarse-to-fine approach that is not trained end-to-end. The method is reviewed in Section 1.2.

Our main contribution is the introduction of a ‘proposer’ module which makes the coarse-to-fine sampling procedure differentiable and amenable to learning via gradient descent. This enables us to train the entire network jointly for the end task. However, training it from scratch is challenging as lack of supervision causes instabilities and produces inferior results. Therefore, we also propose an effective two-stage training strategy where the network is first trained to mimic vanilla NeRF, and then made free to learn better sampling strategies. This approach, named ‘NeRF in detail’ (NeRF-ID), yields better scene representations, achieving state-of-the-art results on two challenging view synthesis benchmarks.

We consider a range of architectures for the ‘proposer’ module. Furthermore, the ‘proposer’ can be trained to produce importance estimates for the sample proposals, which in turn enables us to

adaptively filter out the least promising samples and reduce the amount of computation needed to render a scene, without compromising on the rendering quality.

1.1 Related work

View synthesis. Classical approaches directly interpolate the known images [8, 9, 11, 12, 34, 48] or the light field [18, 27], without requiring the knowledge of the underlying scene geometry, but they often need dense sampling of the scene in order to work well. More recently, learning based approaches were used to blend the images [13, 20, 57], or to construct a multiplane image representation of the scene which can be used to synthesize novel views [29, 36, 66]. Another set of methods generates the views by querying an explicit 3D representation of the scene, such as a mesh [10, 24, 46, 50], a point cloud [1, 5, 19, 68], or a voxel grid [26, 41, 49], but forming this 3D representation can be fragile. We follow an increasingly popular approach where scene geometry and appearance are represented implicitly using a neural network [15, 35, 37, 38, 39, 52] and views are rendered by ray tracing; NeRF [37] is reviewed in Section 1.2.

Proposals. A popular approach in object detection is to first generate a number of proposals, aiming to cover the correct object with high recall, and then classification and refinement is applied to improve the precision [16, 17, 45, 55]. Two-stage temporal action detection approaches [51, 61, 65] follow a similar strategy, where the proposals are temporal segments, and are therefore 1D as in our work where the proposals correspond to the distance from the camera along the ray. However, in both domains full supervision on the locations is available to train the proposers, whereas this level of supervision is not available to us. Differentiable resampling for particle filters [22, 67] can also be seen as a coarse-to-fine approach akin to our proposer, but there the sampling is based purely on coarse samples’ locations and weights, while our proposer also makes use of features computed by the coarse network.

1.2 Overview of Neural radiance field (NeRF)

Here we give a brief overview of NeRF with a particular focus on the issues relevant to this work; full details are available in the original paper [37].

NeRF models the radiance field with a neural network – when queried with 3D world coordinates and 2D viewing direction, the network outputs the density of the space, σ , and the view-dependent color, \mathbf{c} . Rendering an image seen from any given camera can be done by shooting a ray through each pixel and computing its color. The ray color is computed independently for each ray, by sampling points along the ray, querying the network for the points’ densities and colors, and integrating the information from all the samples via the rendering equation:

$$\hat{\mathbf{C}} = \sum_{i=1}^N w_i \mathbf{c}_i, \quad w_i = T_i (1 - \exp(-\sigma_i(t_{i+1} - t_i))), \quad T_i = \exp\left(-\sum_{j=1}^{i-1} \sigma_j(t_{i+1} - t_i)\right) \quad (1)$$

where σ_i , \mathbf{c}_i , t_i is the i -th sample’s density, color and location along the ray, and w_i is its contribution to the ray color estimate $\hat{\mathbf{C}}$.

The network is trained by sampling rays from all pixels of the training set images, and minimizing the L2 loss between the predicted and ground truth ray color. All the operations mentioned so far are differentiable and the network is trained with gradient descent.

With an appropriate choice of the network architecture [64], the density does not depend on the viewing direction, and the color is somewhat restricted in how much it can vary across viewpoints (a trade-off between not overfitting to training views and modelling non-Lambertian effects). This enables NeRF to produce consistent renderings across different viewing directions.

One underlying assumption is that accurate ray rendering can be achieved with a finite number of samples along the ray. With a larger number of samples, the rendering should get better but becomes computationally prohibitive. This is why NeRF follows a coarse-to-fine approach, as illustrated in Figure 1(a). Namely, two networks – coarse and fine – are used, where (i) the coarse network is queried on a few, N_c , equally spaced samples along the ray, (ii) its outputs are used to obtain more, N_f , samples, and (iii) the fine network is queried on the union of the samples ($N_c + N_f$) and produces the final rendering. The sampling of N_f points is done from a piece-wise constant probability density

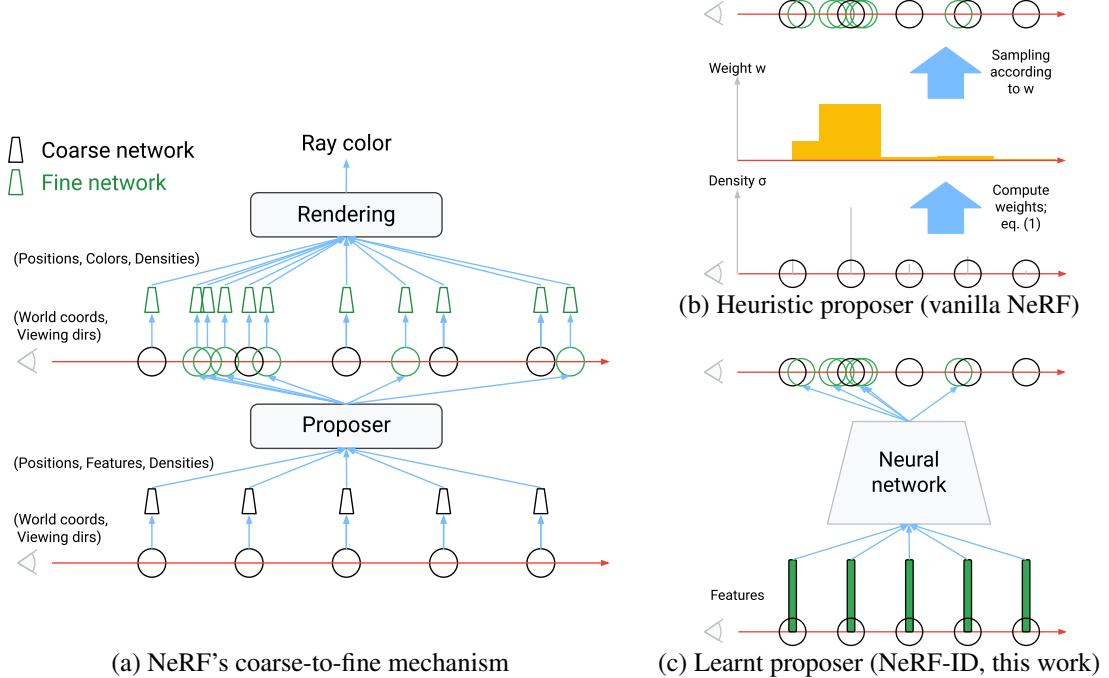


Figure 1: **Overview of NeRF and our method.** NeRF’s coarse-to-fine approach (a) relies on a heuristic ‘proposer’ (b) which acts on the output of the coarse network and produces samples to pass to the fine network. We substitute this mechanism with a learnable proposer (c).

function, where the pdf is computed via a handcrafted procedure involving the densities produced by the coarse network (Figure 1(b)). In brief, the rendering equation can be used on the coarse network’s outputs to calculate the contribution (*i.e.* the weight, w_i in eq. (1)) of every coarse sample to the final color, and these weights are normalized to define the piece-wise constant pdf. The intuition behind this heuristic is that the region of space that contains the object closest to the camera (*i.e.* contributing most to the rendered color) should be sampled the most in order to reveal details captured by the fine network. The whole coarse-to-fine system is not trainable end-to-end but the coarse network is trained independently with the same reconstruction loss.

Recent developments. Due to its impressive performance, NeRF has attracted a lot of attention in the field and has been extended in a variety of ways. Many works adapted NeRF to situations it does not handle out of the box, such as real world scenes with varying lighting and transient objects [33], deformable scenes [40, 42], video [28, 60], unknown cameras [59], *etc.* NeRF requires retraining for every scene and a few approaches alleviate this requirement by sharing parameters across scenes [25, 54, 63]. Combining NeRF with GANs yields promising 3-D aware image generation [7, 47]. However, not many works have focused on improving the ‘core’ algorithm behind NeRF, with the exception of NeRF++ [64] which proposes a better parametrization for large-scale unbounded real-world scenes, and mip-NeRF [3] which addresses aliasing issues. In this work we improve a core component of NeRF that is the hierarchical coarse-to-fine sampling of points along rays, by replacing the heuristic non-trainable sample proposal module with a fully end-to-end trainable one. This is complementary to many of the above-mentioned approaches, *e.g.* [3, 14, 25, 33, 40, 54, 57, 60], since they also use coarse-to-fine sampling, and our module can easily be swapped in.

2 Learning to sample

In this section we describe an improved coarse-to-fine approach in order to facilitate better rendering of details. As explained in the previous section, the coarse-to-fine procedure used in the original NeRF work is a handcrafted heuristic – it matches intuition but it is likely suboptimal. The coarse network is trained independently for reconstruction, and it is not even ‘aware’ of the fine network

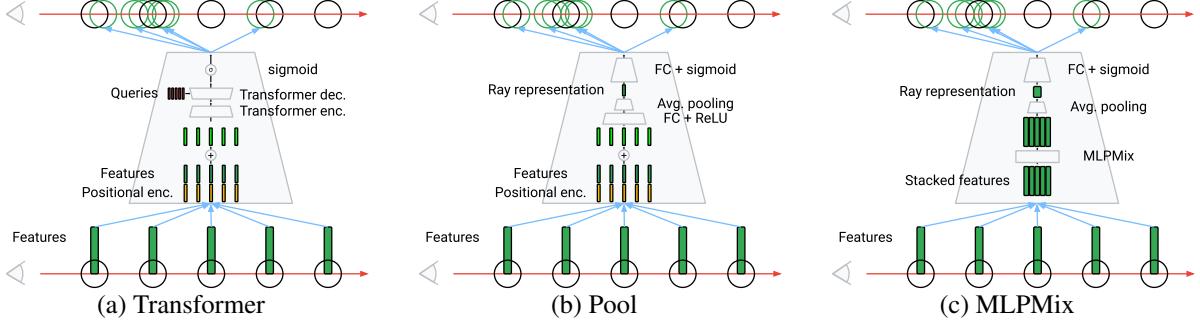


Figure 2: **Trainable proposer architectures.** The input features come from the coarse network, as shown in Figures 1(a) and (c). Full details are in Appendix A and Figure 6.

(no error signal from the fine network reaches the coarse network). Therefore, the coarse network is unable to adjust its outputs to benefit the real end-goal of interest – the reconstruction quality as produced by the fine network.

We replace this non-trainable sample proposal procedure with a differentiable module (Figure 1(c)) and train the whole system end-to-end. As before, the ray is sampled at regular intervals and the outputs of the coarse network are used to decide where to query the fine network. Instead of just using the densities and a hand-engineered proposal mechanism, as in the original NeRF, we pass the features produced by the coarse network at each coarse sample (the features are the values of the last activations before the projection that produced the density) through a neural network that directly produces the locations of the fine samples. So, the input to the “proposer” is a sequence of N_c coarse-network-produced feature vectors and their positions along the ray (scalars normalized to $[0, 1]$ such that 0 corresponds to the near and 1 to the far plane), and the output is a new set of N_f positions along the ray (again in $[0, 1]$ obtained by passing the scalar output through a sigmoid) where the fine network will be queried.

Architectures. We consider a few options for proposer architecture (Figure 2). Here we outline the main features of the architectures and provide full details in Appendix A. The core design choice is to make the proposer network small relative to the coarse and fine networks, in order to have negligible additional parameters and computational overhead.

1. *Transformer*: A transformer [56] encoder processes the input features, and a transformer decoder produces the final set of proposals via cross attention with N_f learnt queries.
2. *Pool*: Each point feature is summed up with the positional encoding of its location along the ray, and passed through a fully connected layer (FC) followed by a ReLU non-linearity. These processed features are averaged together to form a single “ray representation” vector. This vector is then passed through an FC to produce the final set of proposals, where the output dimension of the FC is N_f . The motivation behind the first stage is that simply average pooling (feature+positional encoding) vectors would lose the feature-position association due to all operations being linear, so a non-linearity is required before the pooling. The validity of this choice is confirmed by the ablations in Section 3.2.
3. *MLPMix*: The “ray representation” is obtained by applying a single MLPMix [53] block to the stacked point features sorted by their position along the ray, followed by average pooling. The final set of proposals are again decoded via an FC.

2.1 Effective training

As will be shown in Section 3.2, simply training the entire network (coarse + proposer + fine networks) jointly from scratch sometimes works, but sometimes fails to reach good performance. We hypothesize that this is because the supervision is too weak (just the final color of the rendered ray). There is a chicken and egg problem – training the proposer requires a good fine network that will provide a useful training signal, but the fine network can only become good if it is sampled at informative locations. It can be hard to stop this vicious cycle when training from scratch.

We note that the “proposer” idea has some resemblance to two-stage object detectors such as Faster R-CNN [45], where the first network produces object proposals and the second network examines the proposals in more detail. The analogy with object detection and in particular DETR [6] inspired our Transformer proposer. However, in object detection the proposal network is typically trained with full supervision using object bounding box annotations, while such supervision is not available in our case.

To alleviate the training difficulties, we divide training into two stages. In the *first stage*, the coarse and fine networks are trained as in the vanilla NeRF, *i.e.* using the heuristic proposer. The trainable proposer is trained to predict the proposals coming from the heuristic proposer – the two sets of proposals are matched in a greedy fashion (for speed reasons) and the sum of the L2 distances between the matching proposals is minimized. At the end of the first stage, the coarse and fine networks are identical to the vanilla-NeRF ones, while the proposer hopefully mimics the heuristic proposer. In the *second stage*, we swap in the learnt proposer (*i.e.* the samples seen by the fine network come from the learnt proposer instead of the heuristic one) and simply train the whole system end-to-end to minimize the reconstruction loss of the fine network. To prevent the system from diverging, we still add the reconstruction loss of the coarse network as well, but the coarse network is now optimized to both minimize its reconstruction loss as well as produce good features that enable the proposer to provide informative samples to the fine network and yield good fine network reconstructions.

The two training stages are equal in duration (number of SGD steps) and for fair comparison with vanilla NeRF the total training time is kept constant.

2.2 Learning sample importance

It is also possible to predict the “importance” of each proposal sample. This can be useful to reduce the computational burden of ray rendering as the fine network can be queried only with samples deemed to be important. The proposer can predict importance scores for all the samples by simply regressing another set of $N_c + N_f$ scalars using the same mechanism as with the proposal generation, *e.g.* for the *Pool* and *MLPMix* architectures another FC head operating on the “ray representation” vector produces the $N_c + N_f$ scalars.

To make another parallel with two-stage object detectors – typically the region proposals are assigned confidence scores, facilitated by the fully supervised training. Here, unlike in object detection, such fine-grained supervision is not available, but the output of the fine network can be used as the training signal instead. Namely, the weight w_i (eq. (1)) that each fine sample contributes to the final rendering is exactly what the importance prediction aims to forecast – iff it is high then the sample is important. So the weights are thresholded at the desired accuracy level (0.03 is used as a reasonable value, we did not investigate others) to define the positive and negative samples, and the importance predictor is trained to predict them using the balanced logistic regression loss.

3 Experiments and discussion

In this section we evaluate the performance of our ‘NeRF in detail’ (NeRF-ID) method. First, the experimental protocol and the benchmarks are described, followed by the comparison of various proposer architectures. Next, our method with trainable proposer is contrasted against NeRF and the state-of-the-art in terms of view synthesis quality and speed. Finally, we discuss relation to other methods, limitations and potential avenues for future research.

3.1 Experimental protocol, datasets and evaluation

Datasets. The main benchmarks from the original NeRF [37] are used, namely *Blender* [37] and *LLFF-NeRF* [36, 37]. *Blender* is a realistically rendered 360° synthetic dataset comprising of 8 scenes with 100 training, 100 validation and 200 testing views of resolution 800 × 800. *LLFF-NeRF* contains forward-facing 1008 × 756 images of 8 real scenes, ranging between 20 and 62 images per scene, 1/8 of which are used for testing and the rest for training; since there is no validation set we use the training set in its place.

Performance metrics. Following standard procedure [37], novel view synthesis quality is measured using peak signal-to-noise ratio (PSNR), and structural similarity (SSIM) [58], where higher scores

Table 1: **Comparison of proposer architectures and vanilla NeRF (PSNR).** Most experiments were repeated 5 times and we report the median PSNR, full results are in Appendix C. NeRF denotes the results reported in the original paper [37], NeRF[†] is our reimplementation. Highlighted are the **best** and **second best** results.

	Method	Avg.	Chair	Drums	Ficus	Hotdog	Lego	Materials	Mic	Ship
<i>Blender</i>	NeRF [37]	31.01	33.00	25.01	30.13	36.18	32.54	29.62	32.91	28.65
	NeRF [†]	31.83	34.35	24.98	30.40	37.04	33.56	30.19	34.60	29.52
	Transformer	31.82	<u>34.50</u>	24.91	30.93	36.78	33.84	<u>30.27</u>	<u>34.57</u>	28.75
	Pool	<u>32.23</u>	34.28	25.18	32.37	<u>37.11</u>	34.74	29.92	<u>34.31</u>	29.94
	MLPMix	32.34	34.54	<u>25.15</u>	<u>32.24</u>	37.26	<u>34.73</u>	30.37	34.71	<u>29.75</u>
<i>LLFF-NeRF</i>	Method	Avg.	Fern	Flower	Fortress	Horns	Leaves	Orchids	Room	T-Rex
	NeRF [37]	26.50	25.17	27.40	31.16	27.45	20.92	20.36	32.70	26.80
	NeRF [†]	26.66	24.93	27.88	31.48	27.74	21.12	20.35	32.77	27.04
	Transformer	26.68	<u>25.13</u>	<u>27.87</u>	31.42	27.79	21.10	20.46	32.59	27.05
	Pool	26.79	25.07	27.81	<u>31.50</u>	<u>27.82</u>	<u>21.11</u>	<u>20.40</u>	33.14	27.49
	MLPMix	<u>26.76</u>	25.01	27.85	31.51	27.88	21.09	20.38	<u>32.93</u>	<u>27.45</u>

Table 2: **Comparison with the state-of-the-art.** NeRF denotes the results reported in the original paper [37], NeRF[†] is our reimplementation. Highlighted are the **best** and **second best** results.

	SRN [52]	NV [32]	LLFF [36]	NeRF [37]	NeRF [†]	NSVF [31]	IBRNet [59]	GRF [54]	NeRF-ID
<i>Blender</i> PSNR	22.25	26.05	24.88	31.01	31.83	31.75	28.14	<u>32.06</u>	32.34
	Blender SSIM	0.846	0.893	0.911	0.947	0.954	0.954	0.942	0.960
<i>LLFF-NeRF</i> PSNR	22.84	/	24.13	26.50	26.66	/	<u>26.73</u>	26.64	26.76
	<i>LLFF-NeRF</i> SSIM	0.668	/	0.798	0.811	0.820	/	0.851	0.837
									0.822

signify better performance. We observe very similar trends for the two metrics, so here we mainly report PSNR and the SSIM for all experiments can be found in Appendix C. For most important experiments we train five times with different random seeds.

Training procedure. We use the same optimization procedure and hyper-parameters for all experiments, datasets and scenes. As with the original NeRF, $N_c = 64$ points per ray are processed by the coarse network, while the fine network evaluates 192 samples (reusing the same $N_c = 64$ locations as the coarse network and additional $N_f = 128$ samples obtained from the sample proposal procedure). Batches are compiled by sampling rays randomly across the entire training set, and optimization is ran for 10 billion rays with a total batch size of 66k (*i.e.* roughly 150k SGD steps). The Adam optimizer [23] (with default hyper-parameters) is used with the learning rate first following a linear warmup from 0 to 5×10^{-4} for 1k steps and then switching to the cosine decay schedule. The checkpoint with the best PSNR on a random subset of the validation set is used (*i.e.* early stopping), although in the vast majority of scenes this corresponds to the last checkpoint. The entire system is implemented in JAX [4] and the DeepMind JAX Ecosystem [2] and trained on 16 Cloud TPUs for about 10 hours.

Note that our training procedure is slightly different from the original NeRF, *e.g.* they train with smaller batch sizes and for fewer iterations (around 1.2 billion rays). We also make sure that each ray passes through the center of its pixel instead of the corner. Our reimplementation, which we name NeRF[†], outperforms the original and thus serves as a good baseline for our NeRF-ID.

3.2 Results and discussion

Proposer architectures. The performance of the three proposer architectures – Transformer, Pool and MLPMix – is compared in Table 1. While all three often work well and there are scenes on which each of them performs best, the Transformer is inferior due to significantly worse performance on a few scenes (*e.g.* *Blender*: *Ficus*, *Blender*: *Lego*, *LLFF-NeRF*: *T-Rex*) and we observe its training curves are somewhat unstable. Pool and MLPMix are somewhat on par, but Pool is significantly worse

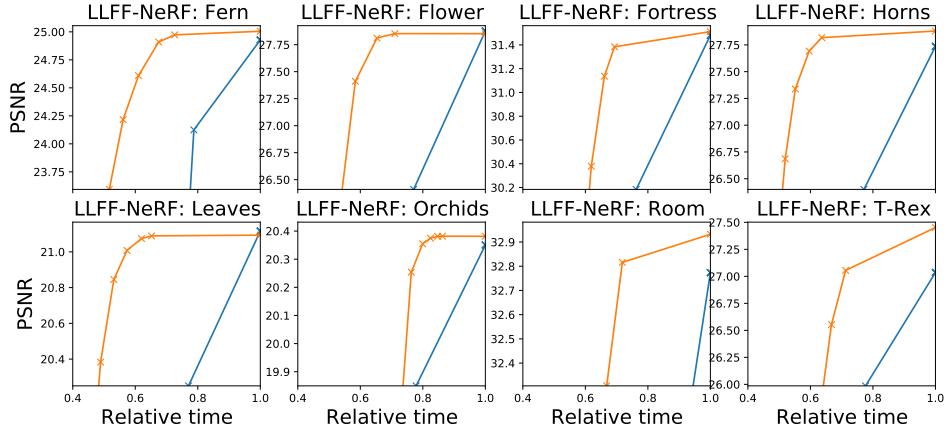


Figure 3: **Speedup via importance prediction, NeRF^\dagger vs. our NeRF-ID .** Samples deemed to be important by the proposer are kept, different operating points are obtained by varying the importance threshold. ‘Relative time’ is the time spent on rendering the scene relative to using all samples.

on two scenes (*Blender: Materials*, *Blender: Mic*) and overall has a higher variance. Therefore, in the rest of the paper, our method NeRF-ID uses the MLPMix proposer. Appendix C contains additional ablations on the proposer architectures, including various ways to integrate sample positions into the Pool architecture, and a “blind” proposer (proposer which ignores input features and therefore always outputs the same proposals; its bad performance verifies our good proposers are not exploiting some aspect of the datasets via a simple degenerate strategy).

Two-stage training. Training from scratch often succeeds but in the majority of those cases it underperforms our two-stage training procedure. It also often fails badly (e.g. -5 PSNR on *Blender: Ship*), validating the need for the two-stage training. Full results are available in Appendix C.

Comparison with the state-of-the-art (Tables 1 and 2). The MLPMix-based proposer consistently outperforms NeRF and NeRF^\dagger on all *Blender* scenes, and is on par or better on the *LLFF-NeRF* scenes, verifying the effectiveness of our approach; the improvements are statistically significant, as demonstrated in Appendix C. Particularly impressive is its performance on *Ficus* and *Lego* in *Blender*, achieving $+1.84$ and $+1.17$ PSNR (-53% and -31% MSE), respectively, and $+0.41$ on *LLFF-NeRF: T-Rex* (-10% MSE). Furthermore, NeRF-ID outperforms state-of-the-art in PSNR (the metric optimized by all approaches) and is on par in SSIM.

Qualitative results. Figure 4 contrasts synthesized views of our NeRF-ID versus NeRF^\dagger . While NeRF^\dagger shows impressive renderings, NeRF-ID does significantly better in capturing some finer details, such as edges, thin structures and small objects. These differences do not feature so prominently in the quantitative evaluations since the proportion of the image that contains such details is typically small, but the improvements are clearly visible.

Speedup. Figure 3 shows speedups obtained by only querying the fine network on samples deemed to be important by the proposer (Section 2.2). For most scenes in *LLFF-NeRF* it is possible to render views 25% faster without any loss in view synthesis quality. NeRF-ID consistently dominates NeRF^\dagger , producing better images using fewer computations. For example, on the *Fern* scene, NeRF-ID that uses 27% fewer computations achieves better results than the NeRF^\dagger , while if NeRF^\dagger uses 21% fewer computations it suffers a loss in PSNR of 0.8 (+20% MSE). Speedups are even more dramatic on the *Blender* dataset (Figure 13) as many rays belong to the uniformly white background and our method automatically learns to allocate them very few samples.

Note that speeding up NeRF is not the main focus of this work, and better approaches that are orders of magnitudes faster exist, such as [21, 30, 43, 44, 62]. However, (i) our approach essentially comes for free without heavily specialized machinery, (ii) it is compatible with some of the approaches such as FastNeRF [14] and BakingNeRF [21] since they still use coarse-to-fine sampling, and (iii) some of

them are not capable of handling real world unbounded scenes [44, 62] like the ones in *LLFF-NeRF* or require significantly more storage to represent a scene [14, 21, 44, 62].

What is learnt? Figure 5 investigates what is being learnt. The heuristic proposer of vanilla NeRF [37] often provides good proposals, but it oversamples some regions while sometimes undersampling the important surfaces. Our learnt proposer rarely misses the important surfaces and yields a much more diverse sampling. This in turn yields a more accurate rendering, but is also beneficial for effective training of the fine network. Importance prediction (Section 2.2) does a good job at picking the most important proposals (*i.e.* proposals that are close to the surface closest to the ray origin). Furthermore, the proposals and their importance increase in density when the (horizontal) ray is tangent or near to tangent to the surface.

Discussion, limitations and future work. Mip-NeRF [3] actually achieves better PSNR performance on the *Blender* dataset but we do not include it in the comparison because (i) it is a concurrent approach, (ii) it is not directly applicable on real unbounded scenes like the ones comprising the *LLFF-NeRF* dataset (it achieves no improvements over NeRF), and (iii) its approach is complementary to ours. In fact, NeRF-ID can be applied on many works that extend NeRF as long as they use or are amenable to using the coarse-to-fine strategy when rendering rays. This includes Mip-NeRF [3], both top competitors from Table 2 (GRF [54], IBRNet [57]), NeRFs for unconstrained [33], deformable [40] and temporal scenes [60], as well as some approaches for speeding up NeRF [14, 21].

NeRF-ID as well as other NeRF-based approaches still produce significant errors especially on the real-world scenes, where on *LLFF-NeRF* the performances have somewhat plateaued (NeRF-ID, mip-NeRF [3], GRF [54], IBRNet [57]). It is likely that the bottleneck is somewhere other than the proposer mechanism, even though we believe that our proposer module will be useful for future NeRF-based approaches. For example, deeper networks might be required to model these complex scenes, but this in turns exacerbates NeRF’s problem with being data hungry and not utilizing the common structure of the world. Few recent works have started tackling the latter issue [25, 54, 63] and we believe the proposer has a place in these approaches as well – learning a universal proposer shared across scenes can still facilitate learning and focus the effort on relevant parts of the scene. This work has kept the proposer very shallow with few parameters in order to make NeRF-ID directly comparable to NeRF, but it is possible that a deeper proposer is required to reach its full potential.

4 Conclusions

We have presented a ‘proposer’ module that learns the hierarchical coarse-to-fine sampling, thus enabling NeRF to be trained end-to-end for the view synthesis task. Multiple architectures for the module are explored and the best consistently outperforms the vanilla NeRF on both challenging benchmarks. NeRF-ID fares well against the state-of-the-art with negligible overhead in the number of parameters and amount of computation. It also enables faster execution since 25% of the samples can be pre-filtered without sacrificing the view synthesis quality.

Acknowledgments. We would like to thank Bojan Vujatović and Yusuf Aytar for helpful discussions.

References

- [1] K.-A. Aliev, A. Sevastopolsky, M. Kolos, D. Ulyanov, and V. Lempitsky. Neural point-based graphics. In *Proc. ECCV*, 2020.
- [2] I. Babuschkin, K. Baumli, A. Bell, S. Bhupatiraju, J. Bruce, P. Buchlovsky, D. Budden, T. Cai, A. Clark, I. Danihelka, C. Fantacci, J. Godwin, C. Jones, T. Hennigan, M. Hessel, S. Kaputowski, T. Keck, I. Kemaev, M. King, L. Martens, V. Mikulik, T. Norman, J. Quan, G. Papamakarios, R. Ring, F. Ruiz, A. Sanchez, R. Schneider, E. Sezener, S. Spencer, S. Srinivasan, W. Stokowiec, and F. Viola. The DeepMind JAX Ecosystem, 2020.
- [3] J. T. Barron, B. Mildenhall, M. Tancik, P. Hedman, R. Martin-Brualla, and P. P. Srinivasan. Mip-NeRF: A multiscale representation for anti-aliasing neural radiance fields. *CoRR*, abs/2103.13415, 2021.
- [4] J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, G. Necula, A. Paszke, J. VanderPlas, S. Wanderman-Milne, and Q. Zhang. JAX: Composable transformations of Python+NumPy programs, 2018.

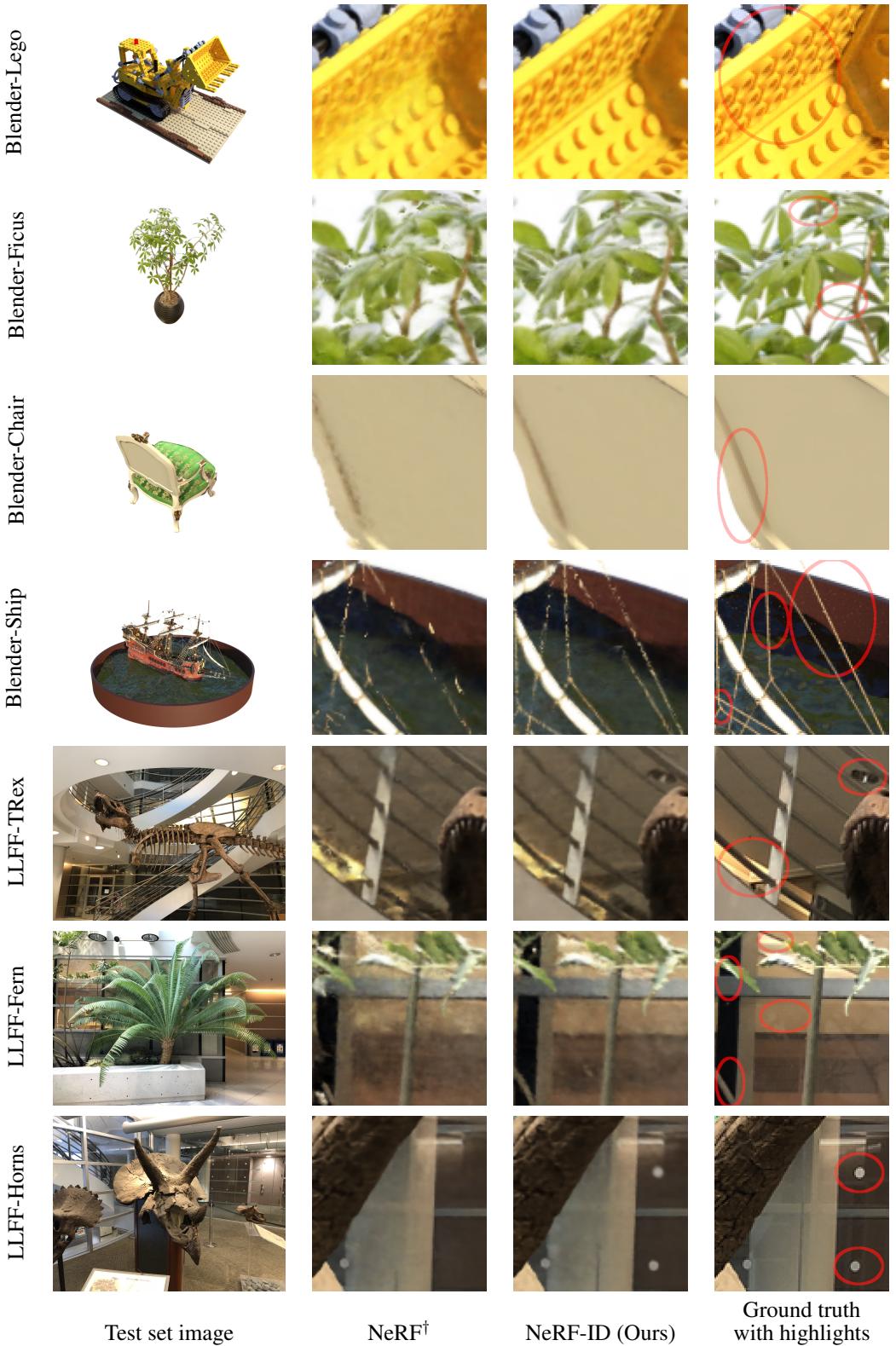


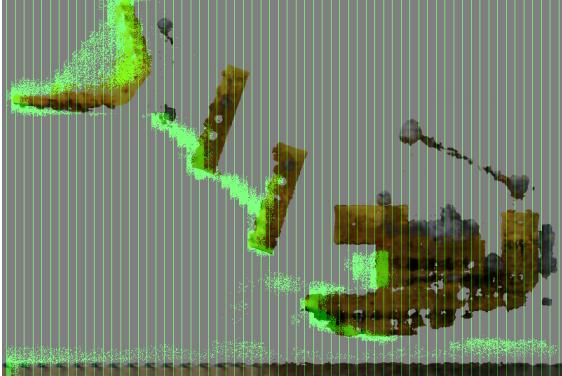
Figure 4: **Qualitative results.** Comparison of our NeRF-ID versus NeRF[†]. Overall both methods produce good renderings, but the difference is especially apparent in fine details that NeRF[†] often misses while NeRF-ID reproduces better, such as thin branches, ropes, markings, edges etc.



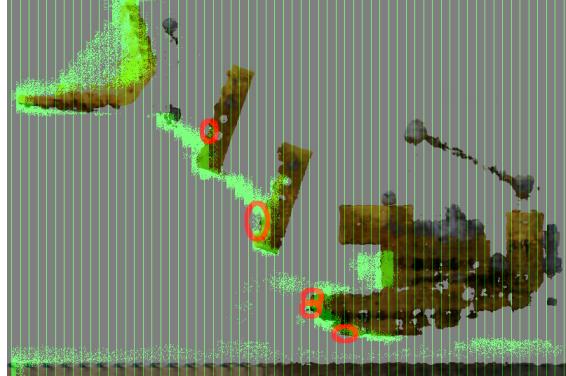
(a) A cross section from the *Blender: Lego* scene.



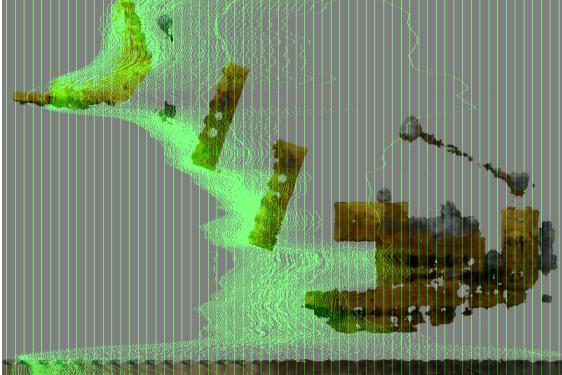
(b) Test image roughly aligned with (a)



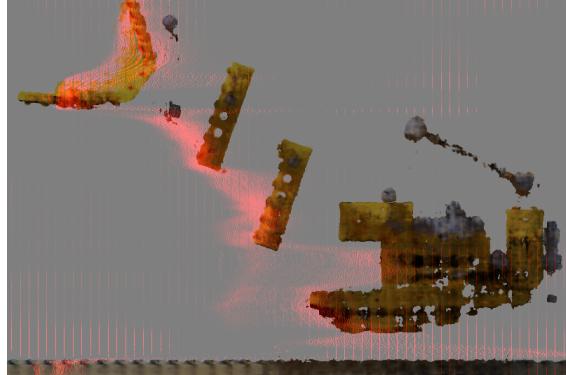
(c) Heuristic proposals (+ coarse samples) [37]



(d) Highlighted mistakes in (c)



(e) NeRF-ID proposals (+ coarse samples)



(f) Importance prediction for NeRF-ID proposals in (e)

Figure 5: What is learnt? (a) A cross section of the *Blender: Lego* scene, produced by querying the fine network densely on the plane and plotting the color masked by the occupancy. (b) An image from the test set whose camera plane is roughly parallel to the cross section in (a). (c) and (e) For each row of the cross section image, a ray is shot from left to right and proposals (along with coarse samples) are overlaid in green over the cross section image; the samples are displayed at pixel resolution even though in reality they are real-valued. (c) shows the heuristic proposals of our reimplementation NeRF[†] of vanilla NeRF [37], while (e) shows our NeRF-ID learnt proposals. (f) All proposals from (e) are colored in red such that the intensity is proportional to the estimated importance (Section 2.2). The heuristic proposals are over-concentrated in a few areas and sometimes undersample the surface closest to the ray origin (d). NeRF-ID proposals rarely miss the closest surface (e), and are much more diverse which provides a more accurate rendering but also a better sampling that facilitates training of the fine network. Importance prediction does a good job at highlighting the the most promising proposals (f).

- [5] G. Bui, T. Le, B. Morago, and Y. Duan. Point-based rendering enhancement via deep learning. *Visual Computer*, 2018.
- [6] N. Carion, G. S. Francisco Massa and, N. Usunier, A. Kirillov, and S. Zagoruyko. End-to-end object detection with transformers. In *Proc. ECCV*, 2020.
- [7] E. Chan, M. Monteiro, P. Kellnhofer, J. Wu, and G. Wetzstein. pi-GAN: Periodic implicit generative adversarial networks for 3D-aware image synthesis. In *Proc. CVPR*, 2021.
- [8] G. Chaurasia, S. Duchêne, O. Sorkine-Hornung, and G. Drettakis. Depth synthesis and local warps for plausible image-based navigation. In *Proc. ACM SIGGRAPH*, 2013.
- [9] S. Chen and L. Williams. View interpolation for image synthesis. In *Proc. ACM SIGGRAPH*, 1993.
- [10] P. Debevec, Y. Yu, and G. Borshukov. Efficient view-dependent image-based rendering with projective texture-mapping. In *Eurographics Workshop on Rendering Techniques*, 1996.
- [11] P. E. Debevec, C. J. Taylor, and J. Malik. Modeling and rendering architecture from photographs: A hybrid geometry- and image- based approach. In *Proc. ACM SIGGRAPH*, pages 11–20, 1996.
- [12] A. W. Fitzgibbon, Y. Wexler, and A. Zisserman. Image-based rendering using image-based priors. In *Proc. ICCV*, volume 2, pages 1176–1183, 2003.
- [13] J. Flynn, I. Neulander, J. Philbin, and N. Snavely. DeepStereo: Learning to predict new views from the world’s imagery. In *Proc. CVPR*, 2016.
- [14] S. J. Garbin, M. Kowalski, M. Johnson, J. Shotton, and J. Valentin. FastNeRF: High-fidelity neural rendering at 200FPS. *CoRR*, abs/2103.10380, 2021.
- [15] K. Genova, F. Cole, A. Sud, A. Sarna, and T. Funkhouser. Local deep implicit functions for 3D shape. In *Proc. CVPR*, 2020.
- [16] R. B. Girshick. Fast R-CNN. In *Proc. ICCV*, 2015.
- [17] R. B. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proc. CVPR*, 2014.
- [18] S. J. Gortler, R. Grzeszczuk, R. Szeliski, and M. F. Cohen. The lumigraph. In *Proc. ACM SIGGRAPH*, pages 43–54, 1996.
- [19] J. P. Grossman and W. Dally. Point sample rendering. *Rendering Techniques*, 1998.
- [20] P. Hedman, J. Philip, T. Price, J. Frahm, G. Drettakis, and G. Brostow. Deep blending for free-viewpoint image-based rendering. *ACM Trans. Graph.*, 2018.
- [21] P. Hedman, P. P. Srinivasan, B. Mildenhall, J. T. Barron, and P. Debevec. Baking neural radiance fields for real-time view synthesis. *CoRR*, abs/2103.14645, 2021.
- [22] R. Jonschkowski, D. Rastogi, and O. Brock. Differentiable particle filters: End-to-end learning with algorithmic priors. In *Proc. RSS*, 2018.
- [23] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *Proc. ICLR*, 2015.
- [24] R. Koch. 3D surface reconstruction from stereoscopic image sequences. In *Proc. ICCV*, pages 109–114, 1995.
- [25] A. R. Kosiorek, H. Strathmann, D. Zoran, P. Moreno, R. Schneider, S. Mokrá, and D. J. Rezende. NeRF-VAE: A geometry aware 3D scene generative model. In *Proc. ICML*, 2021.
- [26] K. N. Kutulakos and S. M. Seitz. A theory of shape by space carving. Technical Report CSTR 692, University of Rochester, 1998.
- [27] M. Levoy and P. Hanrahan. Light field rendering. In *Proc. ACM SIGGRAPH*, 1996.
- [28] Z. Li, S. Niklaus, N. Snavely, and O. Wang. Neural scene flow fields for space-time view synthesis of dynamic scenes. In *Proc. CVPR*, 2021.
- [29] Z. Li, W. Xian, A. Davis, and N. Snavely. Crowdsampling the plenoptic function. In *Proc. ECCV*, 2020.
- [30] D. B. Lindell, J. N. Martel, and G. Wetzstein. AutoInt: Automatic integration for fast neural volume rendering. In *Proc. CVPR*, 2021.
- [31] L. Liu, J. Gu, K. Z. Lin, T.-S. Chua, and C. Theobalt. Neural sparse voxel fields. In *NeurIPS*, 2020.
- [32] S. Lombardi, T. Simon, J. Saragih, G. Schwartz, A. Lehrmann, and Y. Sheikh. Neural volumes: Learning dynamic renderable volumes from images. *ACM Trans. Graph.*, 38(4), 2019.
- [33] R. Martin-Brualla, N. Radwan, M. S. M. Sajjadi, J. T. Barron, A. Dosovitskiy, and D. Duckworth. NeRF in the wild: Neural radiance fields for unconstrained photo collections. In *Proc. CVPR*, 2021.
- [34] L. McMillan and G. Bishop. Plenoptic modeling: An image-based rendering system. In *Proc. ACM SIGGRAPH*, 1995.
- [35] L. Mescheder, M. Oechsle, M. Niemeyer, S. Nowozin, and A. Geiger. Occupancy networks: Learning 3D reconstruction in function space. In *Proc. CVPR*, 2019.
- [36] B. Mildenhall, P. P. Srinivasan, R. Ortiz-Cayon, N. K. Kalantari, R. Ramamoorthi, R. Ng, and A. Kar. Local light field fusion: Practical view synthesis with prescriptive sampling guidelines. *ACM Transactions on Graphics (TOG)*, 2019.
- [37] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng. NeRF: Representing scenes as neural radiance fields for view synthesis. In *Proc. ECCV*, 2020.
- [38] M. Niemeyer, L. Mescheder, M. Oechsle, and A. Geiger. Differentiable volumetric rendering: Learning implicit 3D representations without 3D supervision. In *Proc. CVPR*, 2020.
- [39] J. J. Park, P. Florence, J. Straub, R. Newcombe, and S. Lovegrove. DeepSDF: Learning continuous signed distance functions for shape representation. In *Proc. CVPR*, 2019.
- [40] K. Park, U. Sinha, J. T. Barron, S. Bouaziz, D. B. Goldman, S. M. Seitz, and R. Martin-Brualla. Deformable neural radiance fields. *CoRR*, abs/2011.12948, 2020.
- [41] E. Penner and L. Zhang. Soft 3D reconstruction for view synthesis. In *Proc. SIGGRAPH Asia*, 2017.

- [42] A. Pumarola, E. Corona, G. Pons-Moll, and F. Moreno-Noguer. D-NeRF: Neural radiance fields for dynamic scenes. In *Proc. CVPR*, 2020.
- [43] D. Rebain, W. Jiang, S. Yazdani, K. Li, K. M. Yi, and A. Tagliasacchi. DeRF: Decomposed radiance fields. In *Proc. CVPR*, 2021.
- [44] C. Reiser, S. Peng, Y. Liao, and A. Geiger. KiloNeRF: Speeding up neural radiance fields with thousands of tiny MLPs. *CoRR*, abs/2103.13744, 2021.
- [45] S. Ren, K. He, R. Girshick, and J. Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. In *NIPS*, 2015.
- [46] G. Riegler and V. Koltun. Free view synthesis. In *Proc. ECCV*, 2020.
- [47] K. Schwarz, Y. Liao, M. Niemeyer, and A. Geiger. GRAF: Generative radiance fields for 3D-aware image synthesis. In *NeurIPS*, 2020.
- [48] S. M. Seitz and C. R. Dyer. View morphing. In *Proc. Computer graphics and interactive techniques*, 1996.
- [49] S. M. Seitz and C. R. Dyer. Photorealistic scene reconstruction by voxel coloring. In *Proc. CVPR*, pages 1067–1073, 1997.
- [50] Q. Shan, R. Adams, B. Curless, Y. Furukawa, and S. M. Seitz. The visual turing test for scene reconstruction. In *Proc. 3DV*, 2013.
- [51] Z. Shou, D. Wang, and S.-F. Chang. Temporal action localization in untrimmed videos via multi-stage cnns. In *Proc. CVPR*, 2016.
- [52] V. Sitzmann, M. Zollhöfer, and G. Wetzstein. Scene representation networks: Continuous 3D-structure-aware neural scene representations. In *NeurIPS*, 2019.
- [53] I. Tolstikhin, N. Houlsby, A. Kolesnikov, L. Beyer, X. Zhai, T. Unterthiner, J. Yung, A. Steiner, D. Keysers, J. Uszkoreit, M. Lucic, and A. Dosovitskiy. MLP-Mixer: An all-MLP architecture for vision. *CoRR*, abs/2105.01601, 2021.
- [54] A. Trevithick and B. Yang. GRF: Learning a general radiance field for 3D scene representation and rendering. *CoRR*, abs/2010.04595, 2020.
- [55] K. van de Sande, J. Uijlings, T. Gevers, and A. Smeulders. Segmentation as selective search for object recognition. In *Proc. ICCV*, 2011.
- [56] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. In *NeurIPS*, 2017.
- [57] Q. Wang, Z. Wang, K. Genova, P. Srinivasan, H. Zhou, J. T. Barron, R. Martin-Brualla, N. Snavely, and T. Funkhouser. IBRNet: Learning multi-view image-based rendering. *CoRR*, abs/2102.13090, 2021.
- [58] Z. Wang, A. Bovik, H. Sheikh, and E. Simoncelli. Image quality assessment: From error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13, 2004.
- [59] Z. Wang, S. Wu, W. Xie, M. Chen, and V. A. Prisacariu. NeRF--: Neural radiance fields without known camera parameters. *CoRR*, abs/2102.07064, 2021.
- [60] W. Xian, J.-B. Huang, J. Kopf, and C. Kim. Space-time neural irradiance fields for free-viewpoint video. In *Proc. CVPR*, 2021.
- [61] H. Xu, A. Das, and K. Saenko. R-C3D: Region convolutional 3D network for temporal activity detection. In *Proc. ICCV*, 2017.
- [62] A. Yu, R. Li, M. Tancik, H. Li, R. Ng, and A. Kanazawa. PlenOctrees for real-time rendering of neural radiance fields. *CoRR*, abs/2103.14024, 2021.
- [63] A. Yu, V. Ye, M. Tancik, and A. Kanazawa. pixelNeRF: Neural radiance fields from one or few images. In *Proc. CVPR*, 2021.
- [64] K. Zhang, G. Riegler, N. Snavely, and V. Koltun. NeRF++: Analyzing and improving neural radiance fields. *CoRR*, abs/2010.07492, 2020.
- [65] Y. Zhao, Y. Xiong, L. Wang, Z. Wu, X. Tang, and D. Lin. Temporal action detection with structured segment networks. In *Proc. ICCV*, 2017.
- [66] T. Zhou, R. Tucker, J. Flynn, G. Fyffe, and N. Snavely. Stereo magnification: Learning view synthesis using multiplane images. In *SIGGRAPH*, 2018.
- [67] M. Zhu, K. P. Murphy, and R. Jonschkowski. Towards differentiable resampling. *CoRR*, abs/2004.11938, 2020.
- [68] M. Zwicker, H. Pfister, J. Van Baar, and M. Gross. Surface splatting. In *Proc. ACM SIGGRAPH*, 2001.

Appendix overview

Appendix A contains full details of the model architectures. Appendix B contains additional details of the training procedure. Full results of all experiments are provided in Appendix C.

A Architectures

The architectures of the coarse and fine networks are identical to the ones in NeRF [37]. The proposer architectures are shown in detail in Figure 6.

The most expensive operations of *Pool* and *MLPMix* are the ones applied on each of the $N_c = 64$ coarse features, *i.e.* operations before the average pooling. The bottleneck is the single fully connected

layer of size 256×256 . Recall that the NeRF backbone [37] has 8 of these layers in each of the coarse and fine networks, and therefore an FC of size 256×256 is already being applied on a total of $8 \times (N_c + N_c + N_f) = 2048$ vectors, and the additional application on $N_c = 64$ vectors by these proposers is negligible.

The *Transformer* proposer is somewhat more expensive due to the lack of pooling operations, and to make its cost reasonable we constrain it heavily – features are projected down to 16-D, single transformer blocks are being used for the encoder and the decoder, and only a single attention head is being used.

B Training details

Here we provide further details to complement Section 3.1.

General training details. As with the original NeRF implementation [37], there are two slight differences in the setup for the two datasets – for *LLFF-Nerf* the color of infinity is taken to be black and during training random Normal noise is added to the activations before the ReLU that produces density σ , while for *Blender* no noise is added and the color of infinity is white.

Two-stage training. Recall from Section 2.1 that the training is split into two stages of equal duration, where during the first stage the learnt proposer is trained to mimic the heuristic one, while in the second stage all components are trained in an end-to-end manner. When training NeRF-ID, for fairness, we use exactly the same settings including the number of training rays and SGD steps as for NeRF[†]. The only slight difference is that at the start of the second phase we reset the optimizer state and perform another linear warmup for 1k steps. This is to reduce the shock to the system when switching between the two training regimes, though we still see some undesirable behaviour (loss rises sharply when switching stages).

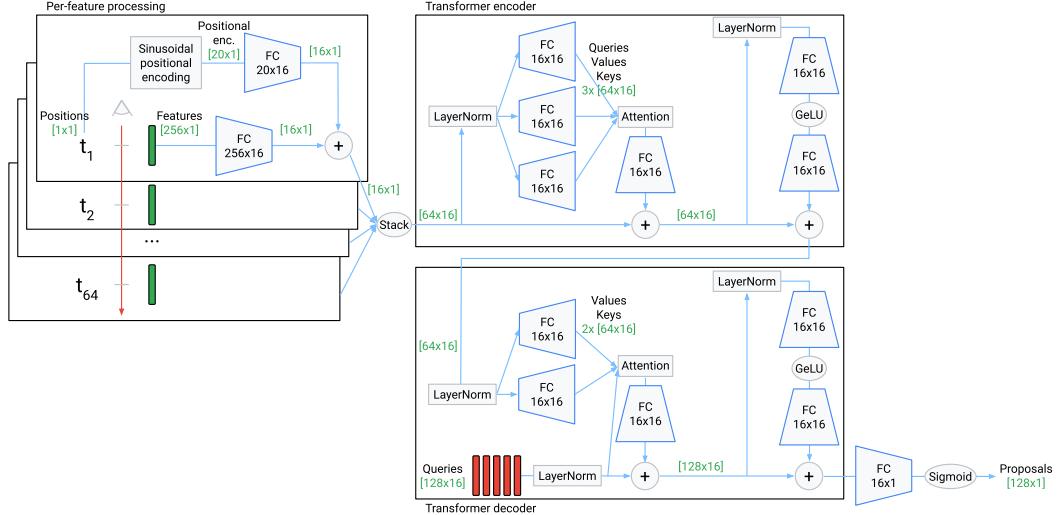
As explained in Section 2.1, during phase 1, the learnt proposer is trying to mimic the heuristic one, which is achieved by minimizing the L2 matching loss between the two sets of proposals. Each heuristic sample is matched with the closest proposal produced by the learnt proposer. The matching is aggressively greedy for speed reasons as all other versions (Hungarian algorithm for optimal matching, greedy 1-1 matching) significantly slowed down training.

Importance prediction. As explained in Section 2.2, the importance of each proposal can be predicted. Note that we place a ‘stop gradient’ operation at the input to the importance predictor, *i.e.* the input coarse-network-features are not affected by training the importance predictor. We have not ablated this design choice, but our intuition is that allowing this gradient propagation path could hurt the overall performance because it could push the proposer and importance predictor towards a trivial solution that yields a low importance prediction loss – produce bad samples and predict they are unimportant. Furthermore, using the ‘stop gradient’ also simplifies comparisons (training the importance predictor does not affect the rest of the network) and removes the need for an extra hyper-parameter that is the weight of the importance predictor loss (as this loss and the main loss do not interact so Adam [23] scales them independently).

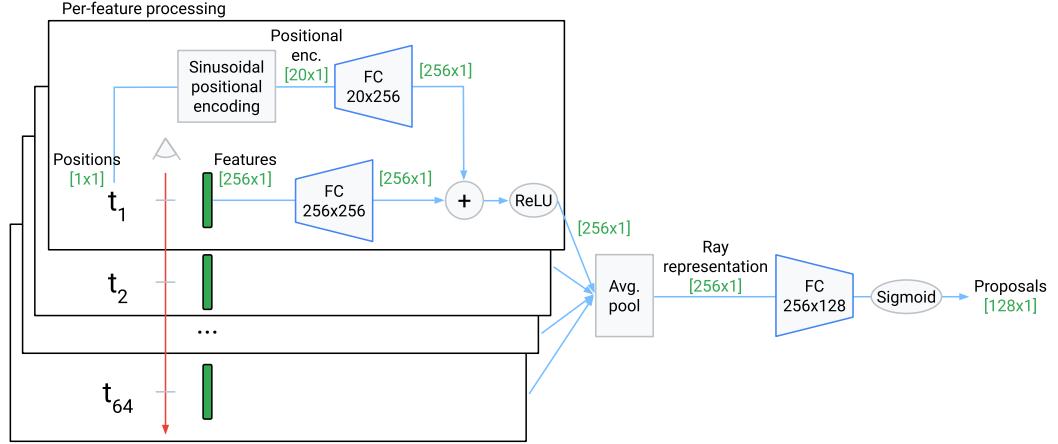
C Results

Proposer architectures. Figure 7 complements Table 1, showing PSNRs for all experimental runs. It confirms that the findings of the main paper are statistically significant, where *Pool* and *MLPMix* proposers tend to be the best, but *Pool* has a higher variance and is significantly worse on two scenes (*Blender: Materials*, *Blender: Mic*). Figure 8 shows the evaluation in terms of SSIM, which reveals the same pattern.

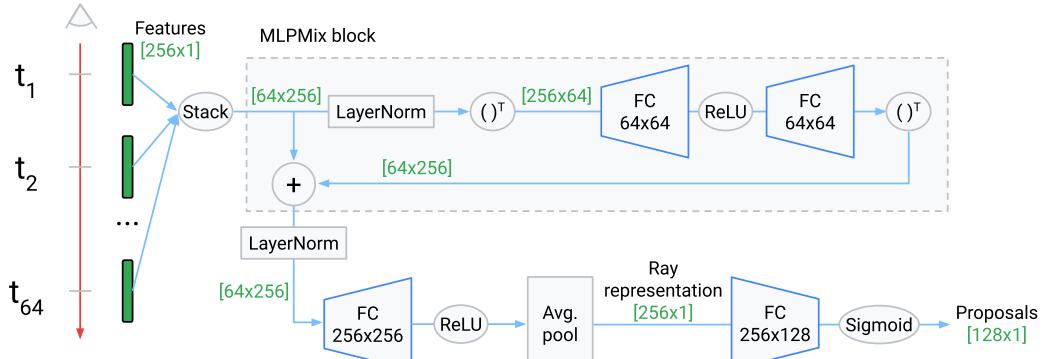
Figure 9 shows additional ablations on the *Pool* proposer architecture. The *Blind* proposer, which ignores input coarse-features and therefore simply always outputs the same proposals, works badly, showing that it is not sufficient to follow such a degenerate strategy. The *No-position* proposer, which ignores the positions of coarse samples along the ray, performs badly on *Blender* because it is insensitive to feature ordering, *e.g.* if produces the same proposals for two colinear rays of opposite directionality, but is competitive on *LLFF-Nerf* as all its scenes are forward-facing. The exact



(a) Transformer



(b) Pool



(c) MLPMix

Figure 6: **Trainable proposer architectures (detailed).** Assumes the number of coarse samples is $N_c = 64$ and the number of proposals is $N_f = 128$, as is default in our NeRF-ID (same values are used in NeRF [37]).

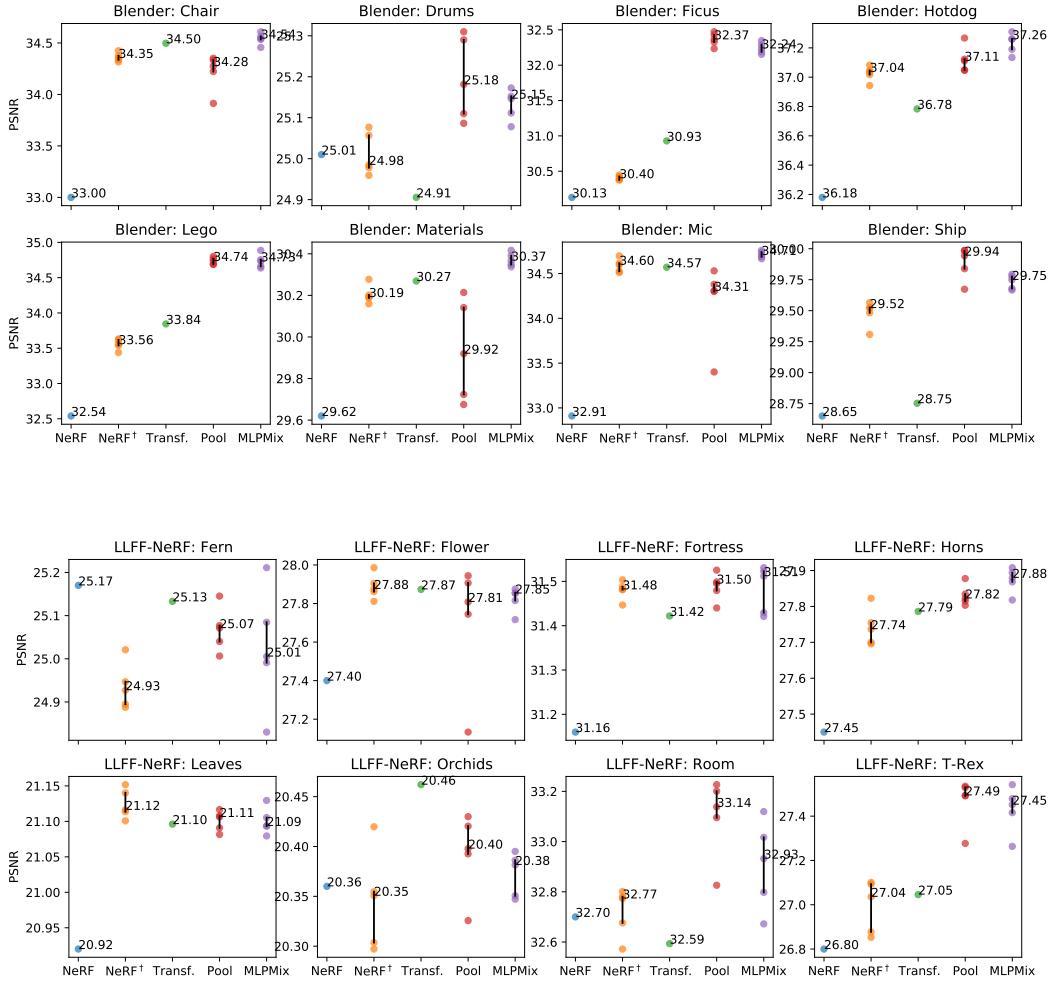


Figure 7: Comparison of proposer architectures and vanilla NeRF (PSNR). For each scene and each method, the performance for all experimental runs is plotted. Vertical black lines connect the 20th and 80th percentiles (*i.e.* the 2nd best and 2nd worst performance out of 5 runs), indicating a robust estimate of the range of performances the method achieves. The median performance is highlighted with its numerical value. NeRF denotes the results reported in the original paper [37], NeRF[†] is our reimplementation.

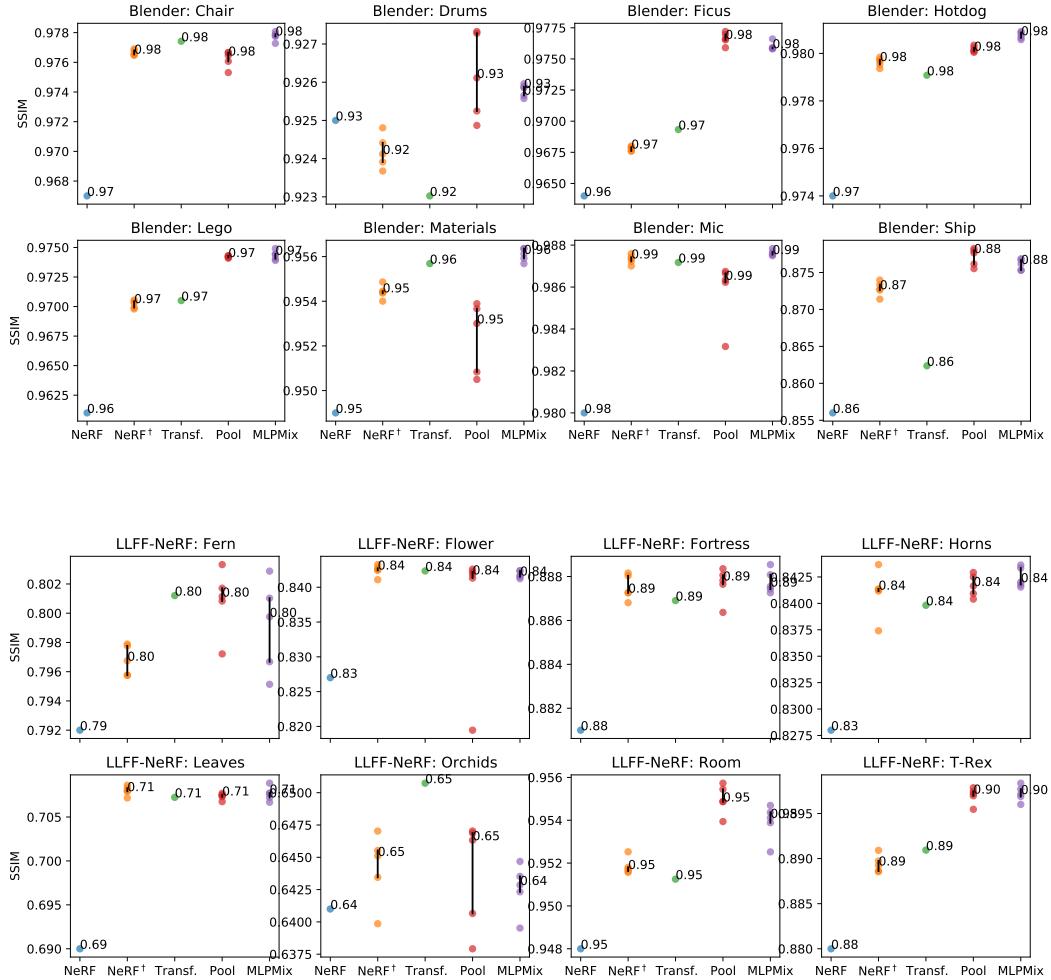


Figure 8: Comparison of proposer architectures and vanilla NeRF (SSIM). For each scene and each method, the performance for all experimental runs is plotted. Vertical black lines connect the 20th and 80th percentiles (*i.e.* the 2nd best and 2nd worst performance out of 5 runs), indicating a robust estimate of the range of performances the method achieves. The median performance is highlighted with its numerical value. NeRF denotes the results reported in the original paper [37], NeRF[†] is our reimplementation.

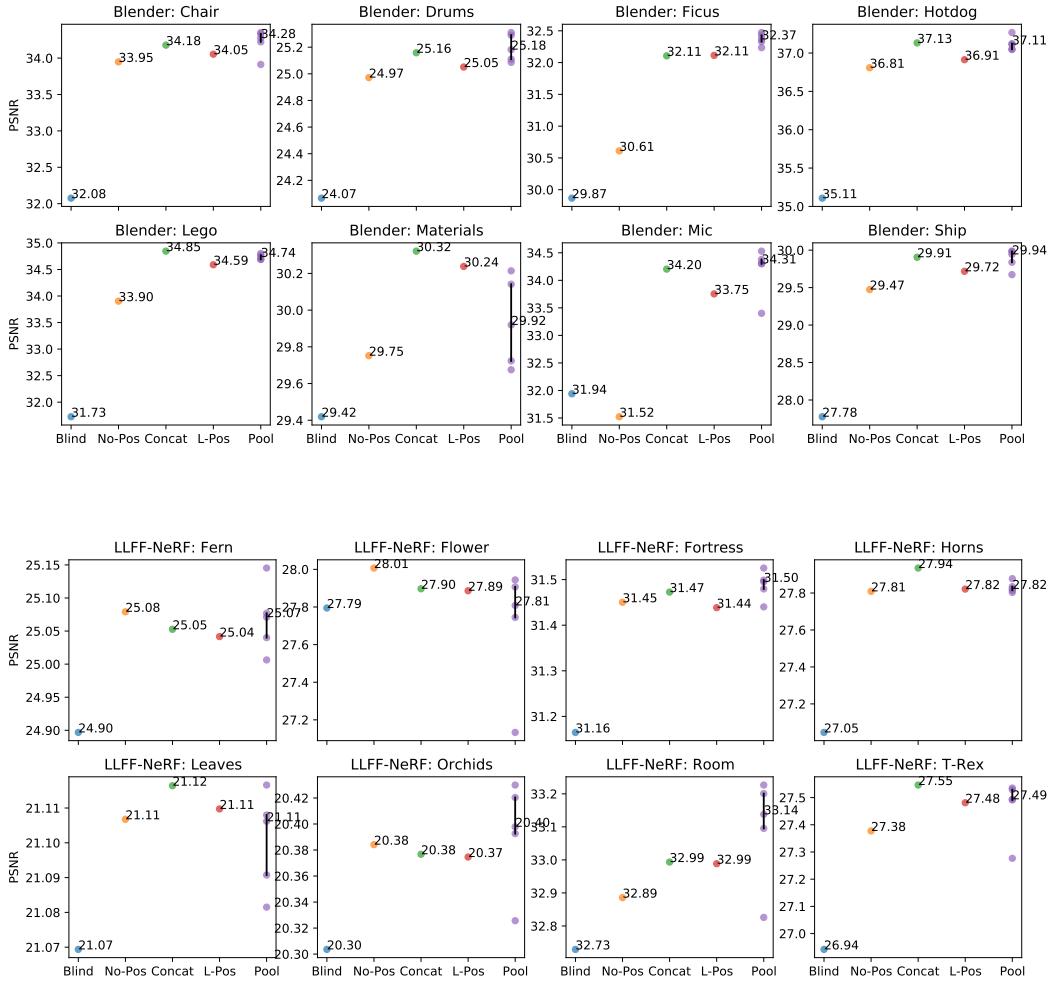


Figure 9: Additional ablations on the *Pool* proposer architecture. For each scene and each method, the performance for all experimental runs is plotted. Vertical black lines connect the 20th and 80th percentiles (*i.e.* the 2nd best and 2nd worst performance out of 5 runs), indicating a robust estimate of the range of performances the method achieves. The median performance is highlighted with its numerical value. The *Blind* proposer ignores input coarse-features and therefore simply always outputs the same proposals. The *No-position* (*No-Pos*) proposer has the same architecture as *Pool* apart from not using the positions of coarse samples along the ray. *Concat* concatenates the positional encodings instead of summing them up as in *Pool*. *Learnt-position* (*L-Pos*) learns positional encodings instead of using the sinusoidal positional encodings as in *Pool*.

manner in which positions are incorporated is not so important – *Concat* (positional encodings are concatenated with the features) and *Learnt-position* (learnt positional encodings are summed with the features) perform similarly to the *Pool* proposer (sinusoidal positional encodings [37] are summed with the features).

Two-stage training. Figure 10 shows that training from scratch sometimes works but often fails, while two-stage training (Section 2.1) is effective.

Comparison with the state-of-the-art. Figures 11 and 12 complement Table 2, showing PSNRs and SSIMs for all experimental runs and all scenes. They confirm that the findings of the main paper are statistically significant.

Speedup. Figure 13 shows speedups on the *Blender* dataset obtained by only querying the fine network on samples deemed to be important by the proposer (Section 2.2); Figure 3 shows the results for *LLFF-NeRF*. Speedups are especially large for *Blender* because many pixels correspond to the background, so the corresponding rays can be heavily pruned. NeRF-ID consistently dominates NeRF † , producing better images using fewer computations. In many cases it is possible to render views 50% faster while still achieving better results than NeRF † .

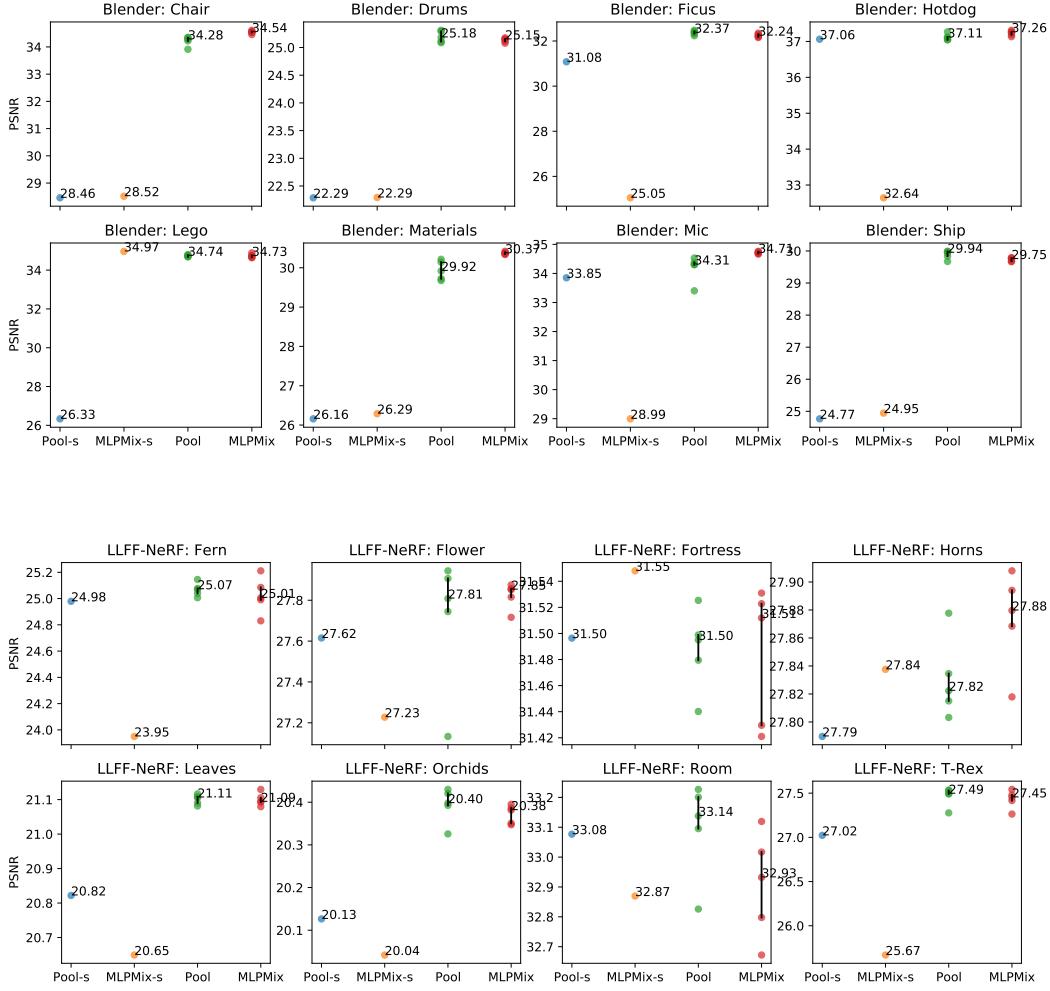


Figure 10: **Training from scratch vs. two-stage training.** For each scene and each method, the performance for all experimental runs is plotted. Vertical black lines connect the 20th and 80th percentiles (*i.e.* the 2nd best and 2nd worst performance out of 5 runs), indicating a robust estimate of the range of performances the method achieves. The median performance is highlighted with its numerical value. *Pool* and *MLPMix* are our proposals trained with the two-stage procedure (Section 2.1), while *Pool-s* and *MLPMix-s* are the same proposers but trained from scratch.

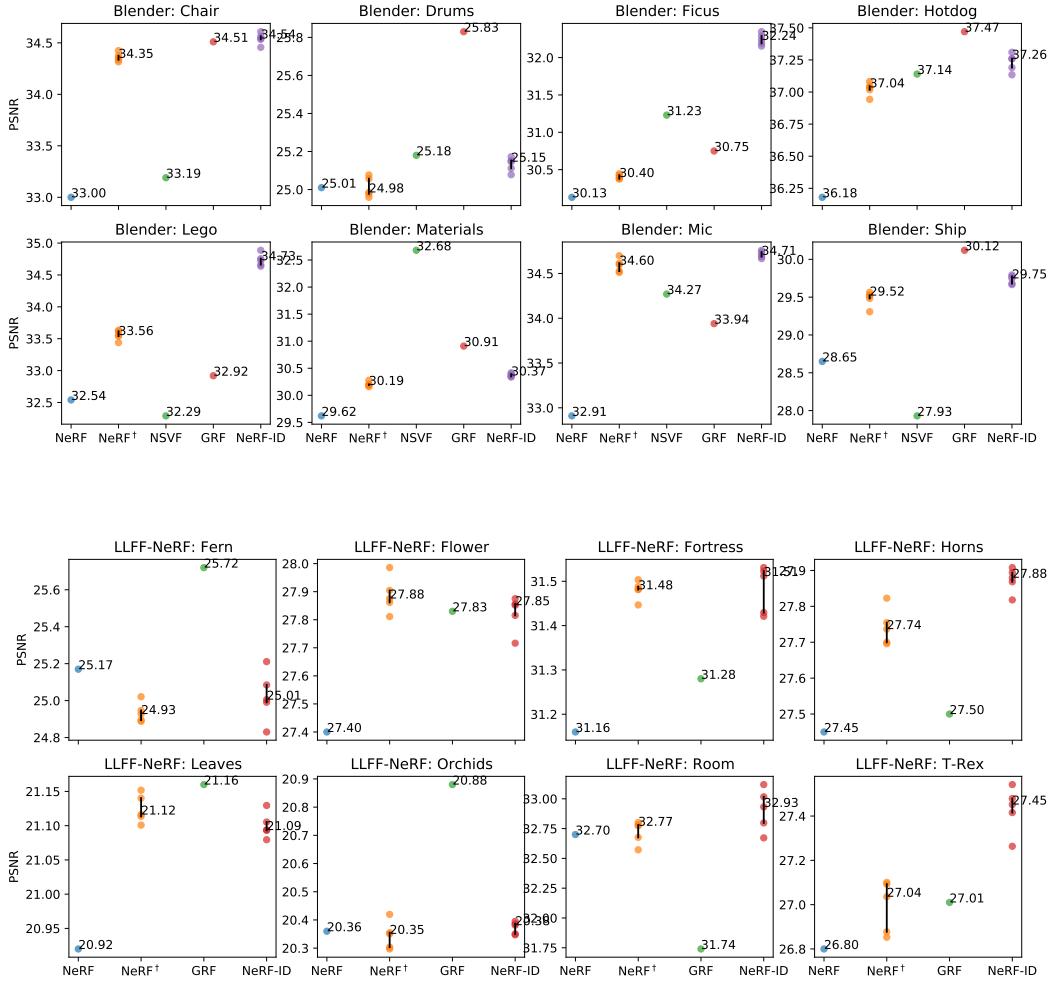


Figure 11: Comparison with the state-of-the-art (PSNR). For each scene and each method, the performance for all experimental runs is plotted. Vertical black lines connect the 20th and 80th percentiles (*i.e.* the 2nd best and 2nd worst performance out of 5 runs), indicating a robust estimate of the range of performances the method achieves. The median performance is highlighted with its numerical value. NeRF denotes the results reported in the original paper [37], NeRF[†] is our reimplementation. GRF is the method of [54]. NSVF [31] only reports results on *Blender* as the method is not capable of handling unbounded scenes in *LLFF-NeRF*. IBRNet [57] is missing because it does not report per-scene performances, see Table 2 for the average results.

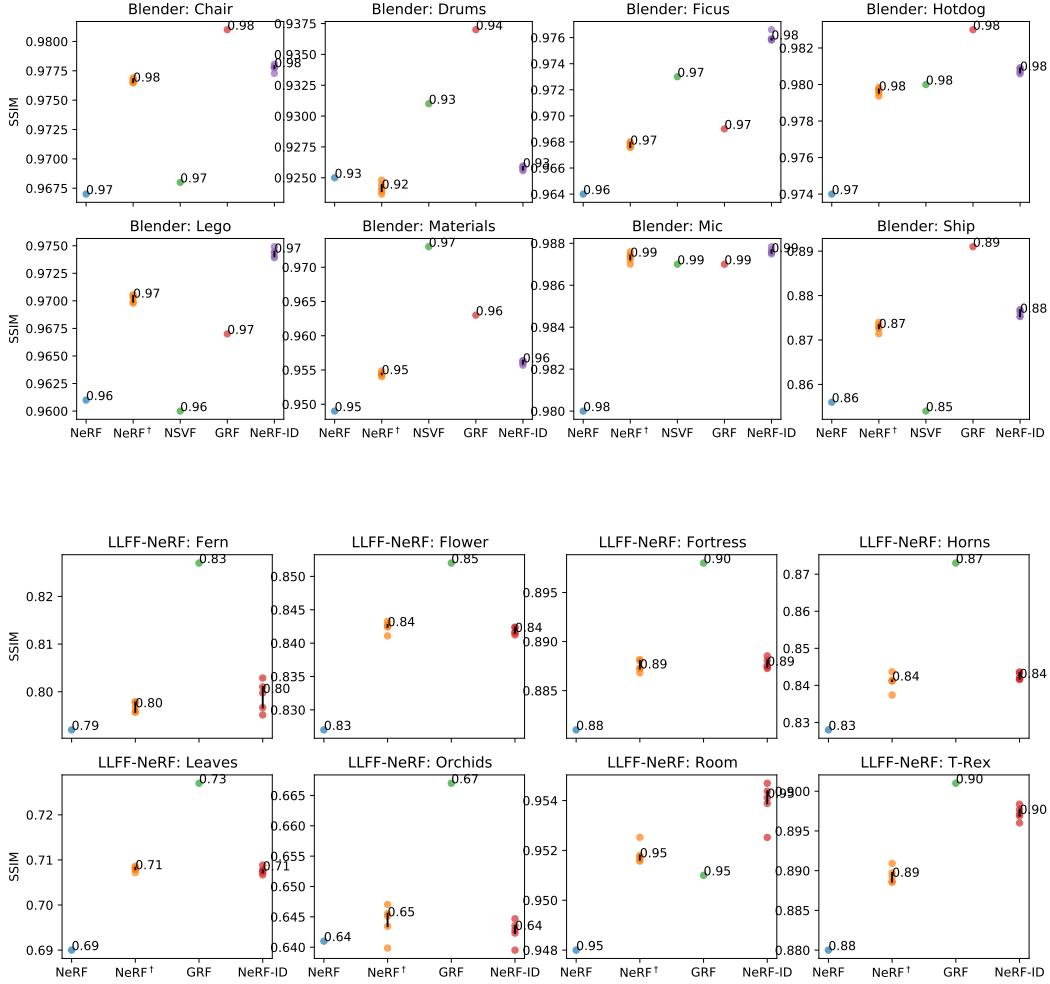


Figure 12: Comparison with the state-of-the-art (SSIM). For each scene and each method, the performance for all experimental runs is plotted. Vertical black lines connect the 20th and 80th percentiles (*i.e.* the 2nd best and 2nd worst performance out of 5 runs), indicating a robust estimate of the range of performances the method achieves. The median performance is highlighted with its numerical value. NeRF denotes the results reported in the original paper [37], NeRF[†] is our reimplementation. GRF is the method of [54]. NSVF [31] only reports results on *Blender* as the method is not capable of handling unbounded scenes in *LLFF-NeRF*. IBRNet [57] is missing because it does not report per-scene performances, see Table 2 for the average results.

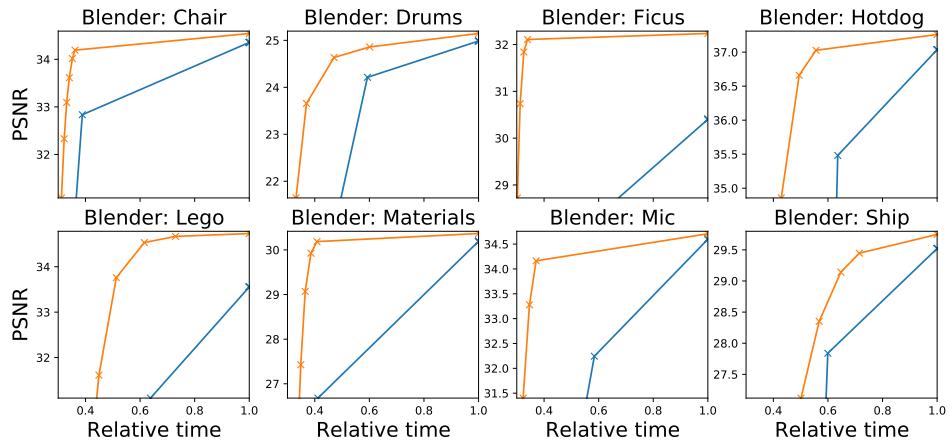


Figure 13: **Speedup via importance prediction, NeRF[†] vs. our NeRF-ID.** Samples deemed to be important by the proposer are kept, different operating points are obtained by varying the importance threshold. ‘Relative time’ is the time spent on rendering the scene relative to using all samples.