Implicit Neural Representations with Levels-of-Experts

Zekun Hao[*][†] hz472@cornell.edu

Arun Mallya[*] amallya@nvidia.com

Serge Belongie[†] sjb344@cornell.edu

Ming-Yu Liu[*] mingyul@nvidia.com

[*]NVIDIA [†]Cornell University

Abstract

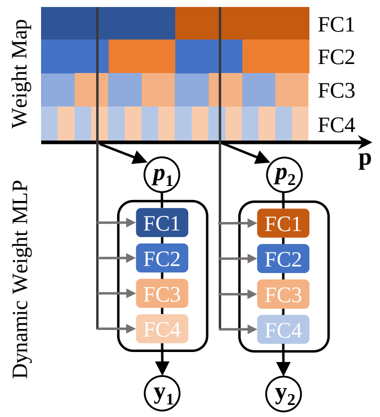
Coordinate-based networks, usually in the forms of MLPs, have been successfully applied to the task of predicting high-frequency but low-dimensional signals using coordinate inputs. To scale them to model large-scale signals, previous works resort to hybrid representations, combining a coordinate-based network with a grid-based representation, such as sparse voxels. However, such approaches lack a compact global latent representation in its grid, making it difficult to model a distribution of signals, which is important for generalization tasks. To address the limitation, we propose the Levels-of-Experts (LoE) framework, which is a novel coordinate-based representation consisting of an MLP with periodic, position-dependent weights arranged hierarchically. For each linear layer of the MLP, multiple candidate values of its weight matrix are tiled and replicated across the input space, with different layers replicating at different frequencies. Based on the input, only one of the weight matrices is chosen for each layer. This greatly increases the model capacity without incurring extra computation or compromising generalization capability. We show that the new representation is an efficient and competitive drop-in replacement for a wide range of tasks, including signal fitting, novel view synthesis, and generative modeling.

1 Introduction

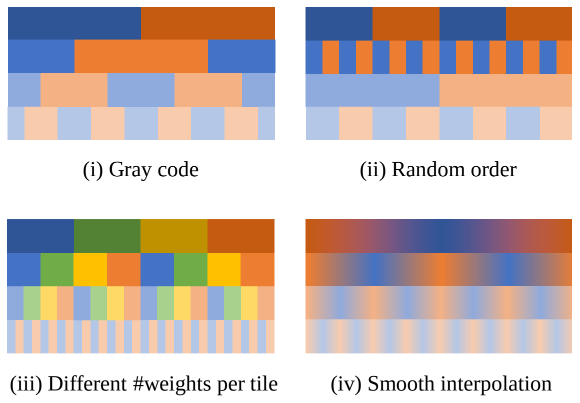
There has been a growing interest in representing low-dimensional but high-frequency signals, such as images, videos, and 3D scenes, with fully-connected neural networks. A common paradigm is to use a coordinate-based multilayer perceptron (MLP) that takes coordinate positions as input and predicts the data value at the specified location [32, 46, 51]. Compared to explicit representations such as point clouds and voxel grids, this kind of implicit neural representation (INR) is memory efficient and can model a distribution of signals for conditional synthesis tasks [3, 28, 36, 39, 45] thanks to its ability to learn a compact and meaningful latent space.

However, scaling up an INR to better represent higher-resolution signals or a distribution of signals, like a distribution of images, is challenging because the mapping can be highly nonlinear. To increase the model capacity to deal with the complexity, we can either make the MLP wider by increasing the dimensions of activations or deeper by stacking more layers. Unfortunately, both options will dramatically increase the computation needed at each data point. Recently, Rebain *et al*. [42] have shown that this results in an undesirable trade-off because the representation power gain diminishes quickly with increased width or depth. We further explore and analyze this issue in Section 4.2.

Many recent works bypass this scaling problem by using a hybrid representation [5, 9, 11, 12, 21, 26, 27, 35, 40, 42, 43, 49, 50]. A discrete data structure, such as sparse voxels, decomposes the space into grids. Within each grid, a lightweight MLP conditioned on the grid embedding produces local detail at a scale finer than the grid resolution. However, such an approach has two major limitations: (1) The reliance on smooth interpolation of grid embeddings [26, 27, 35, 40, 49] or



(a) MLP with position-dependent weights



(b) Examples of different weight tiling patterns

Figure 1: The Levels-of-Experts (LoE) framework. (a) A position-dependent MLP with 1D input, $y = f(x, \theta(x))$ (activation functions omitted for brevity). In this example, each fully-connected (FC) layer has two candidate weight matrices (marked in blue and orange, shade denotes layer depth), arranged in a periodical and hierarchical manner. According to the input location, one of the weight copies is selected for each layer. (b) A variety of hierarchical tiling patterns can be used with LoE. They are not confined to (i) a specific alignment, and can have varying (ii) orders of granularity, or (iii) length of the repetend, and (iv) can be generalized to a smooth interpolation across the weight matrices. Thus, each layer at a different *level*, or scale, has a number of *experts* with their own weight matrices, specializing at different regions of the input space.

output-domain tiling and blending [5, 21, 30, 50] to encourage continuity across the grid boundaries can negatively impact computation efficiency; and, (2) As the underlying signal is described by multiple distributed features stored in a grid, which are associated with fixed locations, it lacks a global, compact latent representation unless it employs another expensive model to generate the grid embedding itself [6, 40, 44]. Still, this is only possible in limited cases, *e.g.*, regular grid, without any sparsity, pruning, or hashing.

Our approach extends the idea of hybrid representation by storing the weights of an MLP on a multi-resolution tiled grid. Conceptually, for each of the linear layers, we assign multiple independent copies of its weights, arranged in a tiling pattern and repeated to fill the space as visualized in Figure 1. This partitions the space so that each copy of weights only needs to handle inputs within certain periodical intervals, essentially making the weights of the MLP position-dependent. We refer to this proposed framework as Levels-of-Experts (LoE). Each layer at a different depth, *level*, has a number of *experts* with their own weight matrices, specializing in different partitions of the input space, depending on the tiling pattern used.

Similar to Fourier features [32, 51], we use different grid resolutions for each layer or depth, so that the weight of different layers repeats at different frequencies. This arrangement has several desired properties: (i) While the weight of each layer is repeating, the learnable combined parameterization of all the layers helps avoid repetition over the input range of interest, and (ii) A large number of uniquely parameterized intervals of the combined model can be obtained and tailored to the underlying problem. In fact, in Section 4.1, we show that our model can fit a scene even without any input position encoding – the position-dependent weight can itself serve as a form of positional encoding! We show that when compared to dividing the space to use different MLPs [42, 43], our layer-level tiling approach can reduce output discontinuities without relying on computationally expensive smooth interpolation or blending, while at the same time improving the representation power. Finally, compared to non-repeating grid embeddings [26, 27, 49], our approach encourages the learning of generalizeable mapping, as shown in Sections 4.1 and 4.2, and also improves parameter efficiency, particularly benefiting generative modeling tasks, as shown in Section 4.4.

Our model has a computational cost comparable to a regular MLP of identical architecture, at the same time being more expressive. Although the parameter count, and thus representation power, is greatly amplified by the use of position-dependent weights, only a single copy of weights is active at each input. In practice, this can be implemented efficiently with an off-the-shelf fused gather-GEMMscatter operator [2] with little speed loss. Our method is a drop-in replacement in many applications without the need for any further modifications. To summarize, we make the following contributions:

1. We introduce a novel hybrid implicit neural representation that is parameterized by a hierarchy of position-dependent and periodic weights (Section 3).
2. We extensively study the effect of various design decisions including the periodicity and hierarchy of weights, weight interpolation methods, and the use of input encodings (Section 4).
3. We demonstrate the efficiency and representation power of our architecture on challenging tasks including high-resolution image fitting, video fitting, novel-view synthesis, and image generation (Section 4).

2 Related Work

Implicit neural representations (INRs). INRs represent a signal with a pointwise (coordinate-based) neural network that takes a coordinate as the input and predicts the data value at the location specified by the input coordinate. With INRs, one can query continuous locations independently and efficiently, which is a desired property for many learning, graphics, and vision tasks [52]. INRs have been used in representing images [3, 46, 48, 51], shapes [14, 31, 39, 46, 51], and scenes [4, 32, 37, 38, 47]. Earlier INRs were based on MLPs with ReLU activations [39, 48] and often failed to represent high-frequency detail in the underlying signal. Recent works have greatly addressed the issue by leveraging better input encoding designs [32, 51], activation functions [41, 46], or network architectures [10, 25]. Our approach, a position dependent MLP architecture, is orthogonal to these approaches and can potentially be used in conjunction with them to achieve better results.

Hybrid representations. Several works propose combining INRs with explicit discrete structure representations to improve both the computation and memory efficiency for modeling large and complex signals. Such a hybrid approach often partitions the input space into smaller regions based on the adopted discrete structure representation, which results in local parameterization, or decomposes the input space to low-rank subspaces [6, 7]. Various discrete structure representations for local parameterization have been explored, including regular grids [5, 21, 35, 40, 43], sparse voxels [16, 26], voronoi cells [42], octrees [27, 49], convex parts [9], and learned shape elements [11, 12]. We can even use another neural network to predict the parameterization, which enables generalization [6, 40, 44]. With the hybrid approach, one first obtains a local feature with the discrete structure and the coordinate, which is then inputted to the pointwise MLP to get the data value.

Note that the discrete nature of the hybrid representation calls for special and often costly designs to ensure smooth transitioning across the subdivision boundaries. One popular approach is to smoothly interpolate the grid feature [26, 27, 35, 40, 49] before handing it off to the MLP. Such an approach only requires one MLP evaluation per sample, but the cost of interpolation can be high for high-dimensional features on a high-dimensional grid. Several approaches [5, 21, 30, 50] allow the MLP to predict signals beyond the grid boundaries so that multiple predictions of the same coordinate from nearby grids can be evaluated and smoothly blended in the output domain, but they are more expensive to compute. There are also hybrid representations that completely abandoned smooth interpolation and achieved considerable speedup [43]. However, it requires distillation from a larger, pretrained network to mitigate discontinuity artifacts. Our approach is also a hybrid approach. It enjoys computation and memory efficiency but does not suffer from the boundary interpolation issue.

Neural networks with input-dependent weights. Our method can be regarded as a special type of hybrid representation that use a multi-level tiled grid to parameterize the weights of the pointwise MLP. Depending on where the input coordinate lies on each level of the grid, a different combination of weights is used for the network. Reiser *et al.* [43] shares the same high-level idea of having coordinate-dependent network weights, but they learn completely disjoint networks for different grid locations, while we use a hierarchical and periodical structure. We will show the importance of our hierarchical and periodical structure in obtaining a smooth and expressive representation.

In a broader context, neural networks with data-dependent weights have been used for modeling 3D animation [8, 19] as a form of a collection of experts and for solving differential equations [34] to represent solutions. Our work is different as we use a layer-wise hierarchical parameterization and is designed for hybrid neural implicit representation.

3 Method

A typical coordinate-based multi-layer perceptron (MLP) can be described as a stack of layers,

$$\hat{f} : \mathbf{p} \rightarrow (g^k \circ \phi \circ g^{k-1} \circ \cdots \circ \phi \circ g^1 \circ \gamma)(\mathbf{p}), \tag{1}$$

where $\mathbf{p}$ is the input coordinate at which the MLP is being evaluated, γ is an input mapping, such as the sine-cosine positional encoding [32], ϕ is a non-linear activation function, and $g^i : \mathbf{x} \rightarrow \mathbf{W}^i\mathbf{x}+\mathbf{b}^i$ is the i^{th} linear layer, which performs an affine transformation on the input $\mathbf{x}$, parameterized by a weight matrix $\mathbf{W}^i$ and a bias vector $\mathbf{b}^i$. During training, $\mathbf{W}^i$ and $\mathbf{b}^i$ are optimized via gradient descent to fit the MLP to the data.

In our Levels-of-Experts (LoE) approach, instead of regarding each $\mathbf{W}^i$ as a single learnable matrix, we additionally model it as a function $\psi^i(\cdot)$ of the input coordinate $\mathbf{p}$. The resulting dynamic-weight linear layer has the form, $h^i : (\mathbf{x}, \mathbf{p}) \rightarrow \psi^i(\mathbf{p})\mathbf{x} + \mathbf{b}^i$, where $\mathbf{x}$ are the inputs to the layer, and $\mathbf{p}$ are the location at which the MLP is being evaluated. By replacing the traditional linear layers g^i in the MLP with dynamic-weight layers h^i, we obtain an MLP with input-dependent weights,

$$f : \mathbf{p} \rightarrow (h^k(\mathbf{p}) \circ \phi \circ h^{k-1}(\mathbf{p}) \circ \cdots \circ \phi \circ h^1(\mathbf{p}) \circ \gamma)(\mathbf{p}). \tag{2}$$

As the resulting position-dependent weight matrix has a much higher dimension compared to its input and output vectors and will be evaluated at a large number of query points, it is important for the weight generation functions $\psi^i(\mathbf{p})$ to be fast, inexpensive, and yet expressive. This rules out the popular weight-prediction networks used in hypernetwork-based approaches [15], in which one has to predict a high-dimensional weight per position. Instead, we use a simple, lightweight function, specifically a coordinate interpolation-based method. Multiple candidate values for the weight matrix are stored in a regular grid (tile) and interpolated in a cyclic manner based on the input coordinates.

Consider the case where we have a grid containing N matrices $\{\mathbf{W}^i_0, \ldots, \mathbf{W}^i_{N-1}\}$, where i is the layer depth, and N is a nonnegative integer. We are only interested in the case that $N > 1$ as $N = 1$ reduces to the original pointwise MLP formulation. Given a 1D coordinate $\mathbf{p} = (p)$, the input-dependent weight for layer i, $\mathbf{W}^i$, is computed as

$$\mathbf{W}^i = \psi^i(\mathbf{p}) = \psi^i(p) = \sum_{j=0}^{n-1} B_{j,N}(\alpha^i p + \beta^i)\mathbf{W}^i_j. \tag{3}$$

where α^i and β^i are hyperparameters that adjust the scale and translation of the grid for each layer and $B_{j,N}$ is the blending function that computes the blending coefficient for the j-th candidate. The blending coefficient can take many different forms. For linear and nearest interpolations, they are defined as follows:

$$B^{\text{linear}}_{j,N}(q) = \max(0, 1 - |(q + 1 - j) \mod N - 1|) \tag{4}$$

$$B^{\text{nearest}}_{j,N}(q) = \begin{cases} 1 & \lfloor q \rfloor \mod N = j \\ 0 & \text{otherwise.} \end{cases} \tag{5}$$

Note that here mod denotes positive remainder operation: $a \mod b = a - b\lfloor \frac{a}{b} \rfloor$. We also note that the above equations can easily be extended to multi-dimensional coordinate spaces.

For linear interpolation, regardless of the tile resolution, only 2 of the blending coefficients are non-zero for each coordinate in our 1D example. On the other hand, the nearest interpolation scheme only has a single non-zero coefficient for each coordinate. This sparsity allows a fast and efficient implementation of performing the dynamic-weight linear layer computation for batched inputs: for each candidate weight matrix, $\mathbf{W}^i_j$ where we only gather input vectors that have $B_{j,N} > 0$ at a time, perform matrix multiplication and scaling, and finally scatter the results to the output matrix.

Empirically, we find it helpful to have different layers of our MLP with different spatial frequencies on the grid. This can be easily achieved by using a different set of α^i and β^i per layer. Using different frequencies at different layers gives an inductive bias to the MLP to capture different repetition patterns. It also serves as a form of regularization that encourages the learning of smooth mapping via weight sharing at different locations. We show this is particularly useful in reducing artifacts for novel view synthesis tasks (see Section 4.3).

A non-exhaustive list of potential grid arrangements—a grid arrangement corresponds to a set of $\{(\alpha^i, \beta^i)\}$—are presented in Fig. 1. We note it is even possible to use a randomized tiling pattern by transforming the grid with a random affine transformation, while still seeing a significant performance gain compared to a regular MLP, as evident by the *Random Affine* experiment in Table 1. Unless otherwise mentioned, we arrange the grids in a progressively growing fashion throughout the paper. We start with the first grid (corresponding to the first MLP layer) covering the full input space without repetition, and progressively subdivide the grids using additional layers. This is shown in Fig. 1(a). This arrangement partitions the input space into uniform-sized grids, with each one having a unique combination of weight matrices. A comparison of different grid arrangements is included in the supplementary material.

4 Experiments

In this section, we validate LoE on 4 challenging tasks. In the first two experiments, we fit our model to high-resolution image and video data, evaluate its performance, and study the effect of various design components. Then in Section. 4.3, we evaluate our model on the indirectly supervised, novel-view synthesis task and study its inductive bias. Finally, in Section. 4.4, we demonstrate its generalization capability by training a generative adversarial network (GAN). All the code will be made publicly available.

4.1 Fitting to a High-resolution Image

We study the effect of our hierarchical weight tiling on model capacity and computational efficiency by fitting networks to a high-resolution image of size 8192×8192 [27] pixels. An image is considered as a set of pixels $\{(\mathbf{p}_i, \Theta(\mathbf{p}_i))\}$ represented by their 2-D coordinates $\mathbf{p}_i = (x_i, y_i)$ and RGB colors $\Theta(\mathbf{p}_i) \in \mathbb{R}^3$. The model $\mathbf{p} \rightarrow f(\mathbf{p})$ takes the coordinate as input and predicts the color at the given coordinate. The goal is to fit the model to the data by minimizing the loss: $\mathcal{L}_2 = \sum_i \|f(\mathbf{p}_i) - \Theta(\mathbf{p}_i)\|^2$.

Table 1 compares our model with several baseline methods and ablations. Our main method significantly outperforms baseline methods that do not use position-dependent weights. Despite sharing the same network architecture and computational cost, an MLP using the sine-cosine positional encoding (PE) as the input mapping [32] performed 12dB worse than our model. We also compare with a hybrid model that learns an input coordinate embedding (CE) [3] for the MLP. Their fitting quality is significantly lower than ours at the same parameter count while incurring a higher computational cost. This suggests that learning a position-dependent weight is more effective than learning a grid of embeddings.

We believe that the effectiveness of our method partially comes from the use of a hierarchical and periodic tiling pattern, which encourages the learning of periodic, spatially-shared features. In Table 1, the *interleaved* model uses a periodic weight tiling scheme but lacks the multi-scale arrangement (Fig. 2(d)). Effectively, multiple independent MLPs are learned, each handling a pixel-skipped subset of the image. This only improves the performance slightly compared to the PE MLP baseline. On the other extreme, in the *chunked* experiment, we partition the input space into uniformly sized chunks and use independent MLPs to handle each chunk. Although the fitting quality is improved, there are large variations in errors across different chunks, as shown in Fig. 2(c). In fact, the chunk boundaries are visible in the fitted image, indicating continuity issues.

Our method achieves the best performance by having the tiled weights repeat at a wide range of intervals. This allows a more efficient data representation by exploiting periodicity at a wide range of frequencies and allows a more adaptive distribution of model capacity and fitting error that is less dependent on the geometry of the data. We also compare our piecewise constant weight parameterization against the smooth, piecewise linear variant, implemented by the bilinear interpolation of weights, shown in Fig. 2(b). Despite having 4 times the computational cost, the fitting quality of the smooth variant is only slightly better (+0.61dB) than the faster, piecewise constant version while also sharing a similarly homogeneous error distribution. This indicates that by using the piecewise constant parameterization, the full performance of the tiled weight models can be enjoyed at only a fraction of the cost. Surprisingly, the tiled weight model is able to achieve reasonable performance even without the use of any input position encoding. In the *Constant Input (no PE)* experiment shown in Table 1, we feed a constant vector to the first layer instead of the coordinate encoding. In fact, the position-dependent tiled weight itself is already a form of positional encoding. It is able to identify a large number of unique intervals in the input space (up to $\prod_i n_i$, where n_i is the number of candidate

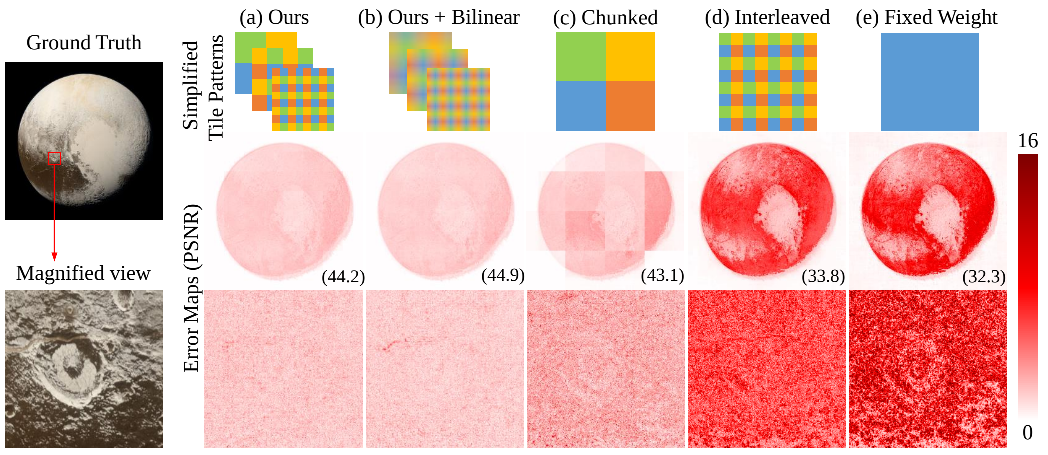


Figure 2: Comparison of errors while fitting to a 64MP image. Our method with discontinuous weights (a) has low and uniformly distributed error comparable to (b), the more expensive version that bilinearly interpolates the weights. (c) and (d) use an ensemble of networks without hierarchical weight tiling, resulting in high error variation, discontinuities, and low fitting quality.

Table 1: Comparison of parameter count, computational cost in number of multiply-accumulates (MACs) per sample, and fitting quality on the 64MP color image of Pluto shown above [27]. All the models use the same number of layers and hidden channels. The model size of our method is chosen to be comparable to ACORN [27]. For the coordinate embedding (CE) baseline, we evaluate multiple grid resolutions and report the best result.

Shown in Fig. 2	Periodic Tiling	Multi-scale		Parameters	MACs	PSNR (dB)	SSIM
			PE MLP [32]	0.59M	0.57M	32.34	0.869
			PE + CE [3]	9.37M	0.65M	39.65	0.967
(d)	✓	✗	Interleaved	9.37M	0.57M	33.80	0.876
(c)	✗	✗	Chunked	9.37M	0.57M	43.13	0.980
	✓	✓	Random Affine	9.37M	0.57M	42.08	0.973
	✓	✓	Constant Input (no PE)	8.92M	0.56M	39.48	0.955
(b)	✓	✓	Ours Bilinear	9.37M	2.28M	**44.85**	**0.985**
(a)	✓	✓	Ours	9.37M	**0.57M**	**44.24**	**0.983**

weights of layer i). Related ideas of using periodical weights in a coordinate-based network are also found in MFN [10] and BACON [25].

4.2 Fitting to a Video

We fit our model to a video [46] with 300 frames and a resolution of 512×512. In this case, each pixel in the video is associated with a 3D coordinate $\mathbf{p}_i = (x_i, y_i, t_i)$. The quantitative results are shown in Table 2, and visual comparisons in Fig. 3. Compared to fixed-weight models such as PE MLP or SIREN [46], the capacity of our model grows favorably with increased input dimensions, without incurring extra computation. For higher-dimensional problems, we can simply use higher dimensional weight tiles. In this case, we use a combination of 2^3 and 4^3 tiles, which amplifies the number of parameters by $8\times$ and $64\times$ compared to a regular linear layer with the same number of channels. This cannot be done in a fully implicit model without greatly increasing the computation. For example, in the SIREN-L experiment, we attempt to increase the model capacity of SIREN by quadrupling the hidden channel count. The resulting model needs $15\times$ more computation, yet the quality is still lacking. This confirms the diminishing return phenomenon associated with coordinate MLPs [42]. Compared to embedding-based hybrid representations such as coordinate embedding (CE), which in this case include a dense $64^3 \times 64$ grid that store the position-dependent features and trilinearly interpolated, the conclusion in Sec.4.1 still holds that our approach performs significantly better under the same parameter count.

Table 2: Comparison of model size, computation and fitting quality on a short video. All of the models have 4 hidden layers and 256 hidden channels, except for SIREN-L, which has 1024 hidden channels.

	Params.	MACs	PSNR(dB)
PE [32]	279K	0.28M	27.33
PE + CE [3]	17.1M	0.29M	35.83
SIREN [46]	265K	0.26M	29.13
SIREN-L	4.21M	4.20M	37.71
Ours	16.9M	0.28M	**39.98**

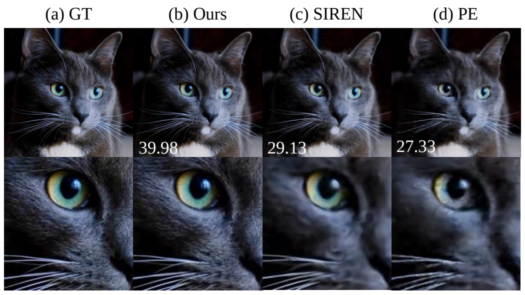


Figure 3: Visual comparison of video fitting results. Numbers indicates PSNR (dB). All the models have the same computational cost.

4.3 Novel View Synthesis

So far our LoE model has demonstrated improved quality in fitting to high-resolution signals via direct supervision. Here, we examine if the model has the desirable inductive bias to work in an under-constrained setting with indirect supervision. We evaluate our method on a novel view synthesis task, where the network models color $\mathbf{c}$ and opacity σ at each 3D location and under different view directions. This kind of volumetric 3D representation is also known as neural radiance fields (NeRF) [32]. Given an image, each pixel in the image can be associated with a ray $\mathbf{r}(t) = \mathbf{o} + t\mathbf{d}$. The color of the pixel can be obtained by sampling points t_i along the ray, querying the neural network at these points to obtain color and opacity $\mathbf{c}_i, \sigma_i = f(\mathbf{r}(t_i), \mathbf{d})$, and perform volumetric rendering via numerical integration [29]:

$$\mathbf{C}(\mathbf{r}) = \sum_{i=1}^{N} T_i(1 - \exp(-\sigma_i \delta_i))\mathbf{c}_i, \text{ where } T_i = \exp\left(-\sum_{j=1}^{i-1} \sigma_j \delta_j\right), \text{ and } \delta_i = t_{i+1} - t_i. \quad (6)$$

The training is done by minimizing the photometric loss between the rendered colors and ground truth pixel values: $\mathcal{L}_2 = \sum_k \|\mathbf{C}(\mathbf{r}_k) - \mathbf{C}_k\|^2$.

We compare our method with baselines on the Tanks and Temples dataset [23, 26], which contains 133 training images at a resolution of 1920×1080. All the models compared have 4 hidden layers and 256 hidden channels. We present the quantitative results in Table 3 and include a visual comparison in Fig. 4. As shown in the zoom-in view, the use of hierarchical, position-dependent weights significantly sets the otherwise identical PE MLP baseline apart by reproducing much better detail.

It has been observed that when the input space is partitioned into grids, and independent MLPs are learned for each grid location, there will exhibit significant free space artifacts in the result [43]. Our model does not have such a problem, despite similarly having position-dependent weights. To gain a better understanding, we experimented with a model that lacked hierarchical arrangement in its weight grids. Despite having a lot more parameters, the ablated model, which indeed suffering from free space artifacts, performed far worst than the baseline PE MLP (Fig. 5). This shows that the use of hierarchically tiled weights is important for providing a good inductive bias for the task.

4.4 Image Generation with GANs

In this section, we demonstrate the generalization capability of the LoE model on the challenging image generation task. The coordinate-based models are used as the generators for the generative adversarial networks (GAN) [13]. More specifically, the model $f(\mathbf{p}, \mathbf{z})$ takes both a coordinate $\mathbf{p}$ and a noise vector $\mathbf{z}$ as input, and map them to a color value. To generate an image, a fixed $\mathbf{z}$ is used and the network is queried at every pixel location $\mathbf{p} \in \mathbf{P}$. We denote the process of generating a full image as $G(\mathbf{z}) = \{f(\mathbf{p}, \mathbf{z}) | \mathbf{p} \in \mathbf{P}\}$. Images of different appearances can be generated by sampling the noise vector from a fixed distribution $\mathbf{z} \sim p_{\mathbf{z}}$. An additional discriminator network D is used to provide the training signal. We use hinge loss [24] as the GAN objective.

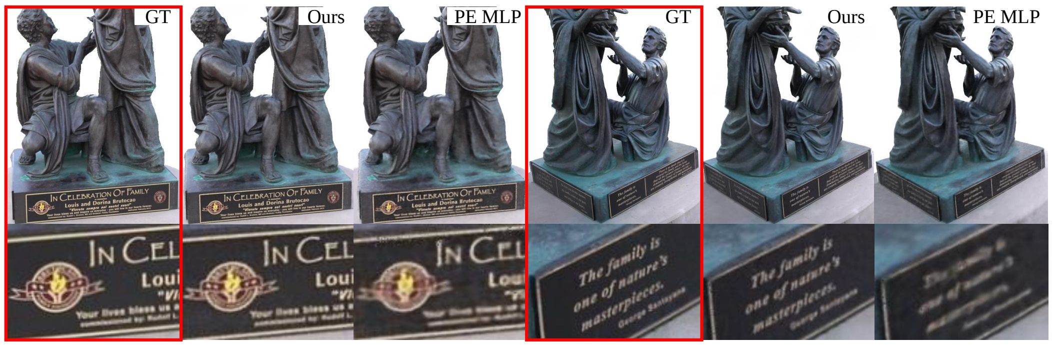


Figure 4: Visual comparison of novel view synthesis results from our model as well as the baseline model. Below each image there are local crops that better show the detail. Our model produces extremely sharp detail compared to the baseline while having the same computational cost. Both models share the same model architecture, with the only difference being the use of hierarchical weight tiling.

Table 3: Comparison of novel view synthesis quality on the *Family* scene. All the models have the same computational cost of 315KMACs per sample.

	#Params.	PSNR(dB)↑	SSIM↑
PE MLP [32]	317K	30.50	0.900
No Hier.	17.8M	27.73	0.861
Ours	17.8M	**31.46**	**0.936**

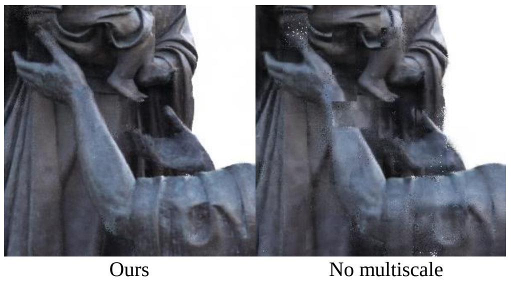


Figure 5: Free-space artifacts in the ablated model that uses tiled weights but lacks the multiscale arrangement. The weight grids in all the layers are aligned and repeated at the same interval.

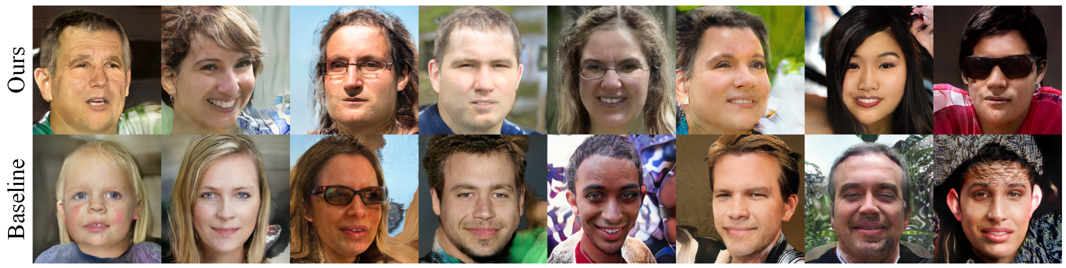


Figure 6: Comparison of image generation results on FFHQ dataset. Images in the top row are generated by our model. Images in the bottom row are generated by a baseline model with coordinate embedding. Both models have comparable parameter counts and computational costs.

To reduce computation and encourage easy reproduction, we use a simplified setting with lightweight components. For the generator, we use an 8-layer network with residual connections. The noise vector is directly fed into the first layer. For the discriminator, we use a multi-resolution patch discriminator [20] with spectral normalization [33]. The model is trained on the Flickr Faces-HQ (FFHQ) dataset [22] at a resolution of 256×256. For the baseline method, we use a coordinate embedding of 256×256 resolution and 256 channels in order to obtain a parameter count comparable to the LoE model. Please refer to the supplementary material for full implementation detail and larger-scale experiments.

We report the model size, computational cost and image quality measured in Fréchet inception distance (FID) [18] in Table 4, and provide a gallery of sample images in Fig. 6. Our method not only achieves a better FID but also produces images free of fixed noise pattern artifacts (shown in Fig. 7).

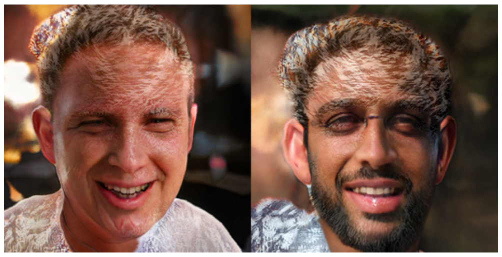


Figure 7: Examples of the fixed noise pattern artifacts in the images generated with the baseline PE + CE method. Our method does not suffer from this issue.

Table 4: Comparison of image generation quality on FFHQ dataset.

	#Params	MACs	FID
PE + CE [3]	19.9M	1.39M	23.5
Ours	19.6M	1.26M	**18.3**

5 Discussion

In this work, we demonstrated a new type of hybrid implicit representation, called Levels-of-Experts (LoE), which is parameterized by hierarchical and periodic, position-dependent weights that are arranged on levels of repeating grids. The new representation offers great versatility, improving the performance on a wide range of tasks. Our method provides greatly increased model capacity compared to fully implicit models while at the same time having a comparably low computational footprint and the same ease of use. Compared to previous grid-based hybrid representations, the LoE model demonstrates good parameter efficiency and generalization capability.

Limitations. The nearest-interpolation variant of our model has undefined derivatives and discontinuities when crossing the grid boundaries. Even though the smooth interpolation variants can be used in these scenarios, compared to SIREN, it does not have smooth, high-order derivatives, limiting its use in applications such as solving differential equations. Similar to previous works, our method required prior knowledge of the underlying signal in order to choose suitable grid scales.

Broader Impact. Our model is orthogonal to various other approaches of improving or extending the increasingly popular implicit neural representations (INRs), such as better activation functions [41], better input encoding [51], the use of hypernetworks [15, 46, 47], or combination with other pre- or post-processing CNNs [6, 40]. Our method can enable higher fidelity data (image, video, volume, *etc*.) synthesis, representation, and compression at reduced computational costs compared to prior works. Like prior works, our method can be misused to negative ends, including for *deepfakes*.

References

[1] Scikit-image: Image processing in python. `https://scikit-image.org/`.

[2] CUTLASS: CUDA templates for linear algebra subroutines. `https://github.com/NVIDIA/cutlass`, 2019.

[3] Ivan Anokhin, Kirill Demochkin, Taras Khakhulin, Gleb Sterkin, Victor Lempitsky, and Denis Korzhenkov. Image generators with conditionally-independent pixel synthesis. In *CVPR*, 2021.

[4] Jonathan T. Barron, Ben Mildenhall, Matthew Tancik, Peter Hedman, Ricardo Martin-Brualla, and Pratul P. Srinivasan. Mip-NeRF: A multiscale representation for anti-aliasing neural radiance fields. In *ICCV*, 2021.

[5] Rohan Chabra, Jan E Lenssen, Eddy Ilg, Tanner Schmidt, Julian Straub, Steven Lovegrove, and Richard Newcombe. Deep local shapes: Learning local SDF priors for detailed 3D reconstruction. In *ECCV*, 2020.

[6] Eric R Chan, Connor Z Lin, Matthew A Chan, Koki Nagano, Boxiao Pan, Shalini De Mello, Orazio Gallo, Leonidas J Guibas, Jonathan Tremblay, Sameh Khamis, et al. Efficient geometry-aware 3d generative adversarial networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16123–16133, 2022.

[7] Anpei Chen, Zexiang Xu, Andreas Geiger, Jingyi Yu, and Hao Su. Tensorf: Tensorial radiance fields. In *European Conference on Computer Vision (ECCV)*, 2022.

[8] Javier Dehesa, Andrew Vidler, Julian Padget, and Christof Lutteroth. Grid-functioned neural networks. In *ICML*, 2021.

[9] Boyang Deng, Kyle Genova, Soroosh Yazdani, Sofien Bouaziz, Geoffrey Hinton, and Andrea Tagliasacchi. CvxNet: Learnable convex decomposition. In *CVPR*, 2020.

[10] Rizal Fathony, Anit Kumar Sahu, Devin Willmott, and J Zico Kolter. Multiplicative filter networks. In *ICLR*, 2020.

[11] Kyle Genova, Forrester Cole, Avneesh Sud, Aaron Sarna, and Thomas Funkhouser. Local deep implicit functions for 3D shape. In *CVPR*, 2020.

[12] Kyle Genova, Forrester Cole, Daniel Vlasic, Aaron Sarna, William T Freeman, and Thomas Funkhouser. Learning shape templates with structured implicit functions. In *ICCV*, 2019.

[13] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. In *NeurIPS*, 2014.

[14] Amos Gropp, Lior Yariv, Niv Haim, Matan Atzmon, and Yaron Lipman. Implicit geometric regularization for learning shapes. In *ICML*, 2020.

[15] David Ha, Andrew Dai, and Quoc V Le. HyperNetworks. In *ICLR*, 2016.

[16] Zekun Hao, Arun Mallya, Serge Belongie, and Ming-Yu Liu. GANcraft: Unsupervised 3D Neural Rendering of Minecraft Worlds. In *ICCV*, 2021.

[17] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[18] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. GANs trained by a two time-scale update rule converge to a local Nash equilibrium. In *NeurIPS*, 2017.

[19] Daniel Holden, Taku Komura, and Jun Saito. Phase-functioned neural networks for character control. *ACM TOG*, 2017.

[20] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. In *CVPR*, 2017.

[21] Chiyu Jiang, Avneesh Sud, Ameesh Makadia, Jingwei Huang, Matthias Nießner, Thomas Funkhouser, et al. Local implicit grid representations for 3D scenes. In *CVPR*, 2020.

[22] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. In *CVPR*, 2019.

[23] Arno Knapitsch, Jaesik Park, Qian-Yi Zhou, and Vladlen Koltun. Tanks and temples: Benchmarking large-scale scene reconstruction. *ACM Transactions on Graphics (ToG)*, 36(4):1–13, 2017.

[24] Jae Hyun Lim and Jong Chul Ye. Geometric GAN. *arXiv preprint arXiv:1705.02894*, 2017.

[25] David B. Lindell, Dave Van Veen, Jeong Joon Park, and Gordon Wetzstein. Bacon: Band-limited coordinate networks for multiscale scene representation. In *CVPR*, 2022.

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

[27] Julien N.P. Martel, David B. Lindell, Connor Z. Lin, Eric R. Chan, Marco Monteiro, and Gordon Wetzstein. ACORN: Adaptive coordinate networks for neural representation. *ACM TOG*, 2021.

[28] Ricardo Martin-Brualla, Noha Radwan, Mehdi SM Sajjadi, Jonathan T Barron, Alexey Dosovitskiy, and Daniel Duckworth. NeRF in the Wild: Neural radiance fields for unconstrained photo collections. In *CVPR*, 2021.

[29] Nelson Max. Optical models for direct volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 1(2):99–108, 1995.

[30] Ishit Mehta, Michaël Gharbi, Connelly Barnes, Eli Shechtman, Ravi Ramamoorthi, and Manmohan Chandraker. Modulated periodic activations for generalizable local functional representations. In *ICCV*, 2021.

[31] Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. Occupancy networks: Learning 3D reconstruction in function space. In *CVPR*, 2019.

[32] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. NeRF: Representing scenes as neural radiance fields for view synthesis. In *ECCV*, 2020.

[33] Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral normalization for generative adversarial networks. In *ICLR*, 2018.

[34] Ben Moseley, Andrew Markham, and Tarje Nissen-Meyer. Finite basis physics-informed neural networks (FBPINNs): a scalable domain decomposition approach for solving differential equations. *arXiv preprint arXiv:2107.07871*, 2021.

[35] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *ACM TOG*, 2022.

[36] Michael Niemeyer and Andreas Geiger. GIRAFFE: Representing scenes as compositional generative neural feature fields. In *CVPR*, 2021.

[37] Michael Niemeyer, Lars Mescheder, Michael Oechsle, and Andreas Geiger. Differentiable volumetric rendering: Learning implicit 3D representations without 3D supervision. In *CVPR*, 2020.

[38] Michael Oechsle, Lars Mescheder, Michael Niemeyer, Thilo Strauss, and Andreas Geiger. Texture fields: Learning texture representations in function space. In *ICCV*, 2019.

[39] Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. DeepSDF: Learning continuous signed distance functions for shape representation. In *CVPR*, 2019.

[40] Songyou Peng, Michael Niemeyer, Lars Mescheder, Marc Pollefeys, and Andreas Geiger. Convolutional occupancy networks. In *ECCV*, 2020.

[41] Sameera Ramasinghe and Simon Lucey. Beyond periodicity: Towards a unifying framework for activations in coordinate-MLPs. *arXiv preprint arXiv:2111.15135*, 2021.

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

[43] Christian Reiser, Songyou Peng, Yiyi Liao, and Andreas Geiger. KiloNeRF: Speeding up neural radiance fields with thousands of tiny MLPs. In *ICCV*, 2021.

[44] Shunsuke Saito, Zeng Huang, Ryota Natsume, Shigeo Morishima, Angjoo Kanazawa, and Hao Li. PIFu: Pixel-aligned implicit function for high-resolution clothed human digitization. In *ICCV*, 2019.

[45] Katja Schwarz, Yiyi Liao, Michael Niemeyer, and Andreas Geiger. GRAF: Generative radiance fields for 3D-aware image synthesis. In *NeurIPS*, 2020.

[46] Vincent Sitzmann, Julien N.P. Martel, Alexander W. Bergman, David B. Lindell, and Gordon Wetzstein. Implicit neural representations with periodic activation functions. In *NeurIPS*, 2020.

[47] Vincent Sitzmann, Michael Zollhöfer, and Gordon Wetzstein. Scene representation networks: Continuous 3D-structure-aware neural scene representations. In *NeurIPS*, 2019.

[48] Kenneth O Stanley. Compositional pattern producing networks: A novel abstraction of development. *Genetic Programming and Evolvable Machines*, 2007.

[49] Towaki Takikawa, Joey Litalien, Kangxue Yin, Karsten Kreis, Charles Loop, Derek Nowrouzezahrai, Alec Jacobson, Morgan McGuire, and Sanja Fidler. Neural geometric level of detail: Real-time rendering with implicit 3D shapes. In *CVPR*, 2021.

[50] Matthew Tancik, Vincent Casser, Xinchen Yan, Sabeek Pradhan, Ben Mildenhall, Pratul P Srinivasan, Jonathan T Barron, and Henrik Kretzschmar. Block-nerf: Scalable large scene neural view synthesis. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8248–8258, 2022.

[51] Matthew Tancik, Pratul P. Srinivasan, Ben Mildenhall, Sara Fridovich-Keil, Nithin Raghavan, Utkarsh Singhal, Ravi Ramamoorthi, Jonathan T. Barron, and Ren Ng. Fourier features let networks learn high frequency functions in low dimensional domains. *NeurIPS*, 2020.

[52] Yiheng Xie, Towaki Takikawa, Shunsuke Saito, Or Litany, Shiqin Yan, Numair Khan, Federico Tombari, James Tompkin, Vincent Sitzmann, and Srinath Sridhar. Neural fields in visual computing and beyond. *Eurographics*, 2022.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] Included in Section 5.
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] Included in Section 5.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] We include all the details needed to reproduce the results in the supplemental material.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] Fully specified in the supplemental material.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] The error bars are included in the supplemental material.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] Included in the supplemental material.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [Yes] In the supplemental material. All the data we used are publicly available.
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes] We include code snippets in the supplemental material.

(d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [Yes] All the data are permissively licensed.

(e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [Yes] The particularly relevant, publicly available FFHQ dataset is originally obtained from permissively licensed sources.

5. If you used crowdsourcing or conducted research with human subjects...

   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A] We did not crowdsource or use human subjects.

   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]

   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

Supplementary Material for
Implicit Neural Representations with Levels-of-Experts

A Comparison of Different Hierarchical Grid Layouts

In the main paper, we have shown that our levels-of-experts framework supports a wide variety of grid layouts (Sec. 3), and arranging the grids in a multi-scale fashion improves the performance (Sec. 4.1 and Sec. 4.3). This section further investigates the performance implications of different hierarchical grid layouts on a 2D toy experiment, where we fit the model to a 512×512 image.

Here we only consider the case of nearest interpolation (piecewise constant) parameterization of the position-dependent weights. In 2D, consider an input coordinate $\mathbf{p} \in [0, 1]^2$, and a 2×2 weight tile for the i-th layer $\{\mathbf{W}^i_{0,0}, \mathbf{W}^i_{0,1}, \mathbf{W}^i_{1,0}, \mathbf{W}^i_{1,1}\}$, the position-dependent weight $\psi^i(\mathbf{p})$ can be computed as follows:

$$\psi^i(\mathbf{p}) = \mathbf{W}^i_{\lfloor x \rfloor \bmod 2, \lfloor y \rfloor \bmod 2}, \tag{7}$$

$$\text{where } \begin{bmatrix} x \\ y \end{bmatrix} = \mathbf{A}^i \mathbf{p} + \mathbf{b}^i. \tag{8}$$

An affine transformation $\mathbf{A}^i$ and $\mathbf{b}^i$ is selected for each layer to allow the weight grids to cover a wide range of spatial frequencies.

We study the effect of different arrangements of $\mathbf{A}^i$ and $\mathbf{b}^i$ via controlled experiments. As shown in Table 5, our models outperform the PE MLP baseline over a wide range of hierarchical grid layouts. In this paper, for consistency, we mainly use coarse-to-fine arrangements in all the experiments, similar to the *Quad Tree* arrangement in this case. However, we also want to point out that the performance of the LoE model can be further improved by tuning the grid layouts, as evident by the better performance achieved with the *Fine to Coarse* arrangement in this case.

Architectures. For all the experiments, we use a 10-layer network with 64 hidden channels, which consists of 9 position-dependent linear layers $i = 1, \ldots, 9$ and a final linear layer. All of the position-dependent linear layers use a 2×2 weight tile. We use Leaky ReLU (0.2 negative slope) activation function between all the layers. We use positional encoding [32] with $L = 8$ frequencies as the input mapping.

Training Details. We use the mean squared error as the reconstruction loss. We use Adam optimizer with $(\beta_1, \beta_2) = (0.9, 0.995)$ and a learning rate of 0.005. We train all the models for 10k iterations, with the learning rate decayed to 0.0005 after the first 5k iterations. We initialize the bias vectors to zeros and initialize the weights using the uniform distribution variant of Kaiming initialization [17]. We sample the full image in each iteration (i.e. no subsampling).

Dataset. We use the "camera" test image from the scikit-image Python package [1], which is a grayscale image with a resolution of 512×512. This image is CC0 licensed.

Runtime & Hardware. Each experiment takes approximately 7 minutes on a NVIDIA Titan V GPU.

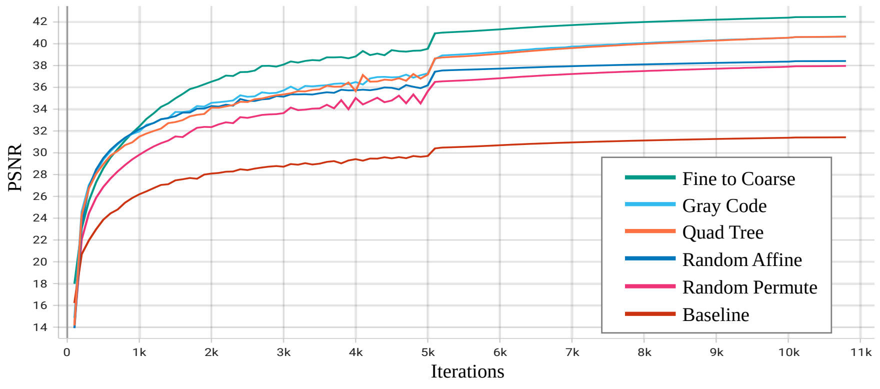
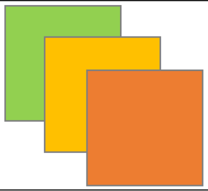
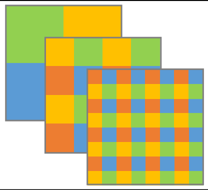
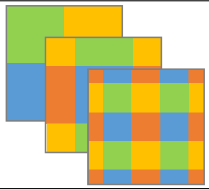
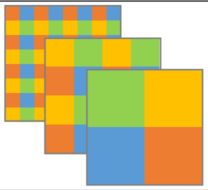
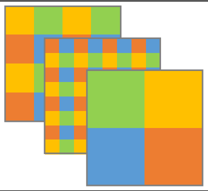
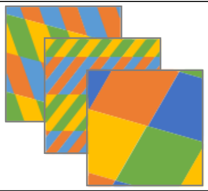


Figure 8: PSNR vs training iterations curve for the 2D toy experiment.

Table 5: Comparison of different hierarchical arrangements of weight grids for a network with 9 position-dependent linear layers and 2×2 weight tiles. The PE MLP baseline is also included in the first row. $\mathbf{I}$ denotes the 2×2 identity matrix.

	Visualization	$\mathbf{A}_i$	$\mathbf{b}_i$	PSNR(dB)
PE MLP		$0\mathbf{I}$	$\mathbf{0}$	31.38
Quad Tree		$2^i\mathbf{I}$	$\mathbf{0}$	40.55
Gray Code		$\begin{cases} 2\mathbf{I} & i = 1 \\ 2^{i-1}\mathbf{I} & i > 1 \end{cases}$	$\begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix}$	40.55
Fine to Coarse		$2^{10-i}\mathbf{I}$	$\mathbf{0}$	42.40
Random Permute		$2^{\text{arr}[i]}\mathbf{I}$, arr = [3,8,1,9,6,2,5,4,7]	$\mathbf{0}$	37.89
Random Affine		$\begin{bmatrix} a_i & b_i \\ c_i & d_i \end{bmatrix}$, $a_i, b_i, c_i, d_i \sim \mathcal{N}(0,\, 16^2)$	$\begin{bmatrix} m_i \\ n_i \end{bmatrix}$, $m_i, n_i \sim \mathcal{U}(0,\, 1)$	38.36

B Details for the Image Fitting Experiments

Architectures. For all the experiments, we use 8-layer networks with [512, 384, 256, 256, 256, 256, 256, 3] respective output channels. We use Leaky ReLU with a negative slope of 0.2 as the activation function between each layer. We use positional encoding with $L = 13$ frequencies as the input mapping for all the experiments. For the coordinate embedding baseline, we bilinearlly interpolate a learnable 128-channel 256×256 embedding and feed it to the first layer of the MLP, concatenated with the encoded input coordinates. For experiments using position-dependent weights, we use 4×4 weight tiles for the first 7 layers and use a regular linear layer for the last layer. The per-layer affine transformation coefficients for these experiments can be found in Table 6. They are defined in the same way as in Sec. A.

Table 6: Per-layer affine transformation coefficients of weight grids for image fitting experiments that use position-dependent weights. All the models have 7 position-dependent linear layers ($i = 1, \ldots, 7$) and use 4×4 weight tiles. The input 2D coordinates have a range of $\mathbf{p} \in [0, 1]^2$. † This prevents the grid pitch from becoming finer than the pixel pitch.

	$\mathbf{A}_i$	$\mathbf{b}_i$
Interleaved	$8192\mathbf{I}$	$\mathbf{0}$
Chunked	$4\mathbf{I}$	$\mathbf{0}$
Random Affine	$\begin{bmatrix} a_i & b_i \\ c_i & d_i \end{bmatrix}$, $a_i, b_i, c_i, d_i \sim \mathcal{N}(0,\, 256^2)$	$\begin{bmatrix} m_i \\ n_i \end{bmatrix}$, $m_i, n_i \sim \mathcal{U}(0,\, 1)$
Constant Input (no PE)	$\begin{cases} 4\mathbf{I} & i = 1 \\ (4^{i-1} \times 2)\mathbf{I} & i > 1 \end{cases}$ †	$\mathbf{0}$
Ours Bilinear	↑ (same as above)	$\mathbf{0}$
Ours	↑	$\mathbf{0}$

Training Details. We use Adam optimizer with $(\beta_1, \beta_2) = (0.9, 0.999)$ and a learning rate of 0.001. For each iteration, we randomly sample 262144 pixels from the image. We train the models for a total of 200k iterations. We decay the learning rate with a multiplier of 0.1 after 100k iterations. We follow the same network initialization scheme described in Sec. A.

Runtime & Hardware. All the experiments are run on a single NVIDIA Tesla V100 GPU (with power consumption capped at 163W). The training time is 8 hours for *PE*, 9 hours for *PE + CE*, 39 hours for *Ours Bilinear*, and 10 hours for the rest of the models.

Dataset. In this experiment, we use the public domain image of the dwarf planet Pluto[1] (NASA/Johns Hopkins University Applied Physics Laboratory/Southwest Research Institute/Alex Parker). The original image is 8000×8000 pixels. We resize it to 8192×8192 following the published implementation of [27].

Uncertainties of the Quantitative Results. We report uncertainties of the quantitative results in Table 7, which extends Table 1 in the main paper, with standard deviations over multiple runs included.

C Details for the Video Fitting Experiments

Architectures. For all the experiments, we use 6-layer networks with 256 hidden channels, with the exception of the SIREN-L experiment, which has 1024 hidden channels. We use Leaky ReLU

[1]`https://solarsystem.nasa.gov/resources/933/true-colors-of-pluto/`

Table 7: Quantitative results for the image fitting experiment. Standard deviations over multiple runs are parenthesized.

	PSNR dB (STD) ↑	SSIM (STD) ↑
PE MLP [32]	32.34 (0.02)	0.869 (6e-4)
PE + CE [3]	39.65 (0.21)	0.967 (1e-3)
Interleaved	33.80 (0.01)	0.876 (7e-5)
Chunked	43.13 (0.01)	0.980 (7e-5)
Random Affine	42.08 (0.78)	0.973 (5e-3)
Constant Input (no PE)	39.48 (0.06)	0.955 (4e-4)
Ours Bilinear	**44.85 (0.02)**	**0.985 (1e-4)**
Ours	**44.24 (0.17)**	**0.983 (4e-4)**

with a negative slope of 0.2 as the activation function between each layer. We use positional encoding with $L = \log_2 512 = 9$ frequencies as the input mapping for PE, PE + CE and LoE experiments. For the PE + CE baseline, similar to the image fitting experiment, we use a 64-channel $64 \times 64 \times 64$ embedding that is trilinearly interpolated and fed to the first layer.

For the LoE experiment, we use $2 \times 2 \times 2$ weight tiles for the first layer, $4 \times 4 \times 4$ tiles for the next 4 layers, and use a regular linear layer for the last layer. The grid resolution for the 5 position-dependent layers are 2, 8, 32, 128, and 512, respectively. Their corresponding affine coefficients are $\mathbf{A}_i$ = grid resolution $\times$ $\mathbf{I}$ and $\mathbf{b}_i = \mathbf{0}$. The 3D input coordinates are normalized to $\mathbf{p} \in [0, 1]^3$.

Training Details. We use Adam optimizer with $(\beta_1, \beta_2) = (0.9, 0.999)$. For PE, PE + CE, and LoE experiments, we use a learning rate of 0.001. For SIREN experiments, we use a learning rate of 5e-5 for stable training. For each iteration, we randomly sample 160000 pixels from the video. We train the models for a total of 200k iterations, decaying the learning rate with a multiplier of 0.1 after 100k iterations. For SIREN experiments, we follow the initialization scheme in [46]. For the rest of the experiments, we follow the same network initialization scheme described in Sec. A.

Runtime & Hardware. All the experiments are run on a single NVIDIA Tesla V100 GPU (with power consumption capped at 163W). The training of PE, PE + CE, and SIREN experiments require 3 hours, while the LoE model requires 4 hours due to the inefficient implementation of the dynamic weight layer. The SIREN-L experiment requires 26 hours of training.

Dataset. The original video is permissively licensed and can be found here[2]. We use the cropped and downsampled version from [46].

Uncertainties of the Quantitative Results. We additionally report the standard deviations of the quantitative results in Table 8, which corresponds to Table 2 in the main paper.

Additional Results. We include the result videos from all the methods in the supplemental material package.

D Details for the Novel View Synthesis Experiment

Architectures. We use an identical network architecture skeleton, as shown in Fig. 9, in all the experiments. We use Leaky ReLU with a negative slope of 0.2 as the activation function between each layer. We use a positional encoding with $L = 10$ frequencies as the input mapping for the coordinates ($\gamma(\mathbf{p})$). We follow the standard settings [32] and use a positional encoding with $L = 4$ for the ray directions ($\gamma_d(\mathbf{d})$).

[2]`https://www.pexels.com/video/the-full-facial-features-of-a-pet-cat-3040808/`

Table 8: Quantitative results of the video fitting experiment, with standard deviations reported in the parenthesis.

	PSNR dB (STD)
PE [32]	27.33 (0.01)
PE + CE [3]	35.83 (0.20)
SIREN [46]	29.13 (0.004)
SIREN-L	37.71 (0.01)
Ours	**39.98 (0.12)**

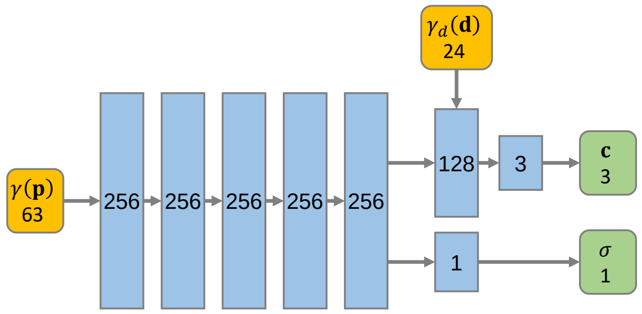


Figure 9: Network architecture for the novel view synthesis experiment. Numbers denote output channels. Yellow blocks denote inputs, while green blocks denote outputs.

For the LoE experiment as well as the *No Hierarchy* ablation, we replace the first 5 layers with position-dependent linear layers and use $4 \times 4 \times 4$ weight tiles in these layers. The grid layouts are shown below in Table 9. For all the experiments, we use identical network architectures for both coarse and fine networks.

Table 9: Affine transformation coefficients for novel view synthesis experiments.

	$\mathbf{A}_i$	$\mathbf{b}_i$
No Hier.	$64\mathbf{I}$	$\mathbf{0}$
Ours	$4^i\mathbf{I}$	$\mathbf{0}$

Training Details. We use Adam optimizer with $(\beta_1, \beta_2) = (0.9, 0.999)$. We use an initial learning rate of 5e-5, and decay it exponentially following the published implementation of [32]. We follow the same network initialization scheme described in Sec. A. For each iteration, we randomly sample 16384 rays. For each ray, we sample 96 coarse samples and 192 important samples. Each model is trained for a total of 500k iterations.

Runtime & Hardware. All the models are trained using $8\times$ NVIDIA A40 GPUs. The LoE model, and the ablation model requires 26 hours of training, while the PE MLP baseline requires 19 hours.

Dataset. We use the preprocessed version of Tanks and Temples dataset [23, 26] for training, which includes 133 training images and 19 test images with a resolution of 1920×1080. The dataset is CC-NC licensed. We isotropically scale the scene so that everything is bounded within $[-0.5, 0.5]^3$ in the world coordinate.

Uncertainties of the Quantitative Results. We additionally report the standard deviations of the quantitative results in Table 10, which corresponds to Table 3 in the main paper.

Additional Results. We also include result images evaluated from test views in the supplemental material package.

Table 10: Quantitative results for the novel view synthesis experiment. Standard deviations are reported in the parenthesis.

	PSNR dB (STD) ↑	SSIM (STD) ↑
PE MLP [32]	30.50 (0.09)	0.900 (1e-3)
No Hier.	27.73 (0.66)	0.861 (0.012)
Ours	**31.46 (0.04)**	**0.936 (1e-3)**

E Details for the Image Generation Experiment

Architectures. In the image generation experiment, we adopt a linearly arranged network with residual connections, as shown in Fig. 10. We use a positional encoding with 8 frequencies as the input mapping, and apply Leaky ReLU to the activations. For the *PE + CE* baseline, we use a 256-channel 256×256 embedding. For the LoE experiment, we use position-dependent weights on the first linear layer of 7 intermediate residual blocks, with the grid resolutions and tile sizes marked in the figure. The definition of these parameters follows Sec. C. The above setting yields approximately the same parameter counts and computational costs, enabling fair comparison. This network architecture is inspired by the "residual" setting used in [3] but greatly simplified and scaled down to accelerate the experiments.

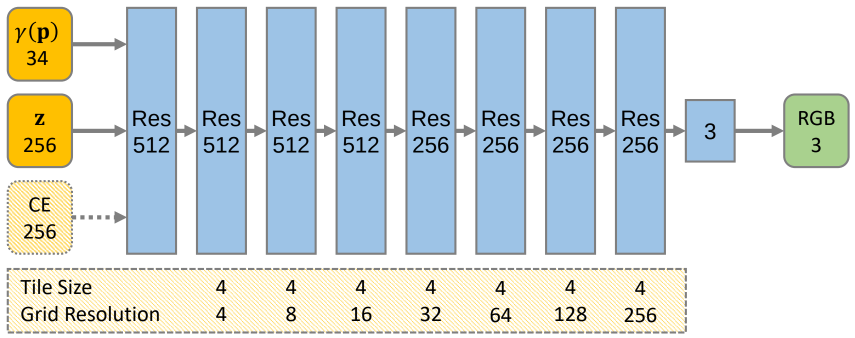


Figure 10: Network architecture for the GAN experiment. Blocks labeled with "Res" are residual blocks. Numbers on the blocks denote output channels. The coordinate embedding (CE) input is used only for the CE experiment, while the position-dependent weight tiling, with tile sizes and grid resolutions noted under each block, is only used in the LoE experiment.

For the discriminator network, we use simple 5-layer patch discriminators [20] constructed with kernel size 3 and stride 2 convolution layers and Leaky ReLU activation functions. We use two such discriminators at two different scales: full image and 1/2 downsampled image. We use spectral normalization [33] on the discriminator weights.

We use hinge loss [24] as the GAN objective. The overall losses for the discriminator and the generator are as follows:

$$L_D = -\mathbb{E}_{\mathbf{x}\sim p_{data}}\left[\min\left(0, -1 + D\left(\mathbf{x}\right)\right)\right] - \mathbb{E}_{\mathbf{z}\sim p_{\mathbf{z}}}\left[\min\left(0, -1 - D\left(G\left(\mathbf{z}\right)\right)\right)\right] \tag{9}$$

$$L_G = -\mathbb{E}_{\mathbf{z}\sim p_{\mathbf{z}}} D\left(G\left(\mathbf{z}\right)\right) \tag{10}$$

Table 11: Image generation quality on FFHQ dataset, with standard deviations over two runs provided in the parenthesis.

	FID (STD)
PE + CE [3]	23.5 (0.7)
Ours	**18.3 (1.6)**

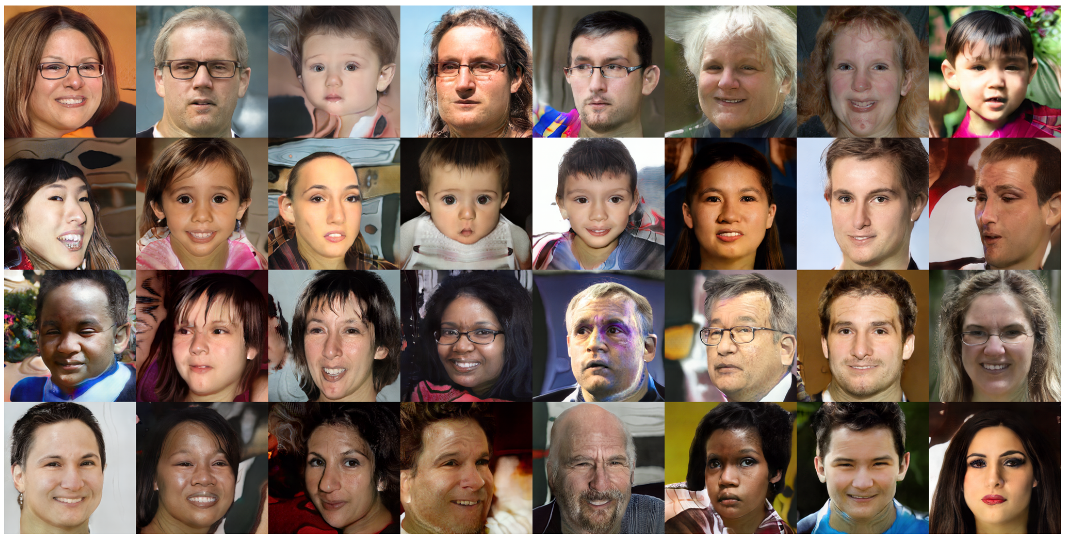
Training Details. We use Adam optimizer with $(\beta_1, \beta_2) = (0.9, 0.999)$ and a learning rate of 1e-4. We follow the same network initialization scheme described in Sec. A for the generators and discriminators. We use a batch size of 4 images per GPU, yielding a combined batch size of 32 images per iteration. Each model is trained for 1M iterations.

Runtime & Hardware. Both models require approximately 60 hours of training on $8\times$ NVIDIA A100 GPUs.

Dataset & Evaluation Metrics. The models are trained on the Flickr Faces-HQ (FFHQ) dataset [22], which contains permissively licensed images from Flickr that were intended for free use and redistribution by their respective authors. In our experiments, we downsample the images from 1024×1024 to 256×256. We report FID scores that are evaluated with a sample size of 50k images.

Uncertainties of the Quantitative Results. We additionally report the standard deviations of the FID scores in Table 11, which corresponds to Table 4 in the main paper.

Additional Results. We include additional examples of generated images in Fig. 11.


(a) Levels-of-Experts

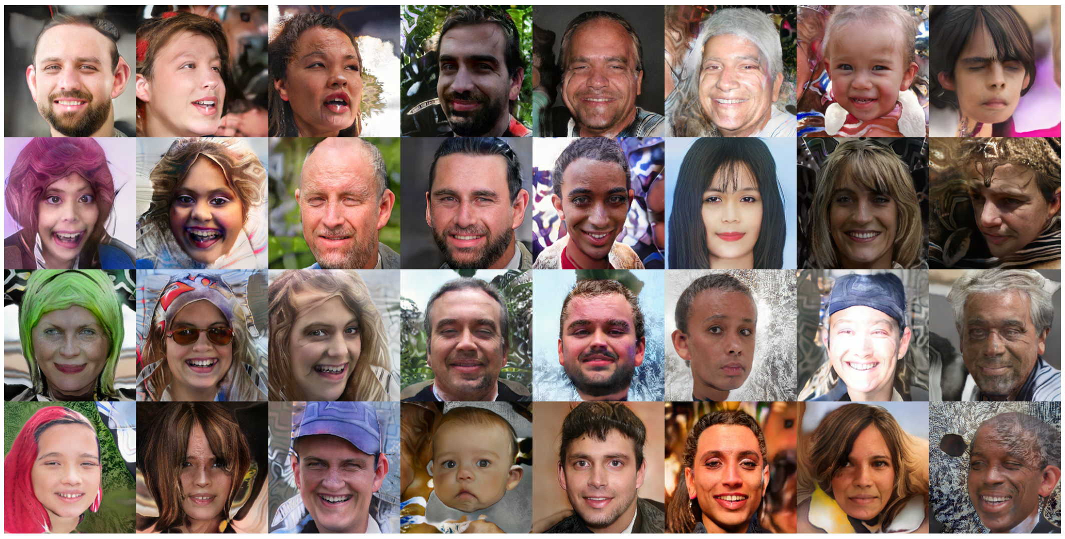

(b) Coordinate Embedding

Figure 11: Additional image generation results from our method (a) and baseline method (b).