A. Additional implementation details

We build on top of the StyleGAN2 framework [40] and change only its generator. All other settings, including the discriminator architecture D, optimizers, losses, training settings and other hyperparameters are kept untouched. This means that we use non-saturating logistic loss for training [23] and use Adam optimizers with the parameters $\beta_1 = 0.0, \beta_2 = 0.98$ and $\epsilon = 1e-8$. Our D is a small version of StyleGAN2 discriminator (i.e. config-e from the paper [40]), regularized with zero-centered R_1 gradient penalty [54] with weight $\gamma = 10$. We apply the regularization on each iteration instead of using the lazy setup, as done by StyleGAN2[40] who applies it only each 16-th iteration. We use the learning rate of 0.00001 for G, 0.0005 for the shared parameters of an INR (which is a part of G) and 0.003 for D. We also employ skip-connections for coordinates inside each multi-scale INR block [37]. We apply them by concatenating the coordinates to inner representations.

For the main experiments, we didn't employ any ProGAN [38], StyleGAN [39] or StyleGAN2 [40] training tricks, like path regularization, progressive growing, equalized learning rate, noise injection, pixel normalization, style mixing, etc. For our additional ablations, we reimplemented our INR-based decoder on top of the StyleGAN2's generator and employed style mixing, equalized learning rate and pixel normalization for it. In terms of implementation, it was equivalent to replacing StyleGAN2's weight modulation-demodulation with our FMM mechanism, reducing kernel size from 3 to 1, concatenating coordinates information at each block and replacing its upfirdn2d upsampling with the nearest neighbour one. We found that for the nearest neighbour upsampling, the model learns to ignore spatial noise injection (by setting noise strengths to zero), because pixels cannot communicate the noise information between each other, making it meaningless and harmful to the generation process.

For our 256×256 experiments (for both LSUN and FFHQ), we use 2 multi-scale INR blocks of resolutions 128 and 256. Each block contains 4 layers of 512 dimensions each. For FFHQ 1024×1024, we used 4 multi-scale INR blocks of resolutions 128, 256, 512 and 1024. First 2 blocks had 3 layers of dimensionality of 512, 512 resolution had 2 layers of resolution 128, the final block had 2 layers of dimensionality of 32.

Our G architecture is the same for all the experiments and consists on 3 non-linear layers with the hidden dimension of 1024 and residual connections. Our noise vector $\boldsymbol{z}$ has the dimensionality of 512.

For additional implementation details, we refer a reader to the accompanying source code.

In all the experiments, we compute FID scores based on 50k images using the tensorflow script from BigGAN pytorch repo[1].

B. Experiments details

B.1. Zooming out

On Fig. 9 we present additional examples of our zooming operation, but evaluating the model on $[-1.5, 1.5]^2$ grid instead of $[-0.3, 1.3]^2$ like we did for Fig. 2 in the main body. This is done to demonstrate the extent to which the model can extrapolate.

B.2. Keypoints prediction

As being said, we train a linear model to predict the keypoints from the latent codes. To train such a model, we first generate $n = 10^4$ latent codes $\boldsymbol{w}_1, ..., \boldsymbol{w}_n$ for each model, then we decode them into images $\boldsymbol{x}_1, ..., \boldsymbol{x}_n$. After that, we predict keypoints vector $\boldsymbol{y}_i$ for each image with Super-FAN model [6]. Then we fit a linear regression model to predict $\boldsymbol{y}_i$ from $\boldsymbol{w}_i$.

Measuring quality based on the synthesized images may be unfair since the variability in keypoints of each model can be different: imagine a generator that always produces a face with the same keypoints. This is why we compute the test quality by embedding real FFHQ images into each model. But to additionally demonstrate that the variability is equal for the both models, we fit a linear regression model on *randomly permuted* latent codes and check its score: if the prediction accuracy is high, than the variability in keypoints is low and the prediction task is much easier. We call this metric KPL (random) and depict the corresponding values on Table 3. It clearly demonstrates that the both models have equal variability in terms of keypoints.

We project FFHQ images into a latent space using the latent space projection procedure from the official repo[2]. We used default hyperparameters except for it, except that we didn't optimize the injected noise since this would take away some of the information that is better to be stored in the latent code. We depict additional qualitative results for random samples of FFHQ on Fig. 10.

[1] https://github.com/ajbrock/BigGAN-PyTorch/blob/master/inception_tf13.py

[2] https://github.com/NVlabs/stylegan2/blob/master/projector.py

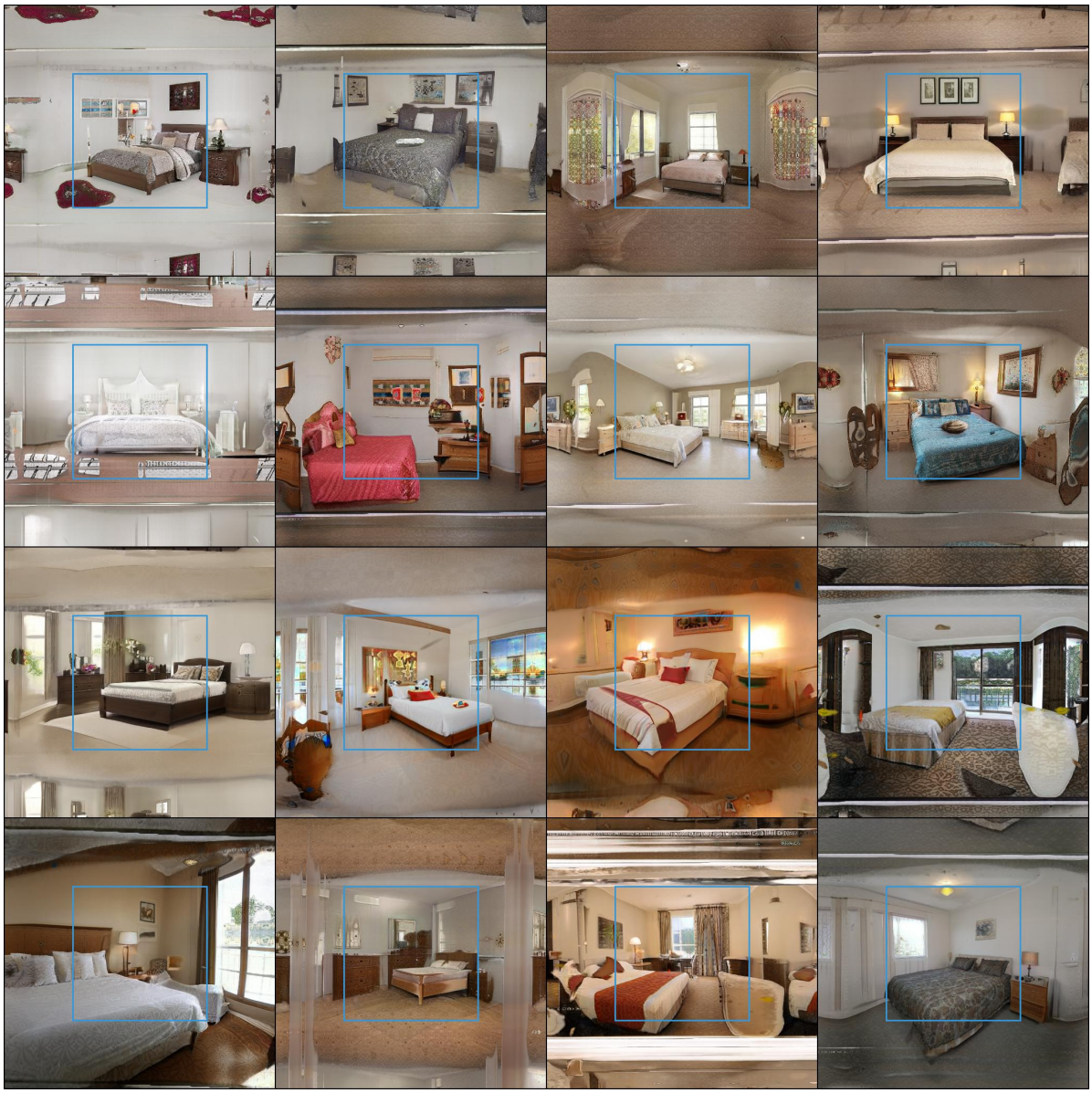

Figure 9: After training our INR-based GAN on LSUN 256×256 dataset, we feed larger coordinates grid $[-0.5, 1.5]^2$ into it. This is larger than on Figure 2 to demonstrate the extent to which the model extrapolates. As one can see, extending the grid outside of $[-0.4, 1.4]^2$ makes the quality to decrease rapidly.

We provide the algorithm on KPL computation in Algorithm 1. We use $N_{\text{tr}} = 10^4$ and $N_{\text{ts}} = 256$.

B.3. Additional samples

On Fig. 12, we present additional samples with the truncation factor of 0.9 from our INR-based model trained on FFHQ1024. We perform the truncation in similar nature to StyleGAN2 [40] by linearly interpolating an inner representation inside G to its averaged value. On Fig. 13, we present common artifacts found in the produced images. On Fig. 14, we present additional superresolution samples from our model trained on LSUN 128^2. On Fig. 15, we present additional uncurated samples of our model trained on LSUN bedroom 256^2.

(a) StyleGAN2 generator

(b) Our INR-based generator

Figure 10: Predicting keypoints from latent codes for random FFHQ images. The corresponding scores are presented in Table 3.

Algorithm 1: Compute Keypoints Prediction Loss (KPL).

Input : Keypoints extractor $\mathcal{K} : \boldsymbol{x} \mapsto \boldsymbol{k} \in \mathbb{R}^{d_k}$.
Input : Generator model $\mathsf{G} : \boldsymbol{w} \mapsto \boldsymbol{x}$
Input : Embedding procedure $\mathcal{E} : \boldsymbol{x} \mapsto \boldsymbol{w}$.
Input : Collection of real face images $X_{\text{test}} = \{\boldsymbol{x}_i\}_{i=1}^{n}$ of size N_{ts}.
Output: KPL score $s \in [0, +\infty)$.
Generate N_{tr} latent codes $W_{\text{train}} = \{\boldsymbol{w}_1, ..., \boldsymbol{w}_{N_{\text{tr}}}\}$;
Generate a dataset of synthetic images $X_{\text{train}} = \{\mathsf{G}(\boldsymbol{w}_i) | \boldsymbol{w}_i \in W_{\text{train}}\}$;
Extract keypoints $K_{\text{train}} = \{\mathcal{K}(\boldsymbol{x}_i) \mid \boldsymbol{x}_i \in X_{\text{train}}\}$ and $K_{\text{test}} = \{\mathcal{K}(\boldsymbol{x}_j) \mid \boldsymbol{x}_j \in X_{\text{test}}\}$;
Embed real images $W_{\text{test}} = \{\mathcal{E}(\boldsymbol{x}_j) \mid \boldsymbol{x}_j \in X_{\text{test}}\}$;
Train a linear keypoints estimator $(\boldsymbol{A}^*, \boldsymbol{b}^*) = \arg\min_{\boldsymbol{A},\boldsymbol{b}} \sum_{i=1}^{N_{\text{tr}}} \|(\boldsymbol{A}\boldsymbol{w}_i + \boldsymbol{b}) - \boldsymbol{k}_i\|_2^2$;
Evaluate its performance on the test set: $s = \sum_{i=1}^{N_{\text{ts}}} \|(\boldsymbol{A}\boldsymbol{w}_j + \boldsymbol{b}) - \boldsymbol{k}_j\|_2^2$;
Return s;

C. Positional encoding of coordinates

Recent works [74, 81] demonstrate that using positional embeddings [84] like Fourier features greatly increases the expressivity of a model, allowing to fit more complex data. Our positional encoding of coordinates follows [81] design and consists on a linear matrix $\boldsymbol{W} \in \mathbb{R}^{n \times 2}$ applied to raw coordinates vector $\boldsymbol{p} = (x, y)$ and followed by sine/cosine non-linearities and concatenated:

(a) INR-GAN latent space projections.

(b) StyleGAN2 latent space projections.

Figure 11: Projection results by projection images from 10. We use the original StyleGAN2's projection procedure [40] to project FFHQ dataset images into the latent space of a generator. All low-frequency details, together with the keypoints are reconstructed well. In our case, the reconstruction quality is lower because we do not optimize for spatial noise as StyleGAN2 does because our vanilla INR-GAN architecture does not use spatial noise injection.

$$\boldsymbol{e}(\boldsymbol{p}) = \begin{bmatrix} \sin(\boldsymbol{W}\boldsymbol{p}) \\ \cos(\boldsymbol{W}\boldsymbol{p})^{\top} \end{bmatrix} \tag{1}$$

Matrix $\boldsymbol{W}$ is produced by our generator G without any factorization since it has only 2 columns. Each row of this matrix corresponds to the parameters of the Fourier transform. The norm of a row corresponds to the frequency of the corresponding wave. We depict frequencies distribution learned by our generator on Figure 16.

D. Geometric prior

Adding coordinates to network input induces powerful prior on the geometric shapes, since now different pixels, otherwise created equal are ordered through the euclidean (or any other) coordinate system. It is a well-know fact that complex geometric shapes could be compactly represented with the use of euclidean coordinates (for example one can write down an ellipse equation as $\frac{(x-x_0)^2}{a^2} + \frac{(y-y_0)^2}{b^2} = 1$). On the other hand without any form of prior one would hope to fit complex patterns with dedicated filters which would potentially consume much more parameters.

It is worthy to discuss the synergy between the coordinate representations and hypernetworks. Lets imagine a learnable system consisting of sequentially connected linear layer $W \in \mathbb{R}^{2\times2}$, which is modulated by a hypernetwork, and INR, taking Euclidean coordinates as an input $f(X), X \in \mathbb{R}^{2\times1}$ The whole model then could be written as $f(W(z) \times X)$. Now, it could be seen, that introducing the hypernetwork to the pipeline allows to apply linear transformation to the coordinates, rotating and zooming the image encoded by INR f. Thus, hypernetworks allow to easily perform transformations non-trivial for traditional deep learning systems.

Figure 12: Random (uncurated) samples from our model trained on FFHQ 1024×1024 dataset with the truncation factor of 0.9. FID: 16.32

Figure 13: Common artifacts found in our model's samples when sampling without truncation. As one can see, the most severe ones are "stains" and patterned texture.

Finer control over form and placement Recent studies show that convolutional NN struggle with such simple and crucial tasks as accurately predicting the coordinates of a drawn point [52] (and vice versa, drawing a point given the coordinates). One should expect, that such an important skill is necessary for the generative model for accurate placement of the different object parts and for precise representation of the object proportions.

E. FMM as a generalizaton of the common weight modulation schemes

In this section we show that Factorized Matrix Multiplication could be seen as a general framework for weight modulation, with Squeeze-and-Excitation, AdaIN and "vanilla" hypernetworks as its particular cases.

Squeeze-and-Excitation Let us look at the l-th FMM layer of our network with the effective rank of 1. In this case $\boldsymbol{A}^l$ and $\boldsymbol{B}^l$ are matrices (actually vectors) of the sizes $n_{in} \times 1$ and $1 \times n_{out}$ respectively. Thus, following the rules of matrix multiplication, we get that $\boldsymbol{W}^l_{h\,i,j} = \boldsymbol{A}^l_i \cdot \boldsymbol{B}^l_j$. On the other hand, lets look at the Squeeze-and-Excitation mechanism. Here we are modulating the output of the each neuron by multiplicating it by the predicted coefficient, which is equivalent to the multiplication of the corresponding weight matrix column by this coefficient. Using our notions and denoting the pre-activation (before non-linearity) vector of modulation coefficients as $\boldsymbol{A}$ we get that in this case $\boldsymbol{W}^l_{h\,i,j} = \boldsymbol{A}^l_i$. While at the

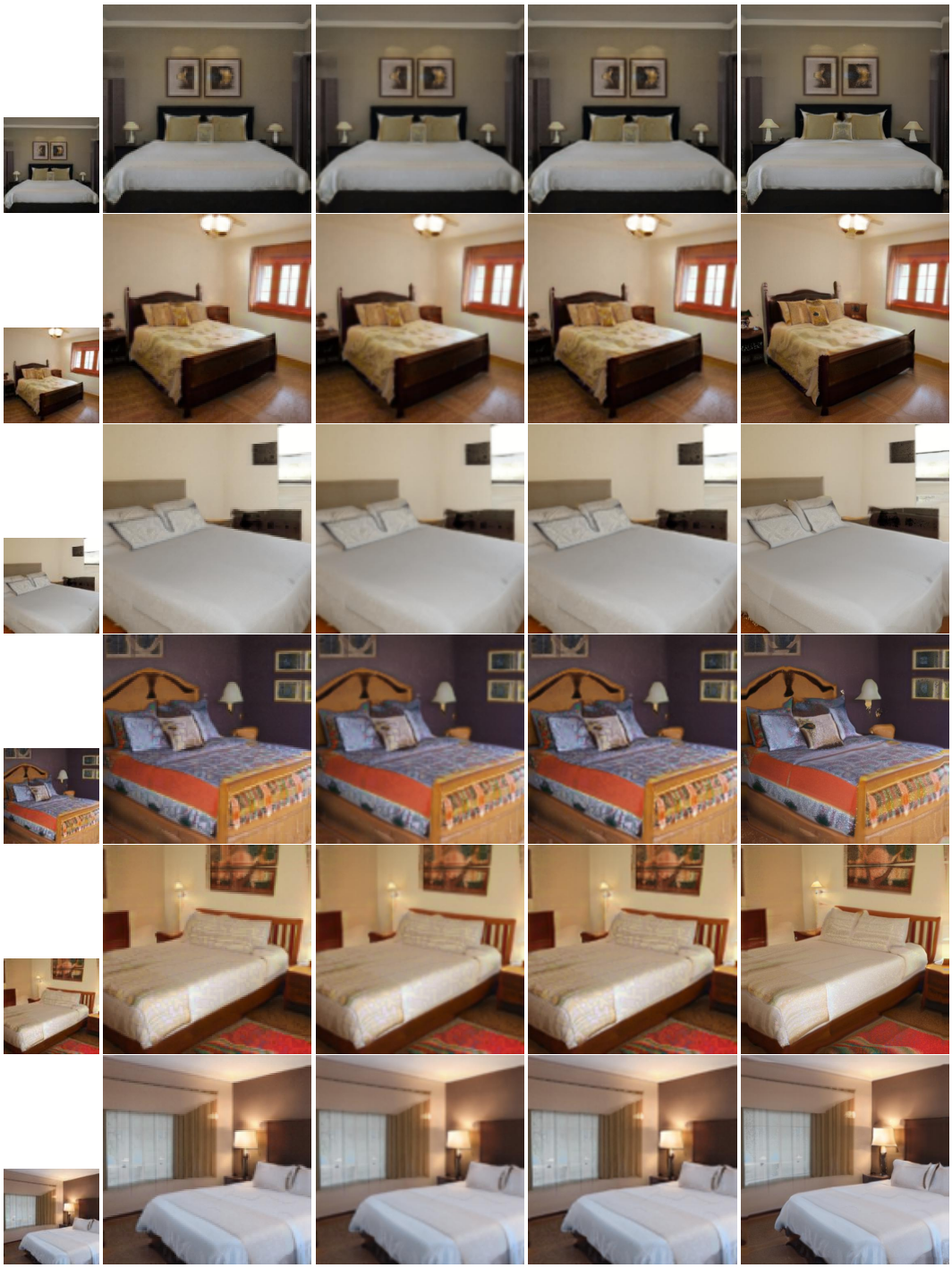

Figure 14: Additional samples from our model to show superresolution properties. We trained the model on LSUN 128×128 and upsampled to 256×256.

first glance it looks like our model is more expressive lets not forget about the fact that the next layer is by itself modulated with its own Squeeze-and-Excitation, from which (omitting relu non-linearity and the fact that it has its own sigmoid) we can get the $\boldsymbol{B}_j^l$ multiplier. Thus it could be seen that squeeze-and-excitation modulation is roughly equivalent to the FMM layer

Figure 15: Random samples of our model on LSUN bedroom 256^2 dataset. FID: 6.27.

with the rank of 1. The same reasoning is applicable to the AdaIN case, though, with AdaIN obviously we have the additional normalization layer and do not have sigmoid non-linearity for the style vector which influences the learning dynamics in its own way. We can say that squeeze-and-excitation is the least powerful weight modulation scheme, which uses the matrix of the rank 1 to modulate the main shared weights, on the other hand it is cheap and simple.

Hypernetworks Vanilla hypernetwork is perhaps the most straightforward (and the most expensive but flexible) approach to the weight modulation. It is as simple as predicting each weight as an output of MLP. So let's demonstrate that any weight

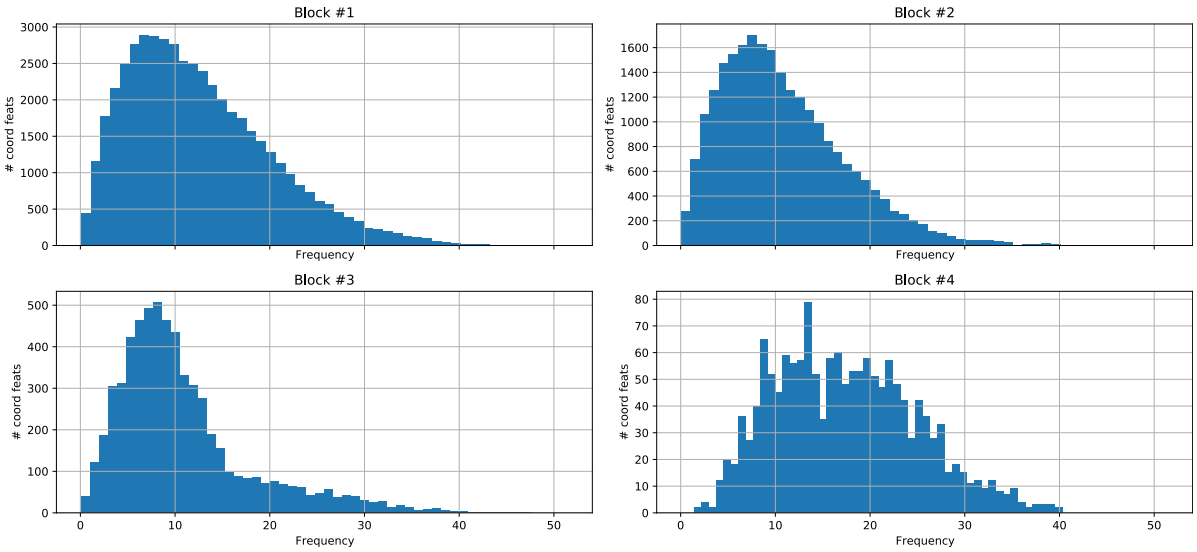

Figure 16: Frequencies distributions for different multi-scale INR blocks of our INR-based GAN trained on FFHQ 1024^2. To produce the plot, we sample 128 images in an INR-based form and computed the norms of the positional encoding layers, i.e. those layers which take raw coordinates as an input. As one can see, the model tries to use more high-frequent positional embeddings for the last layer since they are more important for drawing fine-grained details. Early layers determine the structure of an image and hence use smaller frequencies to operate on a larger scale.

dynamic that can be modeled by hypernetwork could be fitted with the FMM of high enough rank. Let's assume that n_{in} is larger than n_{out} (which is our case, but not essential for the generality of the proof) and choose the FMM rank of n_{in}. In this case $\boldsymbol{A}^l$ and $\boldsymbol{B}^l$ are matrices of the sizes $n_{in} \times n_{in}$ and $n_{in} \times n_{out}$ respectively. Since A is a square matrix we can set it to identity constant (which is a solution easily learnt by a NN just by setting bias) and get $\boldsymbol{W}^l_{h\,i,j} = \boldsymbol{B}^l_{i,j}$. In this case any hypernetwork could be "simulated" with FMM by fitting $\boldsymbol{B}^l_{i,j} = \sigma^{-1}(\frac{\hat{\boldsymbol{W}}^l_{i,j}}{\boldsymbol{W}^l_{s\,i,j}})$, where $\hat{\boldsymbol{W}}$ denotes the weight matrix predicted by the hypernetwork. While σ^{-1} definitely imposes some restrictions, caused by the positiveness and the range, in our experiments adding sigmoid has not resulted in any harm, perhaps because of the flexible calibration of the $\boldsymbol{W}^l_s$.

We have shown that main weight modulation schemes could be seen as the boundary particular cases of our approach. Our approach to the weight modulation is somewhere in between of these two extremes, while reaping the benefits of the both of them. Studying the behaviour of this transmission is specially important for the shading light on the weight modulation at whole and going beyond straightforward approaches.

F. Performance on multi-class datasets

In this section, we conduct experiments on two diverse datasets to demonstrate that our proposed architectural design improves performance in this scenario is well. For this, we employ two datasets: LSUN-10 256^2 and MiniImageNet-100 128^2. LSUN-10 consists on 1M images of 10 LSUN scenes, where we take 100k images of each scene. MiniImageNet-100 consists on 100k images of 100 ImageNet classes[3], where each class provides 1k images. We report the results for different models in Table 4. They demonstrate that our proposed architectural design improves the performance for this setup as well.

G. Accelerated inference of low-resolution images

Several classifiers can produce a prediction faster when an input is easy to classify [82, 32]. A generative model should have a similar property: when we ask a model to generate an image of lower resolution than the model was trained on — it should be able to do it faster. However, traditional convolutional decoders lack this property: to produce a lower-resolution image, one would need to perform the full inference and then downsample the resulted image with standard interpolation techniques. In contrast, INR-based decoders are capable of doing this naturally: for this, we should just evaluate them on

[3] https://github.com/yaoyao-liu/mini-imagenet-tools

Table 4: FID & IS at 300k iterations on multi-class datasets.

Decoder type	LSUN-10		MiniImageNet	
	FID ↓	IS ↑	FID ↓	IS ↑
Basic INR decoder	216.8	1.0	271.5	1.03
+ Hypernetwork-based decoder	OOM	OOM	112.9	8.76
+ Fourier embeddings	OOM	OOM	102.8	9.85
+ FMM	23.78	2.48	84.66	9.32
+ Multi-scale INR	12.47	3.02	59.63	11.29
StyleGAN2	8.99	3.18	52.94	12.32
Validation set	0.42	9.93	0.39	61.79

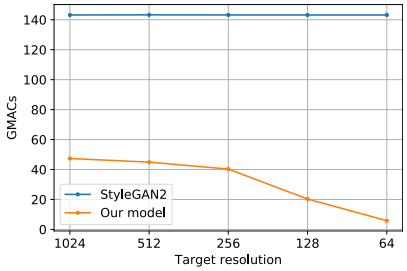

Figure 17: **Accelerated low-resolution image generation**. We measure a decoder's efficiency in terms of #MACs on generating an image of lower resolution compared to what it has been trained on. Since INR can do this by evaluating on a sparser grid, this allows it to save a lot of computation. Traditional convolutional decoders require performing a full inference first and then downsampling the produced image.

a sparser grid compared to what they were trained on. To state the claim rigorously, we compute an amount of multiply-accumulate (MAC) operations for our INR-based generator and StyleGAN2 generator trained on FFHQ 1024^2 for different lower-resolution image sizes. To produce the low-resolution image with StyleGAN2 we first produce the full-resolution image and them downsample it with nearest neighbour interpolation. For our model, we just evaluate it on a grid of the given resolution. The results are reported on Fig. 17. To the best of our knowledge, our work is the first one that explores the accelerated generation of lower-resolution images — an analog of early-exit strategies [82] for classifier models for image generation task.