DiffFNO: Diffusion Fourier Neural Operator

Supplementary Material

1. Time Embedding

As described in the main paper (Section 3.5), we incorporate a sinusoidal time embedding $\mathbf{e}(t)$ to effectively condition our neural network on the diffusion time step t during both training and inference. Here, we provide additional details of its implementation.

To effectively condition our neural network on the diffusion time step t during both training and inference, we incorporate a sinusoidal time embedding $\mathbf{e}(t)$, inspired by positional encodings used in transformer architectures [22]. This embedding captures temporal information across multiple frequencies, enabling the network to learn temporal dynamics at different scales. $\mathbf{e}(t)$ is defined as:

$$\mathbf{e}(t) = \left[\sin\left(\omega_1 t\right), \cos\left(\omega_1 t\right), \ldots, \sin\left(\omega_D t\right), \cos\left(\omega_D t\right)\right], \tag{1}$$

where D is the dimensionality of the embedding (we set $D = 64$), and the frequencies ω_i are determined by an exponentially increasing schedule:

$$\omega_i = \omega_{\min} \cdot \left(\frac{\omega_{\max}}{\omega_{\min}}\right)^{\frac{i-1}{D-1}}, \quad i = 1, 2, \ldots, D, \tag{2}$$

with $\omega_{\min} = 1$ and $\omega_{\max} = 10{,}000$. This configuration ensures coverage of a wide range of frequencies, allowing the network to capture both rapid and slow temporal variations.

$\mathbf{e}(t)$ is concatenated with the encoded image features $\mathbf{v}$ obtained from the convolutional encoder. This enriched feature representation is then passed to both Weighted Fourier Neural Operator (WFNO) and Attention-based Neural Operator (AttnNO), as introduced in the main paper (Section 3.3), ensuring that temporal information is available in subsequent processing stages.

By conditioning on the time step t, the network can adapt its processing to different stages of the diffusion process. Early in the reverse diffusion (at higher t), the network focuses on reconstructing coarse structures from noisy inputs, while at later stages (lower t), it refines fine details. The sinusoidal time embedding facilitates this adaptation by providing a rich temporal context, enhancing the network's ability to model the evolution of image features over time.

This approach is consistent with recent practices in diffusion models [10, 17], where time embeddings play a crucial role in guiding the denoising process. Incorporating the time embedding directly into the network's input features allows DiffFNO to effectively leverage temporal information, contributing to its superior performance in arbitrary-scale super-resolution tasks.

2. Implementation Details

In this section, we provide implementation details of DiffFNO discussed in the main paper, including data preparation, network architecture specifics, training protocols, and computational resources.

Data Preparation and Augmentation. We train our model using the DIV2K dataset [1], which contains 800 high-resolution (HR) images for training and 100 images for validation. To generate training pairs, we extract random HR patches of size 128×128 from the original images. The low-resolution (LR) counterparts are obtained by downsampling these patches using bicubic interpolation with scaling factors uniformly sampled from $\times 1$ to $\times 4$. Specifically, for each HR patch, we randomly select a scaling factor $s \in [1, 4]$ and down-scale the HR patch to size $\lfloor \frac{128}{s} \rfloor \times \lfloor \frac{128}{s} \rfloor$, ensuring that the LR patches align precisely with the HR patches when upscaled.

To enhance the robustness and generalization capability of our model, we apply extensive data-augmentation techniques. These include random horizontal and vertical flips, random rotations by $90°$, $180°$, and $270°$, and random cropping within the HR patches. In addition, we normalize the input LR images by scaling the pixel values to the range $[0, 1]$. We use a batch size of 64 for all experiments.

Encoder (Lifting Layer). We employ two variants of the encoder in our experiments: the baseline Enhanced Deep Super-Resolution (EDSR) network [15] and the Residual Dense Network (RDN) [25]. The encoder is responsible for extracting rich feature representations from the LR input. For EDSR, we use 16 residual blocks, each consisting of two 3×3 convolutional layers with ReLU activations, without batch normalization, as recommended for SR tasks [15]. For RDN, we use a configuration with 16 residual dense blocks, each containing 8 convolutional layers with a growth rate of 64. The final feature map has 64 channels.

Weighted Fourier Neural Operator (WFNO). Our WFNO, as introduced in the main paper (Section 3.2), consists of 8 Fourier layers, each with 64 channels. Unlike the standard FNO, we retain all Fourier modes and apply our Mode Rebalancing to adaptively weight each frequency component. The Fourier transforms are implemented using the Fast Fourier Transform (FFT), and the inverse transforms are implemented using the Inverse FFT (IFFT). To handle complex-valued operations, we separate the real and imaginary parts throughout the network. The spectral convolution operations intrinsically upsample the features from LR to HR spatial dimensions, enabling the network to capture global dependencies across the entire image.

Attention-based Neural Operator (AttnNO). The AttnNO is designed with 8 layers and 64 channels, employing single-head self-attention mechanisms similar to those in [22]. Each layer consists of a self-attention module followed by a feed-forward network with a 3×3 convolutional layer and GELU activation [9]. The attention mechanism allows the network to model local interactions and fine-grained details effectively. Features are upsampled to HR dimensions via bilinear interpolation before being processed by the attention layers.

Gated Fusion Mechanism (GFM). To integrate the features from WFNO and AttnNO, we employ Gated Fusion Mechanism, a novel gating mechanism as described in the main paper (Section 3.3). Specifically, we concatenate the feature maps along the channel dimension and pass them through a 1×1 convolutional layer with 64 output channels, followed by a sigmoid activation function to produce the gating weights. These weights are then used to perform an element-wise weighted summation of the two feature maps. This mechanism allows the network to adaptively balance the contributions of global and local features.

2.1. Adaptive Time-Step (ATS) ODE Solver

Our ATS, as described in the main paper (Section 3.5), employs the Runge-Kutta 4th-order (RK4) method with fixed step sizes determined by our adaptive time-step selection strategy in the main text). Unlike adaptive methods such as Runge-Kutta-Fehlberg (RK45), which adjust step sizes based on local error estimation, our approach uses a precomputed sequence of time steps $\{t_i\}_{i=0}^{N}$ derived from the learned function $\phi_\psi(t)$. This allows us to efficiently allocate computational resources without the computational overhead of error estimation.

The general update equation for advancing from time t_i to t_{i+1} is given by:

$$\mathbf{x}_{i+1} = \mathbf{x}_i + h_i \cdot f_\theta(\mathbf{x}_i, t_i), \quad h_i = t_{i+1} - t_i, \tag{3}$$

where $\mathbf{x}_i$ is the estimate of the image at time t_i, h_i is the step size from the adaptive schedule, and $f_\theta(\mathbf{x}_i, t_i)$ is the approximate drift function.

To achieve higher accuracy, we use the RK4 method:

$$k_1 = f_\theta(\mathbf{x}_i, t_i), \tag{4}$$

$$k_2 = f_\theta\left(\mathbf{x}_i + \frac{h_i}{2}k_1,\, t_i + \frac{h_i}{2}\right), \tag{5}$$

$$k_3 = f_\theta\left(\mathbf{x}_i + \frac{h_i}{2}k_2,\, t_i + \frac{h_i}{2}\right), \tag{6}$$

$$k_4 = f_\theta\left(\mathbf{x}_i + h_i k_3,\, t_i + h_i\right), \tag{7}$$

$$\mathbf{x}_{i+1} = \mathbf{x}_i + \frac{h_i}{6}(k_1 + 2k_2 + 2k_3 + k_4). \tag{8}$$

where k_1, k_2, k_3, and k_4 are intermediate evaluations of the drift function. The RK4 method effectively combines these to compute a more accurate estimate of $\mathbf{x}_{i+1}$.

The benefits of RK4 in Our Context is threefold: (i) Improved Accuracy: The higher-order integration reduces truncation errors, crucial for maintaining image quality in super-resolution tasks. (ii) Stability: RK4 is more stable than lower-order methods with larger step sizes, allowing us to use fewer steps without sacrificing performance. (iii) Efficiency: Although RK4 requires multiple function evaluations per step, the total number of steps N is reduced, leading to faster inference times.

By integrating the ODE with RK4 and using the adaptive time steps from our learned schedule, our ATS efficiently solves the reverse-time diffusion process, achieving high-quality reconstructions with lower computational cost.

Training Protocol. We train our model using the Adam optimizer [13] with $\beta_1 = 0.9$ and $\beta_2 = 0.999$. To stabilize the initial training phase, we employ a learning rate warm-up strategy, gradually increasing the learning rate from 1×10^{-6} to 3×10^{-4} over the first 5 epochs [7]. Subsequently, we use a cosine annealing learning rate scheduler with warm restarts [16], resetting every 50 epochs. The learning rate decays from 3×10^{-4} to 1×10^{-6} following a cosine curve in each cycle. We trained for $1,000$ epochs.

To prevent overfitting, we incorporate an early-stopping mechanism based on the validation loss, with a patience of 50 epochs. Additionally, we apply weight decay regularization with a coefficient of 1×10^{-6}. The loss function is the mean squared error (MSE) between the predicted and true score functions in the diffusion process.

Computational Resources. Training is conducted on a Linux server equipped with an NVIDIA A100 GPU with 40 GB of memory. To maximize computational efficiency, we utilize mixed-precision training with automatic loss scaling provided by PyTorch's AMP module [21]. This allows us to train larger models and use larger batch sizes without exceeding memory constraints.

Implementation Specifics. Our model is implemented in PyTorch [20]. We employ 16 worker threads for data loading to ensure efficient utilization of GPU resources. All convolutional layers are initialized using the He initialization [8], and the biases are initialized to zero. For reproducibility, we set fixed random seeds in all experiments. Code optimizations, including gradient checkpointing [4] are used to reduce memory consumption when necessary.

Inference Details. During inference, we evaluate the model on the LR images without any additional preprocessing. For arbitrary scaling factors beyond the training range (e.g., $\times 6$, $\times 8$, $\times 12$), the model operates without modification, leveraging the resolution-invariant properties of neural operators. The ATS ODE solver dynamically adjusts the integration steps based on the data characteristics, ensuring efficient

CVPR #10261

and accurate reconstruction. We measure inference time by averaging more than 100 runs on the validation set.

3. Hyperparameter Analysis

In this section, we analyze the impact of key hyperparameters on the performance of DiffFNO reported in the main paper (Section 4). We perform ablation studies to understand the contributions of different components and settings to overall performance.

Number of Fourier Modes. To evaluate the effect of retaining all Fourier modes with adaptive weighting versus mode truncation, we conduct experiments with different numbers of modes retained in WFNO. Specifically, we compare the following settings:

1. *Mode Truncation*: Retain only the lowest k frequency modes, with k ranging from 16 to 64.
2. *Full Modes with Uniform Weighting*: Retaining all modes without adaptive weighting.
3. *Full Modes with Mode Rebalancing*: Retaining all modes with our adaptive mode weighting mechanism.

The results, summarized in Table 1, show that retaining all modes with adaptive weighting significantly outperforms mode truncation and full modes with uniform weighting. This highlights the importance of high-frequency components in super-resolution tasks and the effectiveness of our Mode Rebalancing in emphasizing critical frequencies.

Method	# Modes	MR	PSNR
Mode Trunc.	16	No	29.69
Mode Trunc.	32	No	29.91
Mode Trunc.	64	No	30.10
Full Modes	All	No	30.34
WFNO (ours)	All	Yes	**30.88**

Table 1. Impact of Number of Fourier Modes on PSNR (dB) at $\times 4$ Scaling Factor on the DIV2K Validation Set

Gating Mechanism	# Params ↓	PSNR
Simple (ours)	**4K**	30.88
Deep	50K	30.90
Attention-Based	120K	**30.91**

Table 2. Impact of Gating Mechanism Complexity on PSNR (dB) at $\times 4$ Scaling Factor on the DIV2K Validation Set

Gating Mechanism Complexity. We investigate the effect of the complexity of Gated Fusion Mechanism on the performance of the model. We tested different gating schemes:

Tol	PSNR	Inference (s) ↓	# Steps
1×10^{-4}	30.83	**0.85**	**25**
1×10^{-5}	30.85	0.88	28
1×10^{-6} (ours)	**30.88**	0.90	30
1×10^{-7}	**30.88**	0.93	33
1×10^{-8}	30.87	0.97	36

Table 3. Impact of Error Tolerance on PSNR (dB) and Inference Time at $\times 4$ Scaling Factor on the DIV2K Validation Set

1. *Simple Gating*: A single 1×1 convolution followed by sigmoid activation (our default setting).
2. *Deep Gating*: A multi-layer perceptron (MLP) with two hidden layers of sizes 128 and 64, respectively, and ReLU activations.
3. *Attention-Based Gating*: Gating weights are computed using a self-attention module.

The results presented in Table 2, indicate that increasing the complexity of the gating mechanism does not lead to significant performance gains. The simple gating mechanism suffices to effectively integrate global and local features, which is beneficial for computational efficiency. The marginal improvements obtained with more complex gating mechanisms do not justify the much higher computational cost, supporting our choice of the simple 1×1 convolutional gating.

Error Tolerance in ATS ODE Solver. We examine the effect of different error tolerances in the adaptive time-stepping of the ATS ODE solver. We test absolute and relative error tolerances. The results, shown in Table 3, reveal that tighter error tolerances lead to slight improvements in PSNR at the cost of marginally increased inference time. A tolerance of 1×10^{-6} achieves the best balance between reconstruction quality and computational efficiency, justifying its use in our implementation.

Effect of Batch Size. We also investigate the impact of different batch sizes on training stability and convergence. We experiment with batch sizes of 16, 32, 64, 96, and 128. Larger batch sizes lead to more stable training and slightly faster convergence due to better gradient estimation. However, they require more GPU memory. A batch size of 64 provides a good trade-off between computational efficiency and resource utilization on our hardware setup.

Learning Rate Scheduling. We compare different learning rate scheduling strategies, including fixed learning rate, step decay, and our chosen cosine annealing with warm restarts [16]. The cosine annealing scheduler with warm restarts yields better performance by allowing the optimizer to escape local minima, furthering parameter space exploration.

CVPR
#10261

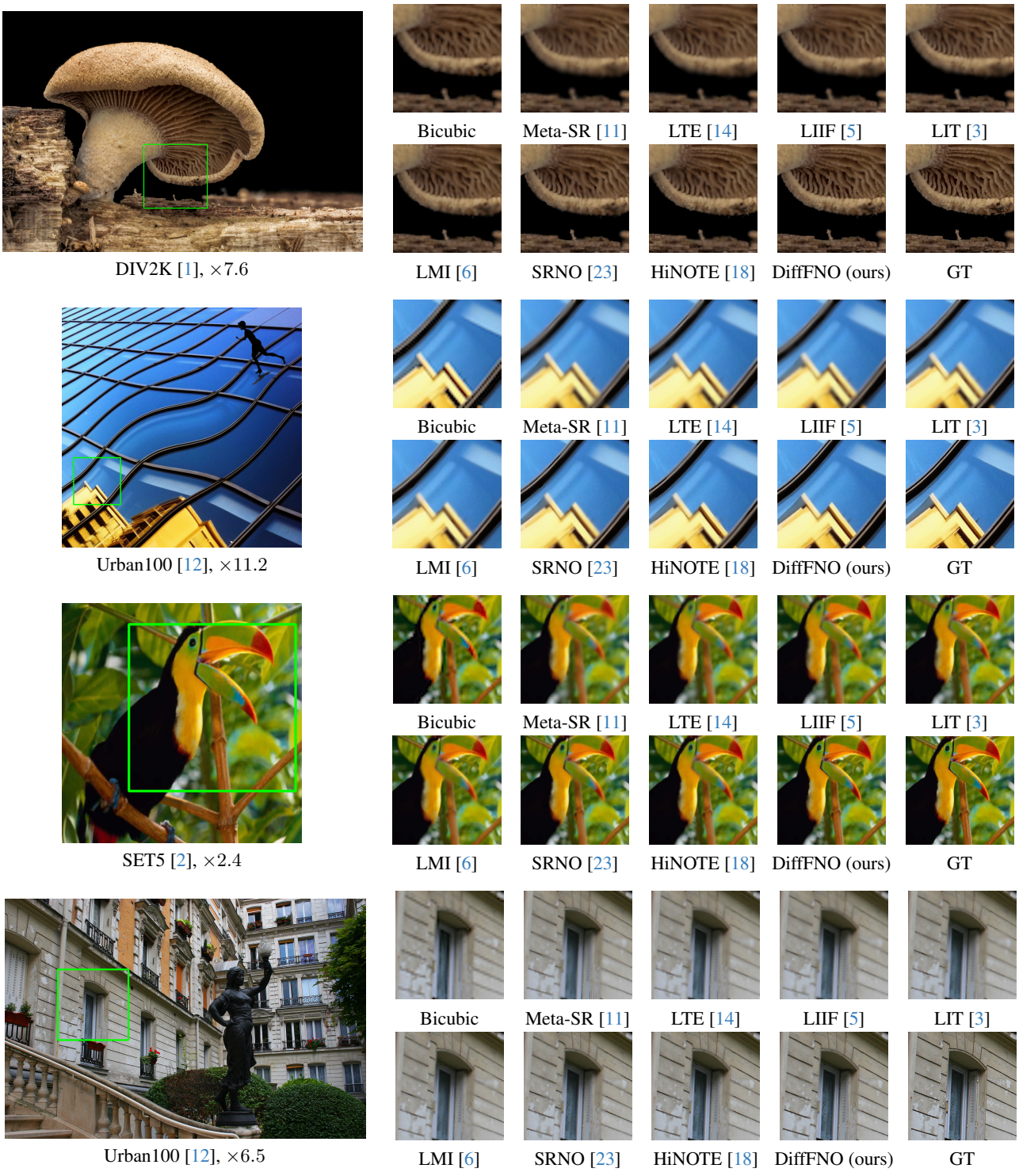

Figure 1. Qualitative results with continuous SR scales. All models used the RDN [25] encoder, except HiNOTE [18] which uses its own.

Conclusion. The hyperparameter analysis confirms the effectiveness of our design choices in DiffFNO. The retention of all Fourier modes with adaptive weighting, the simple yet effective gating mechanism, and the careful selection of error tolerances in the ODE solver all contribute to the superior performance of our model in arbitrary-scale SR.

CVPR
#10261

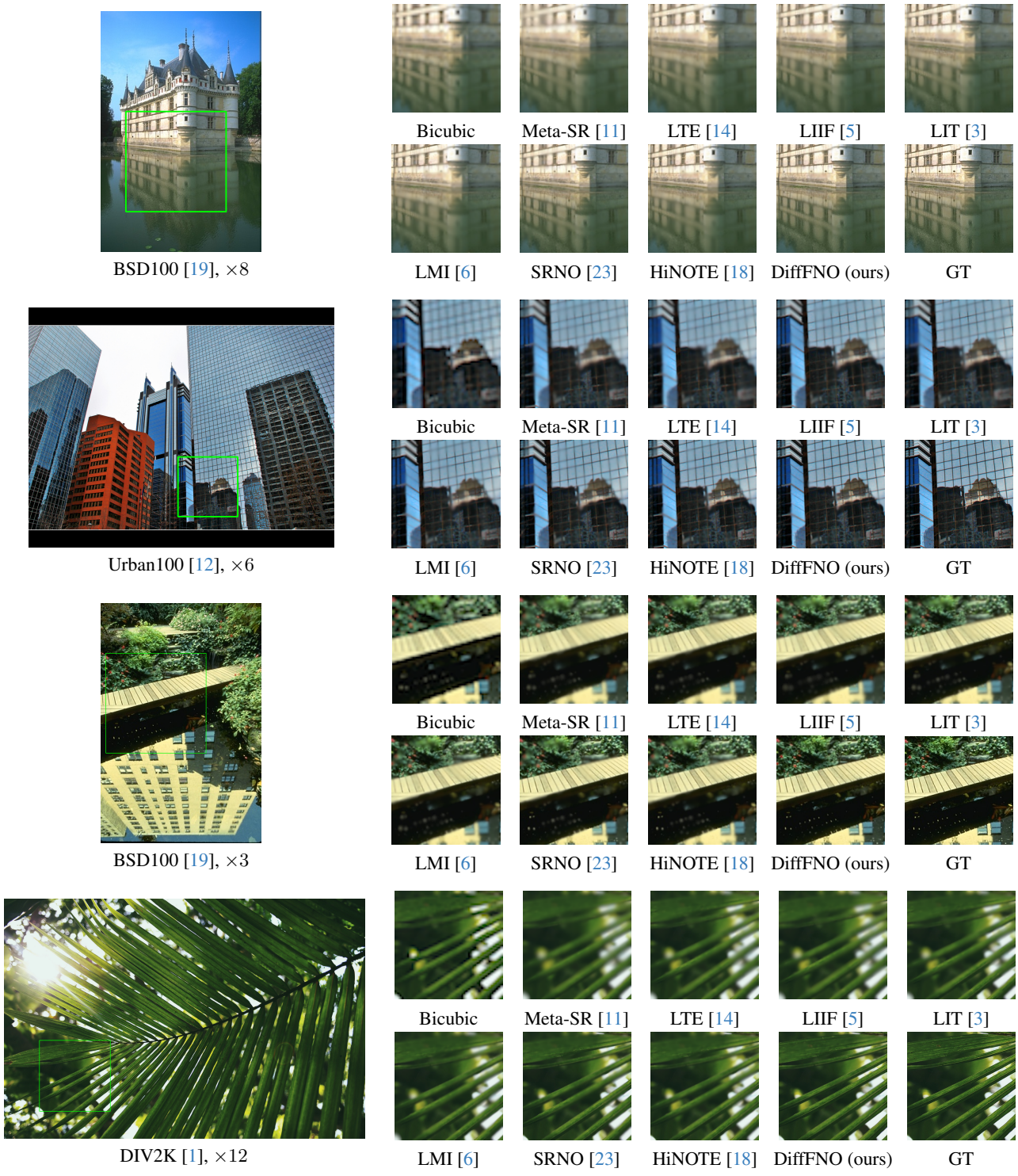

Figure 2. Qualitative results with integer SR scales. All models used the RDN [25] encoder, except HiNOTE [18] which uses its own.

4. More Qualitative Results

To further demonstrate the efficacy of DiffFNO, we provide a comprehensive set of additional qualitative results across various datasets and scaling factors in Fig. 1, 2, and 3, spanning integer and continuous SR scales in and out of the training distribution. These examples highlight

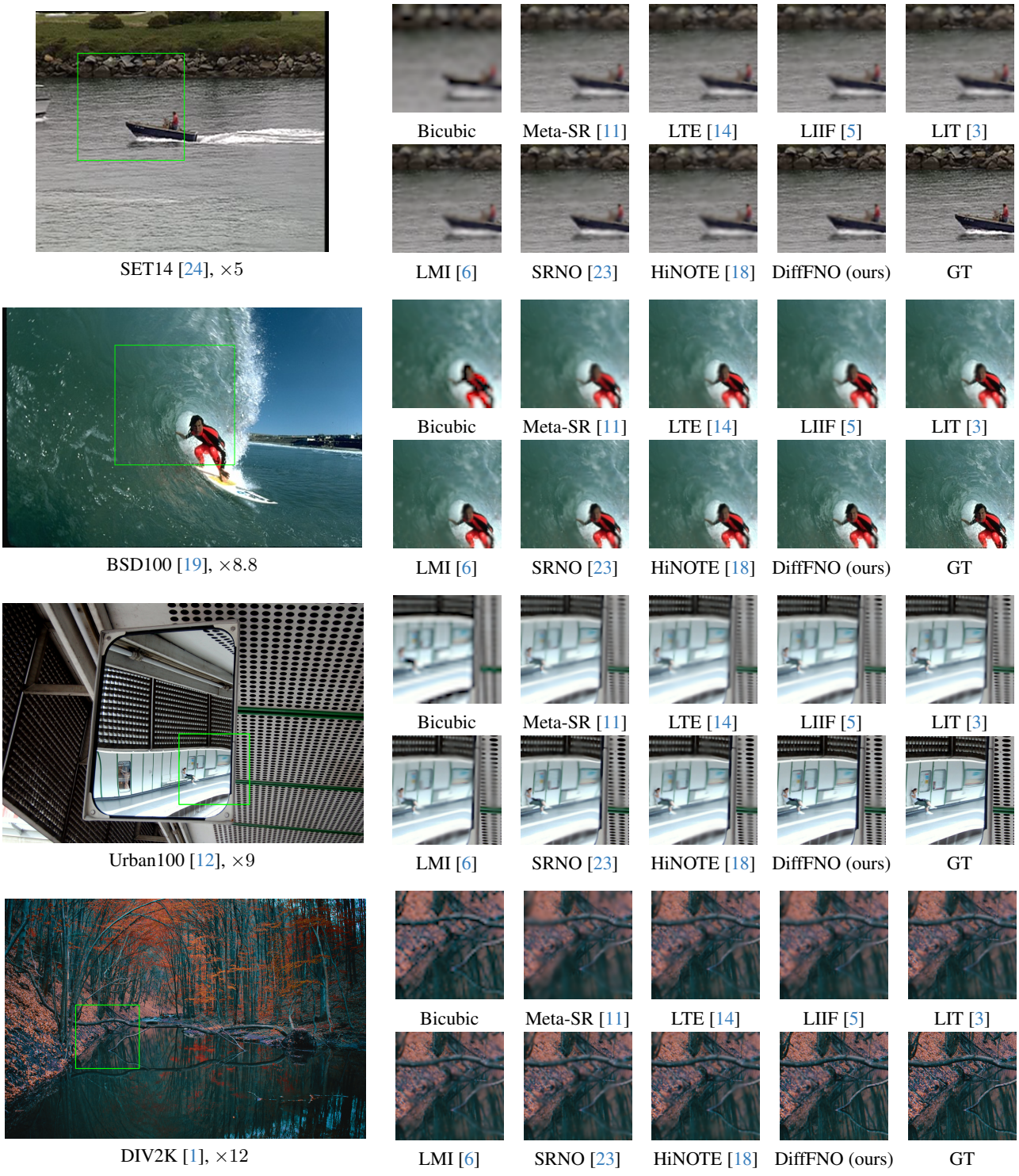

Figure 3. Additional qualitative results with out-of-distribution SR scales. All models used the RDN [25] encoder, except HiNOTE [18].

the model's ability to preserve intricate textures, maintain sharp edges, and accurately reconstruct fine details in diverse scenarios. Comparisons with state-of-the-art methods illustrate DiffFNO's superior performance in handling complex structures and reducing artifacts, thereby validating its robustness and generalization capabilities. Through these

visualizations, our aim is to offer deeper insights into the qualitative advantages of our approach beyond the quantitative metrics presented in the main paper.

References

[1] Eirikur Agustsson and Radu Timofte. Ntire 2017 challenge on single image super-resolution: Dataset and study. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, 2017. 1, 4, 5, 6

[2] Marco Bevilacqua, Aline Roumy, Christine Guillemot, and Marie-line Alberi-Morel. Low-complexity single-image super-resolution based on nonnegative neighbor embedding. In *Proceedings of the British Machine Vision Conference (BMVC)*, pages 135.1–135.10, 2012. 4

[3] Hao-Wei Chen, Yu-Syuan Xu, Min-Fong Hong, Yi-Min Tsai, Hsien-Kai Kuo, and Chun-Yi Lee. Cascaded local implicit transformer for arbitrary-scale super-resolution. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 18257–18267, 2023. 4, 5, 6

[4] Tianqi Chen, Bing Xu, Chiyuan Zhang, and Carlos Guestrin. Training deep nets with sublinear memory cost. In *Proceedings of the 2016 International Conference on Learning Representations (ICLR)*, 2016. 2

[5] Yinbo Chen, Sifei Liu, and Xiaolong Wang. Learning continuous image representation with local implicit image function. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 8628–8638, 2021. 4, 5, 6

[6] Huiyuan Fu, Fei Peng, Xianwei Li, Yejun Li, Xin Wang, and Huadong Ma. Continuous optical zooming: A benchmark for arbitrary-scale image super-resolution in real world. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3035–3044, 2024. 4, 5, 6

[7] Priya Goyal, Piotr Dollár, Ross B. Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch SGD: training imagenet in 1 hour. *CoRR*, abs/1706.02677, 2017. 2

[8] 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 (ICCV)*, pages 1026–1034, 2015. 2

[9] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016. 2

[10] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851, 2020. 1

[11] Xuecai Hu, Haoyuan Mu, Xiangyu Zhang, Zilei Wang, Tieniu Tan, and Jian Sun. Meta-sr: A magnification-arbitrary network for super-resolution. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 1575–1584, 2019. 4, 5, 6

[12] Jia-Bin Huang, Abhishek Singh, and Narendra Ahuja. Single image super-resolution from transformed self-exemplars. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5197–5206, 2015. 4, 5, 6

[13] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 2

[14] Jaewon Lee and Kyong Hwan Jin. Local texture estimator for implicit representation function. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 1929–1938, 2022. 4, 5, 6

[15] Bee Lim, Sanghyun Son, Heewon Kim, Seungjun Nah, and Kyoung Mu Lee. Enhanced deep residual networks for single image super-resolution. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, 2017. 1

[16] Ilya Loshchilov and Frank Hutter. SGDR: Stochastic gradient descent with warm restarts. In *International Conference on Learning Representations*, 2017. 2, 3

[17] Cheng Lu, Yuhao Zhou, Fan Bao, Jianfei Chen, Chongxuan Li, and Jun Zhu. Dpm-solver: A fast ode solver for diffusion probabilistic model sampling in around 10 steps. *Advances in Neural Information Processing Systems*, 35:5775–5787, 2022. 1

[18] Xihaier Luo, Xiaoning Qian, and Byung-Jun Yoon. Hierarchical neural operator transformer with learnable frequency-aware loss prior for arbitrary-scale super-resolution. *arXiv preprint arXiv:2405.12202*, 2024. 4, 5, 6

[19] David Martin, Charless Fowlkes, Doron Tal, and Jitendra Malik. A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 416–423, 2001. 5, 6

[20] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library, 2019. 2

[21] PyTorch Team. *Automatic Mixed Precision (AMP)*. PyTorch, 2023. https://pytorch.org/docs/stable/amp.html. 2

[22] A Vaswani. Attention is all you need. *Advances in Neural Information Processing Systems*, 2017. 1, 2

[23] Min Wei and Xuesong Zhang. Super-resolution neural operator. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 18247–18256, 2023. 4, 5, 6

[24] Roman Zeyde, Michael Elad, and Matan Protter. On single image scale-up using sparse-representations. In *International Conference on Curves and Surfaces*, pages 711–730, 2010. 6

[25] Yulun Zhang, Yapeng Tian, Yu Kong, Bineng Zhong, and Yun Fu. Residual dense network for image super-resolution. In *CVPR*, 2018. 1, 4, 5, 6