From Zero to Detail: Deconstructing Ultra-High-Definition Image Restoration from Progressive Spectral Perspective

Chen Zhao[1*], Zhizhou Chen[1*], Yunzhe Xu[1], Enxuan Gu[2], Jian Li[3]
Zili Yi[1], Qian Wang[4], Jian Yang[1], Ying Tai[1†]
[1]Nanjing University, [2]Dalian University of Technology, [3]Tencent Youtu, [4]China Mobile Institute

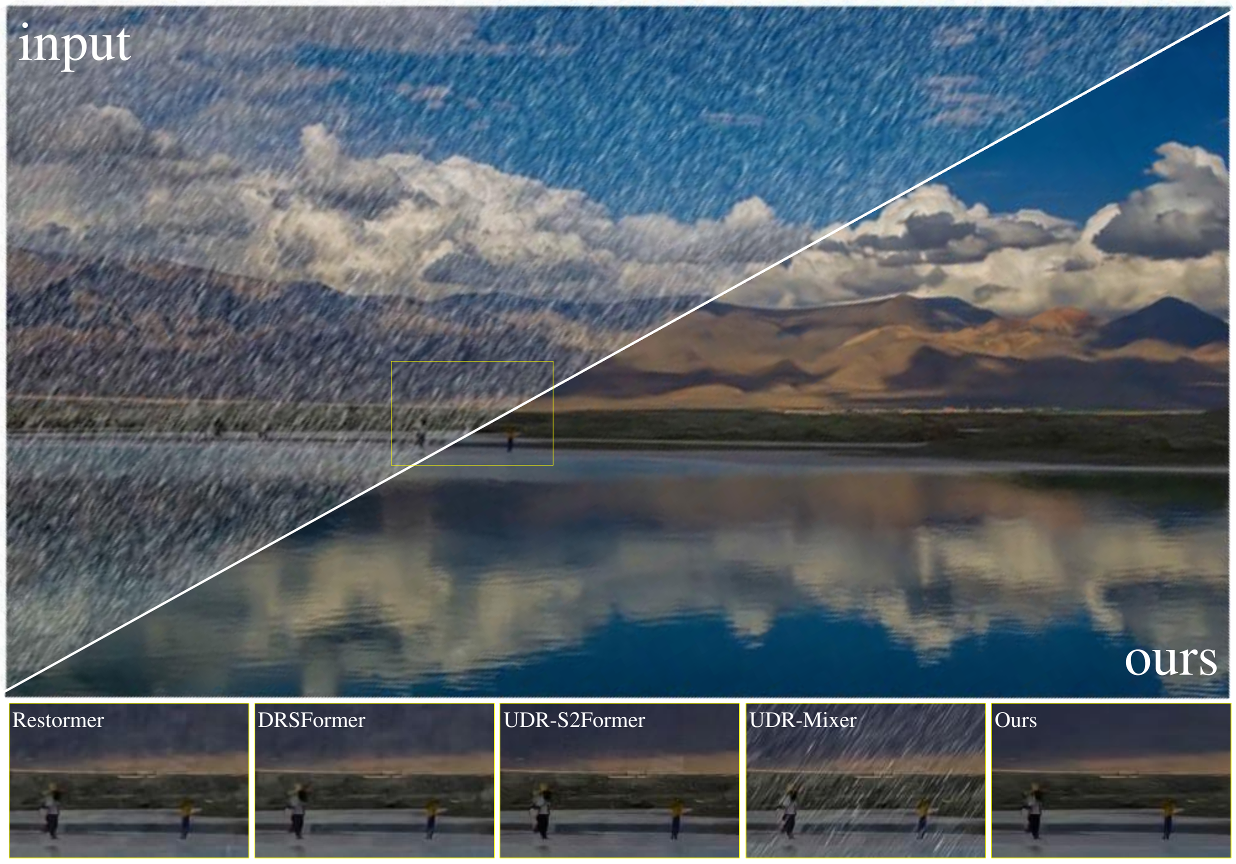

Figure 1. Visualization comparisons on an UHD image. Compared to the state-of-the-art models, our results demonstrate more natural facial details and better structure. (Zoom-in for best view)

Supplementary Material Overview

This supplementary material delves into the key aspects of our approach, offering a detailed exploration of the core concepts and experimental results that complement the main manuscript. We begin with an overview of **State Space Models (SSMs)**, emphasizing their relevance and the integration of Mamba, a state-of-the-art model designed for efficient long-range dependency handling. Next, we discuss the foundational principles of **Kolmogorov–Arnold Networks (KAN)** , highlighting their ability to model complex transformations through learnable base functions. In the **Hybrid Response Analysis**in section , we explore how linear systems capture low-frequency information, while nonlinear systems are employed for the injection of high-frequency features. The **Visualization Results: Nonlinear High-Frequency Injection** section in section presents practical examples illustrating the efficacy of this approach. We further extend our comparative analysis with **Additional**

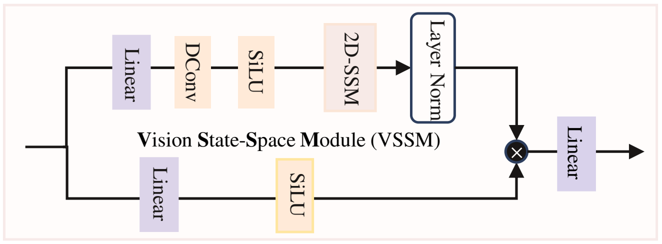

Figure 2. The detail architecture of the Vision State-Space Module (VSSM).

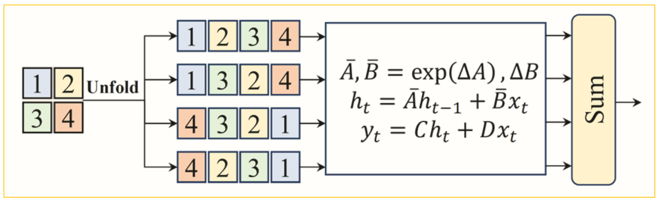
Unfold
$\bar{A}, \bar{B} = \exp(\Delta A), \Delta B$
$h_t = \bar{A}h_{t-1} + \bar{B}x_t$
$y_t = Ch_t + Dx_t$
Sum

Figure 3. The architecture of the 2D selective scanning module (2D-SSM) in the VSS Module.

Visual Comparison Results with SOTA Methods, providing a detailed evaluation of performance against state-of-the-art techniques. Finally, we conclude with a set of **Further Analysis and Discussion** that includes ablation experiments and a deeper interpretation of our findings.

1. Preliminaries: State Space Models

Structured State Space Models (S4), which are based on continuous systems, describe the dynamic interaction between inputs $x(t)$ and outputs $y(t)$ within a linear time-invariant framework. In essence, this model maps a one-dimensional function or sequence $x(t) \in \mathbb{R}^L$ to $y(t) \in \mathbb{R}^L$ through an implicit latent state $h(t) \in \mathbb{R}^N$. Mathematically, this relationship is compactly expressed as a linear ordinary differential equation (ODE), which is outlined as follows:

$$h'(t) = \boldsymbol{A}h(t) + \boldsymbol{B}x(t) \tag{1}$$

$$y(t) = \boldsymbol{C}h(t) + \boldsymbol{D}x(t) \tag{2}$$

where $\boldsymbol{A} \in \mathbb{R}^{N\times N}$, $\boldsymbol{B} \in \mathbb{R}^{N\times 1}$, $\boldsymbol{C} \in \mathbb{R}^{1\times N}$ are the parameters for a state size N, and $\boldsymbol{D} \in \mathbb{R}^1$ denotes the skip connection.

To incorporate State Space Models (SSMs) into deep learning frameworks, researchers discretize the ODE process and synchronize the model with the sample rate of the input data's underlying signal. The discretization commonly uses the zeroth-order hold (ZOH) method, where the time scale parameter Δ is used to convert the continuous parameters $\boldsymbol{A}$ and $\boldsymbol{B}$ into their discrete equivalents $\bar{\boldsymbol{A}}$ and $\bar{\boldsymbol{B}}$. This conversion is defined as follows:

$$h'_t = \bar{\boldsymbol{A}}h_{t-1} + \bar{\boldsymbol{B}}x_t \tag{3}$$

$$y_t = \boldsymbol{C}h_t + \boldsymbol{D}x_t \tag{4}$$

$$\bar{\boldsymbol{A}} = e^{\Delta \boldsymbol{A}} \tag{5}$$

$$\bar{\boldsymbol{B}} = (\Delta \boldsymbol{A})^{-1}(e^{\Delta \boldsymbol{A}} - I) \cdot \Delta \boldsymbol{B} \tag{6}$$

where $\Delta \in \mathbb{R}^D$ and $\boldsymbol{B}, \boldsymbol{C} \in \mathbb{R}^{D\times N}$. The recently introduced state space model, Mamba, has been enhanced by making the parameters $\boldsymbol{B}$, $\boldsymbol{C}$, and Δ input-dependent, which facilitates dynamic feature representation. Essentially, Mamba employs a recursive structure similar to that in Eq. (3), enabling it to handle and retain information from very long sequences. This feature ensures that more pixels contribute to the restoration process. Furthermore, Mamba adopts a parallel scan algorithm, benefiting from the parallel processing advantages discussed in Eq. (3), thereby improving both training and inference efficiency.

Vision State Space Module: Building on Mamba's success in modeling long-range dependencies with linear complexity, we have integrated the VSSM into the UHD IR task. The VSSM effectively captures long-range dependencies using the state space equation. As shown in Figure 2, the VSSM architecture processes the input feature $X \in \mathbb{R}^{H \times W \times C}$ through two parallel branches. In the first branch, a linear layer expands the feature channel to λC, where λ is a predefined expansion factor. This is followed by a depth-wise convolution, SiLU activation, a 2D Selective Scan Module (2D-SSM), and LayerNorm. The second branch also expands the channels to λC using a linear layer, followed by SiLU activation. The outputs of both branches are then combined using a Hadamard product. Finally, the channels are reduced back to C, producing an output X_{out} with the same dimensions as the input. This can be expressed as follows:

$$\begin{aligned} X_1 &= LN(\text{2D-SSM}(SiLU(DWConv(Linear(X))))) \qquad (7)\\ X_2 &= SiLU(Linear(X)) \qquad (8)\\ X_{out} &= Linear(X_1 \odot X_2) \qquad (9) \end{aligned}$$

where $DWConv(\cdot)$ and $Linear(\cdot)$ denote depth-wise convolution and linear projection. $\odot$ denotes the Hadamard product.
2D Selective Scan Module (2D-SSM): Mamba originally processes data causally, which works well for sequential tasks like NLP but struggles with non-sequential data such as images. To address this, we apply the 2D-SSM from [3, 5, 6]. As illustrated in Figure 3, the technique converts a 2D image feature into a linear sequence by scanning in four directions: top-left to bottom-right, bottom-right to top-left, top-right to bottom-left, and bottom-left to top-right. The discrete state space equation is then used to capture long-range dependencies for each sequence. Afterward, the sequences are combined and reshaped to recover the original 2D structure.

2. Preliminaries: Kolmogorov–Arnold Networks

2.1. Kolmogorov-Arnold Representation Theorem

The Kolmogorov-Arnold representation theorem [7] states that any multivariate continuous function f, defined on a bounded domain, can be expressed as a finite composition of continuous univariate functions and addition. Specifically, for a smooth function $f : [0, 1]^n \to \mathbb{R}$, it can be represented as:

$$f(x_1, \ldots, x_n) = \sum_{q=1}^{2n+1} \Phi_q \left(\sum_{p=1}^{n} \phi_{q,p}(x_p) \right)$$

Here, each function $\phi_{q,p} : [0, 1] \to \mathbb{R}$ and $\Phi_q : \mathbb{R} \to \mathbb{R}$ are continuous. This means that the (2d+1)(d+1) univariate functions Φ_q and $\phi_{q,p}$ are enough for an exact representation of a d-variate function.

This theorm can be written in matrix form as follows:

$$f(\mathbf{x}) = \Phi_{\text{out}} \circ \Phi_{\text{in}} \circ \mathbf{x} \qquad (10)$$

where Φ_{in} and Φ_{out} are defined as:

$$\Phi_{\text{in}} = \begin{bmatrix} \phi_{1,1}(\cdot) & \cdots & \phi_{1,n}(\cdot) \\ \vdots & \ddots & \vdots \\ \phi_{2d+1,1}(\cdot) & \cdots & \phi_{2d+1,d}(\cdot) \end{bmatrix} \qquad (11)$$

$$\Phi_{\text{out}} = \begin{bmatrix} \Phi_1(\cdot) & \cdots & \Phi_{2d+1}(\cdot) \end{bmatrix} \qquad (12)$$

This decomposition illustrates how f can be built from simpler functions, showcasing an essential property of multivariate continuous functions.

2.2. Kolmogorov Arnold Networks

Inspired by the Kolmogorov-Arnold representation theorem, [9] define a generalized Kolmogorov-Arnold layer to learn univariate functions on edge, in the form of activation function. Formally, a Kolmogorov-Arnold layer with d_{in}-dimensional inputs and d_{out}-dimensional outputs is illustrated as

$$f(\mathbf{x}) = \Phi \circ \mathbf{x} = \begin{bmatrix} \sum_{i=1}^{d_{in}} \phi_{i,1}(x_i) & \ldots & \sum_{i=1}^{d_{in}} \phi_{i,d_{out}}(x_i) \end{bmatrix}, \text{where} \quad \Phi = \begin{bmatrix} \phi_{1,1}(\cdot) & \cdots & \phi_{1,d_{\text{in}}}(\cdot) \\ \vdots & \ddots & \vdots \\ \phi_{d_{\text{out}},1}(\cdot) & \cdots & \phi_{d_{\text{out}},d_{\text{in}}}(\cdot) \end{bmatrix} \qquad (13)$$

Note that Eq 13 can be seen as a generalized form of Eq 10, such that $\Phi = \Phi_{\text{in}} \circ \Phi_{\text{out}}$. A general KAN network is a stacking of L layers: given an input vector $\mathbf{x}_0 \in \mathbb{R}^{d_0}$, the output of KAN is $KAN(\mathbf{x}_0) = \Phi_{L-1} \circ \Phi_{L-2} \cdots \circ \Phi_0 \circ \mathbf{x}_0$.

In practice, [9] parameterizes Φ use a linear combination of SiLU activation [4] and a B-spline function

$$\phi(x) = w_b \texttt{silu}(x) + w_s \texttt{spline}(x), \text{where} \quad \texttt{silu}(x) = \frac{x}{1+e^{-x}}, \qquad \texttt{spline}(x) = \sum_i c_i B_i(x) \tag{14}$$

2.3. Challenges for original KAN

(1) B-Splines on GPUs are Inefficient. B-splines pose challenges when used in GPU-based computations, primarily because they are not native to CUDA. Implementing B-splines with pure PyTorch or NumPy leads to slower processing on modern GPUs, as there is no CUDA optimization for these operations. Additionally, the localized nature of B-spline computations—where each control point influences a small region of the curve—creates sparse, recursive calculations that GPUs handle inefficiently. While there are existing methods for cubic B-splines [10], extending these techniques to higher-order splines proves difficult.

(2) Parameter and Computational Overhead. KAN introduces a learnable base function for each pair of input-output channels, significantly increasing both the number of parameters and computational load. In contrast, standard neural networks, such as MLPs, have far fewer parameters, with the number of learnable parameters for a KAN layer calculated as:

$$(d_{in} \times d_{out}) \times (G + K + 3) + d_{out}.$$

A regular MLP layer would only require:

$$(d_{in} \times d_{out}) + d_{out}$$

The computational cost for a KAN layer, expressed in FLOPs, becomes:

$$\Big\{\text{FLOPs of non-linear function} \times d_{in} + (d_{in} \times d_{out}) \times [9K \times (G + 1.5K) + 2G - 2.5K + 3]\Big\},$$

while an MLP only requires:

$$\Big\{\text{FLOPs of non-linear function} \times d_{out} + 2 \times (d_{in} \times d_{out})\Big\}.$$

As a result, KAN's complexity in terms of parameters and computation is on the order of $O(G + K)$ and $O(GK)$ times greater than a conventional MLP.

(3) Improper Weight Initialization. Effective weight initialization is crucial for the successful training of deep learning models. Proper initialization ensures variance-preserving properties, meaning that the signal's variance remains consistent as it propagates through layers, both forwards and backwards. However, the KAN paper deviates from this principle. Specifically, B-spline coefficients c_i are initialized as $\mathcal{N}(0, \sigma^2)$ with $\sigma = 0.1$, while w_s and w_b are initialized according to Xavier initialization. When considering a zero-th order spline, the combined variance of the output $\phi(x)$ is given by:

$$Var[\phi(x)] = Var[w_b \texttt{silu}(x)] + Var[w_s \texttt{spline}(x)] = 3\mathbb{E}[\texttt{silu}^2(x)] + \mathbb{E}[\texttt{spline}^2(x)].$$

For an input $x \sim \mathcal{N}(0, \sigma_x^2)$, the variance of the spline function becomes:

$$\mathbb{E}[\texttt{spline}^2(x)] = \sigma^2 = 0.01.$$

Estimations for the SiLU function give:

$$\mathbb{E}[\texttt{silu}^2(x)] \approx 0.355\sigma_x^2,$$

leading to:

$$Var[\phi(x)] \approx 0.01 + 1.064\sigma_x^2 \neq Var[x].$$

Thus, the variance of $\phi(x)$ does not match the input variance, violating the variance-preserving principle, especially for higher-order splines. This initialization strategy undermines stability, making training less efficient.

2.4. Rational Base Functions

Rational functions [1] are used as the base function for [11], instead of the B-spline. Specifically, the function $\phi(x)$ on each edge is parameterized as a rational function over polynomials $P(x)$ and $Q(x)$ of orders m and n, respectively:

$$\phi(x) = wF(x) = w\frac{P(x)}{Q(x)} = w\frac{a_0 + a_1 x + \cdots + a_m x^m}{b_0 + b_1 x + \cdots + b_n x^n} \tag{15}$$

where a_n and b_m are the coefficients of the rational function, and w is the scaling factor. This function is referred to as having degree m/n. The goal is to learn the parameters a_n, b_m, and w through end-to-end backpropagation.

To avoid instability caused by poles, where $Q(x) \to 0$ and $\phi(x) \to \pm\infty$, a Safe Padé Activation Unit (PAU) is employed as the basis. The PAU is a modified form of the standard rational function, given by:

$$F(x) = \frac{a_0 + a_1 x + \cdots + a_m x^m}{1 + |b_1 x + \cdots + b_n x^n|} \tag{16}$$

Implement Rational Function on GPU. A core contribution of [11] is the efficient implementation of the rational function on parallelized devices such as GPUs. Instead of using `pytorch` with automatic differentiation, the function is implemented entirely with CUDA.

- Explicit gradients of $\frac{\delta F}{\delta a_m}$, $\frac{\delta F}{\delta b_n}$, and $\frac{\delta F}{\delta x}$ are computed as follows:

$$\frac{\delta F}{\delta a_m} = \frac{x^m}{Q(x)}, \quad \frac{\delta F}{\delta b_n} = ax^n \frac{A(x)}{|A(x)|}\frac{P(x)}{Q(x)^2}, \quad \text{and} \quad \frac{\delta F}{\delta x} = \frac{\delta P(x)}{\delta x}\frac{1}{Q(x)} - \frac{\delta Q(x)}{\delta x}\frac{P(x)}{Q^2(x)} \tag{17}$$

  where $A(x) = b_1 x + \cdots + b_n x^n$, $\frac{\delta P(x)}{\delta x} = a_1 + 2a_2 x + m a_m x^{m-1}$, and $\frac{\delta Q(x)}{\delta x} = \frac{A(x)}{|A(x)|}(b_1 + 2b_2 x + n b_n x^{n-1})$.
- To optimize the evaluation of polynomials, Horner's method [8] is employed, which reformulates a polynomial in a nested form to reduce the computation:

$$a_0 + a_1 x + \cdots + a_m x^m = a_0 + x(a_1 + x(a_2 + x(\ldots))) \tag{18}$$

  This allows the evaluation of a polynomial of degree n with only n multiplications and n additions. By default, $m = 5$ and $n = 4$ are used.

2.5. Group-Rational KAN

Rational function from Section 2.4 is combined with group-wise parameters to implement the Group-Rational KAN (GR-KAN). In practice, the parameter for the rational function F is shared across each group; however, each edge retains a unique scalar w.

Suppose i is the index of the input channel. With g groups, each group contains $d_g = d_{in}/g$ channels, where i/d_g is the group index. The operation of GR-KAN on the input vector $\mathbf{x}$ can be expressed as:

$$\text{GR-KAN}(\mathbf{x}) = \Phi \circ \mathbf{x} = \left[\sum_{i=1}^{d_{in}} w_{i,1} F_{\left\lfloor \frac{i}{d_g} \right\rfloor}(x_i) \quad \ldots \quad \sum_{i=1}^{d_{in}} w_{i,d_{out}} F_{\left\lfloor \frac{i}{d_g} \right\rfloor}(x_i)\right]. \tag{19}$$

With a simple rewrite, this can be expressed in matrix form as the product of a weight matrix $\mathbf{W} \in \mathbb{R}^{d_{\text{in}} \times d_{\text{out}}}$ and an input-wise rational function $\mathbf{F}$:

$$\texttt{GR-KAN}(\mathbf{x}) = \mathbf{W}\mathbf{F}(\mathbf{x}) = \begin{bmatrix} w_{1,1} & \cdots & w_{1,d_{\text{in}}} \\ \vdots & \ddots & \vdots \\ w_{d_{\text{out}},1} & \cdots & w_{d_{\text{out}},d_{\text{in}}} \end{bmatrix} \times \left[F_{1/d_g}(x_1) \quad \ldots \quad F_{d_{in}/d_g}(x_{d_{in}})\right]^\top \tag{20}$$

Thus, the GR-KAN layer can be implemented as a group-wise rational function $\mathbf{F}$ followed by a linear layer:

$$\texttt{GR-KAN}(\mathbf{x}) = \texttt{linear}(\texttt{group\_rational}(\mathbf{x})) \tag{21}$$

In this form, sharing parameters across each input channel allows direct application of the rational function to the input vector, which is equivalently applied across each grouped edge. In this way, GR-KAN functions as a specialized MLP, with 1) learnable non-linear functions, 2) activation preceding the linear layer, and 3) unique activation functions tailored for each group of edges. Consequently, GR-KAN [11] addresses the high complexity and optimization challenges of the original KAN [9] by introducing rational activation functions. In our study, we introduce the GR-KAN into our framework.

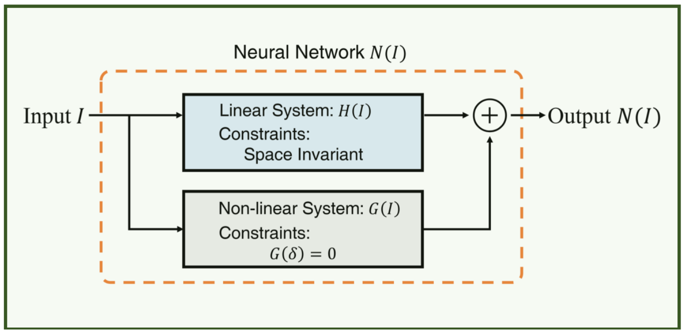

Figure 4. Conceptual diagram of the HyRA.

3. Preliminaries: Hybrid Response Analysis

In this section, we describe the Hybrid Response Analysis (HyRA) [2], which treats the neural network as a combination of a linear system and a non-linear system. The core concept of HyRA is illustrated in Figure 4. Deep network is denoted as $N(I)$, where I is the input image. $N(I)$ is a non-linear system that can be expressed as the sum of a linear system and a non-linear system:

$$N(I) = H(I) + G(I). \tag{22}$$

In this equation, $H(I)$ represents a linear system, and $G(I)$ represents a non-linear system. Without constraints, such a representation is meaningless because $H(I)$ can be arbitrarily chosen, leading to an infinite variety of representations with the same form but different meanings. To give meaning to this representation, a constraint is introduced: the impulse response of $G(I)$ is zero. With this constraint, both $H(I)$ and $G(I)$ can be uniquely determined. Lemma 1 demonstrates that under this constraint, $N(I)$ can still be expressed in the form of Eq. 22. This straightforward method is the essence of HyRA.

Lemma 1. *A neural network $N(I)$ can be expressed as a combination of a linear system $H(I)$ and a non-linear system with an impulse response of zero, i.e.,*

$$N(I) = H(I) + G(I), \quad \textit{where} \quad G(\delta) = 0.$$

Here, δ represents the Dirac delta function.

Proof. 1) **When $G(\delta) = 0$:** In this case, the decomposition $N(I) = H(I) + G(I)$ holds directly by definition. The non-linear component $G(I)$ does not contribute to the impulse response, and the system behaves as expected.

2) **When $G(\delta) \neq 0$:** If the non-linear system's impulse response is non-zero, we redefine the components as follows:

$$H_1(I) = H(I) + G(\delta) * I, \quad G_1(I) = G(I) - G(\delta) * I.$$

Here, $G(\delta) * I$ represents the convolution of the non-linear impulse response with the input. Now, $H_1(I)$ remains a linear system because it is the sum of the original linear system $H(I)$ and the convolution of the Dirac delta with the input, while $G_1(I)$ remains a non-linear system. The equation

$$N(I) = H_1(I) + G_1(I)$$

holds, and it satisfies the condition that the impulse response of the non-linear system $G_1(\delta)$ is zero. Therefore, even when $G(\delta) \neq 0$, it is possible to transform the network into a form where the non-linear impulse response is zero.

□

Once the impulse response of the linear system $H(I)$ is obtained, it can be used to compute the response of the linear system to any input I by convolution. For any input I, the response of the linear system can be computed as:

$$H(I) = I * H(\delta)$$

where $*$ denotes the convolution operation.

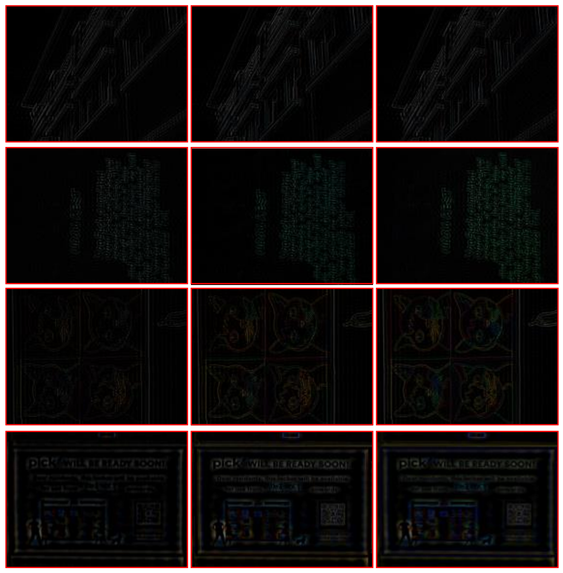

Figure 5. Visual comparison with linear system. Linear system means UHDformer without all non-linear functions. The visual results suggests that the non-linear functions mainly injects high-frequency information.

The non-linear response $G(I)$ is then computed by subtracting the linear response $H(I)$ from the total output $N(I)$ of the network:

$$G(I) = N(I) - H(I) = N(I) - I * H(\delta)$$

This method allows the isolation of the non-linear response $G(I)$, which is crucial for understanding the network's high-frequency behavior. Finally, the study [2] based on the Hybrid Response Analysis introduces an intriguing perspective: the linear system $H(I)$ acts as a low-frequency learner, while *the non-linear system $G(I)$ injects high-frequency information*. The details can be found in [2]. Inspired by this, we further investigate this behavior in UHDformer by removing all non-linear activation functions.

4. Visualization Results: Nonlinear High-Frequency Injection

The visual comparison with the linear system is presented in Figure 5. In this context, the term "linear system" refers to the UHDformer model without the incorporation of any non-linear functions. The visual results highlight a significant difference in the image outputs when comparing the linear system with the fully non-linear model. Specifically, it is evident that the non-linear functions play a crucial role in injecting high-frequency information into the restoration process. This injection of high-frequency content enhances fine details and sharp features in the image, which are absent in the linear system output. The linear model, by contrast, generates smoother and more blurred results, indicating that it lacks the capacity to preserve or emphasize intricate details, a capability enabled by the non-linear components of the model. Thus, the inclusion of non-linear functions in the UHDformer architecture proves to be essential for achieving high-quality, detailed image restoration.

5. Additional Visual Comparison Results with SOTA Methods

As illustrated in Figure 1, our method achieves the most compelling visual results, demonstrating its superior performance in ultra-high-definition (UHD) image restoration tasks. The qualitative comparison highlights the effectiveness of our approach in delivering highly accurate and visually appealing outputs, surpassing existing state-of-the-art (SOTA) methods in multiple dimensions.

Furthermore, additional visual results are presented in Figures 6-15, where our method consistently outperforms other techniques across a diverse range of UHD restoration challenges. Whether addressing various degradation types, including noise, blur, or compression artifacts, our approach stands out for its ability to preserve fine details and recover high-frequency components, producing visually pristine and artifact-free results. These extensive visual evaluations not only substantiate the robustness of our method but also underline its capacity to generalize across different problem settings, positioning it as the leading solution in the field.

The exceptional visual quality achieved by our method reaffirms its superiority, establishing new benchmarks for future research in the UHD image restoration domain.

6. Further Analysis and Discussion

As the proposed Enhanced Resolution Restoration (ERR) framework adopts a multi-stage design, its overall performance is highly sensitive to the chosen training and optimization strategies. Each stage in the framework builds upon the output of the previous one, making the training dynamics complex and interdependent. Consequently, optimizing these stages effectively requires careful consideration of initialization, loss functions, and inter-stage interactions. **Future work will aim to explore more advanced and efficient training methodologies tailored to the multi-stage nature of ERR**. This includes investigating adaptive optimization algorithms, stage-wise fine-tuning techniques, and dynamic loss weighting schemes to balance contributions from different frequency components. Additionally, developing strategies to reduce training time and computational costs while maintaining or even improving performance will be a key focus. By refining these aspects, the framework has the potential to deliver even better restoration quality, further advancing its applicability to challenging UHD image restoration tasks.

References

[1] Nicolas Boullé, Yuji Nakatsukasa, and Alex Townsend. Rational neural networks. *Advances in neural information processing systems*, 33:14243–14253, 2020. 5

[2] Haoyu Deng, Zijing Xu, Yule Duan, Xiao Wu, Wenjie Shu, and Liang-Jian Deng. Exploring the low-pass filtering behavior in image super-resolution. *arXiv preprint arXiv:2405.07919*, 2024. 6, 7

[3] Chenyu Dong, Chen Zhao, Weiling Cai, and Bo Yang. O-mamba: O-shape state-space model for underwater image enhancement. *CoRR*, abs/2408.12816, 2024. 3

[4] Stefan Elfwing, Eiji Uchibe, and Kenji Doya. Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. *Neural networks*, 107:3–11, 2018. 4

[5] Tiehan Fan, Kepan Nan, Rui Xie, Penghao Zhou, Zhenheng Yang, Chaoyou Fu, Xiang Li, Jian Yang, and Ying Tai. Instancecap: Improving text-to-video generation via instance-aware structured caption. *arXiv preprint arXiv:2412.09283*, 2024. 3

[6] Hang Guo, Jinmin Li, Tao Dai, Zhihao Ouyang, Xudong Ren, and Shu-Tao Xia. Mambair: A simple baseline for image restoration with state-space model. In *European Conference on Computer Vision*, pages 222–241. Springer, 2025. 3

[7] Robert Hecht-Nielsen. Kolmogorov's mapping neural network existence theorem. In *Proceedings of the international conference on Neural Networks*, pages 11–14. IEEE press New York, NY, USA, 1987. 3

[8] WG Horner. A new method of solving numerical equations of all orders, by continuous approximation. In *Abstracts of the Papers Printed in the Philosophical Transactions of the Royal Society of London*, pages 117–117. JSTOR, 1815. 5

[9] Ziming Liu, Yixuan Wang, Sachin Vaidya, Fabian Ruehle, James Halverson, Marin Soljačić, Thomas Y Hou, and Max Tegmark. Kan: Kolmogorov-arnold networks. *arXiv preprint arXiv:2404.19756*, 2024. 3, 4, 5

[10] Daniel Ruijters and Philippe Thévenaz. Gpu prefilter for accurate cubic b-spline interpolation. *The Computer Journal*, 55(1):15–20, 2012. 4

[11] Xingyi Yang and Xinchao Wang. Kolmogorov-arnold transformer. *arXiv preprint arXiv:2409.10594*, 2024. 5

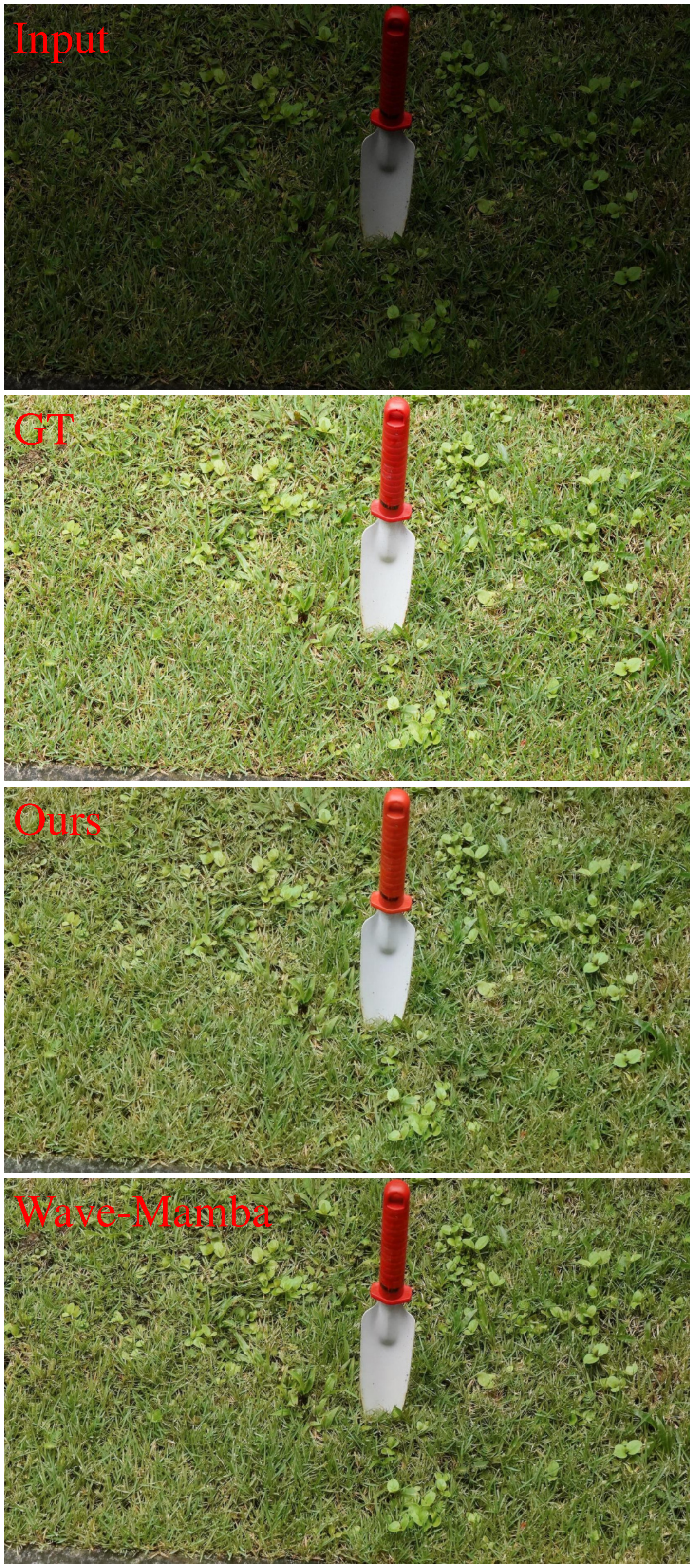

Figure 6. Visual comparison with SOTA method.

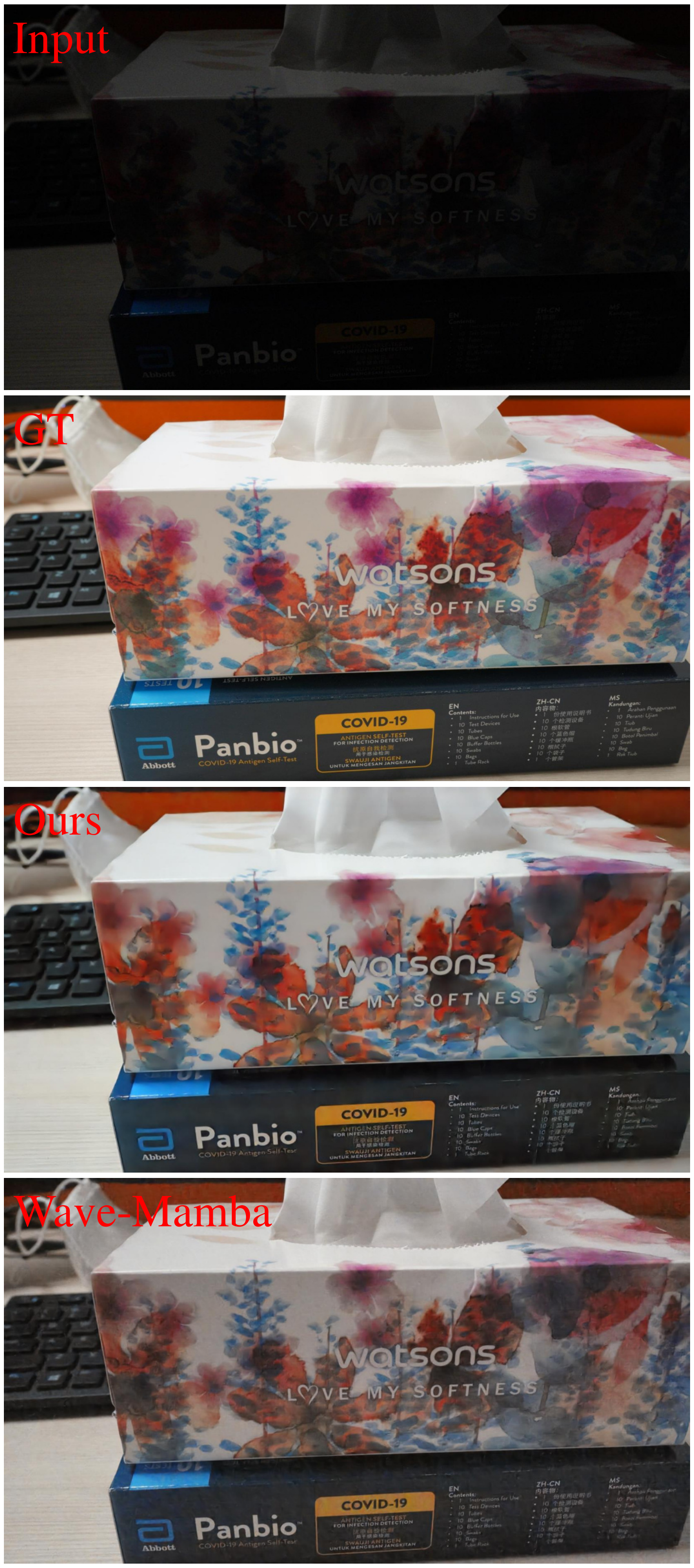

Figure 7. Visual comparison with SOTA method.

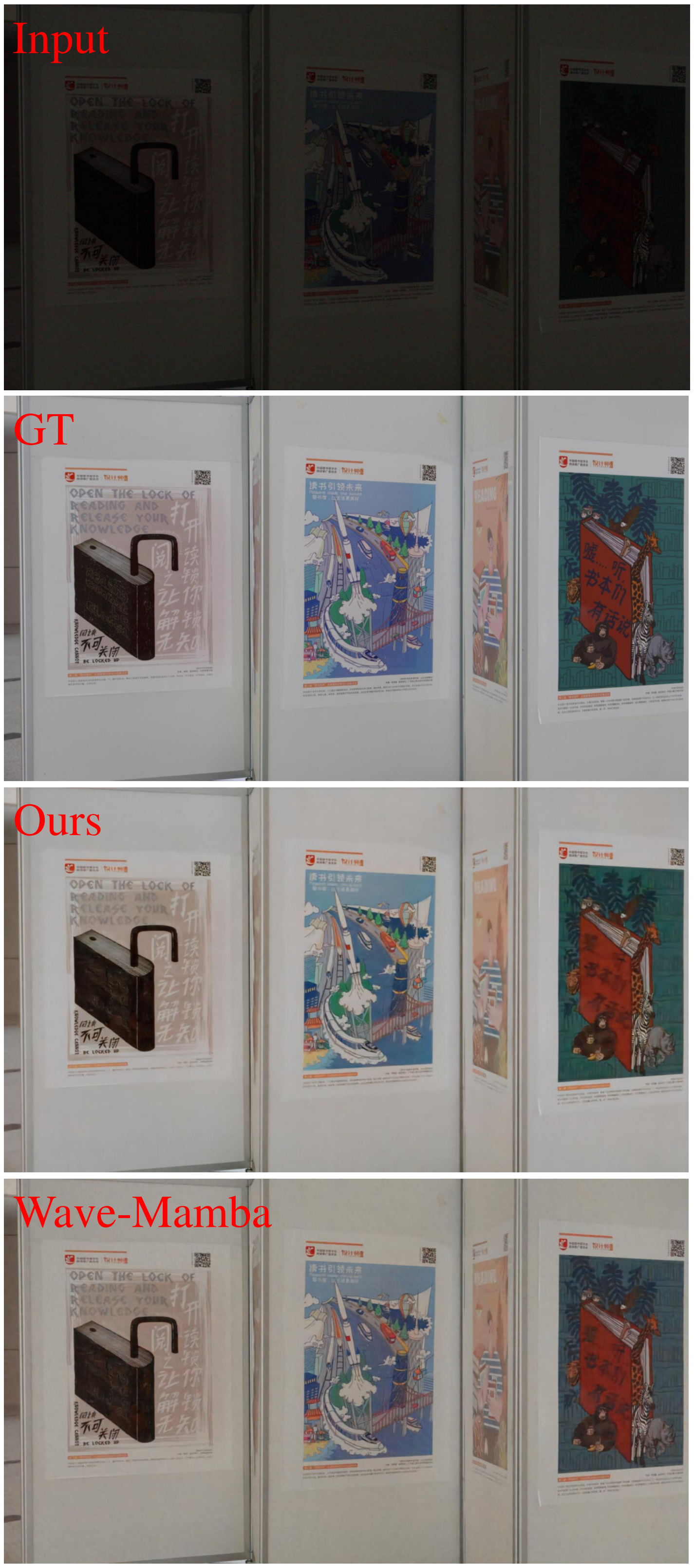

Figure 8. Visual comparison with SOTA method.

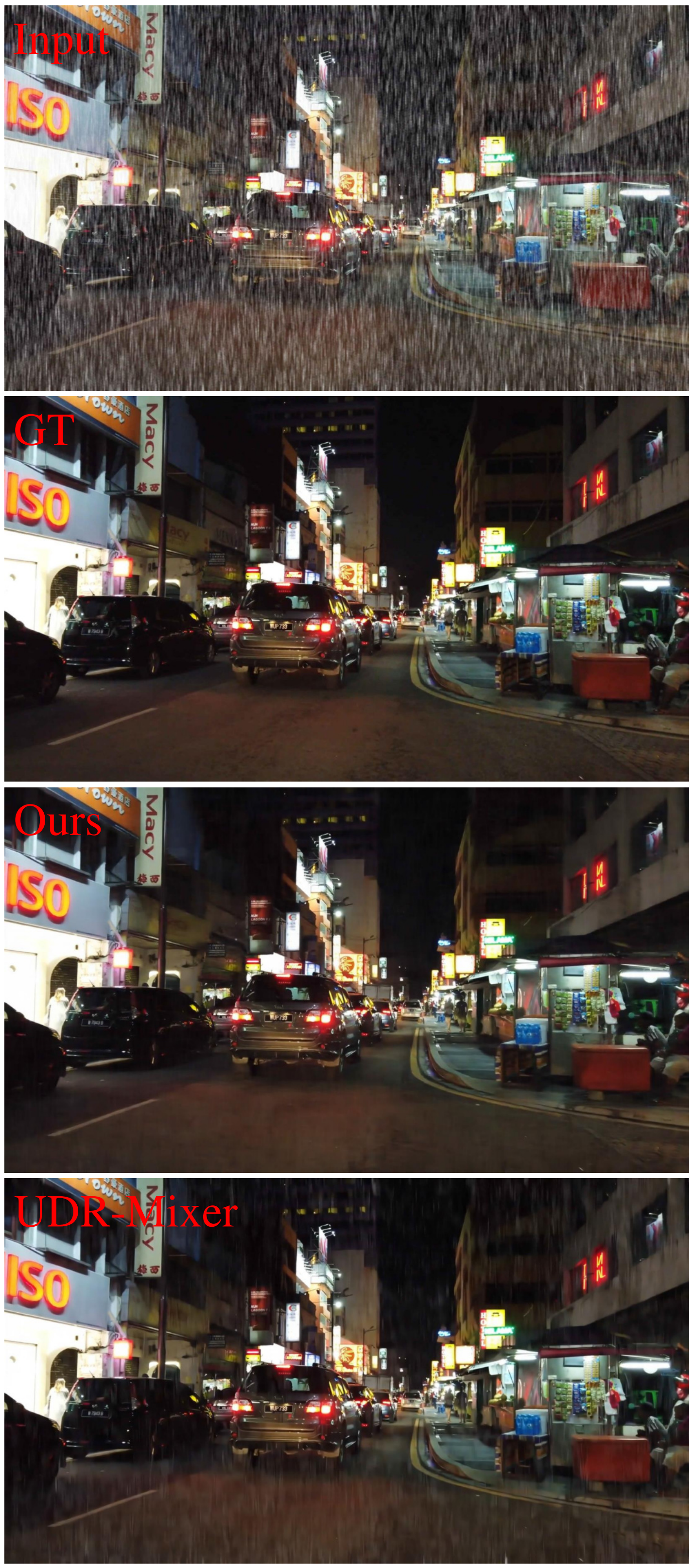

Figure 9. Visual comparison with SOTA method.

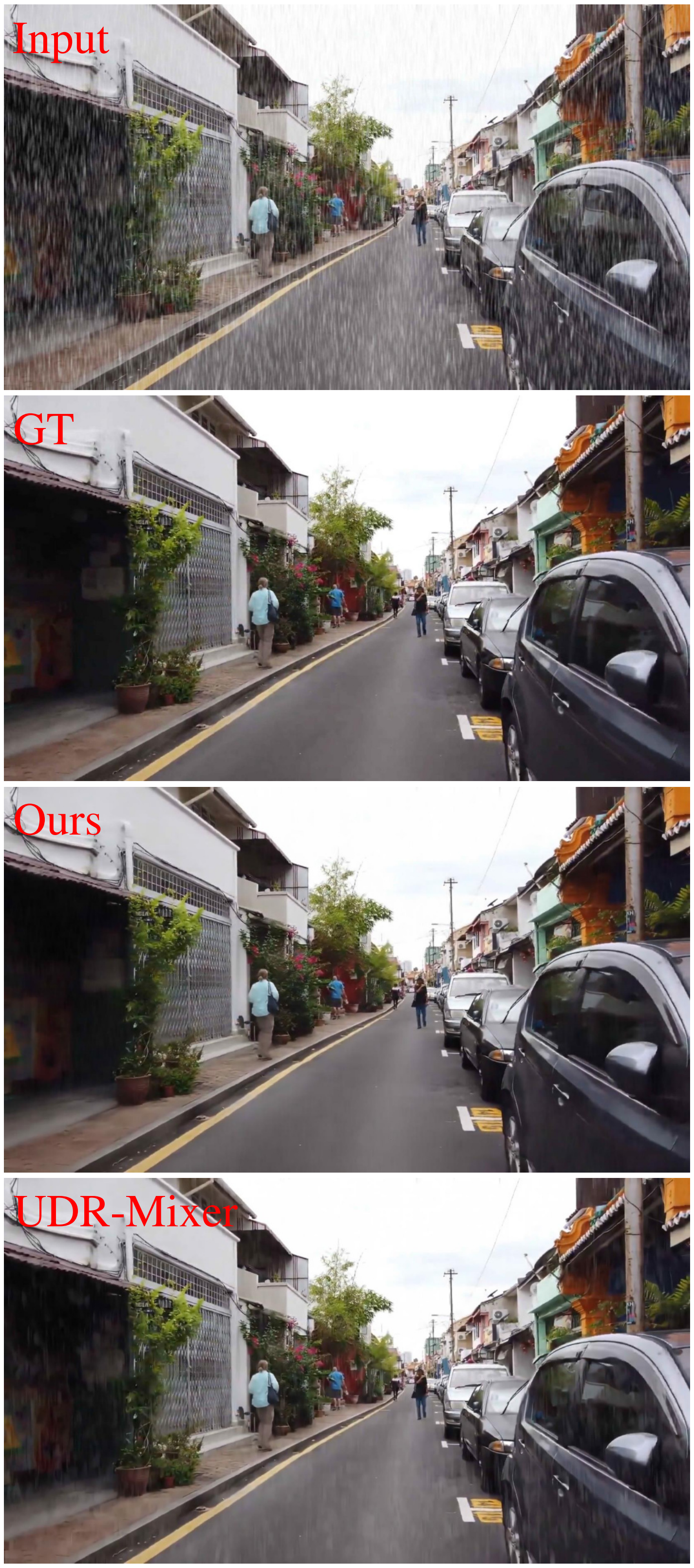

Figure 10. Visual comparison with SOTA method.

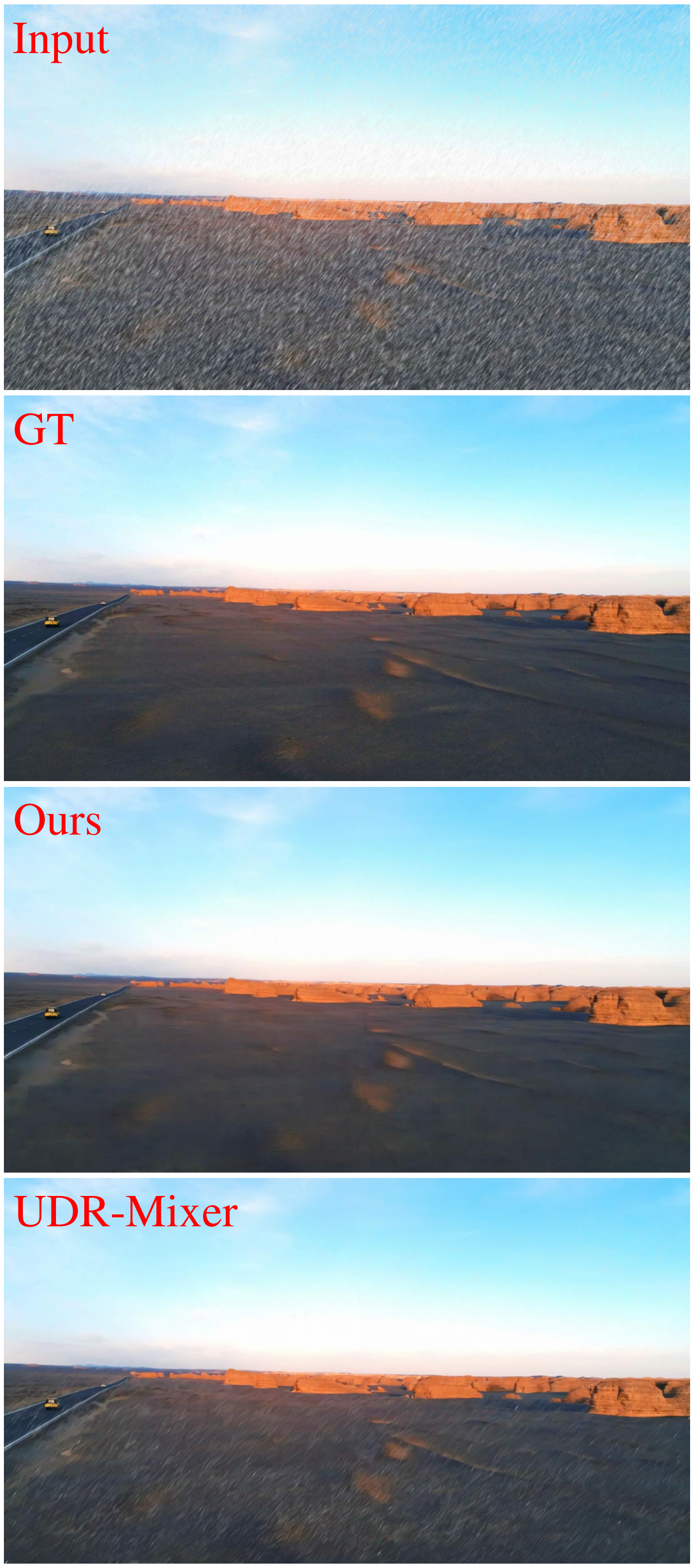

Figure 11. Visual comparison with SOTA method.

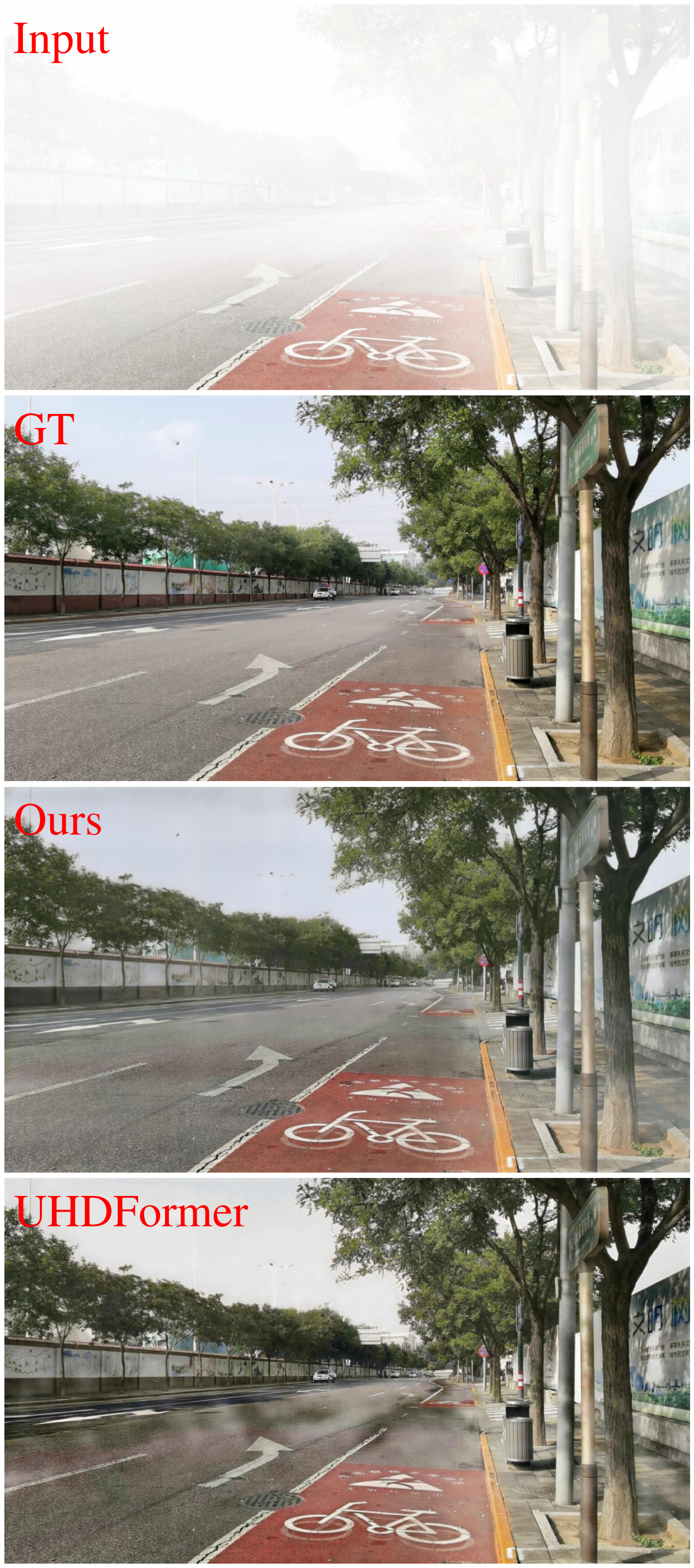

Figure 12. Visual comparison with SOTA method.

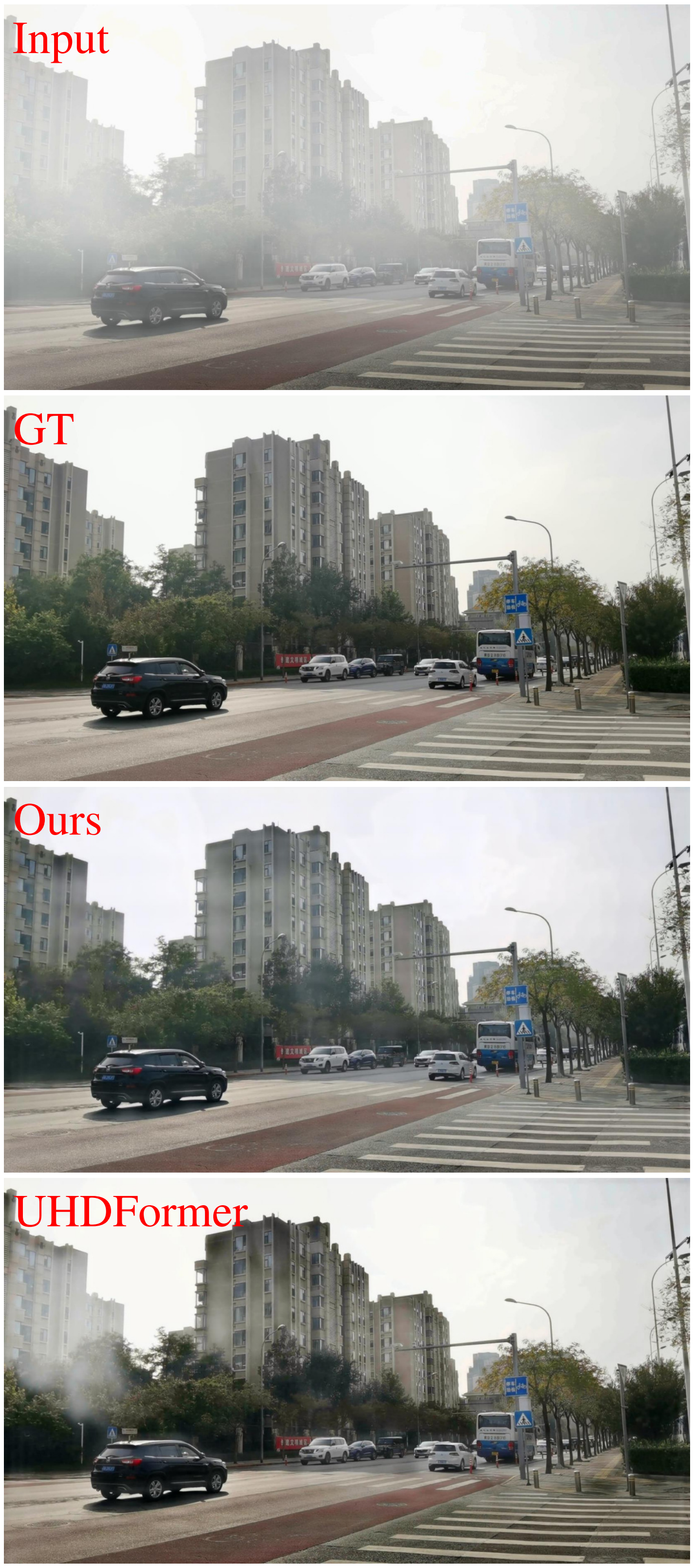

Figure 13. Visual comparison with SOTA method.

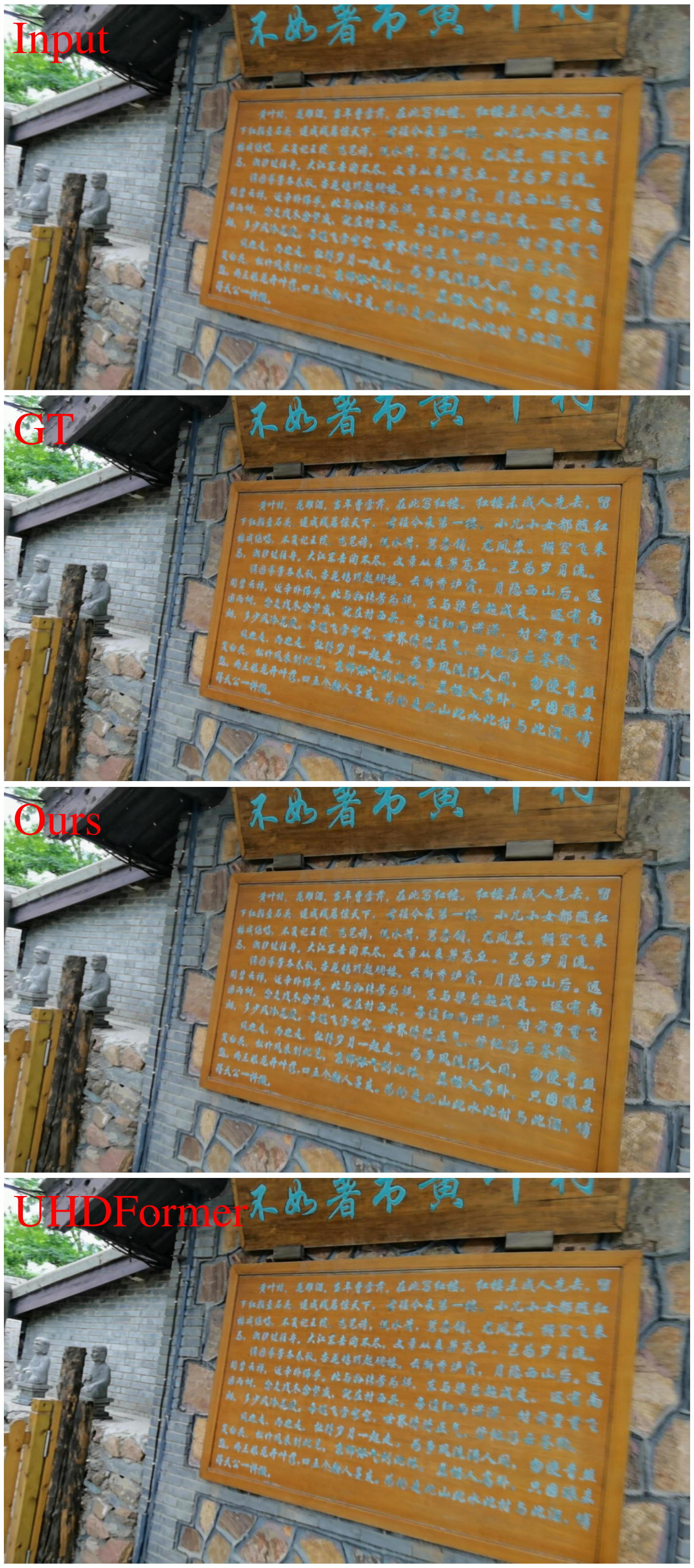

Figure 14. Visual comparison with SOTA method.

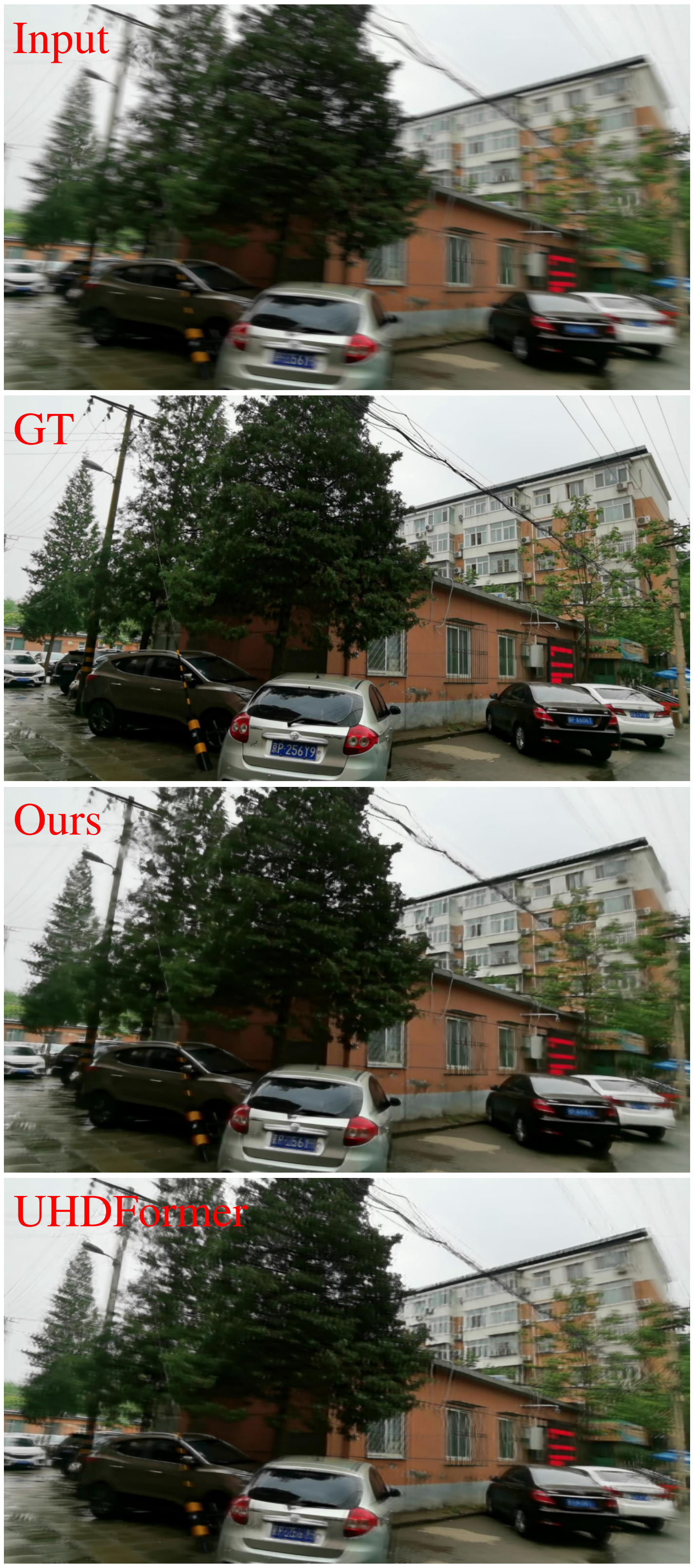

Figure 15. Visual comparison with SOTA method.