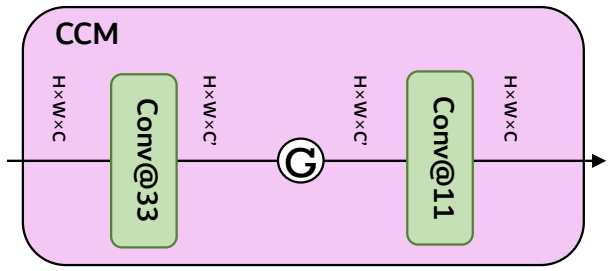


Figure 9. **CCM(Convolutional Channel Mixer)** proposed in SAFMN [40].

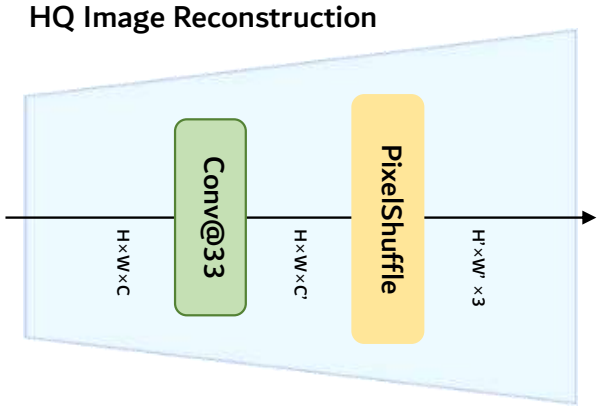


Figure 10. High Quality Image Reconstruction module.

A. CCM and HQ Image Reconstruction

CCM instead of MLP. Since the feed-forward network (FFN) in the original transformer [45] is a fully connected layer, we assume that using it in ViT might disrupt the spatial information of the features. Therefore, we apply the convolutional channel mixer (CCM) [40] instead, an FFN based on FMBConv [42], to preserve spatial information. CCM is a module that mixes each convolution channel. Specifically, the features pass through two convolution layers. The first layer has a 3×3 kernel and expands the channels. Then, GELU [15] is applied for non-linear mapping. Finally, a convolution layer with a 1×1 kernel restores the channels to their original state. In our method, the features pass through Layer Normalization [2], LMLT, and another Layer Normalization before being input to CCM [40]. Detailed structure can be seen in Figure 9.

HQ Image Reconstruction. PixelShuffle [38] is a technique effective for converting low-resolution images to high-resolution ones. The core idea of PixelShuffle is to rearrange the channel dimension of the image tensor into the spatial dimension to increase spatial resolution. This technique spatially expands the information contained in each channel, thereby increasing the resolution.

Specifically, the input tensor is first processed through an initial convolutional layer, which increases the number of channels. PixelShuffle then divides these channels by the square of the upscaling factor and rearranges them spatially to enhance the resolution of the final output image. For example, with an upscaling factor of 2, four channels are transformed into a 2×2 spatial block, effectively doubling the image size.

This process can be summarized as follows. First, `nn.Conv2d`(dim, $3 \times upscaling\_factor^2$, 3, 1, 1) is applied to increase the number of channels in the input tensor. Next, `nn.PixelShuffle`($upscaling\_factor$) rearranges the channels into the spatial dimensions, thereby increasing the image resolution. This method enables HQ Image Reconstruction module to enhance the input image's resolution and restore fine details effectively. The detailed structure is shown in Figure 10.

Table 8. Quantitative comparison of SwinIR-light, CAMixer, and the proposed model on Test4k and Test8k. The values for #GPU Mem and #Time indicate the memory usage and inference time when processing Test4k (left) and Test8k (right), respectively. A dash ("-") denotes that the measurement is not feasible due to GPU memory limitations on an RTX 3090 GPU. The best results are highlighted in **bold**.

Method	#GPU Mem [M]	#Time [ms]	Test4k	Test8k
SwinIR-light [25]	2997.69 / -	1824.85 / -	27.79	33.67
CAMixerSR-M [48]	2266.56 / 14866.0	380.34 / 2027.1	27.80	33.72
LMLT-Large(Ours)	**1830.43 / 10682.3**	**233.68 / 1155.7**	**27.82**	**33.75**

B. Realworld Scenario

Comparisons on LMLT with Real-world Scenario. In Table 8, we analyze the performance of our model on large-image SR tasks, such as Test4K and Test8K, using SwinIR-light and CAMixerSR as baselines on an RTX 3090 GPU. Here, the GPU memory usage and inference time are measured based on the inference on the Test4k and Test8k datasets. As shown in the Table 8, our proposed LMLT-Large model reduces memory requirements on the Test4k dataset by 39% and inference time by 87% compared to SwinIR-light [25], while also reducing memory usage by 19% and inference time by 38% compared to CAMixerSR-M [48], all while achieving better performance. Additionally, on the Test8k dataset, SwinIR-light cannot run due to insufficient memory, whereas CAMixerSR-M is executable. However, the proposed model reduces GPU memory usage by 28% and inference time by 43% compared to CAMixer-M, while maintaining a clear performance advantage.

Table 9. Performance results when the low-to-high element-wise connection removed. Better results are highlighted.

Scale	Method	Set5	Set14	B100	Urban100	Manga109
×2	LMLT-Tiny	**38.01**/0.9606	**33.59/0.9183**	**32.19**/0.8999	**32.04/0.9273**	**38.90**/0.9775
	LMLT-Tiny w/o connection	38.00/0.9606	33.58/0.9181	32.18/0.8999	31.99/0.9268	38.88/0.9775
	LMLT-Large	38.18/0.9612	**33.96/0.9212**	32.33/0.9017	32.75/0.9336	39.41/0.9786
	LMLT-Large w/o connection	**38.19/0.9613**	33.83/0.9199	32.33/**0.9018**	32.75/**0.9338**	**39.43/0.9787**
×3	LMLT-Tiny	34.36/0.9271	**30.37/0.8427**	**29.12/0.8057**	**28.10/0.8503**	**33.72**/0.9448
	LMLT-Tinyw/o connection	**34.40/0.9272**	30.35/0.8425	29.11/0.8056	28.05/0.8496	33.71/0.9448
	LMLT-Large	34.64/0.9293	**30.60**/0.8471	29.26/0.8097	**28.72/0.8626**	34.43/0.9491
	LMLT-Largew/o connection	34.64/0.9293	30.59/**0.8473**	29.26/0.8097	28.69/0.8622	34.43/**0.9492**
×4	LMLT-Tiny	**32.19/0.8947**	28.64/0.7823	27.60/**0.7369**	**26.08/0.7838**	30.60/**0.9083**
	LMLT-Tiny w/o connection	32.16/0.8944	28.64/0.7823	27.60/0.7368	26.04/0.7827	**30.64**/0.9078
	LMLT-Large	**32.48**/0.8987	28.87/**0.7879**	27.75/0.7421	26.63/**0.8001**	**31.32**/0.9163
	LMLT-Large w/o connection	32.47/0.8987	**28.88**/0.7877	27.75/**0.7422**	26.63/0.7999	31.30/0.9163

Table 10. The comparison table between SAFMN and its variant with low-to-high element-wise sum added. For each model, the better results are highlighted in **bold**.

Scale	Method	Set5	Set14	B100	Urban100	Manga109
×2	SAFMN [40]	**38.00/0.9605**	**33.54/0.9177**	32.16/0.8995	31.84/0.9256	38.71/0.9771
	SAFMN [40] $w/$ connection	37.99/0.9604	33.52/0.9174	**32.17/0.8996**	**31.86/0.9257**	**38.77/0.9773**
×3	SAFMN [40]	34.34/0.9267	**30.33**/0.8418	29.08/0.8048	27.95/0.8474	33.52/0.9437
	SAFMN [40] $w/$ connection	34.34/**0.9269**	30.32/0.8418	**29.09/0.8049**	**27.96/0.8476**	**33.55/0.9438**
×4	SAFMN [40]	**32.18/0.8948**	**28.60/0.7813**	27.58/**0.7359**	**25.97/0.7809**	30.43/0.9063
	SAFMN [40] $w/$ connection	32.10/0.8937	28.59/0.7812	27.58/0.7358	25.96/0.7799	**30.44**/0.9063

C. Low-to-high connection and Pooling

Effects of Low-to-high connection. In the Table 5 and Table 9, we confirm performance differences when the low-to-high connection is not applied to LMLT. Notably, at LMLT-Tiny, we observe performance improvements across all scales, particularly on the Urban100 dataset. Inspired by this, we also apply low-to-high connections between heads in SAFMN [40] and verify the experimental results. Table 10 shows that adding low-to-high connection to the upper head in SAFMN [40] does not yield significant performance differences. Moreover, at the $\times 4$ scale, the SSIM for the Urban100 [18] and Set5 [3] datasets decreases by 0.0010 and 0.0011, respectively, indicating a reduction in performance.

We then visualize the features of LMLT-Tiny to understand the effect of the low-to-high connection. Each column of Figure 11 illustrates the original image, the aggregated feature visualization of LMLT-Tiny combining all heads 11(a), and the aggregated feature visualization of LMLT-Tiny without low-to-high connection 11(b). In 11(b), the images show pronounced boundaries in areas such as stairs, buildings, and the sky. In contrast, 11(a) shows these boundaries as less pronounced. This demonstrates that the low-to-high connection can address the border communication issues inherent in WSA.

To further validate the architectural benefits of the low-to-high connection, we visualize the Layer Activation Maps (LAM) for LMLT-Large with and without this component(Fig 12). This analysis provides clear evidence that the connection enables the model to capture features from a significantly wider receptive field. As further substantiated in Figure 14, this capability allows our proposed architecture to reference a broader spatial context than even standard Multi-Head Self-Attention, demonstrating its superior effectiveness in aggregating global information.

Effects of Pooling. In this section, we experiment with the efficiency of our LMLT, which varies spatial size for each head. Unlike LMLT, which divides features by head and then applies query, key, and value mapping while varying the spatial size for each feature, here we keep the spatial size of all features the same without reduction, and therefore, the low-to-high connection is not applied. The results of this experiment and the proposed LMLT-Tiny model can be found in Table 11. Experimental results show that LMLT-Tiny outperforms the model without pooling across all datasets, indicating that harmoniously combining local and global information is more effective than merely retaining spatial information.

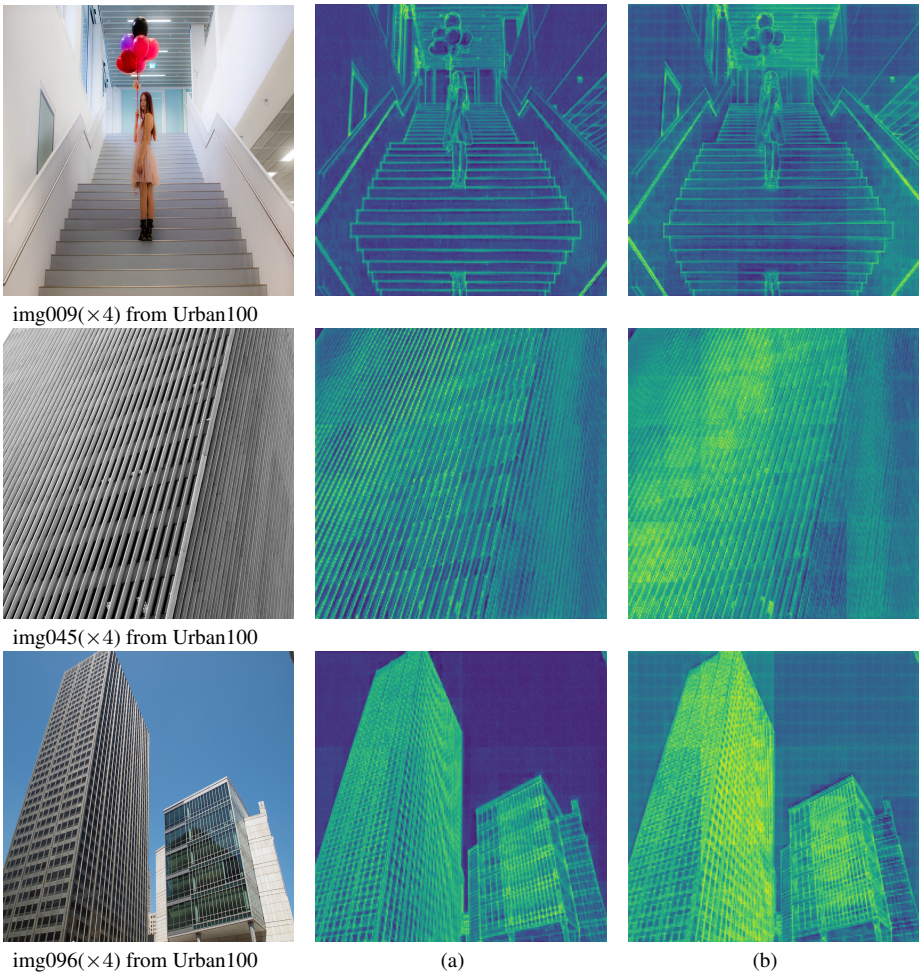


Figure 11. Visualization of features with low-to-high connection(a) and without connection(b) on Urban100×4. As shown in the images, without the low-to-high connection, the boundaries between windows are clearly visible.

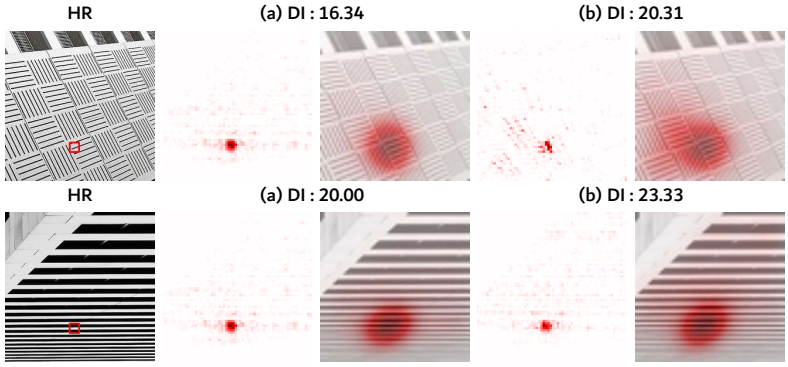


Figure 12. Visualization of LAM-Large. From left to right: (a) LMLT-Large without low-to-high connection and (b) LMLT-Large.

Subsequently, we investigate the reason behind this through feature visualization. Figure 13 visualizes the features when no pooling is applied to any head in LMLT. The leftmost image is the original Urban100 [18] image. Figure 13(a) shows the aggregated features of all heads in LMLT-Tiny. Column Figure 13(b) visualizes the features

Table 11. Performance with or without pooling and merging. Best results are highlighted in **bold**.

Scale	Method	#Params	#FLOPs	Set5	Set14	B100	Urban100	Manga109
×2	LMLT-Tiny	239K	59G	**38.01/0.9606**	**33.59/0.9183**	**32.19/0.8999**	**32.04/0.9273**	**38.90/0.9775**
	LMLT-Tiny w/o pool	239K	67G	37.98/0.9605	33.56/0.9178	32.16/0.8996	31.87/0.9255	38.79/0.9773
	LMLT-Tiny w/o pool and merge	229K	64G	37.95/0.9604	33.51/0.9173	32.14/0.8993	31.76/0.9245	38.68/0.9771
×3	LMLT-Tiny	244K	28G	**34.36/0.9271**	**30.37/0.8427**	**29.12/0.8057**	**28.10/0.8503**	**33.72/0.9448**
	LMLT-Tiny w/o pool	244K	32G	**34.36**/0.9270	30.34/0.8421	29.10/0.8051	28.02/0.8488	33.66/0.9445
	LMLT-Tiny w/o pool and merge	234K	31G	34.28/0.9265	30.31/0.8417	29.07/0.8044	27.94/0.8467	33.55/0.9438
×4	LMLT-Tiny	251K	15G	**32.19/0.8947**	**28.64/0.7823**	**27.60/0.7369**	**26.08/0.7838**	**30.60/0.9083**
	LMLT-Tiny w/o pool	251K	17G	32.12/0.8940	28.61/0.7820	27.58/0.7362	26.01/0.7815	30.51/0.9074
	LMLT-Tiny w/o pool and merge	240K	17G	32.07/0.8934	28.60/0.7817	27.56/0.7355	25.95/0.7795	30.45/0.9064

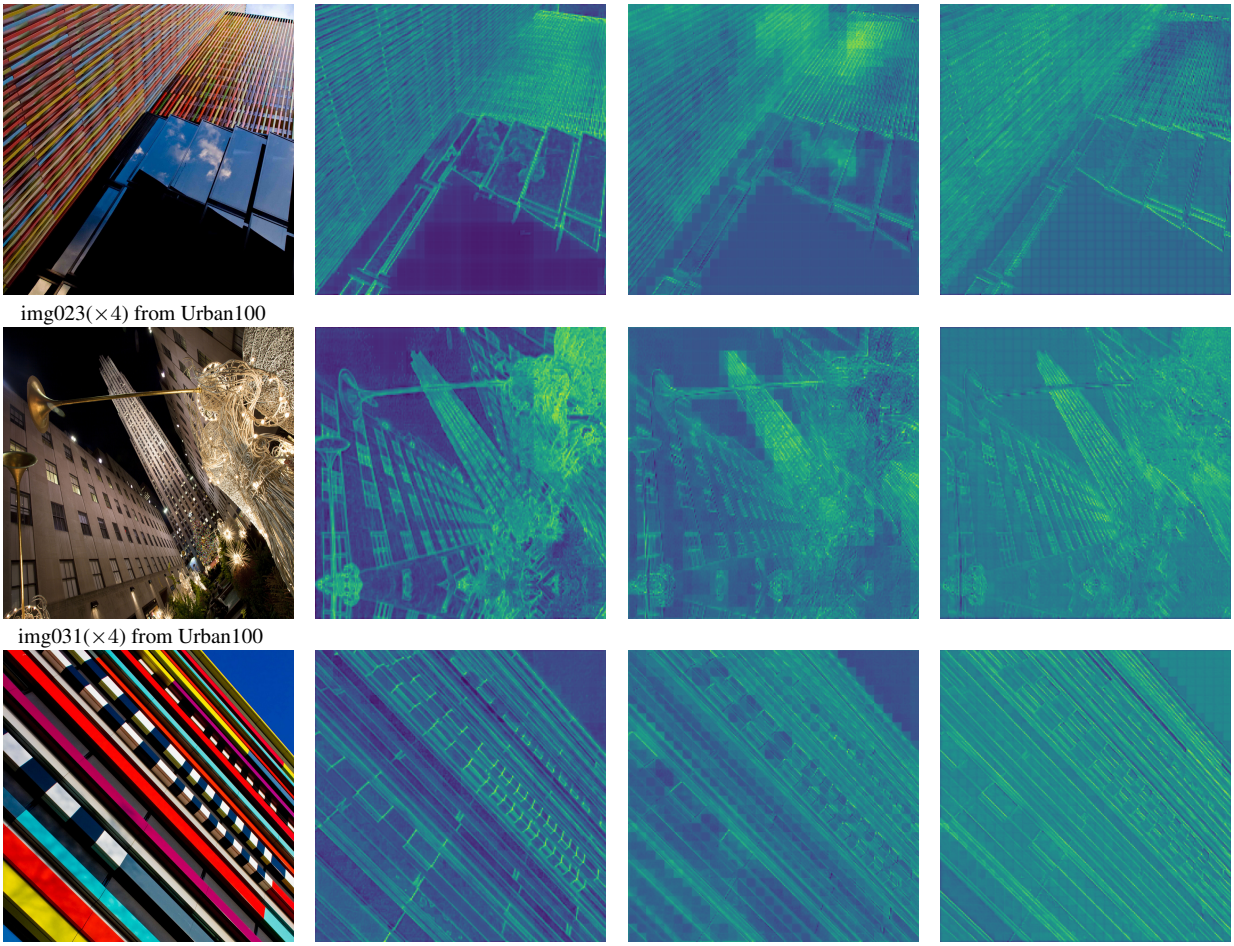


Figure 13. Comparison without pooling on Urban100 ×4 scale. From left to right: (a) LMLT with pooling applied, (b) without any pooling, (c) without pooling and without multiplication by activation. In (b) and (c), the boundaries between windows are visible.

without pooling, and Figure 13(c) visualizes the features without both pooling and merging, all at the ×4 scale. In 13(b) and 13(c), grid patterns are evident across the images, indicating that the disadvantages of being limited to local windows outweigh the benefits of maintaining the original spatial size.

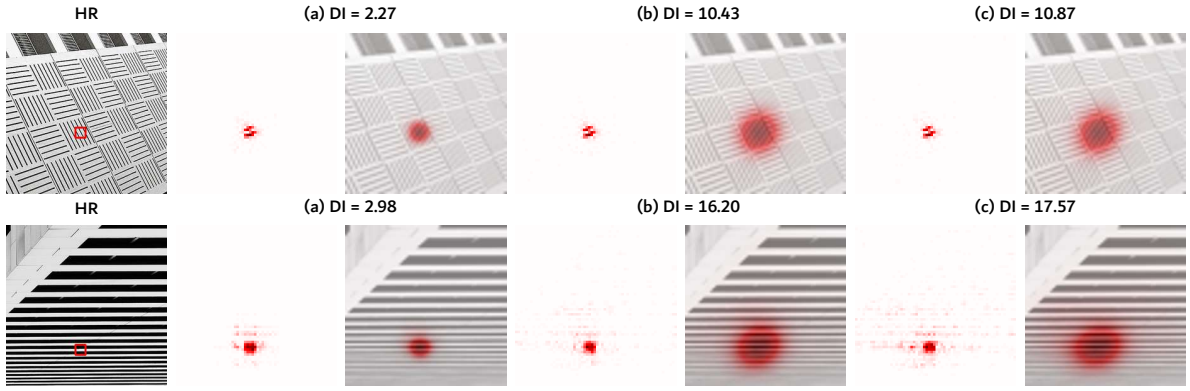


Figure 14. Visualization of LAM-Tiny. From left to right: (a) LMLT-Tiny without pooling, (b) LMLT-Tiny without low-to-high connection and (c) LMLT-Tiny.

Additionally, we compare the LAM of our proposed model, LMLT-Tiny (Figure 14(c)), with a version of the model that does not include pooling for each head (Figure 14(a)), and a version where the low-to-high connection is removed (Figure 14(b)), demonstrating that our proposed model effectively references a broader region. The results show that, even when the spatial size of the model is maintained without pooling, it fails to process information from a wider area. Moreover, the low-to-high connection proves to be effective in enabling the model to capture information from a larger region.

D. LMLT with Multi-Head Self Attention

In this section, we demonstrate how our proposed LMLT is more efficient than traditional Multi-Head Self-Attention (MHSA) across various datasets. Table 12 shows that the proposed Low-to-high Multi-Level Self Attention reduces maximum GPU memory usage while maintaining performance. At each scale, our model achieves an average reduction of 8% in the number of parameters, 18% in FLOPs, and 59% in memory usage compared to LMLT with MHSA. Meanwhile, it achieves increases in SSIM of 0.0009, 0.0008, and 0.0002 on the Urban100 dataset, respectively. Furthermore, Figure 15 uses the LAM [12] to illustrate that proposed model effectively references a broader area of information compared to MHSA.

E. Analysis of FLOPs

This section analyzes the relationship between theoretical computational cost (FLOPs) and practical execution time for the core modules of our LMLT-Large model and the SwinIR-Light [25] baseline.

As detailed in Table 13, our analysis of SwinIR-Light [25] indicates that its self-attention module accounts for approximately 50% of the total FLOPs, while the MLP contributes about 32%. However, a significant discrepancy appears in the runtime analysis. Across all scales, the self-attention module's execution time shows a significant difference compared to the MLP, ranging from a minimum of three times to well over ten times longer. This identifies the standard self-attention operation as the primary computational bottleneck, as its runtime is substantially higher than its theoretical FLOP count would suggest. This discrepancy arises because the runtime of self-attention is dominated by the cost of large matrix multiplications (QK^T) and ($Attention \times V$) and the handling of large intermediate tensors (Q, K, V, and the attention map). These operations incur substantial GPU memory access latency in addition to their quadratic computational complexity with respect to spatial size, resulting in poor scaling between FLOPs and actual runtime.

Our proposed LMLT architecture, however, is designed to address this specific bottleneck. By replacing the standard monolithic self-attention with four parallel, spatially downscaled streams, the attention mechanism in LMLT-Large constitutes only 8.5% of the total FLOPs (an over 80% reduction compared to SwinIR-Light), and its runtime is correspondingly reduced by approximately 89%. Although the CCM module is responsible for nearly 90% of the model's FLOPs, executes approximately 30% faster than our attention module. This demonstrates the practical efficiency of our proposed attention design, which is not fully captured by FLOPs alone.

Additionally, in Figure 16, we visualize the peak GPU memory usage of each module during a $\times 4$ upscaling task for both LMLT-Large and SwinIR-Light [25]. The memory footprint was measured at the end of each module's execution step using `torch.cuda.max_memory_allocated`.

For our proposed LMLT-Large, the memory profile shows a controlled and stable pattern. The CCM module records the highest peak memory at approximately 180MB, slightly higher than the LHSB module's peak

Table 12. Comparison of the results between LMLT-Tiny and LMLT-Tiny with Multi-Head Self-Attention (MHSA) applied instead of Multi-Level Self-Attention. Better results are highlighted.

Scale	Method	#Param	#FLOPs	GPU Mem	Set5	Set14	B100	Urban100	Manga109
$\times 2$	LMLT-Tiny	**239K**	**59G**	**324.01M**	38.01/0.9606	**33.59/0.9183**	**32.19/0.8999**	**32.04/0.9273**	**38.90**/0.9775
	LMLT-T $w/$ MHSA	260K	72G	789.67M	38.01/**0.9607**	33.57/0.9178	32.18/0.8998	31.97/0.9264	38.84/0.9775
$\times 3$	LMLT-Tiny	**244K**	**28G**	**151.96M**	34.36/**0.9271**	30.37/0.8427	**29.12/0.8057**	**28.10/0.8503**	**33.72**/0.9448
	LMLT-T$w/$ MHSA	265K	34G	369.97M	34.36/0.9270	**30.38/0.8428**	29.11/0.8056	28.06/0.8495	33.68/0.9448
$\times 4$	LMLT-Tiny	**251K**	**15G**	**81.44M**	**32.19/0.8947**	28.64/0.7823	27.60/0.7369	26.08/**0.7838**	30.60/0.9083
	LMLT-T $w/$ MHSA	271K	19G	198.356M	32.15/0.8945	**28.66/0.7829**	27.60/**0.7370**	26.08/0.7836	**30.61/0.9087**

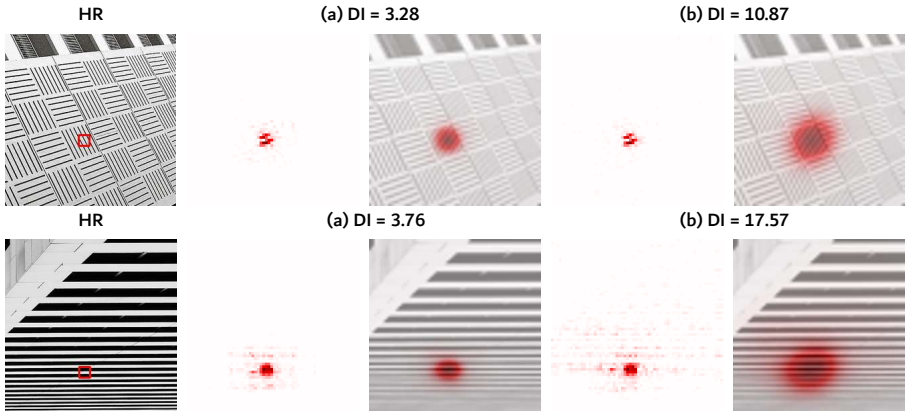


Figure 15. Visualization of LAM-Tiny. From left to right: (a) LMLT-Tiny with Multi-Head Self Attention, (b) LMLT-Tiny.

of around 170MB, due to differences in their internal processing. While LHSB processes features in parallel across downscaled channels, CCM employs a channel expansion layer (`Conv2d(D, D*growth_rate)`) on the full-dimensional features, which transiently requires more memory. The other modules, such as the Shallow Extractor and HQ Image Reconstruction, have a negligible memory footprint.

In sharp contrast, the profile of SwinIR-Light [25] reveals that the Self-Attention module is a major memory bottleneck. The Self-Attention block produces sharp, periodic spikes, demanding more than 300 MB of memory. The subsequent MLP block peaks at around 170 MB, roughly half the Self-Attention's requirement. Although the MLP also contains a channel expansion structure, it lacks the multiple linear projections and matrix multiplications on large, full-dimensional feature maps that make Self-Attention so memory-intensive. This clearly illustrates the inherently memory-hungry nature of ViT-based architectures.

F. Impact of Blocks, Channels, and Heads

In this section, we analyze how the performance of our proposed model changes based on the number of blocks, heads and channels.

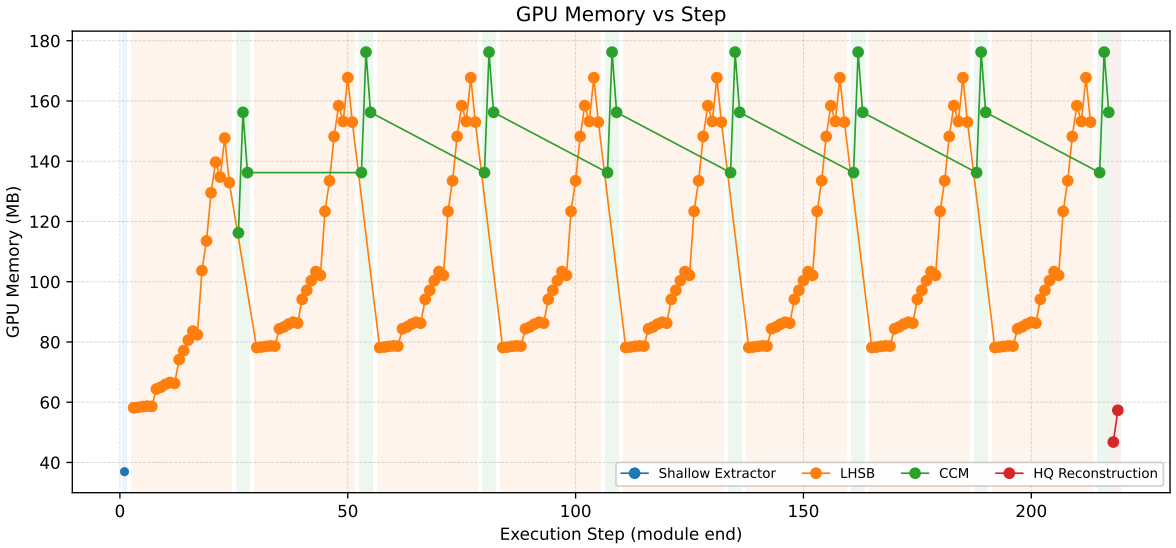
Impact of Number of Blocks. First, We evaluate the performance by varying the number of blocks to 4, 6, 8, 10, and 12. Experiments are conducted on $\times 2$ scale and the performance is evaluated using benchmark datasets, and analyzed in terms of the number of parameters, FLOPs, GPU memory usage, and average inference time.

As shown in Table 14, the increase in the number of parameters, FLOPs and inference time tends to be proportional to the number of blocks, and performance also gradually improves. For the Manga109 [36] dataset, as the number of blocks increases from 4 to 12 in increments of 2, PSNR increases by 0.27 db, 0.16 db, 0.10 db, and 0.10 db, respectively. Interestingly, despite the increase in the number of blocks from 4 to 12, the GPU memory usage remains almost unchanged. While the number of parameters nearly triples, the GPU memory usage remains stable, 323.5M to 324.5M. We observe the overall increase in PSNR with the increase in the number of blocks and designate the model with 8 blocks as LMLT-Tiny and the model with 12 blocks as LMLT-Small.

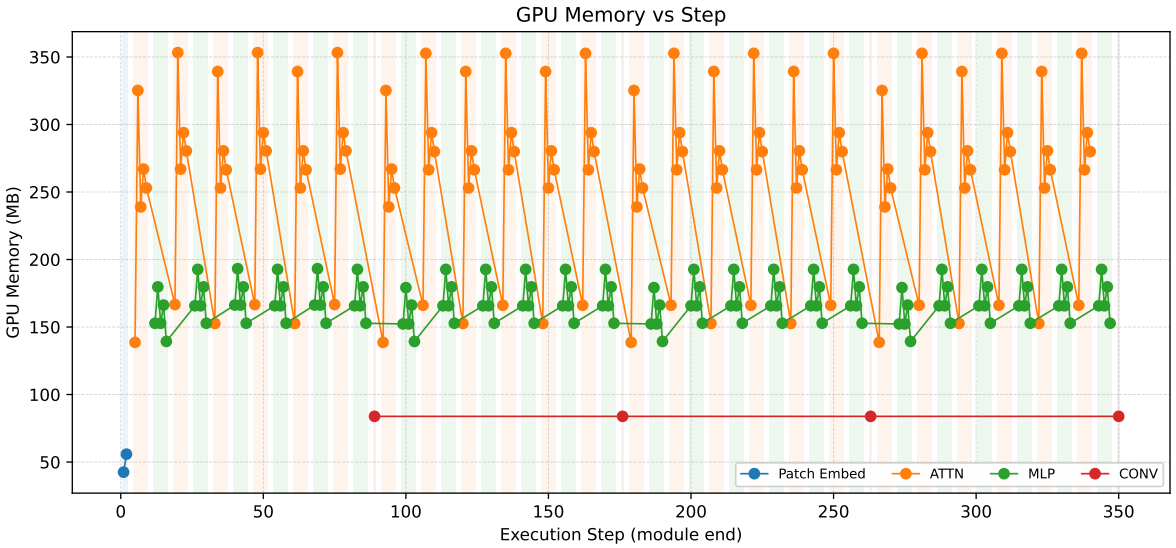
Impact of Number of Channels. Next, we evaluate how performance changes with the number of channels. Similar to the performance evaluation based on the number of blocks, this experiment evaluates performance using benchmark datasets, the number of parameters, FLOPs, GPU memory usage, and average inference time as performance

Table 13. Analysis of the FLOPs and runtime of Self-Attention and MLP(CCM [40]) modules.

scale	method	Attention Runtime	Attention FLOPs	MLP(CCM) Runtime	MLP(CCM) FLOPs
×2	SwinIR-Light [25]	1084.57ms	122.1G (50.0%)	89.33ms	79.6G(32.6%)
	LMLT-Large(Ours)	**68.97**ms	**26.2G (8.5%)**	**48.99**ms	**277.4G(90.6%)**
×3	SwinIR-Light [25]	336.00ms	54.9G (49.6%)	36.55ms	35.8G(32.3%)
	LMLT-Large(Ours)	**32.72**ms	**12.2G (8.5%)**	**23.58**ms	**129.5G(89.9%)**
×4	SwinIR-Light [25]	185.23ms	31.2G (49.1%)	48.99ms	20.4G(32.0%)
	LMLT-Large(Ours)	**22.66**ms	**6.5G (8.4%)**	**13.22**ms	**69.4G(88.8%)**



(a) LMLT-Large



(b) SwinIR-light

Figure 16. The memory consumption of each execution step on (a) LMLT-Large and (b) SwinIR-light.

metrics.

As shown in Table 15, LMLT's performance increases with channels, along with parameters and FLOPs. However, unlike the variations in the number of blocks, increasing the

Table 14. Performance difference of LMLT based on the number of blocks.

#Block	#Params	#FLOPs	#GPU Mem	#AVG Time	Set5	Set14	B100	Urban100	Manga109
4	122K	30G	323.52M	29.75ms	37.88/0.9601	33.40/0.9166	32.06/0.8984	31.49/0.9217	38.47/0.9765
6	181K	44G	323.77M	43.51ms	37.94/0.9601	33.52/0.9175	32.15/0.8995	31.82/0.9253	38.74/0.9771
8	239K	59G	324.01M	57.37ms	38.01/0.9606	33.59/0.9183	32.19/0.8999	32.04/0.9273	38.90/0.9775
10	298K	73G	324.26M	70.55ms	38.05/0.9608	33.66/0.9188	32.22/0.9003	32.17/0.9286	39.00/0.9778
12	357K	88G	324.5M	84.22ms	38.05/0.9608	33.65/0.9187	32.24/0.9006	32.31/0.9298	39.10/0.9780

Table 15. Performance difference of LMLT based on the number of channels.

#Channel	#Params	#FLOPs	#GPU MEM	#AVG Time	Set5	Set14	B100	Urban100	Manga109
24	109K	27G	255.77M	38.34ms	37.91/0.9602	33.44/0.9169	32.09/0.8988	31.62/0.9231	38.58/0.9768
36	239K	59G	324.01M	57.37ms	38.01/0.9606	33.59/0.9183	32.19/0.8999	32.04/0.9273	38.90/0.9775
48	420K	103G	460.32M	65.66ms	38.06/0.9609	33.67/0.9189	32.25/0.9007	32.33/0.9299	39.14/0.9780
60	652K	158G	567.75M	81.64ms	38.10/0.9610	33.76/0.9201	32.28/0.9012	32.52/0.9316	39.24/0.9783
72	935K	226G	684.82M	108.74ms	38.17/0.9612	33.83/0.9205	32.32/0.9016	32.65/0.9329	39.36/0.9786
84	1,270K	306G	717.31M	123.07ms	38.18/0.9612	33.96/0.9212	32.33/0.9017	32.75/0.9336	39.41/0.9786

number of channels results in a more significant increase in the number of parameters, FLOPs, and memory usage. Inference time, however, increases proportionally with the number of channels. For instance, with 36 channels, the average inference time is 57.16ms, and when doubled, it requires approximately 108.74ms, nearly twice the time. As the number of channels increases from 24 to 84 in increments of 12, the PSNR on the Urban100 [18] dataset increases by 0.42 db, 0.29 db, 0.19 db, 0.13 db, and 0.10 db, respectively. Based on the overall performance increase, we designate the model with 60 channels as the Base model and the model with 84 channels as the Large model. In this context, the Small model has an inference time about 3ms longer than the Base model, but it has fewer parameters, lower memory usage, and fewer FLOPs, thus justifying its designation.

Impact of Number of Heads. In this paragraph, we compare the performance differences based on the number of heads. In our model, as the number of heads decreases, the channel and the number of downsizing operations for each head decrease. For example, in our baseline with 4 heads and 36 channels, the lowest head has a total of 9 channels and is pooled 3 times. However, if there are 2 heads, the lowest head has 18 channels and is pooled once. Additionally, the maximum pooling times and the number of heads are related to the number of windows and the amount of self-attention computation. According to equation 2, as the number of heads decreases, the computation increases. As a result, as the number of heads decreases, the number of parameters, FLOPs, and GPU memory usage increase.

As shown in Table 16, the performance with 4 heads and 3 heads is similar across all scales and test datasets. However, reducing the number of heads to 1 significantly degrades performance, despite an increase in parameters, FLOPs, and GPU memory usage. Notably, on the Urban100 [18] dataset at scale $\times 2$, the LMLT with 4 heads achieves a PSNR of 32.04 dB with 239K parameters, 59G FLOPs, and 324M memory usage. In contrast, with 1 head, the parameters increase to 270K, FLOPs to 75G, and memory usage to 437M, while the PSNR drops to 31.93 dB, a decrease of 0.11 dB. This highlights the inefficiency of relying on fewer heads despite increased computational demands. Additionally, when the scale is $\times 3$ and $\times 4$, the PSNR decreases by 0.05 dB and 0.04 dB, respectively. This suggests that while maintaining the spatial size of all features with a single head increases parameters, FLOPs, and channels per head, incorporating global and cross-window information is more beneficial for achieving better performance.

G. LMLT with the Other Components

Importance of Aggregation and Activation. We analyze the impact of aggregating features from each head using a 1×1 convolution or applying activation before multiplying with the original input. Results are shown in the 'Act / Aggr' row of Table 17. Without aggregation, PSNR decreases by 0.10 dB on the Urban100 [18] dataset. If features are directly output without applying activation and without multiplying with the original input, PSNR decreases by 0.12 dB. Omitting both steps leads to an even greater decrease of 0.22 dB, indicating that including both aggregation and activation is more efficient. Conversely, multiplying features directly to the original feature without the activation func-

Table 16. Performance difference of LMLT based on the number of heads. #Chan is the number of channels in each heads. Best results are highlighted.

Scale	#Levels	#Chan	#Params	#FLOPs	#GPU	Set5	Set14	B100	Urban100	Manga109
×2	1	36	270K	75G	437.21M	38.00/0.9606	33.58/0.9179	32.17/0.8997	31.93/0.9260	38.83 /0.9774
	2	18	250K	64G	385.96M	**38.01**/0.9606	**33.59**/0.9180	32.18/**0.8999**	32.02/0.9270	38.87/**0.9776**
	3	12	243K	60G	346.71M	38.00/0.9606	**33.59**/0.9182	**32.19/0.8999**	32.02/**0.9273**	38.88/0.9775
	4	9	239K	59G	324.01M	**38.01**/0.9606	**33.59/0.9183**	**32.19/0.8999**	**32.04/0.9273**	**38.90**/0.9775
×3	1	36	275K	35G	205.52M	34.37/0.9271	**30.39/0.8431**	29.11/0.8054	28.05/0.8495	33.71/0.9449
	2	18	255K	30G	181.14M	34.37/0.9271	30.37/0.8427	29.11/0.8056	28.05/0.8496	**33.73/0.9450**
	3	12	248K	29G	161.90M	**34.41/0.9273**	30.37/0.8426	**29.12/0.8059**	28.09/0.8502	**33.73**/0.9449
	4	9	244K	28G	151.96M	34.36/0.9271	30.37/0.8427	**29.12**/0.8057	**28.10/0.8503**	33.72/0.9448
×4	1	36	282K	19G	111.28M	32.14/0.8943	**28.65/0.7826**	27.60/0.7366	26.04/0.7825	30.57/0.9080
	2	18	261K	17G	98.69M	32.18/**0.8948**	28.63/**0.7826**	27.60/**0.7370**	26.07/0.7839	30.59/0.9085
	3	12	254K	16G	87.04M	**32.19**/0.8947	28.63/0.7821	27.60/0.7367	**26.08**/0.7834	30.58/0.9080
	4	9	251K	15G	81.44M	**32.19**/0.8947	28.64/0.7823	27.60/0.7369	**26.08/0.7838**	**30.60/0.9083**

tion improves performance by 0.1 dB.

Therefore, inspired by this, we experiment with a version of LMLT without GeLU [15] across various scales. Table 18 shows the results for our LMLT and the model without the activation function across different scales and channels. As shown, with 36 channels, there is minimal performance difference across all scales, with the largest being a 0.04 higher PSNR on the Set5 [3] $\times 4$ scale when GeLU [15] is removed. However, when expanded to 60 channels, our LMLT performs better on most benchmark datasets for both $\times 3$ and $\times 4$ scales. Specifically, on the $\times 4$ scale of the Urban100 [18] dataset, PSNR and SSIM are higher by 0.05 dB and 0.0013, respectively. This demonstrates that adding GeLU [15] after aggregating features is more beneficial for performance improvement.

Importance of Positional Encoding. Lastly, we examine the role of Positional Encoding (PE) in performance improvement. Results are shown in the 'PE' row of Table 17. Removing PE results in decreased performance across all benchmark datasets, notably with a PSNR drop of 0.06 dB and an SSIM decrease of 0.0006 on the Urban100 [18] dataset. Using RPE [29] results in a maximum PSNR increase of 0.03 dB on the Set14 [51] dataset, but has little effect on other datasets. Additionally, parameters and GPU memory increase by 5K and 45M, respectively.

H. Comparisons with Other Methods

Image Reconstruction comparisons. Here, we first compare the LMLT-Tiny and LMLT-Small with other CNN-based image super-resolution models such as CARN-m, CARN [1], EDSR-baseline [26], PAN [62], LAPAR-A [24], ECBSR-M16C64 [55], SMSR [46], Shuffle-Mixer [39], and SAFMN [40], so that we can demonstrate that our LMLT is not only lighter but also achieves superior performance compared to other CNN-based state-of-the-art models. Table 19 shows that our LMLT significantly reduces number of parameters and computation overheads while achieving substantial performance gains on various datasets. LMLT-Small performs well on most datasets, and the LMLT-Tiny also performs second and third best on the BSD100 [35] and Manga109 [36] datasets, except for the Manga109 $\times 4$ SSIM [49]. In particular, the number of parameters and FLOPs are the second smallest after SAFMN [40].

In Table 20, we compare our proposed LMLT model with ViT-based SR models, where LMLT-Base is evaluated against efficient SR models [4, 10, 11, 19, 27, 33, 61] and LMLT-Large is compared with lightweight SR models [4, 25, 28, 31, 47, 52, 56, 58, 63, 64]. As shown in Table 20, LMLT-Base achieves the best or second-best performance across almost all scales and datasets. Notably, on the Manga109 dataset, it demonstrates significant performance improvements, achieving 0.27dB, 0.29dB, and 0.29dB higher PSNR at scales $\times 2$, $\times 3$, and $\times 4$, respectively, while reducing the number of parameters by approximately 30% compared to the second-best model, NGswin [4].

Similarly, LMLT-Large also achieves the best or second-best performance across several scales and datasets, with particularly notable results on the Set14 dataset. In addition to this, we demonstrate that our model has a significant advantage in inference time and GPU memory usage in the next paragraph.

Memory and Running time Comparisons. In this paragraph, we present the memory usage and average inference time of our proposed LMLT compared to other super-resolution methods. Similar to the experimental setup in Table 2, #GPU Mem represents the maximum memory usage during inference, measured using PyTorch's

Table 17. Ablation studies on each component of our method at scale $\times 2$. LMLT-Tiny is used.

Ablation	Variants	#Param	#Flops	#GPU Mem	Set5	Set14	B100	Urban100	Manga109
Baseline	-	**239K**	**59G**	**324M**	**38.01/0.9606**	**33.59/0.9183**	**32.19/0.8999**	**32.04/0.9273**	**38.90/0.9775**
Act / Aggr	No Aggregation	229K	56G	324M	37.99/0.9606	33.55/0.9178	32.17/0.8997	31.94/0.9262	38.84/0.9774
	No Activation	239K	59G	324M	37.99/0.9605	33.55/0.9180	32.16/0.8997	31.92/0.9260	38.83/0.9774
	No Aggr, No Act	229K	56G	324M	37.95/0.9606	33.53/0.9175	32.15/0.8994	31.82/0.9250	38.73/0.9771
	GELU $\rightarrow$ None	239K	59G	324M	38.03/0.9606	33.60/0.9184	32.19/0.9000	32.05/0.9272	38.91/0.9776
PE	No PE	236K	59G	309M	37.98/0.9606	33.55/0.9176	32.18/0.8998	31.98/0.9267	38.86/0.9774
	LePE $\rightarrow$ RPE[29]	244K	59G	369M	38.02/0.9606	33.62/0.9182	32.20/0.9000	32.05/0.9275	38.90/0.9775

Table 18. Performance difference of LMLT with GELU and without GELU. The better results are highlighted in **bold**.

Scale	Ablation	#Channel	Set5	Set14	B100	Urban100	Manga109
$\times 2$	LMLT	36	38.01/0.9606	33.59/0.9183	32.19/0.8999	32.04/**0.9273**	38.90 /0.9775
	LMLT w/o GELU	36	**38.03**/0.9606	**33.60/0.9184**	32.19/**0.9000**	**32.05**/0.9272	**38.91/0.9776**
	LMLT	60	38.10/0.9610	33.76/**0.9201**	32.28/**0.9012**	**32.52/0.9316**	39.24/0.9783
	LMLT w/o GELU	60	38.10/0.9610	**33.80**/0.9200	32.28/0.9011	32.51/0.9315	**39.26**/0.9783
$\times 3$	LMLT	36	34.36/0.9271	**30.37/0.8427**	**29.12**/0.8057	**28.10/0.8503**	**33.72/0.9448**
	LMLT w/o GELU	36	**34.37/0.9272**	30.36/0.8425	29.11/0.8057	28.08/0.8502	33.71/0.9447
	LMLT	60	**34.58/0.9285**	**30.53/0.8458**	**29.21/0.8084**	**28.48/0.8581**	**34.18/0.9477**
	LMLT w/o GELU	60	34.53/0.9283	30.51/0.8457	29.20/0.8080	28.45/0.8576	34.17/0.9476
$\times 4$	LMLT	36	32.19/0.8947	**28.64/0.7823**	27.60/0.7369	26.08/**0.7838**	**30.60/0.9083**
	LMLT w/o GELU	36	**32.23/0.8949**	28.62/0.7820	27.60/0.7369	26.08/0.7836	30.59/0.9082
	LMLT	60	32.38/0.8971	**28.79/0.7859**	**27.70/0.7403**	**26.44/0.7947**	**31.09/0.9139**
	LMLT w/o GELU	60	**32.39/0.8973**	28.78/0.7858	27.69/0.7399	26.39/0.7934	31.04/0.9132

`torch.cuda.max_memory_allocated()`. #AVG Time indicates the average time taken to upscale a total of 50 random images by $\times 2$, $\times 3$, and $\times 4$ scales. The experiments were conducted three times, and the average inference time is reported. Each random image has sizes of 640×360 for $\times 2$ scale, 427×240 for $\times 3$ scale, and 320×180 for $\times 4$ scale.

As shown in Table 21, our proposed LMLT-Tiny uses less memory at all scales compared to all models except SAFMN [40]. Although LMLT-Small requires more inference time compared to other models, its GPU usage is almost similar to LMLT-Tiny, and its performance is significantly superior as demonstrated in Table 19.

Particularly, compared to EDSR [26], which has the next highest GPU memory usage after LMLT, LMLT-Tiny achieves a 44%, 72%, and 83% reduction in memory usage at scales $\times 2$, $\times 3$, and $\times 4$, respectively, while delivering 0.02dB, 0.09dB, and 0.06dB higher performance on the Set14 dataset. Similarly, LMLT-Small reduces memory usage by the same percentages at each scale, while outperforming EDSR on the Set14 dataset by 0.08dB, 0.19dB, and 0.16dB, respectively. These results highlight that the proposed model is not only lightweight but also achieves higher performance compared to CNN-based SR models.

Additionally, we discuss the memory usage and inference speed of LMLT-Base, LMLT-Large, and other ViT-based SR models [4, 11, 25, 28, 52, 56, 58, 63, 64]. We observe that LMLT-Base and LMLT-Large are highly efficient in terms of inference speed and memory usage compared to other ViT-based SR models. Specifically, compared to NGswin [4], LMLT-Base maintains similar performance while reducing memory usage by 61%, 62%, and 61% at scales $\times 2$, $\times 3$, and $\times 4$, respectively, and decreasing inference time by an average of 78%, 76%, and 78%. Similarly, compared to SwinIR-light [25], LMLT-Large achieves a 44%, 43%, and 46% reduction in memory usage at each scale, while reducing inference time by 87%, 80%, and 81%, respectively, while maintaining similar performance.

Notably, as discussed in Table 2, our most complex model, LMLT-Large, is lighter and faster than all ViT-based comparison models. Furthermore, as shown in Table 21, LMLT-Large achieves 40%, 38%, and 40% lower memory usage than HNCT, while reducing inference time by 65%, 50%, and 51% at scales $\times 2$, $\times 3$, and $\times 4$, respectively. Moreover, LMLT-Large achieves 0.31dB, 0.16dB, and 0.16dB higher PSNR on the Set14 dataset, demonstrating both the efficiency and effectiveness of the proposed model.

Qualitative Comparisons. In this paragraph, we examine the qualitative comparisons of the LMLT-Tiny model and

Table 19. Comparisons with existing methods. Best and second-best performance are in red and blue, and third-best is underlined. Unreported results are left blank.

Scale	Method	#Params	#FLOPs	Set5	Set14	B100	Urban100	Manga109
×2	CARN-M [1]	412K	91G	37.53/0.9583	33.26/0.9141	31.92/0.8960	31.23/0.9193	-
	CARN [1]	1,592K	223G	37.76/0.9590	33.52/0.9166	32.09/0.8978	31.92/0.9256	-
	EDSR-baseline [26]	1,370K	316G	37.99/0.9604	33.57/0.9175	32.16/0.8994	31.98/0.9272	38.54/0.9769
	PAN [62]	261K	71G	38.00/0.9605	33.59/0.9181	32.18/0.8997	32.01/0.9273	38.70/0.9773
	LAPAR-A [24]	548K	171G	38.01/0.9605	33.62/0.9183	32.19/0.8999	32.10/0.9283	38.67/0.9772
	ECBSR-M16C64 [55]	596K	137G	37.90/0.9615	33.34/0.9178	32.10/0.9018	31.71/0.9250	-
	SMSR [46]	985K	132G	38.00/0.9601	33.64/0.9179	32.17/0.8990	32.19/0.9284	38.76/0.9771
	ShuffleMixer [39]	394K	91G	38.01/0.9606	33.63/0.9180	32.17/0.8995	31.89/0.9257	38.83/0.9774
	SAMFN [40]	228K	52G	38.00/0.9605	33.54/0.9177	32.16/0.8995	31.84/0.9256	38.71/0.9771
	LMLT-Tiny(Ours)	239K	59G	38.01/0.9606	33.59/0.9183	32.19/0.8999	32.04/0.9273	38.90/0.9775
	LMLT-Small(Ours)	357K	88G	38.05/0.9608	33.65/0.9187	32.24/0.9006	32.31/0.9298	39.10/0.9780
×3	CARN-M [1]	415K	46G	33.99/0.9236	30.08/0.8367	28.91/0.8000	27.55/0.8385	-
	CARN [1]	1,592K	119G	34.29/0.9255	30.29/0.8407	29.06/0.8034	28.06/0.8493	-
	EDSR-baseline [26]	1,555K	160G	34.37/0.9270	30.28/0.8417	29.09/0.8052	28.15/0.8527	33.45/0.9439
	PAN [62]	261K	39G	34.40/0.9271	30.36/0.8423	29.11/0.8050	28.11/0.8511	33.61/0.9448
	LAPAR-A [24]	594K	114G	34.36/0.9267	30.34/0.8421	29.11/0.8054	28.15/0.8523	33.51/0.9441
	SMSR [46]	993K	68G	34.40/0.9270	30.33/0.8412	29.10/0.8050	28.25/0.8536	33.68/0.9445
	ShuffleMixer [39]	415K	43G	34.40/0.9272	30.37/0.8423	29.12/0.8051	28.08/0.8498	33.69/0.9448
	SAFMN [40]	233K	23G	34.34/0.9267	30.33/0.8418	29.08/0.8048	27.95/0.8474	33.52/0.9437
	LMLT-Tiny(Ours)	244K	28G	34.36/0.9271	30.37/0.8427	29.12/0.8057	28.10/0.8503	33.72/0.9448
	LMLT-Small(Ours)	361K	41G	34.50/0.9280	30.47/0.8446	29.16/0.8070	28.29/0.8544	33.99/0.9464
×4	CARN-M [1]	412K	33G	31.92/0.8903	28.42/0.7762	27.44/0.7304	25.62/0.7694	-
	CARN [1]	1,592K	91G	32.13/0.8937	28.60/0.7806	27.58/0.7349	26.07/0.7837	-
	EDSR-baseline [26]	1,518K	114G	32.09/0.8938	28.58/0.7813	27.57/0.7357	26.04/0.7849	30.35/0.9067
	PAN [62]	272K	28G	32.13/0.8948	28.61/0.7822	27.59/0.7363	26.11/0.7854	30.51/0.9095
	LAPAR-A [24]	659K	94G	32.15/0.8944	28.61/0.7818	27.61/0.7366	26.14/0.7871	30.42/0.9074
	ECBSR-M16C64 [55]	603K	35G	31.92/0.8946	28.34/0.7817	27.48/0.7393	25.81/0.7773	-
	SMSR [46]	1,006K	42G	32.12/0.8932	28.55/0.7808	27.55/0.7351	26.11/0.7868	30.54/0.9085
	ShuffleMixer [39]	411K	28G	32.21/0.8953	28.66/0.7827	27.61/0.7366	26.08/0.7835	30.65/0.9093
	SAFMN [40]	240K	14G	32.18/0.8948	28.60/0.7813	27.58/0.7359	25.97/0.7809	30.43/0.9063
	LMLT-Tiny(Ours)	251K	15G	32.19/0.8947	28.64/0.7823	27.60/0.7369	26.08/0.7838	30.60/0.9083
	LMLT-Small(Ours)	368K	23G	32.31/0.8968	28.74/0.7846	27.66/0.7387	26.26/0.7894	30.87/0.9117

other models on the Urban100 [18] ×4 scale. The comparison includes CARN [1], EDSR [26], PAN [62], ShuffleMixer [39], and SAFMN [40]. The results can be seen in Figure 17. As mentioned in section 4.1, we observe that our model reconstructs images with continuous stripes better than other models.

Additionally, we compare our proposed models LMLT-Base and LMLT-Large with IMDN [19], NGswin [4], SwinIR-light [25], and SwinIR-NG [4] on the Manga109 [36] dataset at×4 scale. As explained earlier in section 4.1, our model shows strength in areas with continuous lines compared to other models. Figure 18 illustrates the differences between our LMLT-Base, LMLT-Large and other state-of-the-arts models.

Table 20. Comparisons with our LMLT-Small, LMLT-Base, LMLT-Large and other Super-Resolution models on multiple benchmark datasets. Best and second-best performance are in red and blue color.

Scale	Method	#Params	#FLOPs	Set5	Set14	B100	Urban100	Manga109
×2	IMDN [19]	694K	156G	38.00/0.9605	33.63/0.9177	32.19/0.8996	32.17/0.9283	38.88/0.9774
	LatticeNet [33]	756K	170G	38.06/0.9607	33.70/0.9187	32.20/0.8999	32.25/0.9288	38.94/0.9774
	RFDN-L [27]	626K	146G	38.08/0.9606	33.67 /0.9190	32.18/0.8996	32.24/0.9290	38.95/0.9773
	SRPN-Lite [61]	609K	140G	38.10/0.9608	33.70/0.9189	32.25/0.9005	32.26/0.9294	-
	HNCT [11]	357K	82G	38.08/0.9608	33.65/0.9182	32.22/0.9001	32.22/0.9294	38.87/0.9774
	FMEN [10]	748K	172G	38.10/0.9609	33.75/0.9192	32.26/0.9007	32.41/0.9311	38.95/0.9778
	NGswin [4]	990K	140G	38.05/0.9610	33.79/0.9199	32.27/0.9008	32.53/0.9324	38.97/0.9777
	LMLT-Base(Ours)	652K	158G	38.10/0.9610	33.76/0.9201	32.28/0.9012	32.52/0.9316	39.24/0.9783
	SwinIR-light [25]	910K	244G	38.14/0.9611	33.86/0.9206	32.31/0.9012	32.76/0.9340	39.12/0.9783
	ESRT [31]	751K	-	38.03/0.9600	33.75/0.9184	32.25/0.9001	32.58/0.9318	39.12/0.9774
	ELAN [56]	621K	203G	38.17/0.9611	33.94/0.9207	32.30/0.9012	32.76/0.9340	39.11/0.9782
	SwinIR-NG [4]	1,181K	274G	38.17/0.9612	33.94/0.9205	32.31/0.9013	32.78/0.9340	39.20/0.9781
	SRformer-Light [64]	853K	236G	38.23/0.9613	33.94/0.9209	32.36/0.9019	32.91/0.9353	39.28/0.9785
	OSFFNet [47]	516K	83G	38.11/0.9610	33.72/0.9190	32.29/0.9012	32.67/0.9331	39.09/0.9780
	SMFANet+ [63]	480K	108G	38.19/0.9611	33.92/0.9207	32.32/0.9015	32.70/0.9331	39.46/0.9787
	LMLT-Large(Ours)	1,270K	306G	38.18/0.9612	33.96/0.9212	32.33/0.9017	32.75/0.9336	39.41/0.9786
×3	IMDN [19]	703K	72G	34.36/0.9270	30.32/0.8417	29.09/0.8046	28.17/0.8519	33.61/0.9445
	LatticeNet [33]	765K	76G	34.40/0.9272	30.32/0.8416	29.10/0.8049	28.19/0.8513	33.63/0.9442
	RFDN-L [27]	633K	66G	34.47/0.9280	30.35/0.8421	29.11/0.8053	28.32/0.8547	33.78/0.9458
	SRPN-Lite [61]	615K	63G	34.47/0.9280	30.38/0.8425	29.16/0.8061	28.22/0.8534	-
	HNCT [11]	363K	38G	34.47/0.9275	30.44/0.8439	29.15/0.8067	28.28/0.8557	33.81/0.9459
	FMEN [10]	757K	77G	34.45/0.9275	30.40/0.8435	29.17/0.8063	28.33/0.8562	33.86/0.9462
	NGswin [4]	1,007K	67G	34.52/0.9282	30.53/0.8456	29.19/0.8078	28.52/0.8603	33.89/0.9470
	LMLT-Base(Ours)	660K	75G	34.58/0.9285	30.53/0.8458	29.21/0.8084	28.48/0.8581	34.18/0.9477
	SwinIR-light [25]	918K	111G	34.62/0.9289	30.54/0.8463	29.20/0.8082	28.66/0.8624	33.98/0.9478
	ESRT [31]	751K	-	34.42/0.9268	30.43/0.8433	29.15/0.8063	28.46/0.8574	33.95/0.9455
	ELAN [56]	629K	90G	34.61/0.9288	30.55/0.8463	29.21/0.8081	28.69/0.8624	34.00/0.9478
	SwinIR-NG [4]	1,190K	114G	34.64/0.9293	30.58/0.8471	29.24/0.8090	28.75/0.8639	34.22/0.9488
	SRformer-Light [64]	861K	105G	34.67/0.9296	30.57/0.8469	29.26/0.8099	28.81/0.8655	34.19/0.9489
	OSFFNet [47]	524K	38G	34.58/0.9287	30.48/0.8450	29.21/0.8080	28.49/0.8595	34.00/0.9472
	SMFANet+ [63]	487K	48G	34.66/0.9292	30.57/0.8461	29.25/0.8090	28.67/0.8611	34.45/0.9490
	LMLT-Large(Ours)	1,279K	144G	34.64/0.9293	30.60/0.8471	29.26/0.8097	28.72/0.8626	34.43/0.9491
×4	IMDN [19]	715K	41G	32.21/0.8948	28.58/0.7811	27.56/0.7353	26.04/0.7838	30.45/0.9075
	LatticeNet [33]	777K	44G	32.18/0.8943	28.61/0.7812	27.57/0.7355	26.14/0.7844	30.54/0.9075
	RFDN-L [27]	643K	38G	32.28/0.8957	28.61/0.7818	27.58/0.7363	26.20/0.7883	30.61/0.9096
	SRPN-Lite [61]	623K	36G	32.24/0.8958	28.69/0.7836	27.63/0.7373	26.16/0.7875	-
	HNCT [11]	373K	22G	32.31/0.8957	28.71/0.7834	27.63/0.7381	26.20/0.7896	30.70/0.9112
	FMEN [10]	769K	44G	32.24/0.8955	28.70/0.7839	27.63/0.7379	26.28/0.7908	30.70/0.9107
	NGswin [4]	1,019K	36G	32.33/0.8963	28.78/0.7859	27.66/0.7396	26.45/0.7963	30.80/0.9128
	LMLT-Base(Ours)	672K	41G	32.38/0.8971	28.79/0.7859	27.70/0.7403	26.44/0.7947	31.09/0.9139
	SwinIR-light [25]	930K	64G	32.44/0.8976	28.77/0.7858	27.69/0.7406	26.47/0.7980	30.92/0.9151
	ESRT [31]	751K	-	32.19/0.8947	28.69/0.7833	27.69/0.7379	26.39/0.7962	30.75/0.9100
	ELAN-light [56]	640K	54G	32.43/0.8975	28.78/0.7858	27.69/0.7406	26.54/0.7982	30.92/0.9150
	HPINet-S [28]	463K	88G	32.47/0.8987	28.80/0.7872	27.69/0.7416	26.59/0.8016	30.92/0.9143
	SwinIR-NG [4]	1,201K	63G	32.44/0.8980	28.83/0.7870	27.73/0.7418	26.61/0.8010	31.09/0.9161
	SRformer-Light [64]	873K	63G	32.51/0.8988	28.82/0.7872	27.73/0.7422	26.67/0.8032	31.17/0.9165
	SPIN [52]	555K	42G	32.48/0.8983	28.80/0.7862	27.70/0.7415	26.55/0.7998	30.98/0.9156
	OSFFNet [47]	537K	22G	32.39/0.8976	28.75/0.7852	27.66/0.7393	26.36/0.7950	30.84/0.9125
	HIT-SIR [58]	792K	54G	32.51/0.8991	28.84/0.7873	27.73/0.7424	26.71/0.8045	31.23/0.9176
	SMFANet+ [63]	496K	28G	32.51/0.8985	28.87/0.7872	27.74/0.7412	26.56/0.7976	31.29/0.9163
	LMLT-Large(Ours)	1,295K	78G	32.48/0.8987	28.87/0.7879	27.75/0.7421	26.63/0.8001	31.32/0.9163

Table 21. The memory consumption and inference times are reported. All experiments were conducted on a single RTX 3090 GPU. Methods marked with † are CNN-based, while unmarked methods are ViT-based.

Scale	Method	#GPU Mem [M]	#Avg Time [ms]	Set14
×2	CARN-M [1]	2707.82	67.56	33.26/0.9141
	CARN [1]	2716.80	73.55	33.52/0.9166
	EDSR-baseline [26]	577.61	43.58	33.57/0.9175
	LAPAR-A [24]	1812.60	43.50	33.62/0.9183
	SAFMN [40]	259.56	33.61	33.54/0.9177
	LMLT-Tiny(Ours)	**324.01**	**57.37**	**33.59/0.9183**
	LMLT-Small(Ours)	**324.5**	**84.22**	**33.65/0.9187**
	HNCT [11]	1200.55	351.49	33.65/0.9182
	NGswin [4]	1440.40	375.19	33.79/0.9199
	SwinIR-light [25]	1278.64	944.11	33.86/0.9206
	SwinIR-NG [4]	1,227.0	1126.53	33.94/0.9205
	SRformer-Light [64]	1176.15	1006.48	33.94/0.9209
	LMLT-Base(Ours)	**567.75**	**81.64**	**33.76/0.9201**
	LMLT-Large(Ours)	**717.31**	**123.07**	**33.96/0.9212**
×3	CARN-M [1]	1213.10	37.56	30.08/0.8367
	CARN [1]	1222.08	41.08	30.29/0.8407
	EDSR-baseline [26]	541.61	26.14	30.28/0.8417
	LAPAR-A [24]	1813.84	35.95	30.34/0.8421
	SAFMN [40]	114.70	17.38	30.33/0.8418
	LMLT-Tiny(Ours)	**151.96**	**31.06**	**30.37/0.8427**
	LMLT-Small(Ours)	**152.5**	**44.22**	**30.47/0.8446**
	HNCT [11]	545.64	117.20	30.44/0.8439
	NGswin [4]	696.97	168.49	30.53/0.8456
	SwinIR-light [25]	587.63	287.96	30.54/0.8463
	SwinIR-NG [4]	564.7	393.68	30.58/0.8471
	SRformer-Light [64]	529.28	312.37	30.57/0.8469
	LMLT-Base(Ours)	**266.31**	**41.43**	**30.53/0.8458**
	LMLT-Large(Ours)	**338.36**	**58.68**	**30.60/0.8471**
×4	CARN-M [1]	680.84	21.39	28.42/0.7762
	CARN [1]	689.83	20.50	28.60/0.7806
	EDSR-baseline [26]	492.39	19.86	28.58/0.7813
	LAPAR-A [24]	1811.47	32.24	28.61/0.7818
	SAFMN [40]	65.26	11.28	28.60/0.7813
	LMLT-Tiny(Ours)	**81.44**	**23.54**	**28.64/0.7823**
	LMLT-Small(Ours)	**81.92**	**31.01**	**28.74/0.7846**
	HNCT [11]	312.72	69.61	28.71/0.7834
	NGswin [4]	372.94	118.13	28.78/0.7859
	SwinIR-light [25]	342.46	176.76	28.77/0.7858
	ELAN-light [56]	241.3	42.93	28.78/0.7858
	HPINet-S [28]	445.9	100.4	28.80/0.7872
	SwinIR-NG [4]	329.6	239.64	28.83/0.7870
	SRformer-Light [64]	320.95	180.42	28.82/0.7872
	SPIN [52]	441.5	701.29	28.80/0.7862
	SMFANet+ [63] †	247.5	18.12	28.87/0.7872
	HIT-SIR [58]	1,339.0	139.84	28.84/0.7873
	LMLT-Base(Ours)	**144.00**	**26.15**	**28.79/0.7859**
	LMLT-Large(Ours)	**185.68**	**34.07**	**28.87/0.7879**

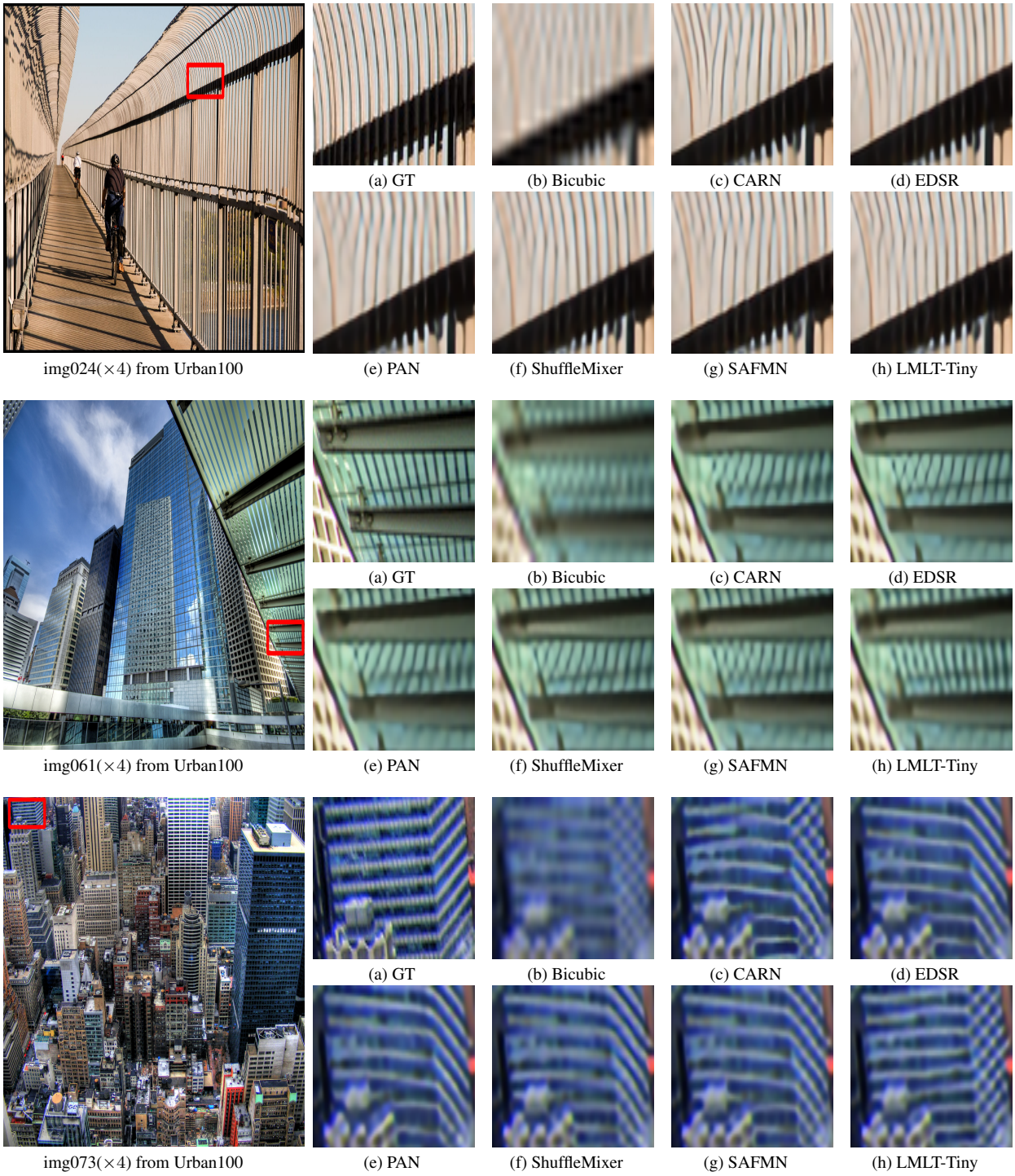


Figure 17. Visual comparisons for $\times 4$ SR on Urban100 dataset. Compared with the results in (c) to (g), the Ours(LMLT-Tiny, (h)) restore much more accurate and clear images.

Figure 18. Visual comparisons for $\times 4$ SR on Manga109 dataset. Compared with the results in (c) to (f), the Ours(LMLT-Base and LMLT-Large, (g) to (h)) restore much more accurate and clear images.