Editable Image Elements for Controllable Synthesis Supplementary Materials

Jiteng Mu[1], Michaël Gharbi[2], Richard Zhang[2], Eli Shechtman[2], Nuno Vasconcelos[1], Xiaolong Wang[1], and Taesung Park[2]

[1] University of California, San Diego, USA {jmu,nuno,xiw012}@ucsd.edu
[2] Adobe Research, USA {mgharbi,rizhang,elishe,tapark}@adobe.com
https://jitengmu.github.io/Editable_Image_Elements/

In this supplementary materials, we provide more details of the submission: We show additional editing results and pixel editing baselines in Section 1 complementing Section 4.2 in the paper; Furthermore, reconstruction evaluations are shown in Section 2; More implementation details, including architecture designs (paper Section 3.2 and Section 3.3), training recipes (paper Section 4.1), and image element partition algorithms (paper Section 3.1) are discussed in Section 3.

1 Additional Editing Comparison

In Figure 4, 5, and 6 of Section 4.2 in the main text, we have shown our editing results as well as comparisons to ***Self-Guidance***, ***Paint-by-Example***, and ***InstructPix2Pix***. Here we provide more details and show additional comparisons to these methods. Furthermore, we devise additional pixel editing related baselines built on our proposed image element partition, namely, ***pixel editing***, ***pixel editing + SDEdit***, and ***pixel editing + SD-Inpaint***. We run user studies on the pixel editing related baselines and show the results in Figure 1. More visual comparisons are presented in Figure 7, 8, 9, 10, 11, 12, 13, and 14. We elaborate on the detailed implementations of each baseline below.

Self-Guidance. The inversion of a real image is implemented following Self-Guidance, where a set of intermediate attention maps can be recorded by running a set of forward-process denoisings of a real input. Edits can then be performed with respect to the obtained attention maps. Since self-guidance modifies the gradient of each diffusion sampling step, it is sensitive to the hyperparameters choices. We observe that with small guidance strength, the editing operation is not respected, and with higher guidance, the realism is negatively affected. Instead of using separate parameters for individual images as in the paper, we employ the same parameter set for all testing images.

Paint-by-Example. Paint-by-Example requires a source image with a mask specifying the region to inpaint, plus a reference image for the target inpainting content. To achieve spatial editing, we take the pre-trained model and run inference as follows. We manually annotate each image with three regions: a source region for the object of interest, a target region for where to put the object, and a background region for inpainting the source region. In the first step, we treat the deliberately cropped background region as the reference to remove the object

in the source region. Intuitively, this can be interpreted as object removal. Next, we directly regard the cropped source region as the reference image to inpaint the target location. Though the first step can potentially achieve object removal, it is difficult to find appropriate background region as reference for some images and edits, leading to degraded image realism. The second stage poses further challenges in inpainting the source region content faithfully to the target location with expected size, especially when the source region is of low resolution or the source region contains only part of an object.

InstructPix2Pix. Though InstructPix2Pix is known to be great at texture transfer, we find its performance for spatial editing to be limited. We tried various prompts and found the model tends to either not respond to the spatial instructions or only modifies the global textures. In addition, it is also worth noting that using language only is limiting to achieve precise spatial editing. In comparison, our proposed method directly takes coordinates to represent the locations and sizes, making it easy to change image elements following the requested spatial edits.

Pixel editing. Pixel editing is a simple baseline implemented by copy-pasting image elements in pixel space, where the image partition is the same as used in our algorithm. Specifically, we directly copy the image patches obtained with our algorithm to the target location and resize them as desired. As expected, pixel editing does not handle the source region properly, leading to object duplication and unrealistic images. In addition, it is also challenging to scale up the source region while maintaining high quality with simple pixel space resizing. To address the challenges, we further propose to use SDEdit and SD-Inpaint as described below.

Pixel editing + SDEdit. SDEdit is a simple idea by first manipulating pixel space, and then adding Gaussian noise to the edited image and running the reverse sampling process for image synthesize. A sweet spot balancing realism and faithfulness can be identified for some specific intermediate time steps. Intuitively, adding more Gaussian noise and running the SDE for longer synthesizes more realistic images, but they are less faithful to the give input. To use SDEdit for spatial editing, we provide the masked pixel editing results as input, where the source region is left blank instead of maintaining the original pixels. We tested various time step choices (0.5, 0.7, 0.9) and find it is non-trivial to identify a single sweet spot for all images and editing operations. For a fair comparison, the diffusion model is chosen to be Stable Diffusion v1.5.

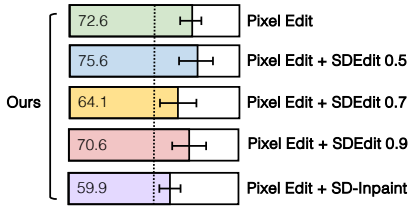

Fig. 1: Perceptual study where users are asked to choose which image better reflects both the editing and image quality. Results show that ours is preferred compared to all cases.

	Supervision	Bottleneck	MSE ↓	PSNR ↑	SSIM ↑	LPIPS ↓	FID ↓
VAE-KL-Adv	L1+LPIPS+Adv	$16 \times 16 \times 32$	0.0052	24.56	0.6701	0.2180	5.96
VAE-KL	L2	$16 \times 16 \times 32$	0.0037	25.47	0.6754	0.4617	62.46
AE	L2	$16 \times 16 \times 32$	0.0027	27.55	0.7503	0.3759	55.65
Ours - DDIM	L2	256×32	0.0069	22.98	0.6354	0.3376	10.82

Table 1: Reconstruction Comparison. Our method achieves better LPIPS and FID scores compared to AE and VAE (all trained with L2 loss), and competitive compared with variational autoencoder trained with adversarial loss. LPIPS and FID scores are more informative as we prefer faithful reconstruction instead of pixel-perfect results. Note MSE and PSNR are known to prefer blurry results as visualized in Figure 2. VAE-KL-Adv denotes variational autoencoder with KL divergence regularization plus losses following Latent Diffusion Model, VAE-KL for variational autoencoder with KL loss, AE for autoencoder, and ours-DDIM is with content encoder and diffusion decoder by running 50-step DDIM sampling steps. Numbers are computed on $5,000$ samples.

Pixel editing + SD-Inpaiting. We also test Stable Diffusion Inpainting model, where the UNet has 5 additional input channels (4 for the encoded masked-image and 1 for the mask itself) to inpaint the 'holes' left in the source region. We find though it produces reasonable inpainting results for some editing operations, it still suffers from scaling up the source region with high quality because the scaled source region is not modified by the diffusion model with masked input. In addition, we observe that it tends to keep inpainting another object rather than blending seamlessly with the background in the source region. Note that since we find SD-Inpainting model does not handle the small gaps between patches well, to get better results, we run the morphological operation on the masks to fill in the gaps first before passing it to the diffusion model.

2 Reconstruction Comparison

To quantitatively evaluate the reconstruction of our method, we compare the reconstruction quality of the proposed method to various convolutional autoencoder approaches, as shown in Table 1. The bottleneck size of all methods is the same for a fair comparison. Qualitative comparisons are presented in Figure 2.

Specifically, our approach is composed of a content encoder and a diffusion decoder. The first stage trains the content encoder plus a lightweight transformer decoder. Then in the second stage, a diffusion decoder is learned with the content encoder frozen. The reconstruction results presented are obtained by providing all image patches to the content encoder to obtain corresponding image elements, then decoding with the diffusion decoder. We run 50 DDIM sampling steps for the results, showing that our method obtains better LIPIPS and FID compared to the autoencoder (AE) and variational autoencoder (VAE). From the visualizations in Figure 2, it is clear that our reconstruction maintains more details and is more visually appealing compared to AE and VAE, though showing slightly

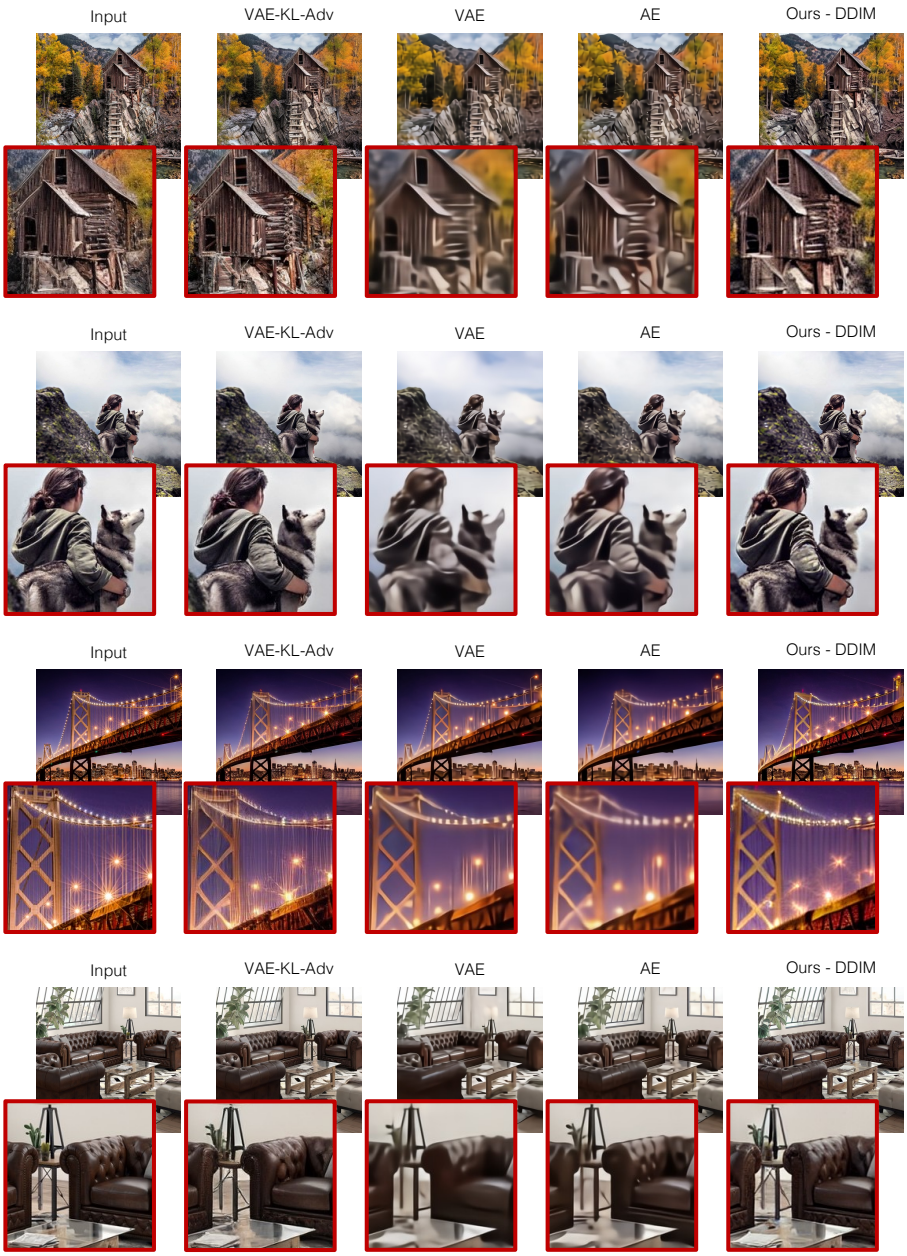

Fig. 2: Reconstruction comparisons. Ours-DDIM, trained with only L2 loss, preserves more details compared to autoencoder (AE) and variational autoencoder (VAE-KL), and achieves competitive results compared with variational autoencoder trained with adversarial loss (VAE-KL-Adv).

lower MSE and PSNR (which is widely known for preferring blurry results). We also trained the VAE in an adversarial manner following the Latent Diffusion Model, showing on the top row for reference. Our method achieves competitive numbers, and from Figure 2, shows similar visual results (sharper details). Note that the simple L2 loss can be trained much faster than the adversarial loss. Adversarial training could potentially be used on our method for even better quality and we leave this for future work.

The input image size to all methods are 512×512. The architecture of all baselines are based on the autoencoder borrowed from the Latent Diffusion Model, which is a convolutional autoencoder with multiple residual blocks, downsampling layers and upsampling layers. The main differences are in that we use more downsampling and upsampling layers to obtain a compact latent representation ($32\times$ downsampling rather than $8\times$ downsampling). In addition, we also increase the number of channels of the bottleneck to 32 for a fair comparison. The VAE-KL-Adv is also included for reference, which is trained in an adversarial manner with an additional patch-based discriminator optimized to differentiate original images from reconstructions. To avoid arbitrarily scaled latent spaces, an Kullback-Leibler-term is implemented to regularize the latent. It also uses L1 loss combined with LPIPS loss for better reconstruction.

3 Implementation Details

In Section 3.1, we provide more details of the content encoder and transformer decoder architectures, the pseudo-code for the fused attention block in our diffusion decoder. Detailed training recipes and hyper parameters are then presented in Section 3.2. We present the image partition algorithm and comparisons of different variants in Section 3.3.

3.1 Architectures

The architectures of the content encoder and transformer decoder described in paper Section 3.2 are illustrated in Figure 3. We also provide more details of the implementation of our fused attention block for the diffusion decoder (paper Section 3.3) in Algorithm 1.

Content Encoder. Each image patch is first resized to 32×32 before input to the content encoder. This design ensures the features extracted are agnostic to positional and size information. The content encoder then maps the input image patch to a feature vector of 32 channels, as shown in Figure 3. The content encoder consists of multiple residual blocks, followed by convolutional downsampling layers with stride 2. One additional middle block consists of 2 Res-Blocks and 1 attention layer is implemented to further process the intermediate features. Then another convolutional layer outputs a feature of 32 channels for this patch. The overall architecture is based on the convolutional encoder of the Latent Diffusion Model, but with less residual blocks at each level and a different

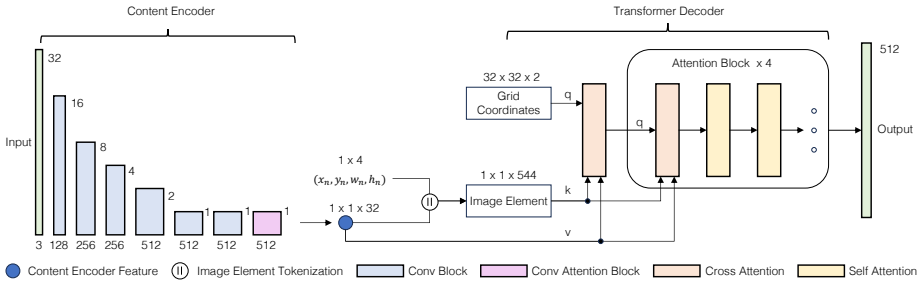

Fig. 3: Architecture for content encoder and transformer decoder. Only one image element is shown for simple illustration, but in practice, all image elements are jointly decoded.

output dimension. To compensate for the missing spatial information, the patch feature is combined with the a 4-dimensional vector, indicating its position and size, to form an image element.

Transformer Decoder. To decode the image element, a set of grid coordinates of shape $32 \times 32 \times 2$ is passed to positional embedding to form the queries as the input of the transformer decoder. Then the image elements are served as keys and the features (excluding the spatial information) are used as values for the cross attention layer. Figure 3 only shows one image element as an illustration. In practice, all image elements are jointly decoded using the transformer decoder. The transformer decoder is modified from Masked Autoencoders. The network consists of 4 attention blocks, each is with 1 cross attention and 2 self-attention layers. All self-attention layers are with 512 channels and 16 heads, and cross attention layers with 512 channels and 1 head.

Diffusion Decoder. While the auto-encoder aforementioned produces meaningful image elements, the synthesized image quality is limited. We modify pretrained Stable Diffusion model to condition on our image elements for better reconstruction and editing quality. The Stable Diffusion base model is implemented with a UNet architecture, which contains a set of residual blocks and attention blocks. To incorporate the image element into the UNet model, we modify the attention blocks to jointly take the text embedding and the image elements, as shown in Algorithm 1.

Specifically, for each layer with a cross attention to text embeddings in the original UNet architecture, we insert a new cross attention layer with parameters to take the image elements. Both cross attention layers, one on text and the other on image elements, are processed using the output features of the previous self-attention layers. The output features of the cross attention are then added to the self-attention layer features with equal weights.

Algorithm 1: Fused Attention Block

```
# x: input features
# context_image: image features mapped from image elements
# context_text: text features obtained from text encoders

# self attention
x = x + self_attention(norm_self(x))

# cross attention
if context_image is not None and context_text is not None:
    # fuse image and text attention outputs
    y_text = cross_attention_text(norm_text(x), context=context_text)
    y_image = cross_attention_image(norm_image(x), context=context_image)
    y = y_text + y_image
if context_image is not None and context_text is None:
    # image attention only
    y_image = cross_attention_image(norm_image(x), context=context_image)
    y = y_image
if context_text is not None and context_image is None:
    # text attention only
    y_text = cross_attention_text(norm_text(x), context=context_text)
    y = y_text
x = x + y

# feed forward
x = feedforward(norm_feedforward(x)) + x
```

3.2 Training Details

Our dataset contains $3M$ images from the LAION Dataset, filtered with an aesthetic score above 6.25 and text-to-image alignment score above 0.25. We randomly select $2.9M$ for training and leave the held-out 100k data for evaluation.

For the content encoder and lightweight transformer decoder, we train this autoencoder for 30 epochs with MSE loss. We use the AdamW optimizer, setting the learning rate at $1e-4$, weight decay to 0.01, and beta parameters to $(0.9, 0.95)$. The model is trained on 8 GPUs, with a total batch size of 128. In addition, all image elements are presented to the decoder for efficient training, which means no dropout is performed for this stage.

Our diffusion decoder is built on Stable Diffusion v1.5, trained with the same losses as Stable Diffusion. For the training phase, we use the AdamW optimizer, setting the learning rate at $6.4e-5$, weight decay to 0.01, and beta parameters to $(0.9, 0.999)$. We report the results after around $180k$ iterations. The model is trained across 8 GPUs, with a total batch size of 64. During inference, we use classifier-free guidance with equal weights on text $\mathbf{C}$ and image element $\mathbf{S}$ to generate our examples: $\epsilon(z, \mathbf{C}, \mathbf{S}) = \epsilon(z, \emptyset, \emptyset) + w * [\epsilon(z, \mathbf{C}, \mathbf{S}) - \epsilon(z, \emptyset, \emptyset)]$. The examples in the paper are generated with $w = 3.0$ using 50 sampling steps with the DDIM sampler. During training, we randomly set only $\mathbf{C} = \emptyset$ for 30% of examples, $\mathbf{S} = \emptyset$ for 10% of examples, and both $\mathbf{C} = \emptyset$ and $\mathbf{S} = \emptyset$ for 10% of examples.

Algorithm 2: Image Element Partition

Input: Pretrained SAM model M, Image $\mathbf{x}$, Grid coordinates Q, Total number of iterations T, Centoid adjustment ratio β_c
Output: Image partition $A = \{a_1, a_2, \cdots, a_N\}$, centoid locations $C = \{c_1, c_2, \cdots, c_N\}$, bounding box sizes $Z = \{z_1, z_2, \cdots, z_N\}$

$C_0 = Q$
for $i = 1, 2, \cdots, T$ **do**
  $s \leftarrow M(\mathbf{x}, C_{i-1})$ `// Compute SAM scores`
  $g \leftarrow \text{ComputeAssignment}(s)$ `// SAM assignment`
  $C_i \leftarrow \text{ComputeCentroids}(g)$ `// initial centroid`
  $\tilde{C}_i \leftarrow \beta_c \cdot C_i + (1 - \beta_c) \cdot Q$ `// Centroid Adjustment`
  $\tilde{g} \leftarrow \text{ComputeAssignmentWithDistance}(s, \tilde{C}_i, Q)$ `// Distance regularization (Eq 1)`
  $\tilde{g} \leftarrow \text{ConnectedComponent}(\tilde{g})$ **if** $i = T - 1$
  $C_i \leftarrow \text{ComputeCentroids}(\tilde{g})$
end for

`// Compute Outputs`
$A \leftarrow \text{ComputePatches}(\tilde{g})$
$C \leftarrow \text{ComputeCentroids}(\tilde{g})$
$Z \leftarrow \text{ComputeSizes}(\tilde{g})$

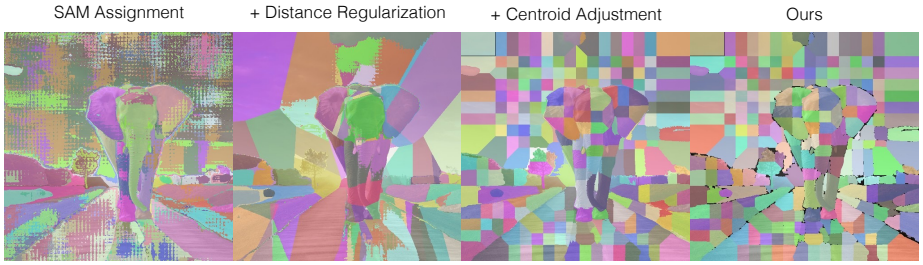

Fig. 4: Image element partition algorithm intermediate results. We show the step-by-step results complementing Algorithm 2. Starting from the noisy scores produced by SAM, distance regularization, centroid adjustment, morphological operation plus connected components are used sequentially to produce our final partitions.

3.3 Image Element Partition

We here provide more details of the image partition algorithm for Section 3.1 in the paper. Specifically, we implement Algorithm 2 for extracting image elements, with intermediate outputs shown in Figure 4.

To divide the image into patches, we borrow the insight of the Simple Linear Iterative Clustering (SLIC) to operate in the feature space of the state-of-the-art point-based Segmentation Anything Model (SAM). We start with $N = 256$ query points using 16×16 regularly spaced points Q on the image, resulting in at most 256 image element partitions in the end. To start with, SAM takes an image $\mathbf{x}$ and initial centroids C_0 as inputs, and predicts association scores s for each query point and pixel locations. Then it follows by computing the cluster assignment g for all pixel locations. However, since the segments tend to vary too much in shape and size, and extreme deviation from the regular grid is not amenable to downstream encoding and decoding, see Figure 4 SAM Assignment.

So we propose to add distance regularization as described in Eq 1 in the paper with hyper parameters β balancing the distance and scores, visualized in Figure 4 Distance Regularization. Nevertheless, it is observed that centroids tend to collapse for semantically close regions. To address this, we further modify the obtained centroids by computing a linear interpolation of the centroids C_i and grid coordiantes Q, with a hyper parameter $\beta_c = 0.2$. This effectively avoids centroid collapse as illustrated in Figure 4 Centroid Adjustment. Finally, morphological operations and connected components are applied to remove small regions. We set the number of iterations to be 1, but potentially more iterations could be used for better partitions.

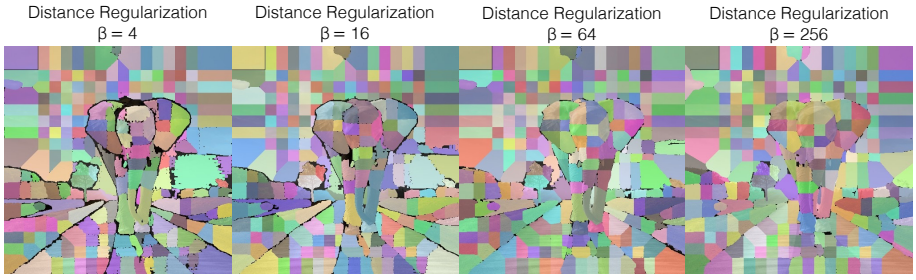

Fig. 5: Comparisons on different distance regularization strengths corresponding to Eq 1 in the paper. Smaller β tends to drop more pixels and larger β produces less accurate superpixels. $\beta = 64$, achieving a sweet spot between reconstruction and editability, is used throughout the paper.

We also study various parameter choices for β in Eq 1 as shown in Figure 5. We empirically choose $\beta = 64$, which achieves a good balance between reconstruction quality and editability. We also compare with various image partition algorithms, such as SLIC algorithm in pixel space and grid partition. From Figure 6, our algorithm yields best editing results, whereas other methods presents various types of failures. We observe that for SLIC algorithm, the image partitions tend to show wiggling boundaries and the superpixels obtained are also not well aligned with object boundaries as ours, making the learning harder. For grid partition, since the location and size parameters are constants across all images, it fails to relocate and resize objects. This again shows that our design of image element partition and resizing are the keys to learn a disentangled representation.

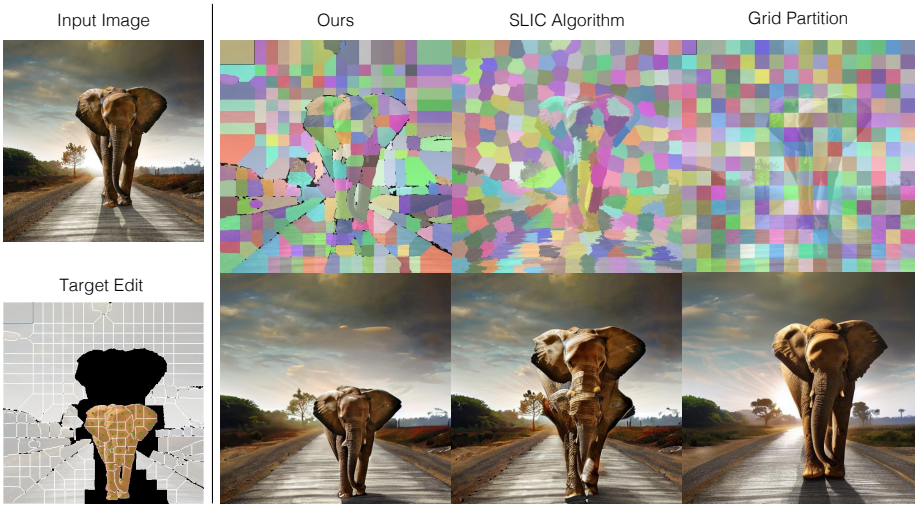

Fig. 6: Comparisons on different image partition algorithms. We test various image partitions on the same edit and results are shown on the right: ours, SLIC algorithm, and grid partition. Ours follows the edit operations and preserves object details correctly. In comparison, SLIC algorithm produces unrealistic images and grid partition fails to follow the edit.

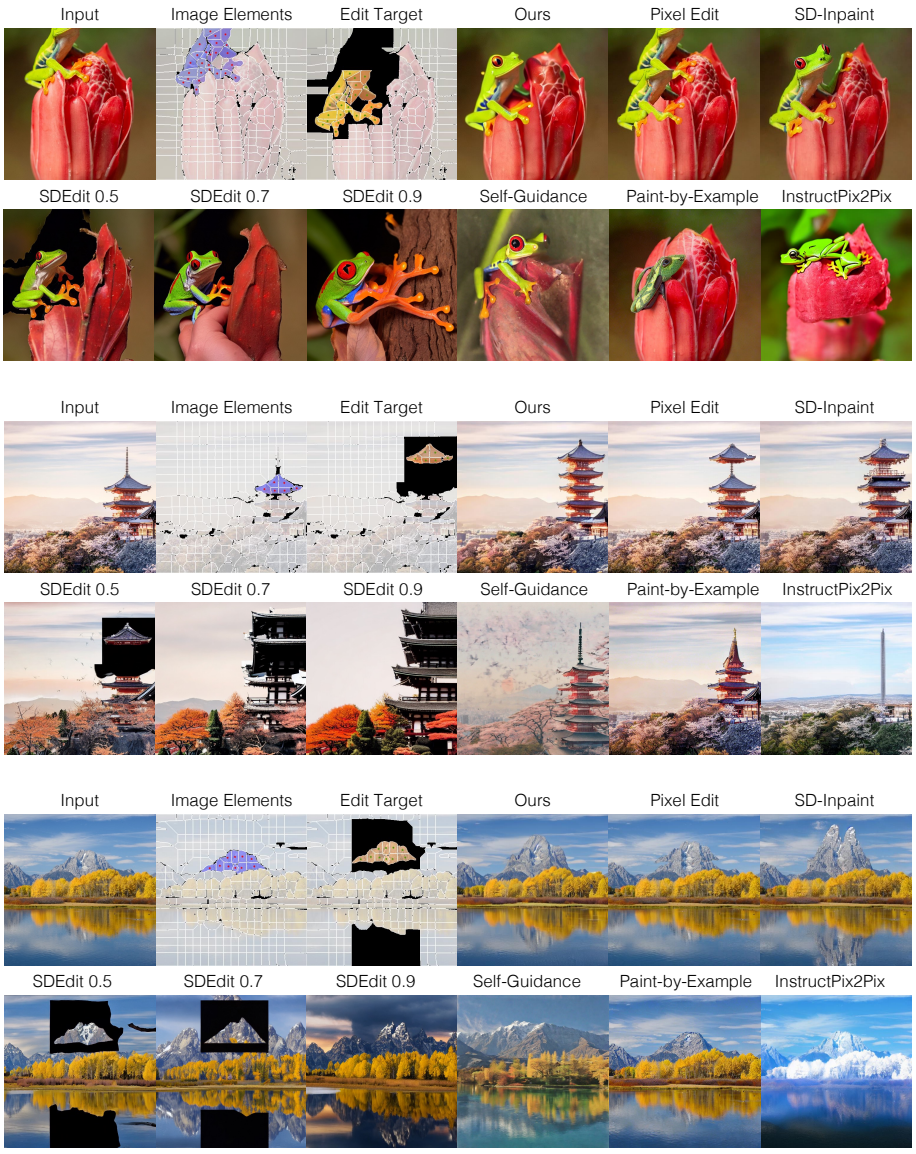

Fig. 7: Comparisons complementing Figure 4, 5, and 6. Our method is compared to pixel editing, pixel editing with Stable Diffusion Inpainting model (SD-Inpaint), pixel editing with SDEdit of various schedules, Self-guidance, Paint-by-Example, and InstructPix2Pix on various edits. Our results attain superior results in preserving the details of the input as well as following the new edits. The baseline results show various types of failures, such as decline in image quality, floating textures, and unfaithful to the edits.

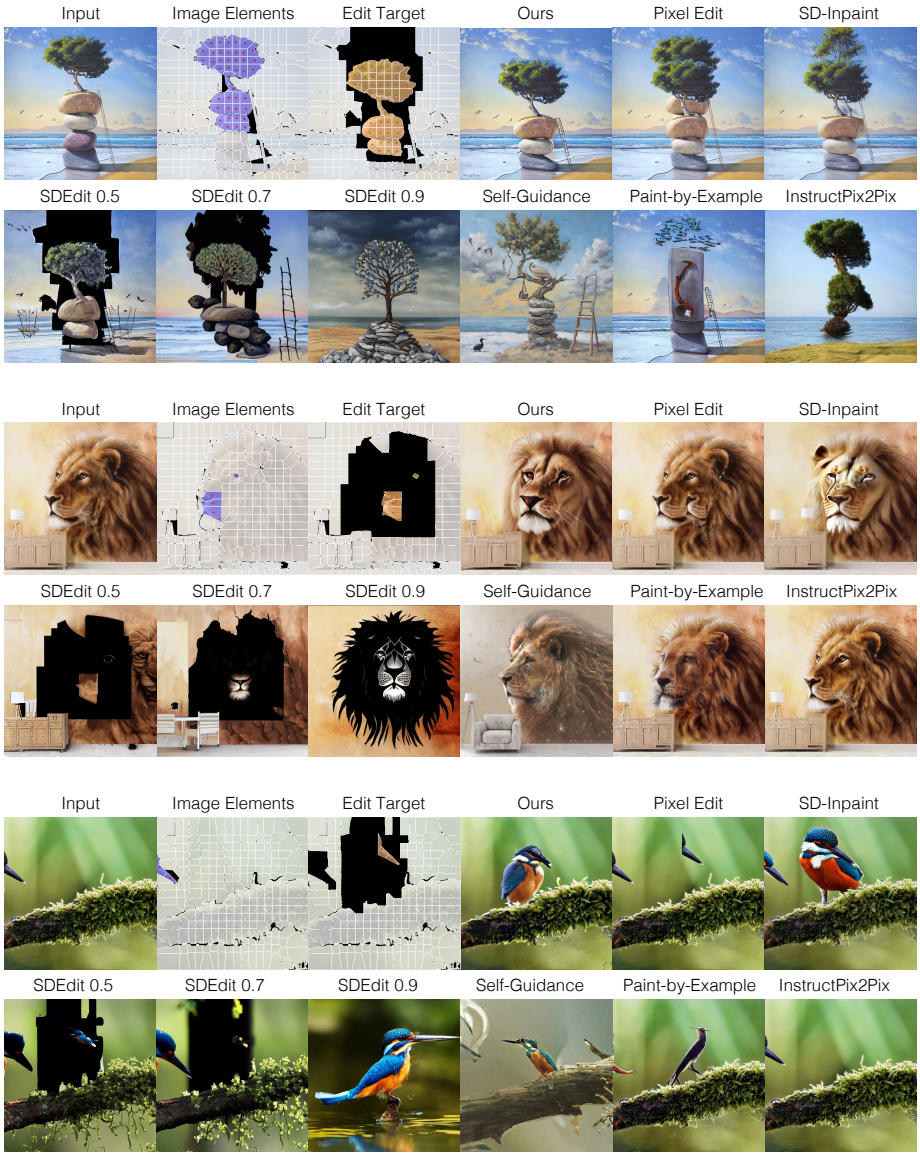

Fig. 8: Comparisons complementing Figure 4, 5, and 6. Our method is compared to pixel editing, pixel editing with Stable Diffusion Inpainting model (SD-Inpaint), pixel editing with SDEdit of various schedules, Self-guidance, Paint-by-Example, and InstructPix2Pix on various edits. Our results attain superior results in preserving the details of the input as well as following the new edits. The baseline results show various types of failures, such as decline in image quality, floating textures, and unfaithful to the edits.

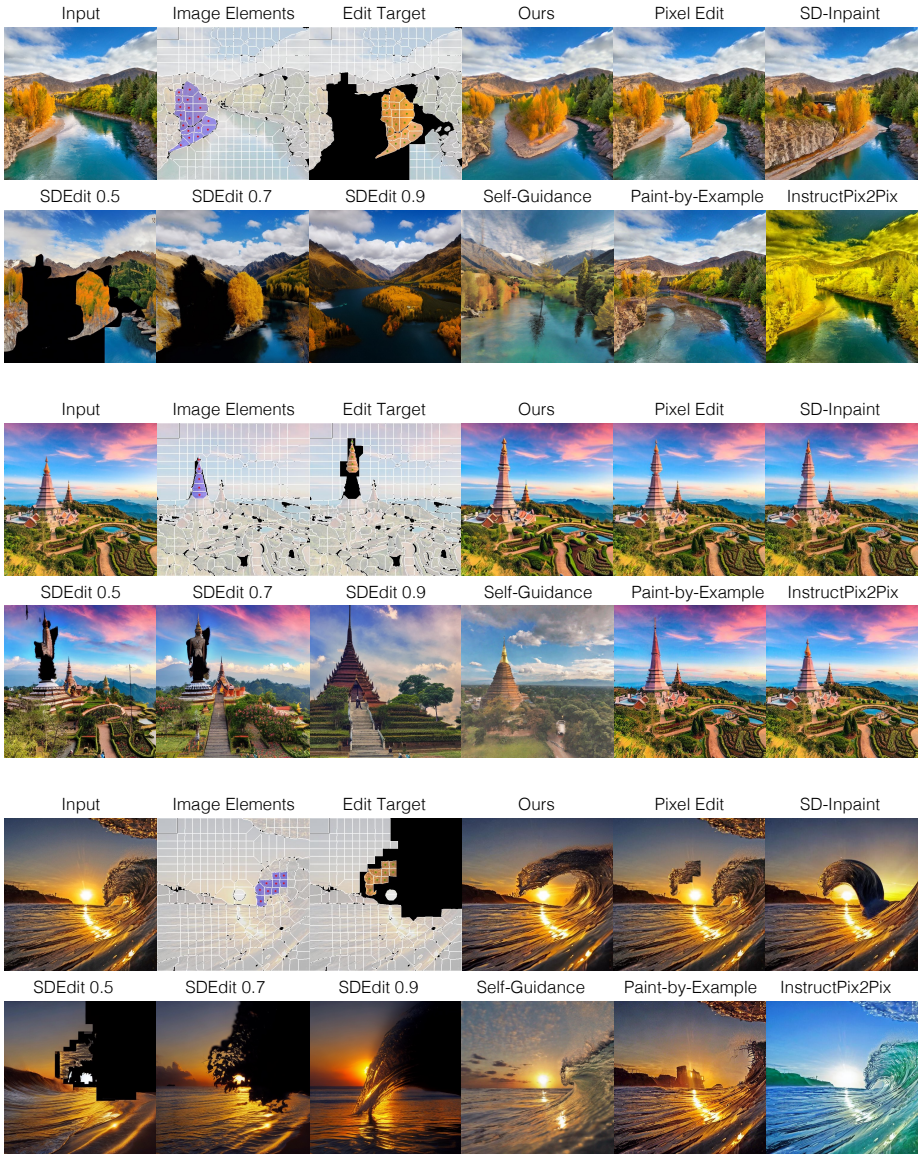

Fig. 9: Comparisons complementing Figure 4, 5, and 6. Our method is compared to pixel editing, pixel editing with Stable Diffusion Inpainting model (SD-Inpaint), pixel editing with SDEdit of various schedules, Self-guidance, Paint-by-Example, and InstructPix2Pix on various edits. Our results attain superior results in preserving the details of the input as well as following the new edits. The baseline results show various types of failures, such as decline in image quality, floating textures, and unfaithful to the edits.

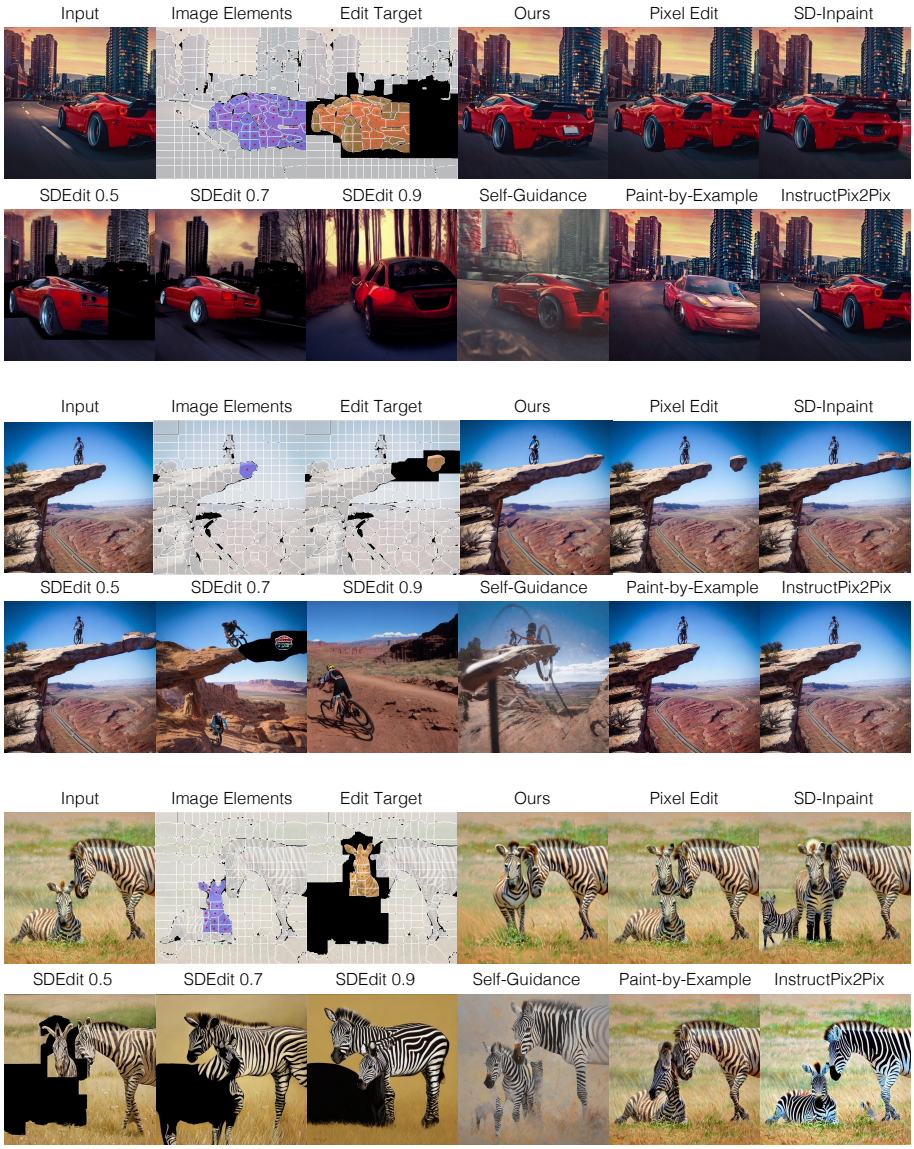

Fig. 10: Comparisons complementing Figure 4, 5, and 6. Our method is compared to pixel editing, pixel editing with Stable Diffusion Inpainting model (SD-Inpaint), pixel editing with SDEdit of various schedules, Self-guidance, Paint-by-Example, and InstructPix2Pix on various edits. Our results attain superior results in preserving the details of the input as well as following the new edits. The baseline results show various types of failures, such as decline in image quality, floating textures, and unfaithful to the edits.

Fig. 11: Comparisons complementing Figure 4, 5, and 6. Our method is compared to pixel editing, pixel editing with Stable Diffusion Inpainting model (SD-Inpaint), pixel editing with SDEdit of various schedules, Self-guidance, Paint-by-Example, and InstructPix2Pix on various edits. Our results attain superior results in preserving the details of the input as well as following the new edits. The baseline results show various types of failures, such as decline in image quality, floating textures, and unfaithful to the edits.

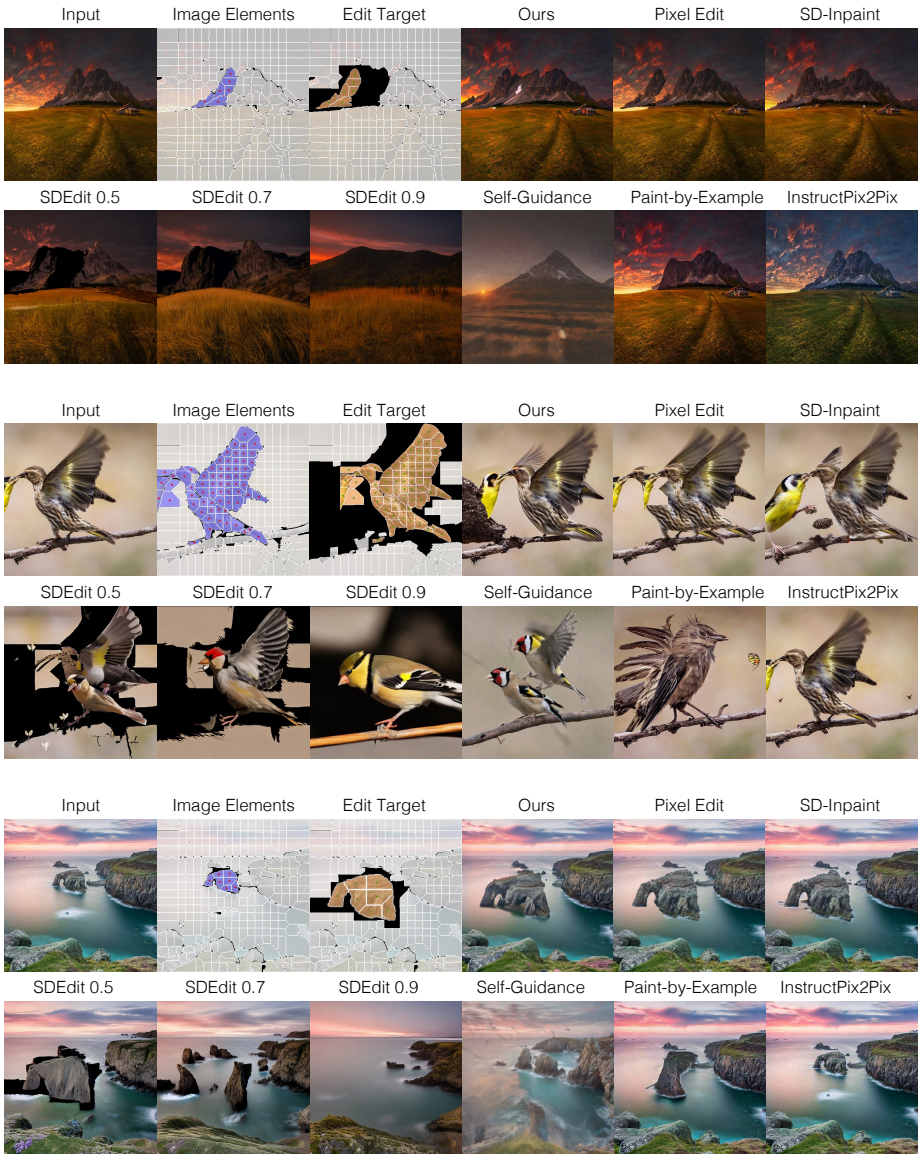

Fig. 12: Comparisons complementing Figure 4, 5, and 6. Our method is compared to pixel editing, pixel editing with Stable Diffusion Inpainting model (SD-Inpaint), pixel editing with SDEdit of various schedules, Self-guidance, Paint-by-Example, and InstructPix2Pix on various edits. Our results attain superior results in preserving the details of the input as well as following the new edits. The baseline results show various types of failures, such as decline in image quality, floating textures, and unfaithful to the edits.

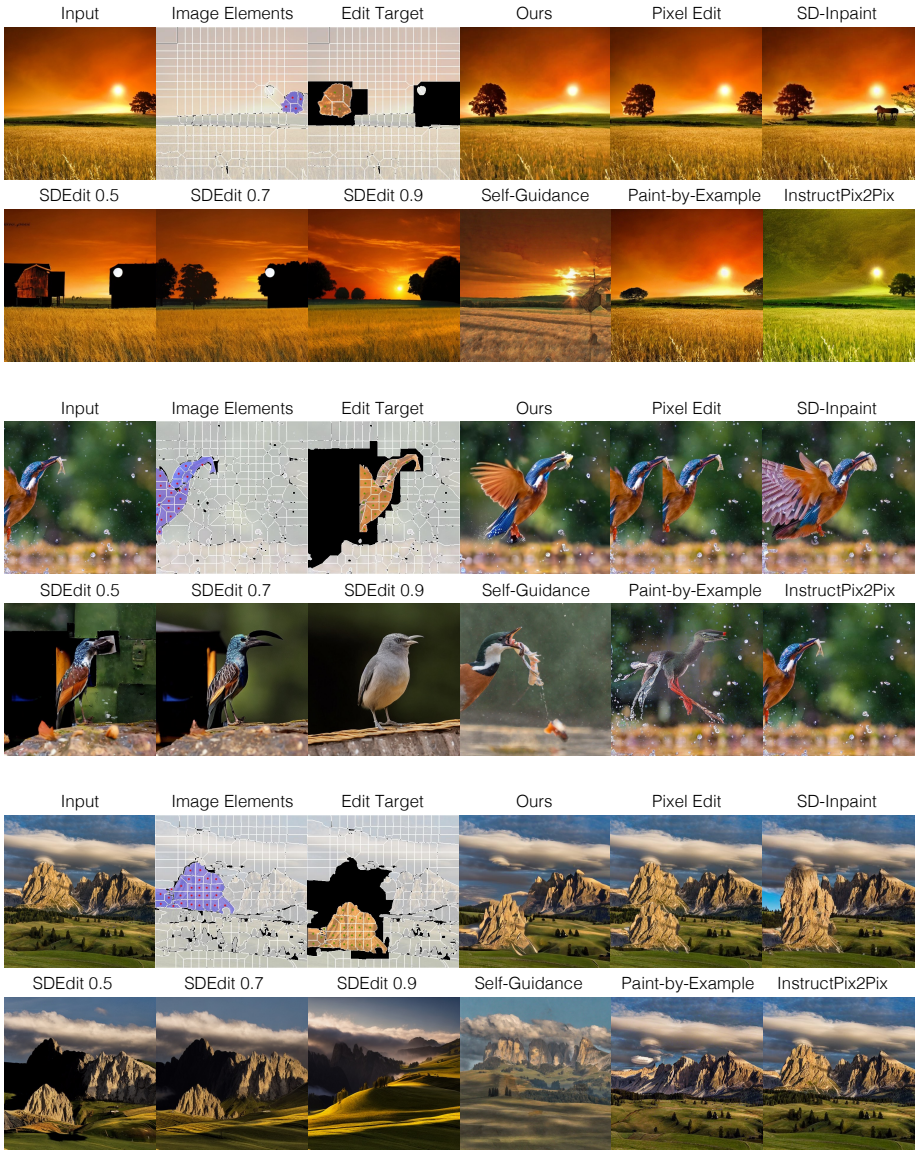

Fig. 13: Comparisons complementing Figure 4, 5, and 6. Our method is compared to pixel editing, pixel editing with Stable Diffusion Inpainting model (SD-Inpaint), pixel editing with SDEdit of various schedules, Self-guidance, Paint-by-Example, and InstructPix2Pix on various edits. Our results attain superior results in preserving the details of the input as well as following the new edits. The baseline results show various types of failures, such as decline in image quality, floating textures, and unfaithful to the edits.

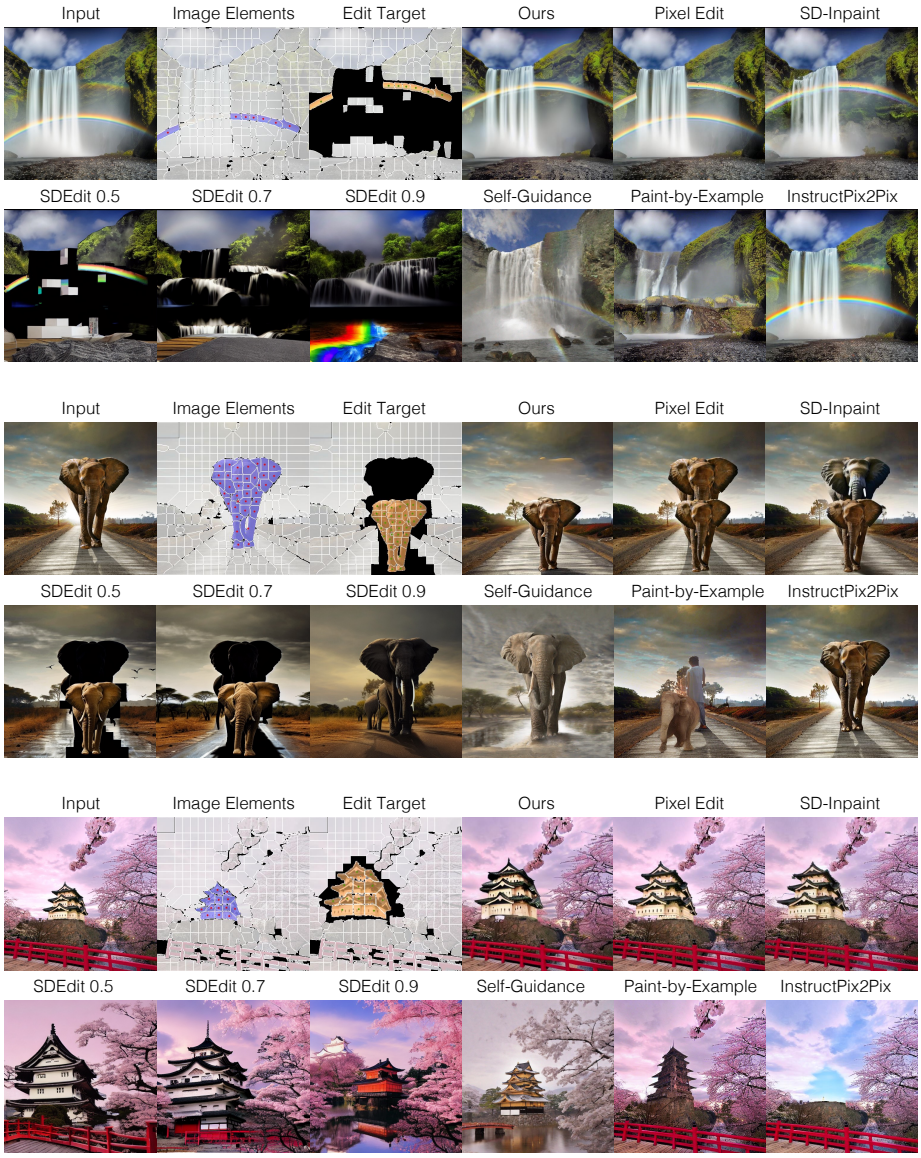

Fig. 14: Comparisons complementing Figure 4, 5, and 6. Our method is compared to pixel editing, pixel editing with Stable Diffusion Inpainting model (SD-Inpaint), pixel editing with SDEdit of various schedules, Self-guidance, Paint-by-Example, and InstructPix2Pix on various edits. Our results attain superior results in preserving the details of the input as well as following the new edits. The baseline results show various types of failures, such as decline in image quality, floating textures, and unfaithful to the edits.