Emu Edit: Precise Image Editing via Recognition and Generation Tasks

Supplementary Material

7. Data

Fig. 7 shows the tasks composing our dataset and their distribution.

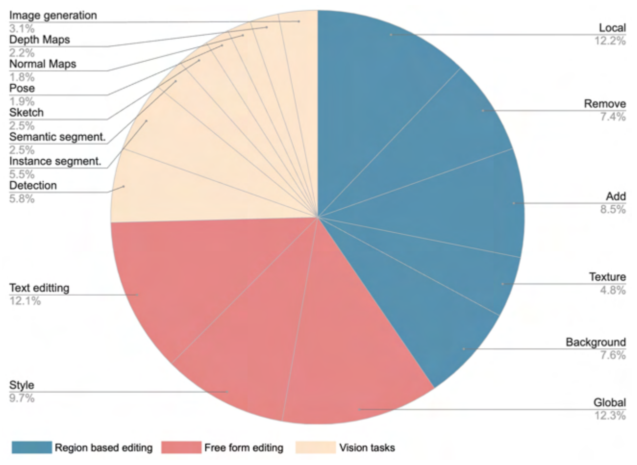


Figure 7. Distribution of the tasks in our training dataset.

7.1. Instruction Generation

We generate instructions utilizing the dialogue-optimized 70B parameter Llama 2 variant. We use a temperature of 0.9 and set the top-p value to 0.9. We employ LLM in-context learning to generate instructions. Figs. 16-17 demonstrate the prompts used for task Add. A similar approach is used for the remaining tasks. We instruct the LLM to generate instructions similar to, but diverse from, the examples provided.

To achieve this, we supply the LLM with the following: (1) a system message describing the input and output formats, (2) an introduction message in which we outline the problem and the goal for each key in the output, and, (3) a historical context of the conversation with the LLM containing examples for possible outputs. We then prompt the LLM with a new input caption and ask it to provide a new instruction. To encourage more variance and randomness in the LLM-generated instructions, we perform the following on the historical context: (1) shuffling between examples, (2) randomly sampling 60% of the examples, and, (3) randomly changing the verbs in the examples from a set of words.

7.2. Image Pairs Generation

Below we describe in detail our image generation methods for all the tasks. The image pair generation phase uses an image caption, and the corresponding output caption, "original object", and "edited object" that the LLM generated in the instruction generation phase.

7.2.1 Grounded Precise Editing

As described in Sec. 3.2, we integrate the mask m of the edited area, during the editing process, to ensure seamless blending of edited regions with the original image. We call this operation *mask-based attention control*. Blending is defined as follows: $x_t \cdot m + (1-m) \cdot y_t$, where x_t is the noisy edited image in step t, and, y_t is the noisy version of the input image in step t. In the first $blend_s$ percent of the steps we replace each of the noisy generated images with the corresponding noisy version of the input images. In the rest of the steps we use blending. The purpose of this, is to ensure structure preservation between the input and the edited image. We continue by following P2P and inject the self attention layers. Cross attention layers are injected on the common tokens between the input and output captions. We denote by $\mathcal{N}_c$, and $\mathcal{N}_s$ the portion of steps where we share cross attention and self attention maps, correspondingly.

7.2.2 Mask Extraction

Region-based editing includes all the editing instructions that perform changes to the image in a limited region, leaving the rest of the image unchanged. To adjust a particular object or location while preserving the rest of the details, we utilize a mask of the local area in the editing process. We utilize DINO [14] to detect the area that needs to be masked, using the "original object" and "edited object" fields that were generated in the previous stage (Sec. 7.1).

Dilation, Gaussian Blurring and Bounding Box Masks: We observe that when utilizing mask-based attention control to generate an edited image, it often replaces the object with a similar object type instead of removing it. For example, when masking the region around a dog, we confine the editing to that specific area, resulting in the generation of a new variation of the dog. We address this issue by creating three different types of masks. The first employs the original precise mask, created by DINO and SAM [11]. The second involves expanding the mask beyond the added object through dilation and then refining it using Gaussian blurring. Finally, the third approach uses the bounding box around the object (created by DINO), thereby eliminating the constraints of a specific shape. We generate multiple images, each with a different mask, and then filter for the best image. Our filtering is described in Sec. 3.2.

Possessive words: In some cases the "original object" and "edited object" generated by the LLM contain possessive words (e.g, "a dog's tail"). We observe that, in many cases, DINO struggles to detect the object in these cases. To this end, we employ an additional prompting to the LLM to identify the object without possession.

7.2.3 Region-Based Editing Tasks

Local/Texture Given the input caption, we first generate the input image. Then, we utilize the "original object" (as described in Sec. 7.2) to extract the local mask (using Sec. 7.2.2). Lastly, we apply masked-based attention control using the obtained mask to generate the edited image. We repeat this entire process for 10 iterations, where in each iteration, we sample the guidance scale from $[4, 8]$, $\mathcal{N}_c$ and $\mathcal{N}_s$ from $[0.3, 0.9]$, and $blend_s$ from $[0.02, 0.2]$.

Add Extracting the mask of the "edited object" (the object that was added in this case) is not possible in advance because the object does not exist in the input image. To overcome this challenge, we address this as follows:

1. We generate the output image y using the output caption. Note that the image y contains the "edited object".
2. The mask m of the "edited object" in y is extracted.
3. We apply the mask-based attention control to generate the input image x using the input caption, the image y and the mask m

The main problem with this approach is that in certain instances, we generate a different version of the object, instead of eliminating it, as described in Sec 7.2.2.

Remove. The process of generating data for Remove task is similar to the one of Add task. The only difference is that we first generate the image x (using the input caption), then extract the mask m of the object to remove, and finally generate the image y using the output caption, image x and the mask m.

Background. Given an input image, input caption and the edited object (in this case, the alternative background), we first extract the background mask. To eliminate artifacts in the contour, we apply minimum filter which extends the background mask and then smooth it using Gaussian filtering. Next, we provide the image and the resulting mask as input to an inpainting model, which creates a new background. Lastly, we blend the input image and the edited image in the mask region. We generate 10 edited images, with different noise and guidance scale, and pick the best according to CLIP metrics.

7.2.4 Free-Form Editing Tasks

Global. The global task includes editing instructions that are not restricted to a specific area. Therefore, we generate the image pairs using mask-based attention control with a blank mask. $blend_s$ is sampled from $[0.1, 0.2]$ to encourage better image faithfulness. We sample $\mathcal{N}_c$ and $\mathcal{N}_s$ from $[0.4, 0.9]$.

Style. We use Plug-and-Play (PNP) [25] to generate the stylized edited images. The goal of this task is to alter the image style according to the editing instruction while preserving the image structure. We apply PNP on the real input images using DDIM inversion. For each sample, we generate 10 edited images, each with the following parameters sampled: guidance scale sampled from $[6.5, 10.0]$, $\mathcal{N}_s$ from $[0.5, 1.0]$, and, the portion of spatial features to share is set to 0.8.

Text Editing. The text editing task includes adding text to the image, removing text from the image, and replacing one text with the other. In addition, we allow the user to choose the font and the color of the added text. We generate a mask, m, of the text found in the input image, x, using OCR [7]. We utilize mask m to inpaint the image, denote the new image y. For adding text, we use y as the input image and x as the edited image. For removing text and replacing text, we use the reverse. When replacing text, we overlay the inpainted region in image y with a text in a specific font and color.

7.2.5 Vision tasks

Detect/Segment. Given an input image, we detect the "edited object" using DINO. To formalize detection as a generative task, we create a new image y by drawing the detected bounding box. For segmentation, we paint the detected object pixels.

Color. We define the Color task as a modification to the overall colors of an image. We generate samples by applying the following filters: (1) color filters - randomly changing the brightness, contrast, saturation and hue of an image, (2) blurring - applying random-sized Gaussian kernels, and (3) sharpening and defocusing.

Image-to-Image Translation Tasks that involve bi-directional mapping from conditioning images to target images. For instance, sketch-to-image and image-to-sketch. We follow [30], to generate depth maps, segmentation maps, human poses, normal maps and sketches.

Table 5. **Data generation pipeline evaluation.** We compare our data generation pipeline with that of InstructPix2Pix. We also report the automatic metrics on the InstructPix2Pix training dataset.

Task	Method	CLIP$_{dir}$	CLIP$_{im}$	CLIP$_{out}$	L1↓	DINO↑
Local	IP2P	0.329	0.922	0.270	0.046	0.917
	Our	0.402	0.927	0.289	0.029	0.908
Texture	IP2P	0.282	0.876	0.297	0.189	0.671
	Our	0.373	0.957	0.296	0.033	0.923
Remove	IP2P	0.204	0.818	0.254	0.067	0.755
	Our	0.279	0.913	0.266	0.046	0.841
Add	IP2P	0.263	0.897	0.278	0.157	0.934
	Our	0.318	0.962	0.304	0.007	0.925
Global	IP2P	0.281	0.916	0.276	0.103	0.845
	Our	0.315	0.919	0.289	0.081	0.869
Background	IP2P	0.106	0.829	0.271	0.082	0.725
	Our	0.214	0.843	0.283	0.201	0.771
IP2P Dataset		0.172	0.855	0.271	0.119	0.809

8. Dataset Evaluation

In Sec. 3 we introduce our dataset generation pipeline, which includes methods that address the unique difficulties associated with each particular task. In this section we compare our approach with that of InstructPix2Pix. We begin by sampling 6,000 random samples from the same distribution of Sec. 3, following the instruction generation stage. Hence, each sample contains the input image, input caption, editing instruction, output caption, and the edited objects. We then generate image pairs using both our data generation pipeline, and that of InstructPix2Pix, which employs Prompt-to-Prompt and CLIP-based filtering. In Tab. 5 we report automatic metrics comparing the outputs of each pipeline. As can be seen, our method for data generations outperforms that of InstructPix2Pix (IP2P) on all the tasks. Additionally, to isolate the effect of our instruction generation stage, we also directly evaluate the InstructPix2Pix training dataset, which also underperforms when compared to ours.

Table 6. Number of images per task and split in our Image Editing Benchmark

Task	Validation set	Test set
Add	264	533
Background	266	373
Color	262	519
Global	220	219
Remove	264	531
Local	256	446
Style	227	434

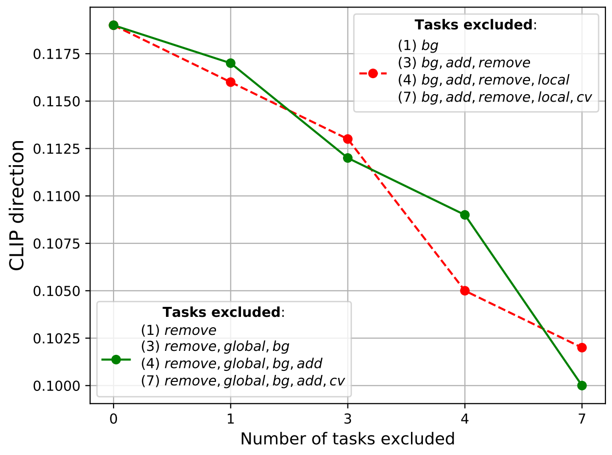


Figure 8. Ablation on the model performance (CLIP$_{dir}$) on Style and Texture tasks as we progressively exclude tasks that don't fall within these categories.

9. Image Editing Benchmark

We take the images from the MagicBrush benchmark [29] and undergo a three-step annotation process utilizing crowd workers: (i) instruction generation, (ii) instruction filtering, and, (iii) caption annotation. In the first step, three crowd workers are assigned to generate an instruction for each (image, task) pair. Moving to the second stage, five different crowd workers classify each (image, instruction) pair's task type and whether the instruction is relevant to the image. Instructions with at least one irrelevant annotation are then filtered out, and for the remaining ones, the task is determined through majority voting among the five workers. At this juncture, we select, at most, a single instruction for each (image, task) pair to preserve the benchmark's diversity.

Finally, we task crowd workers with annotating two captions for each remaining (image, instruction) pair - one for the image, and one for the desired image after having edited it. This facilitates automatic evaluation using the methodologies outlined in [9, 25]. Throughout this annotation phase, workers are presented with the input image and instruction, and are tasked with providing captions that faithfully describe the image while aligning with the given instruction. See Tab. 6 for the number of images per task and split in our benchmark.

10. Additional Results

10.1. Performance on Vision Tasks

We also evaluate the performance of our model on tasks other than edit, specifically: detection, segmentation, and depth estimation. We report: (i) Mean Average Precision (mAP@0.5) on MS-COCO [13] for detection task, (ii) Mean Intersection over Union (mIoU) on ADE20K [31, 32]

Table 7. Comparison of Emu Edit to task-specific experts on image-editing tasks. We report automatic metrics and human preference ratings. Human evaluation (%) is shown as a percentage of majority votes in favor of our multi-task model compared to an expert model.

Task	Method	CLIP$_{dir}\uparrow$	CLIP$_{img}\uparrow$	CLIP$_{out}\uparrow$	L1$\downarrow$	DINO$\uparrow$	Text Align.	Image Faith.
Local	Expert	0.139	0.879	0.244	0.057	0.841	-	-
	Our	0.142	0.885	0.252	0.047	0.891	57.5	56.9
Global	Expert	0.106	0.820	0.227	0.096	0.823	-	-
	Our	0.118	0.852	0.235	0.072	0.847	58.4	62.6
Add	Expert	0.119	0.851	0.237	0.059	0.828	-	-
	Our	0.123	0.917	0.240	0.036	0.892	61.1	59.6
Background	Expert	0.145	0.689	0.229	0.240	0.560	-	-
	Our	0.157	0.852	0.240	0.223	0.586	64.3	62.5

for segmentation task, and, (iii) Root Mean Square Error (RMSE) on NYUv2 [16] for monocular depth estimation. Emu Edit was not trained on those datasets, therefore, we report zero-shot results on both tasks, see Tab. 8.

Table 8. Emu Edit performance on vision tasks. For object detection we use mAP@0.5, for segmentation we use mIoU, and, for depth estimation we use RMSE.

Method	Object Detection$\uparrow$	Semantic Segmentation$\uparrow$	Depth Estimation$\downarrow$
Emu Edit	61.467	50.028	0.246

10.2. Additional ablations.

To further isolate each component's contribution, we train 3 additional variants and compare them to InstructPix2Pix (IP2P) in Tab.9. Each variant modifies one component, starting from IP2P baseline: (1) we replace IP2P's base model with Emu to isolate the base model contribution (**A1**), (2) we add task embedding to **A1**, demonstrating its significant impact on performance (**A2**). Although **A1**-**A2** and IP2P are trained on IP2P data, the best performance is achieved when utilizing the task embedding, reinforcing its importance. Last, we train IP2P on our dataset (**B**), yielding better results than IP2P but inferior to Emu Edit, highlighting both data and model contributions.

10.3. Controlling the Task Embedding

As depicted in Fig. 9, altering the task embedding controls the task executed by the model, resulting in different generations for a given instruction.

10.4. Influence of Number of Tasks

We report results for the ablation of the number of tasks in Fig. 8.

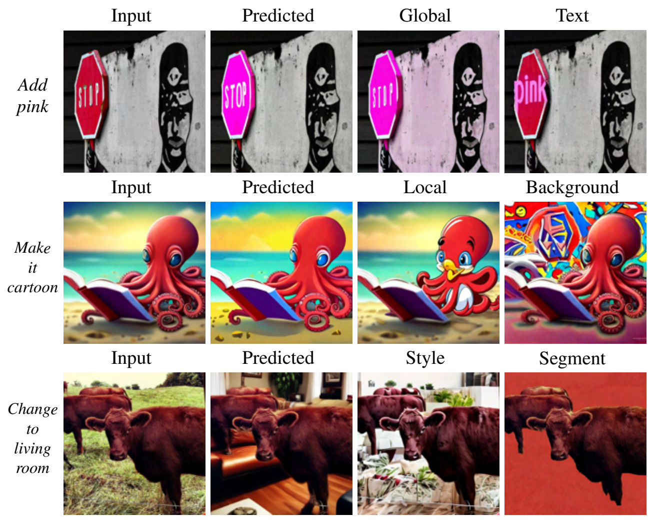


Figure 9. Controlling the Task Embedding. For each sample, we present the edited image using the task predicted by the task predictor. In addition, we present the edited image generated using the same input image and instruction, but with different task embeddings. For instance, in the first row we generate the edited image using the predicted task (Add), Global task, and Text task.

10.5. Few-Shot Learning of New Tasks

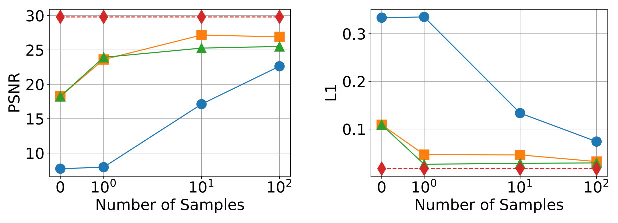
Fig. 12 illustrates generation examples produced by our model for various tasks learned in a few-shot setting. Additionally, the performance results of our model on the tasks of super resolution and contour detection are presented in Fig. 10.

10.6. Qualitative Comparisons with Existing Approaches

Figs. 14 and 15 shows qualitative comparisons with baselines on generated samples. In addition, in Figs. 18 and 19 we present qualitative comparisons of our model with baselines on Emu Edit test set.

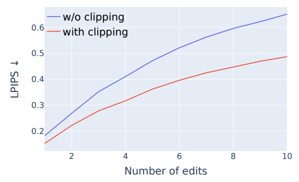
Table 9. Human evaluation shows the % of raters prefer the our model variant over InstructPix2Pix.

	Emu Edit Test Set						MagicBrush Test Set					
Variant	CLIP$_{dir}$	CLIP$_{img}$	CLIP$_{out}$	L1	Text Align.	Img Faith.	CLIP$_{dir}$	CLIP$_{img}$	CLIP$_{out}$	L1	Text Align.	Img Faith.
A1	0.089	0.849	0.226	0.117	63.3	61.8	0.120	0.864	0.256	0.086	60.9	63.7
A2	0.095	0.857	0.227	0.113	69.6	66.0	0.127	0.875	0.257	0.084	68.7	67.1
B	0.092	0.862	0.226	0.086	68.5	71.1	0.129	0.876	0.261	0.051	69.0	70.0



(a) Super Resolution (b) Contour Detection

Figure 10. Few-shot performance for different tasks over 1, 10, and 100 samples. Each line represents a different training setting: 'Scratch' finetune (Blue, ◯), Emu Edit finetune (Orange, □), task inversion (Green, △), all compared to an upper-bound expert trained on 100k samples (Red dashed line, ◇).



Metric	α				
	0	0.01	0.03	0.1	0.2
LPIPS	0.652	0.596	0.487	0.410	**0.348**
CLIP$_{out}$	0.225	0.225	**0.225**	0.216	0.194
CLIP$_{dir}$	0.135	0.139	**0.152**	0.138	0.101

Figure 11. Sequential editing in multi-turn. **left**-the influence on image quality degradation, **right**-the influence of different threshold (α) values.

10.7. Sequential Edit Thresholding

To demonstrate the generic utility of our sequential editing technique, we conduct the following experiment: we sample 200 images from Emu Edit development set, and generate 10 consecutive instructions for each. We then apply multi-turn editing with and without our technique. We report LPIPS results between the input and edited images for each turn in Fig.11-left and observe a significant quality improvement with our technique. Tab.11-right examines the influence of different thresholds after 10 edits. As can be seen, a value of 0.03 strikes a good balance between image quality and text faithfulness.

Qualitative comparison. In Fig. 13, a qualitative comparison is provided to demonstrate the effectiveness of the proposed technique in maintaining image quality during multi-turn editing scenarios. Specifically, we vary the value of the hyperparameter α, which controls the degree to which pixel values are used during the editing process. With $\alpha = 0$, no thresholding is applied and the output image is simply the result of passing the input image through the model. Conversely, when $\alpha = 1$, the input image is used as the output image without any editing. We present results for several values of α, including $0.5, 0.25, 0.1, 0.05, 0.025, 0.01$, and the baseline value of 0. As can be observed from the figure, when no clipping is applied (i.e., $\alpha = 0$), artifacts tend to accumulate and manifest as general noise in the output image. On the other hand, applying a threshold helps preserve the quality of the output image even when multiple edit turns are applied. However, using a large value of α can interfere with the editing process and result in poor edit quality. Based on these observations, we opt to use a value of $\alpha = 0.03$ in our experiments, as it strikes a balance between preserving image quality and allowing for effective editing.

11. Implementation Details

We use a scaled-down version of [6] which is conditioned on CLIP ViT-L [18] and T5-XL [19], and generates images at a resolution of 512×512. We adapt it to obtain image inputs by concatenating to the input channels following [2]. We condition on the text and task embeddings both through cross-attention and by addition to the timestep embeddings. For training, we employ the Adam optimizer with a batch size of 512. We use a learning rate of 2e-5 with a cosine decay schedule and a linear warmup of 2,000 iterations. Emu Edit is trained for 50k iterations on 128 A100 GPUs for 19 hrs. During inference, Emu Edit runs for 50 diffusion steps in 4 secs. Optimizing a task embedding for new tasks takes 15 min on a single GPU.

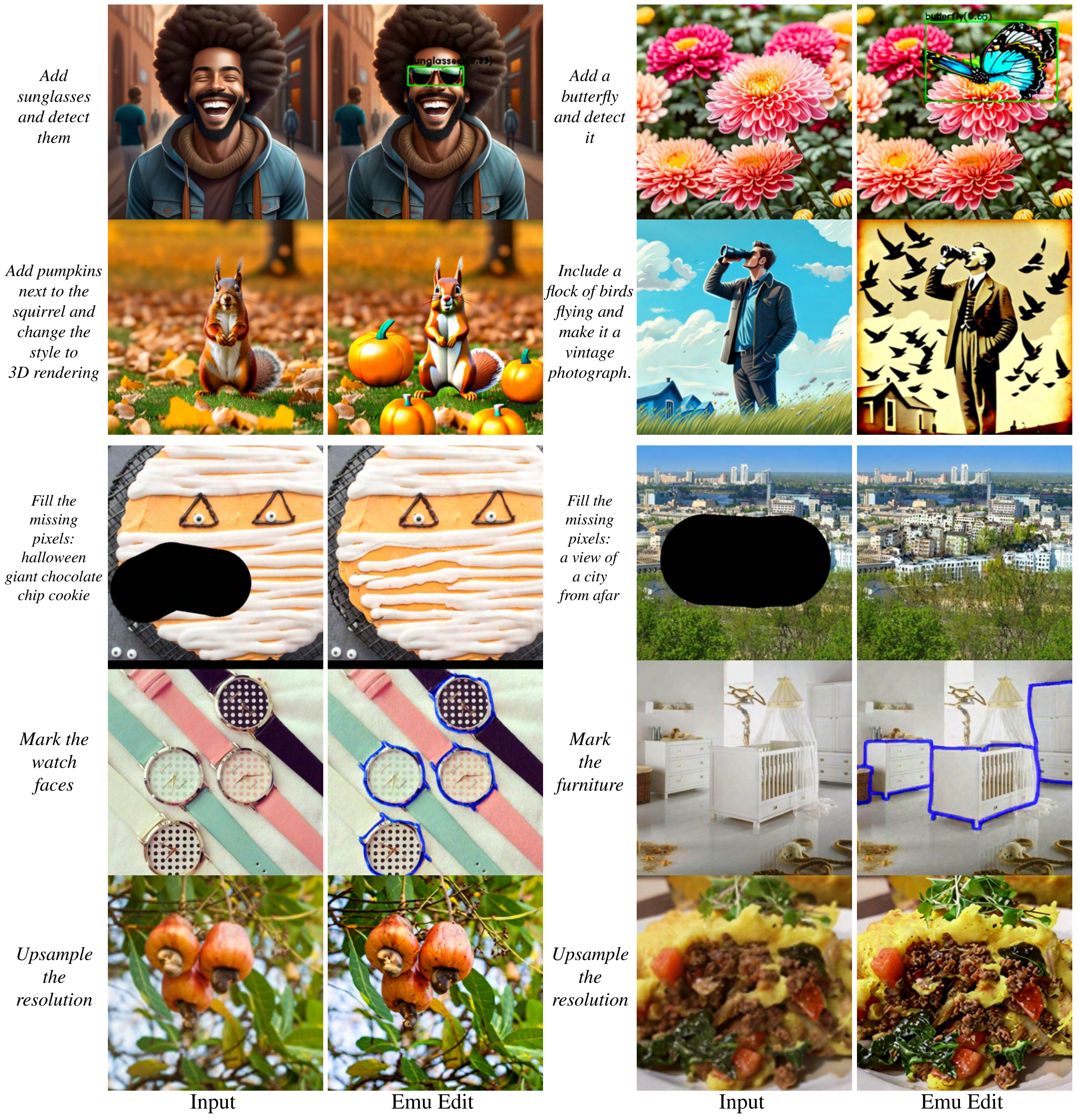

Add sunglasses and detect them
Add a butterfly and detect it
Add pumpkins next to the squirrel and change the style to 3D rendering
Include a flock of birds flying and make it a vintage photograph.
Fill the missing pixels: halloween giant chocolate chip cookie
Fill the missing pixels: a view of a city from afar
Mark the watch faces
Mark the the furniture
Upsample the resolution
Upsample the resolution
Input
Emu Edit
Input
Emu Edit

Figure 12. Generations of our model on unseen tasks with task inversion. From top to bottom: (i) composition of add and detect tasks, (ii) composition of add and style tasks, (iii) image in-painting, (iv) contour detection, (v) super-resolution.

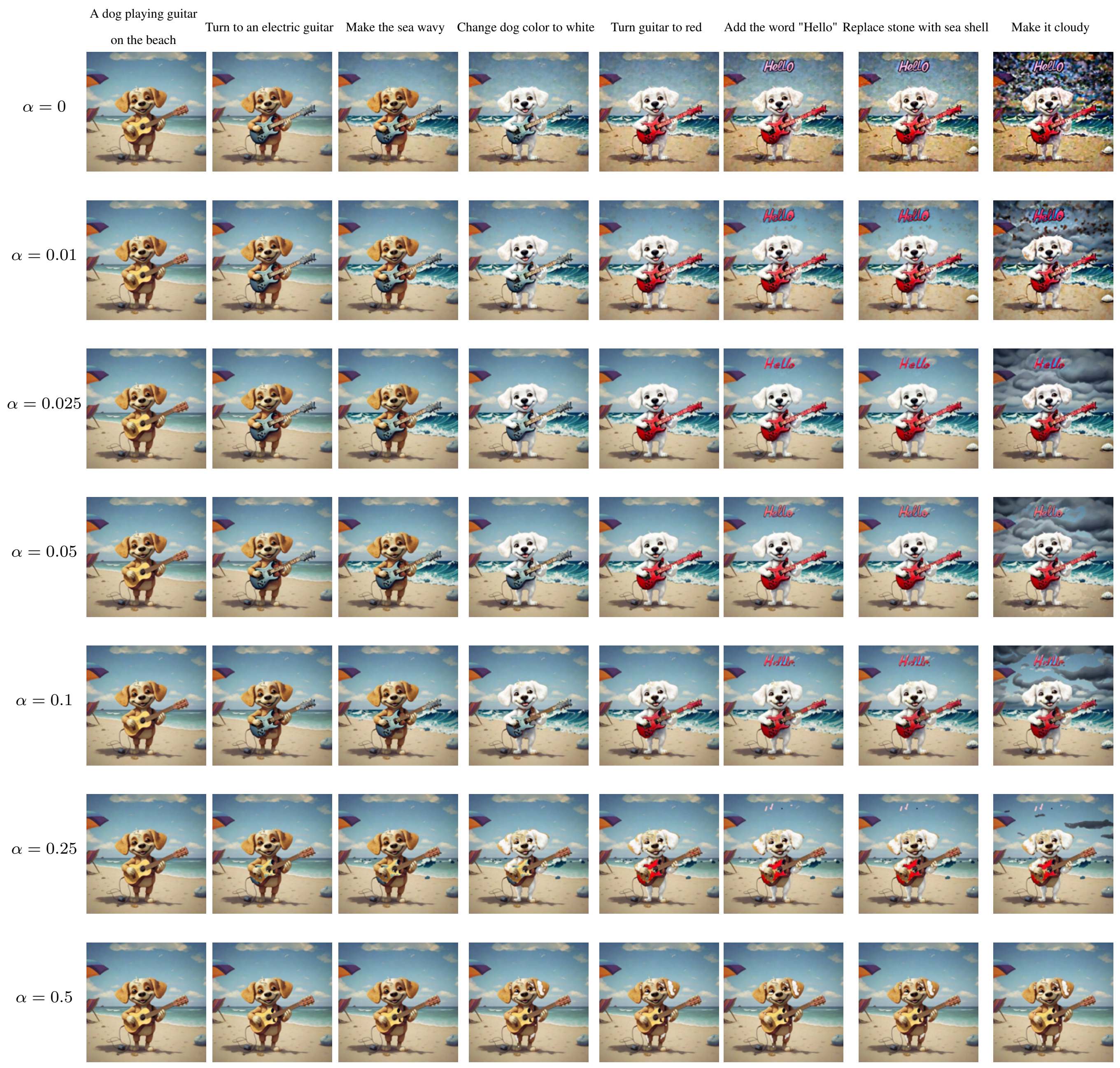


Figure 13. Effect of Sequential Edit Thresholding during sequential edits (from left to right) with different α values.

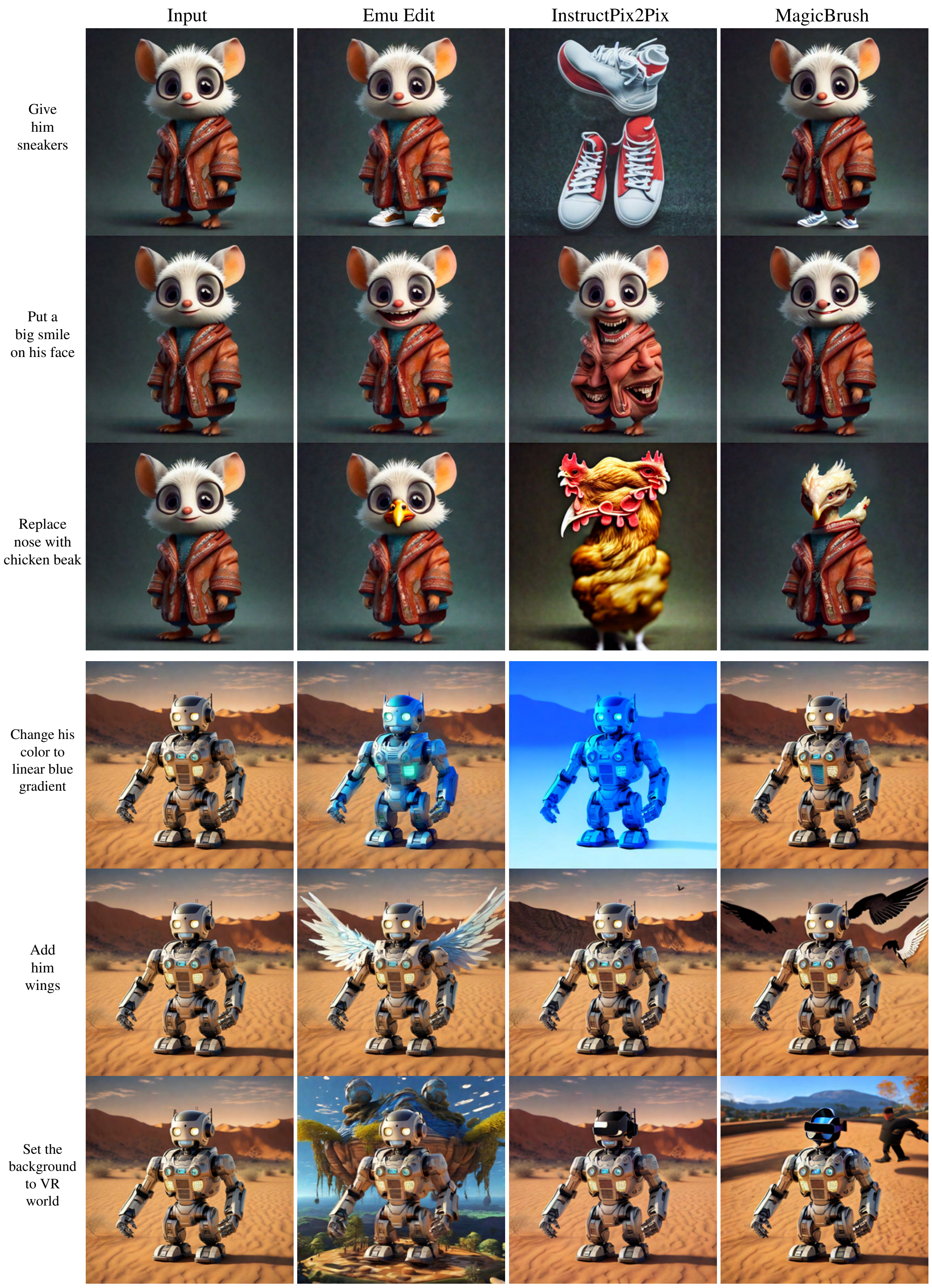

Input
Emu Edit
InstructPix2Pix
MagicBrush
Give him sneakers
Put a big smile on his face
Replace nose with chicken beak
Change his color to linear blue gradient
Add him wings
Set the background to VR world

Figure 14. Qualitative comparison with baselines.

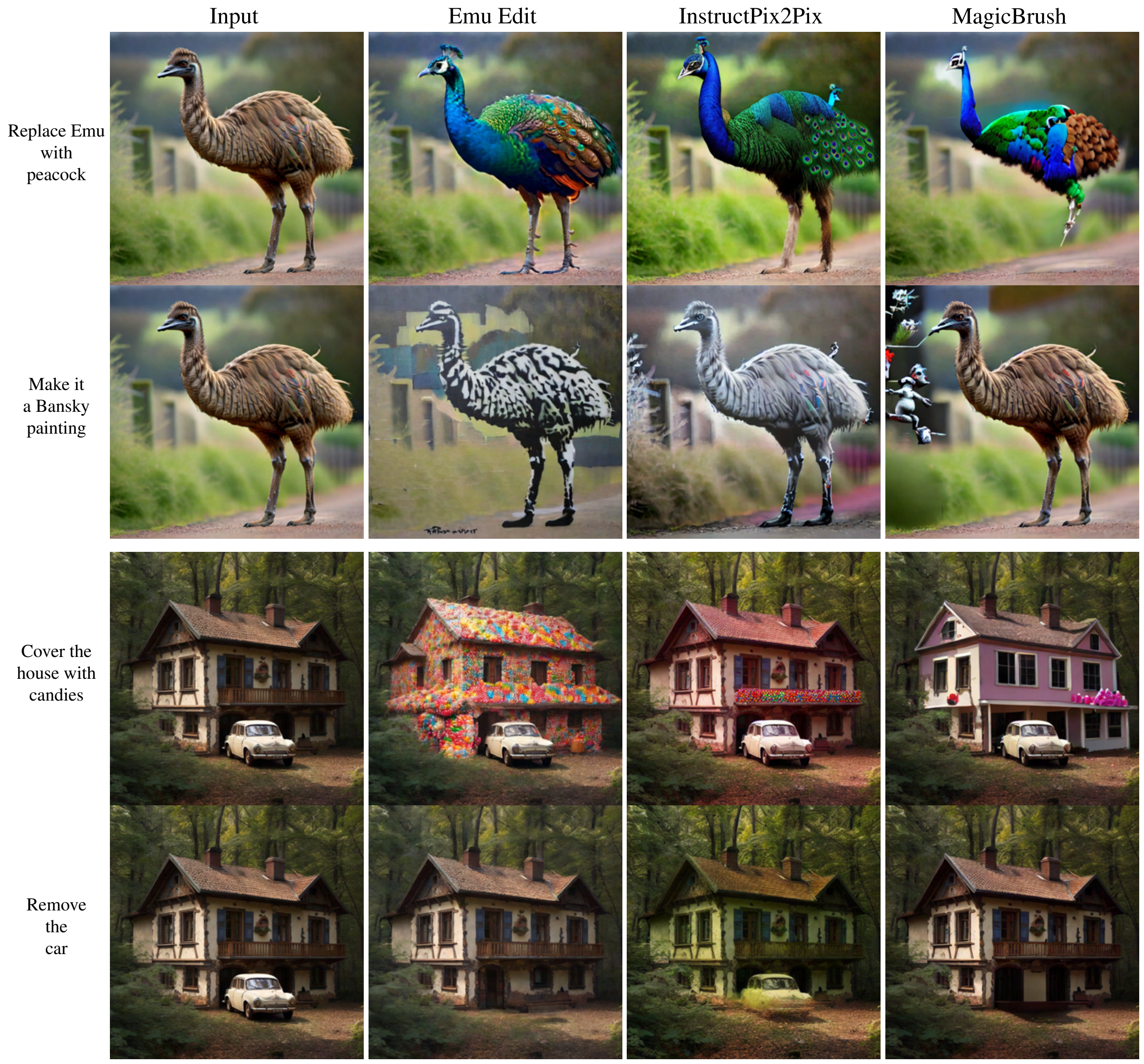


Figure 15. Qualitative comparison with baselines.

```python
def get_content_instruction(new_prompt):
    optional_verbs = choice(["include", "place", "position", "set",
    "incorporate", "alongside", "give", "put", "insert", "together with",
    "with", "make", "integrate", "have", "append", "make", "add", "include"])

    # system message #
    system_message =
    f"<<SYS>>
    You are an assistant that only speaks JSON. Do not write normal text.
    The assistant answer is JSON with the following string fields:
    'edit', 'edited object','output'. Here is the latest conversation between
    Assistant and User.
    <</SYS>>"

    # introduction message #
    intro_message =
    f"[INST]User: Hi, My job to take a given caption ('input') and to output the following:
        an instruction for {optional_verbs} an object to the image ('edit'), the object to
        {optional_verbs} ('edited object'), and the caption with the object ('output').
        Please help me do it. I will give you the 'input', and you will help.
        When you reply, use the following format:
        {"edit": '<instruction>', 'edited object': '<object>', 'output': '<caption>'}[/INST]
    Assistant: Sure, I'd be happy to help! Please provide the actual input caption you'd
        like me to read and I'll assist you with writing an instruction to {optional_verbs} an
        object to the image, writing the added object and writing the caption with the object."


    # shuffling #
    random.seed(torch.randint(1 << 32, ()).item())
    shuffle(few_shot_examples)
    few_shot_examples = few_shot_examples[:int(len(few_shot_examples) * 0.6)]
    prompt = system + intro_message + "".join(few_shot_examples)

    # add the test prompt #
    prompt = prompt + f"[INST]User: {new_prompt}[/INST]"

    return prompt
```

Figure 16. An example of in-context learning for the task of Add.

```
few_shot_examples = [
[INST]User: "Beautiful cat with mojito sitting in a cafe on the street"[/INST]
Assistant: {"edit":  "include a hat", "edited object":  "hat", "output":  "Beautiful cat wearing a
    hat with mojito sitting in a cafe on the street"}
[INST]User:  "robot playing chess at home."[/INST]
Assistant: {"edit":  "add a cheerful smiling face.", "edited object":  "robot", "output":  "robot
    playing chess at home with a cheerful smiling face."}
[INST]User:  "A cute creature sits at the beach."[/INST]
Assistant: {"edit":  "set a dog besides the creature", "edited object":  "dog", "output":  "A cute
    creature and a dog sit at the beach."}
[INST]User:  "Superhero on the street in sunny day working on his tablet."[/INST]
Assistant: {"edit":  "put a vintage tie on the superhero.", "edited object":  "tie", "output":
    "Superhero with a vintage tie on the street in sunny day working on his tablet."}
[INST]User:  "Picture clouds, birds, the wind, foliage, rainbow, hill, art, pair, guy"[/INST]
Assistant: {"edit":  "together with a dog on the left", "edited object":  "dog", "output":
    "Picture clouds, birds, the wind, foliage, rainbow, hill, art, pair, guy, dog on the left"}
[INST]User:  "horse on a red Boat Near Mountains During Golden Hour"[/INST]
Assistant: {"edit":  "give the horse sunglassess", "edited object":  "sunglassess", "output":
    "horse with sunglassess on a red Boat Near Mountains During Golden Hour"}
[INST]User:  "An animal family on studio background."[/INST]
Assistant: {"edit":  "make them hold a teddy bear.", "edited object":  "teddy bear", "output":  "An
    animal family holding a teddy bear on studio background."}
[INST]User:  "Baked Salmon With Bell Peppers"[/INST]
Assistant: {"edit":  "insert kale pesto to the dish", "edited object":  "kale pesto", "output":
    "Baked Salmon With Kale Pesto And Bell Peppers"}
[INST]User:  "An airplaine is flying in the sky in rainy day."[/INST]
Assistant: {"edit":  "add flowers in the windows", "edited object":  "flowers", "output":  "An
    airplaine with flowers in the windows is flying in the sky in rainy day."}
[INST]User:  "photo of mountains and trees"[/INST]
Assistant: {"edit":  "position a castle between the trees", "edited object":  "castle", "output":
    "photo of mountains, trees and castle between the trees"}
[INST]User: "Little bunny in the park"[/INST]
Assistant: {"edit":  "Make the bunny play with a kite.", "edited object":  "kite", "output":
    "Little bunny playing with a kite in the park"}
[INST]User: "Attic Bedroom With Large Ceilings"[/INST]
Assistant: {"edit":  "decorate the room with beautiful chandeliers", "edited object":
    "chandeliers", "output":  "Attic Bedroom With Beautiful Chandeliers on Large Ceilings"}
[INST]User: "Wedding rings and yellow flower on a red background"[/INST]
Assistant: {"edit":  "place a bird by the yellow flower", "edited object":  "bird", "output":
    "Wedding rings, a bird, and yellow flower on a red background"}
[INST]User: "Tree Near the lake in the morning"[/INST]
Assistant: {"edit":  "Give it autumn leaves on top", "edited object":  "leaves", "output":  "Tree
    with autumn leaves on top Near the lake in the morning"}
[INST]User: "robot and alien sitting on hanging bridge at daytime"[/INST]
Assistant: {"edit":  "make them hold three books.", "edited object":  "three books", "output":
    "robot and alien holding three books while sitting on hanging bridge at daytime"}
[INST]User: "Skogafoss waterfall in the south of Iceland"[/INST]
Assistant: {"edit":  "Set a colorful rainbow in the backhground!", "edited object":  "rainboe",
    "output":  "Skogafoss waterfall with a colorful rainbow in the south of Iceland"}
[INST]User: "Polar Bear with rubber gloves pushing shopping carts"[/INST]
Assistant: {"edit":  "Make it wear a coat", "edited object":  "coat", "output":  "Polar Bear with a
    coat pushing shopping carts"}
]
```

Figure 17. Examples of prompts for Add task.

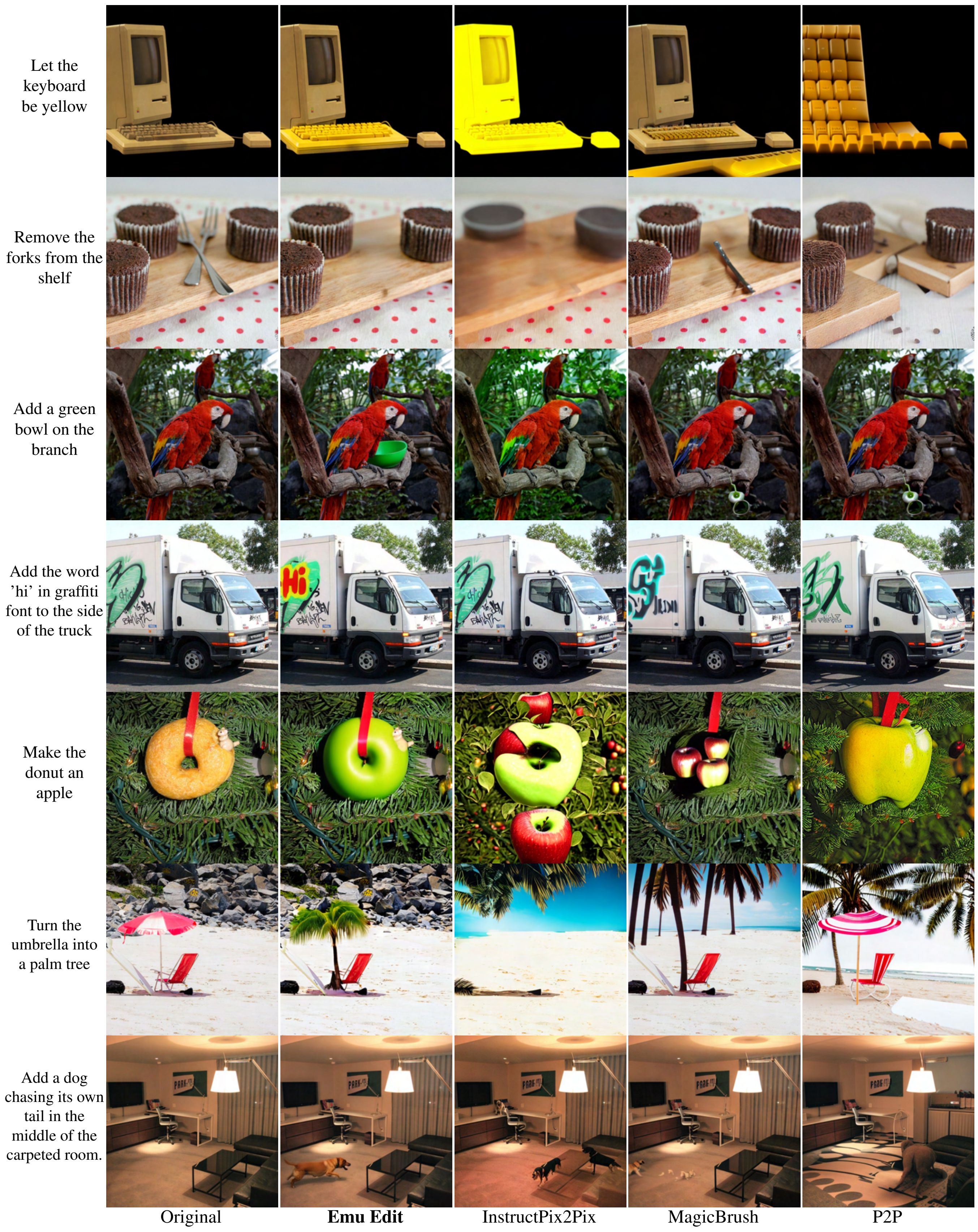

Let the keyboard be yellow
Remove the forks from the shelf
Add a green bowl on the branch
Add the word 'hi' in graffiti font to the side of the truck
Make the donut an apple
Turn the umbrella into a palm tree
Add a dog chasing its own tail in the middle of the carpeted room.
Original
Emu Edit
InstructPix2Pix
MagicBrush
P2P

Figure 18. Qualitative comparison of our model to baselines on Emu Edit Test Set.

Turn the refrigerator into a bookshelf with books

Change the image to have a 1970s pop art style.

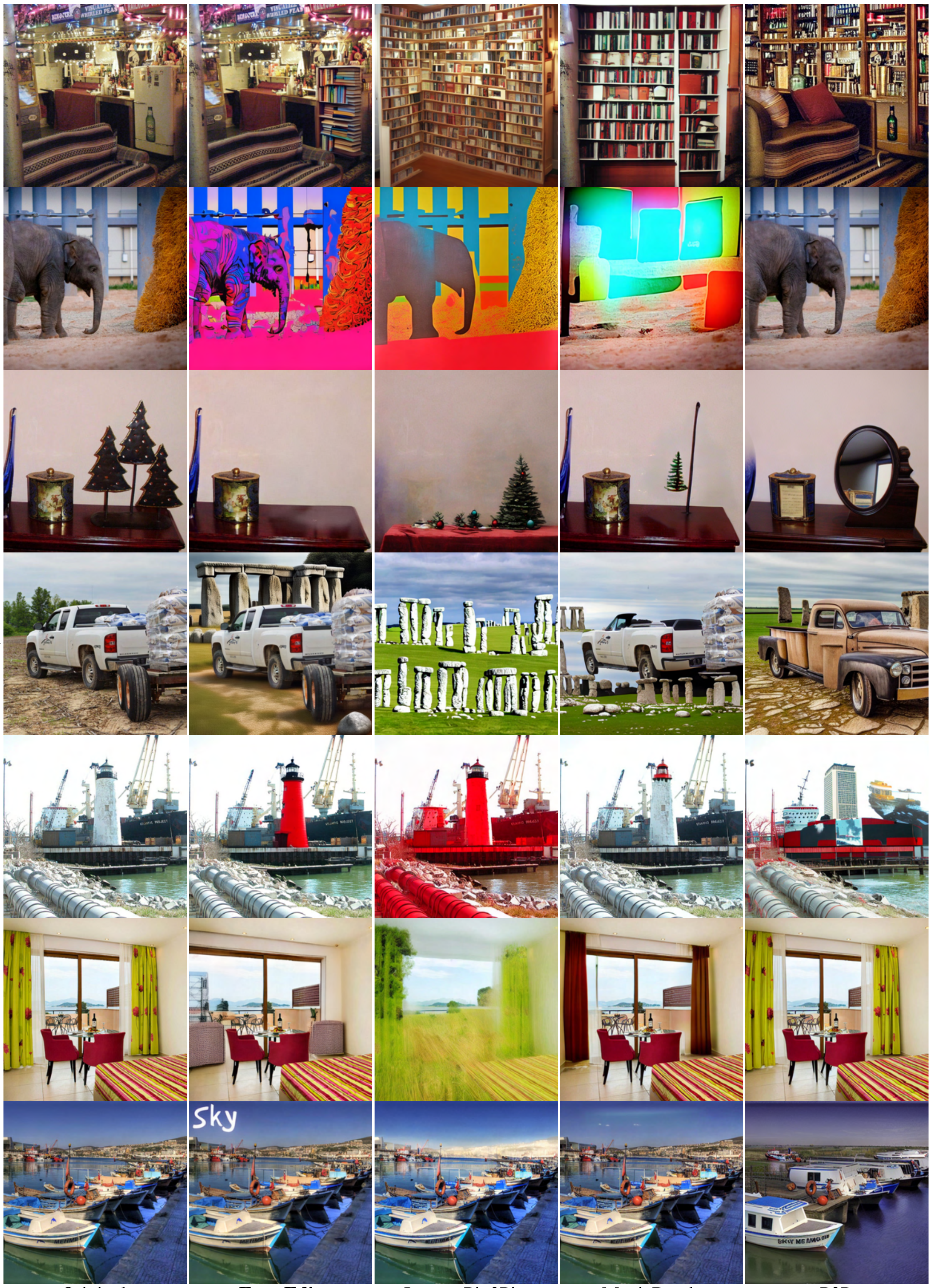
Remove the Christmas trees on the table

Put Stone Henge as the background of the scene.

Change the color of the lighthouse into completely red.

Remove the curtains.

Add the word 'sky' in white to the sky.



Original **Emu Edit** InstructPix2Pix MagicBrush P2P

Figure 19. Qualitative comparison of our model to baselines on Emu Edit Test Set.