Emu Edit: Precise Image Editing via Recognition and Generation Tasks

Shelly Sheynin*, Adam Polyak*, Uriel Singer*, Yuval Kirstain*, Amit Zohar*, Oron Ashual,
Devi Parikh and Yaniv Taigman

GenAI, Meta

Dress the emu with a fireman outfit *Let's see it graduating* *Mark the drinks*

Figure 1. Emu Edit is a multi-tasking model that combines various editing (left, middle) and vision (right) tasks for precise image editing.

Abstract

Instruction-based image editing holds immense potential for a variety of applications, as it enables users to perform any editing operation using a natural language instruction. However, current models in this domain often struggle with accurately executing user instructions. We present Emu Edit, a multi-task image editing model which sets state-of-the-art results in instruction-based image editing. To develop Emu Edit we train it to multi-task across an unprecedented range of tasks, such as region-based editing, free-form editing, and Computer Vision tasks, all of which are formulated as generative tasks. Additionally, to enhance Emu Edit's multi-task learning abilities, we provide it with learned task embeddings which guide the generation process towards the correct edit type. Both these elements are essential for Emu Edit's outstanding performance. Furthermore, we show that Emu Edit can generalize to new tasks, such as image inpainting, super-resolution, and compositions of editing tasks, with just a few labeled examples. This capability offers a significant advantage in scenarios where high-quality samples are scarce. Lastly, to facilitate a more rigorous and informed assessment of instructable image editing models, we release a new challenging and versatile benchmark that includes seven different image editing tasks.[1]

1. Introduction

Image editing is a widely-used application that millions engage with every day. Popular image editing tools, however, either demand considerable expertise and are time-consuming to use, or are quite limited, providing only a predefined set of editing operations, such as specific filters. Instruction-based image editing [2, 29] attempts to resolve these limitations by allowing users to effortlessly describe their editing goals using natural language instructions. For instance, a user can provide a model with an image and instruct it to "Dress the emu with a fireman outfit" or "Let's see it graduating" (see Fig. 1).

Nevertheless, while instruction-based image editing models like InstructPix2Pix [2] are *designed* to process any given instruction, they often struggle to *accurately* interpret and execute such instructions. Moreover, their generalization is limited, often falling short on tasks that deviate slightly from those they were trained on (see Fig. 3). To address these gaps, we introduce Emu Edit, the first image editing model trained on an extensive and diverse set of tasks, including both image editing and computer vision tasks. Emu Edit provides a substantial improvement in both compliance with the edit instruction and preservation of the visual fidelity of the original image. As we show through both automatic metrics [18] and human judgments on two benchmarks [29], Emu Edit achieves state-of-the-art results in instruction-based image editing.

The success of Emu Edit stems from two key contributions. First, we train our model to multi-task across sixteen distinct image editing tasks. These tasks span region-based

[1]Project Page: https://emu-edit.metademolab.com/
*Equal contribution.

(a) *A cat* (b) *Remove the tail* (c) *Add a pink jacket* (d) *Make it rainy* (e) *Have the cat look shocked*

(f) *Extract the depth map* (g) *Generate a rainy day image of a hedgehog in a dress using the depth map* (h) *Replace the dress with an astronaut outfit* (i) *Segment the spacesuit, and detect the hands* (j) *Add the text "purple cat" using a purple font*

Figure 2. **Multi-turn image editing.** Each subsequent image is derived from the prior one, using its associated caption. The initial image is based on a zeroed reference.

editing tasks, free-form editing tasks and computer vision tasks, all formulated as generative tasks. Unlike previous work, we develop a distinct data curation pipeline for each task, allowing us to gather a training set that is not only more diverse but also more precise in its examples. We find that training a single model on all tasks yields better results than training expert models on each task independently. Additionally, we show that as the number of training tasks increases, so does the performance of Emu Edit. Furthermore, we discover that surprisingly, computer vision tasks such as detection, segmentation, and others, significantly enhance editing performance, as validated both by human raters as well as quantitative measures.

Second, to process this wide array of tasks effectively, we introduce the concept of *learned task embeddings*, which are used to steer the generation process toward the correct generative task. Concretely, for each task, we learn a unique task embedding vector, and integrate it into the model through cross-attention interactions, and by adding it to the timestep embeddings. We demonstrate that learned task embeddings significantly enhance the ability of our model to accurately infer the appropriate edit type from the free-form instruction and execute the correct edit.

Equipped with a robust model trained across a broad spectrum of tasks and guided by learned task embeddings, we explore few-shot adaptation to unseen tasks via *task inversion*. In this process, we maintain the model weights untouched, and solely update a task embedding to fit the new task. Our experiments demonstrate that Emu Edit can swiftly adapt to new tasks, such as super-resolution, contour detection, and others. Notably, for some tasks, fine-tuning the model on just a *handful* of examples yields results that nearly match those of an expert model trained on *one hundred thousand* examples. This makes task inversion with Emu Edit particularly advantageous in scenarios where labeled examples are limited, or when the compute budget is low. Finally, to support future research for instruction-based image editing, we publicly release a diverse and challenging benchmark that includes seven different image editing operations, as well as our model's generations on this dataset.

In summary, this work addresses the limitations of instruction-based image editing models in accurately following user instructions. We demonstrate that by employing multi-task training across a diverse array of tasks, including recognition, generation, and editing, we can enhance our model's performance. Furthermore, by incorporating learned task embeddings into our model's architecture, we not only improve its results but also enable efficient few-shot learning for new tasks. With these improvements, our model sets a new standard in the field, offering significantly more precise and robust instruction-based image editing capabilities than existing models.

2. Related Work

The emergence of high-performing text-to-image diffusion models [8, 20, 21, 23] facilitated the development of effective text-based image editing methods. Such methods usually employ aligned and detailed descriptions of the input and edited image to perform a specific edit. Prompt-to-

Prompt (P2P) [9] injects the input caption attention maps to the target caption attentions maps. Null-Text Inversion [15] inverts an input image using the null-text embedding to support editing of a real image. Plug-and-Play (PNP) [25] injects spatial features in addition to attention maps and obtains better performance at global editing. Imagic [10] finetunes the diffusion model to support complex textual instructions. EDICT [26] suggests an image inversion based on two noise vectors enabling better image reconstruction and textual faithfulness. Another class of image editing models, employs an input mask as additional input [1, 27, 28]. Blended Diffusion [1] modifies the diffusion step by blending the input image in the unmasked regions. Imagen Editor [27] and SmartBrush [28] finetune the text-to-image model to be conditioned on both the input image and mask. While the text-based image editing methods detailed above enable humans to edit images, they frequently exhibit inconsistent performance and require multiple inputs, such as aligned and detailed descriptions of both the input and target images, or at times, input masks.

To offer a more intuitive and user-friendly interface, and significantly enhance ease of use for humans, InstructPix2Pix [2] introduced an instructable image editing model. They developed this model by utilizing both GPT-3 [3] and Prompt-to-Prompt [9], to generate a large synthetic dataset for instruction-based image editing, and employed the dataset to train an instructable image editing model. Unlike InstructPix2Pix which used a synthetic dataset, MagicBrush [29] developed a manually-annotated instruction-guided image editing dataset by requesting humans to use an online image editing tool [17]. Finetuning InstructPix2Pix on this dataset led to improved image editing capabilities. However, even though there has been progress and improvement in instruction-based image editing models, we show in Sec. 5 that state-of-the-art image editing models still struggle with accurately interpreting and precisely executing editing instructions.

In this paper, we drastically narrow such performance gaps by leveraging multi-task training and a matching architecture. Unlike prior work that solely focuses on image editing [2, 29], we train our model to perform various tasks and learn a very diverse set of capabilities. The quality and versatility of our training procedure and dataset, together with our improved architecture for multi-task learning, enables us to make a big leap in performance and differentiates us from prior work in the field. Fig. 3 include several challenging editing samples as examples.

3. Multi-Task Dataset for Image Editing

Training a robust and accurate image editing model requires a highly diverse dataset of input images, editing instructions, and output edited images. However, manually collecting such examples is impractically time-consuming, existing sources on the web (e.g. communities and forums on social media) are limited in size, and publicly available synthetic datasets often lack in diversity or quality. Therefore, we construct a new dataset that encompasses sixteen distinct tasks and ten million examples. Each example (c_I, c_T, x, i) in our dataset, contains an input image c_I, a text instruction c_T, a target image x, and a task index i (out of the sixteen). The following section outlines the process of the data construction. In Sec. 3.2 we describe the instruction generation process, and in Sec. 3.3 the image pairs (c_I, x) generation and filtering.

1. Region-Based Editing
- **Local**: Substituting one object for another, altering an object's attributes (e.g., "make it smile").
- **Remove**: Erasing an object from the image.
- **Add**: Inserting a new object into the image.
- **Texture**: Altering an object's visual characteristics without affecting its structure (e.g., painting over, filling or covering an object).
- **Background**: Changing the scene's background.

2. Free-Form Editing
- **Global**: Edit instructions that affect the entire image, or that can not be described using a mask (e.g., "let's see it in the summer").
- **Style**: Change the style of an image.
- **Text Editing**: This involves text-related editing tasks such as adding, removing, swapping text, and altering the text's font and color.

3. Vision tasks
- **Detect**: Identifying and marking a specific object within the image with a rectangle bounding box.
- **Segment**: Isolating and marking an object in the image.
- **Color**: Color adjustments like sharpening and blurring.
- **Image-to-Image Translation**: Tasks that involve bi-directional image type conversion, such as sketch-to-image, depth map-to-image, normal map-to-image, pose-to-image, segmentation map-to-image, and so on.

Table 1. Description of the tasks forming the Emu Edit dataset.

3.1. Task Categories

The dataset is composed of tasks which are divided into three main categories: region-based editing, free-form editing, and vision tasks. Tab. 1 includes the full list of tasks, and their distribution in the train set is visualized in Fig. 7.

3.2. Instruction Generation

To generate editing instructions, we leverage the dialogue-optimized 70 billion parameter Llama 2 variant [24]. We observed that using a single agent to generate the instructions for all tasks leads to a lack of diversity in the dataset. Notably, the LLM exhibits a bias towards particular tasks

and instruction phrasings. To address this, we utilize in-context learning to create a task-specific agent for each task. Concretely, we provide the LLM with a task description, a few task-specific exemplars, and a real image caption. To increase diversity we sample the exemplars and randomize their order. Given such input, we expect the LLM to output (1) an editing instruction, (2) an output caption for an ideal output image, and (3) which objects should be updated or added to the original image. We refer the readers to Fig. 15-16 for examples of our prompts. Further details on instruction generation are provided in Appendix 7.1.

3.3. Image Pairs Generation

Our aim is to generate pairs of input and edited images that adhere to the edit instructions and preserve image elements that should remain intact. In order to address the unique challenges associated with each task and create a high quality dataset, we develop a novel generation technique for each task. In the following subsections, we first detail our core improvements to the Prompt-to-Prompt algorithm, utilized when generating data for Local, Add, Remove, Texture, and Global tasks. Secondly, we describe our post-processing filtering. Full details can be found in Appendix 7.2. For region-based tasks, see Appendix 7.2.3. Free-form tasks are described in detail in Appendix 7.2.4. Finally, all vision tasks are described in Appendix 7.2.5.

Grounded Precise Editing. A crucial prerequisite when creating a pair of input and edited images is to guarantee that the two images differ only in specific elements or locations, while remaining identical in all other aspects. Previous instruct-based image editing methods [2] rely on Prompt-to-Prompt (P2P) to build an image-editing dataset. P2P injects cross-attention maps from the input image generation to the edited image generation. To support local edits, P2P additionally approximates a *mask* of the edited part, based on the cross-attention maps and constrains the edit to this local area. P2P relies on word-to-word alignment between the input image caption and the edited image caption (e.g. "a cat riding a *bicycle*" and "a cat riding a *car*") to produce editing image pairs. However, when there is no word-to-word alignment, the resulting mask tends to be imprecise due to its reliance on cross-attention maps. Furthermore, as word-to-word alignment is not a practical assumption in most of the image editing tasks, this approach often fails to preserve structure and identity. To address this challenge, we propose a mask extraction method, which is applied before the editing process. Our approach involves: (i) identifying the edited areas from the editing instruction via an LLM and creating corresponding masks before image generation, and (ii) integrating these masks during the editing process to ensure seamless fusion of edited regions with the original image. Further description of the method is found at Appendix 7.2.1.

Distinct editing challenges, such as adding or removing objects, require tailored solutions. We utilize various techniques, including dilation and Gaussian blurring, to refine the masks. We describe the mask extraction process in detail in Appendix 7.2.2.

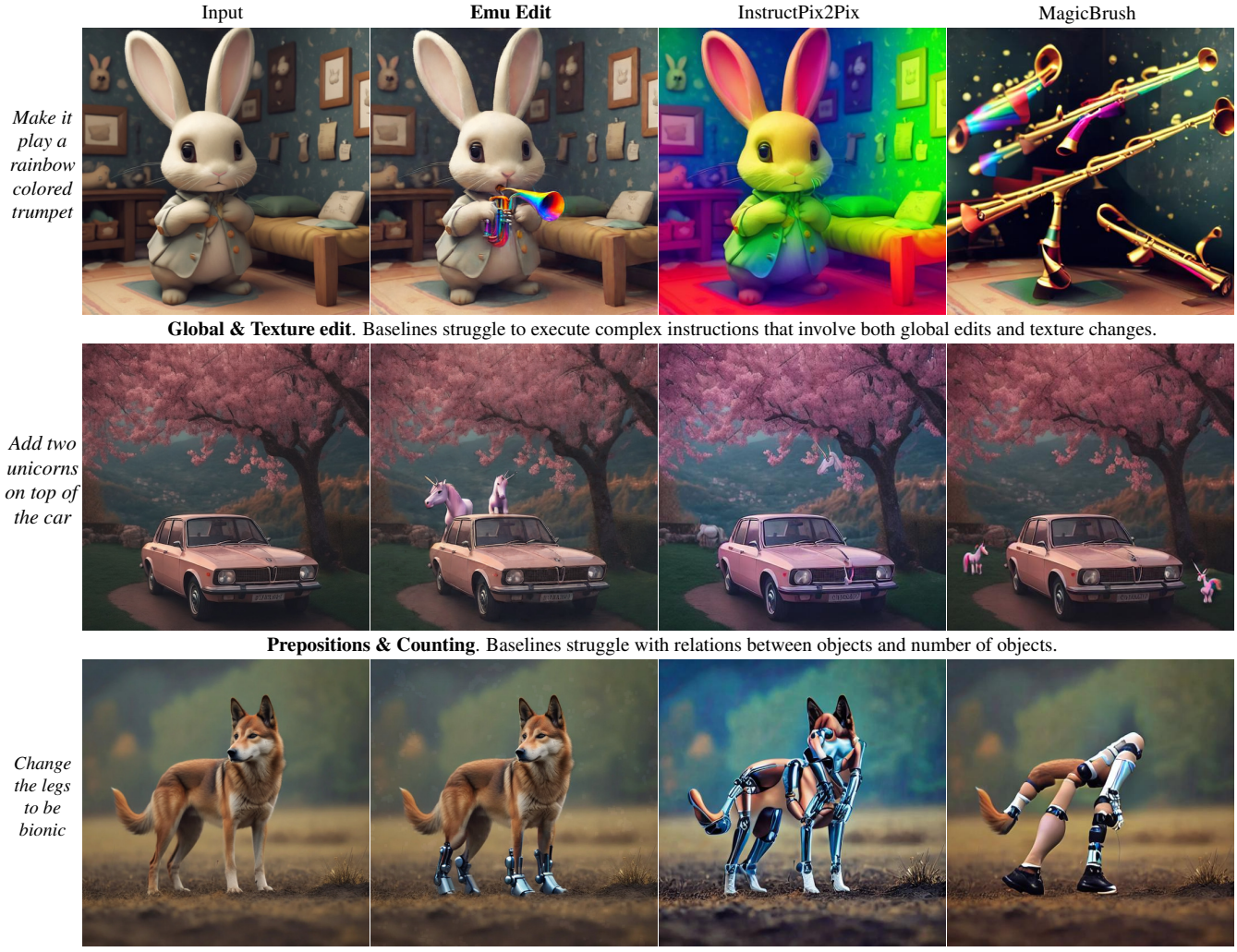
Filtering. We employ a comprehensive filtering approach to ensure the fidelity of the dataset. This includes: (i) using the task predictor (Sec. 4.2) to reassign samples with instructions that should belong to another task, (ii) applying CLIP filtering metrics [2], (iii) employing structure preserving filtering based on the L1 distance between the depth map of the input image and the edited image, and (iv) applying image detectors to validate the presence (in Add task), the absence (in Remove task) or replacement (in Local task) of elements, according to the objects specified in the instruction. This process filters out 70% of the data, resulting in a final dataset of ten million samples.

4. Method

Emu Edit is a diffusion model designed to multi-task across a broad spectrum of editing tasks. These include region-based and free-form image editing tasks, as well as traditional computer vision tasks like detection, segmentation, and depth estimation, all of which are formulated as generative tasks. As Emu Edit is trained on various tasks, a crucial aspect is the ability to identify the semantic edit (e.g., global/local/texture) that needs to be applied, based on the user instruction. However, in cases where the instruction is very unique (such as "fix the bumper" in Fig. 4), or when there is ambiguity regarding the edit type (e.g. "change the sky to be gray" in Fig. 4 can be interpreted as both Global edit and Texture edit), the model may encounter difficulty determining the expected edit type. To provide the model with a strong condition that will steer the generation process toward the correct task, we propose learning a unique task embedding for each task, which we integrate into the model. During training, the task embeddings are learned together with model weights. Post training, Emu Edit is able to adapt to new tasks via few-shot learning a new task embedding, leaving the rest of the model frozen. Last, we introduce a method to preserve the quality of the generated images in multi-turn editing scenarios. We follow next with a detailed description of each part of our method.

4.1. Architecture

Our model builds upon the foundation set by Emu, which is outlined in [6]. The Emu model is a two-stage approach that begins with a pre-training phase and concludes with a quality fine-tuning stage. The pivotal aspect of the method is that the fine-tuning dataset is relatively small, comprising only a few thousand images, but must be of exceptional quality, often necessitating manual annotation. Emu adapted the latent diffusion model architecture [22] to sup-



Global & Texture edit. Baselines struggle to execute complex instructions that involve both global edits and texture changes.

Prepositions & Counting. Baselines struggle with relations between objects and number of objects.

Local. Baselines struggle to perform intricate local edits.

Figure 3. Failure cases of baseline instruction-based image editing models.[2]

port high-resolution image generation and incorporated a 16-channel autoencoder with encoder E and decoder D. In the following section, we adapt the notation of [2]. A large U-Net, ϵ_θ, with 2.8 billion parameters, θ, text embeddings from CLIP ViT-L [18] and T5-XXL [19], and a substantial pre-training dataset of 1.1 billion images facilitate the model's ability to learn complex semantics and finer details, with a noise-offset strategy contributing to high-contrast and aesthetically pleasing image generation. Given the encoded latent of an image $z = E(x)$, the diffusion process generates a noisy latent z_t where the noise level increases over timesteps $t \in T$. To convert Emu to an instruction-based image editing model, we condition it on the image to be modified c_I and the instruction c_T. Emu Edit is trained to minimize the following optimization problem,

$$\min_\theta \mathbb{E}_{y,\epsilon,t}\left[\|\epsilon - \epsilon_\theta(z_t, t, E(c_I), c_T)\|_2^2\right] \quad (1)$$

where $\epsilon \in N(0,1)$ is the noise added by the diffusion process and $y = (c_T, c_I, x)$ is a triplet of instruction, input image and target image from the dataset. In practice, we initialize the weights of Emu Edit with the weights of Emu. To support the image conditioning, we follow [2] and increase the number of input channels. New weights are initialized to zero. During inference, we perform classifier-free guidance on both image and text conditions. In our experiments we use a scale of $\gamma_I = 1.5$ for the image condition and $\gamma_T = 5.0$ for the text condition. Furthermore, we apply a rescaling of the diffusion scheduler to achieve a zero signal-to-noise ratio (SNR) at the terminal timestamp, as suggested in [12]. This is crucial in order to avoid any mismatch between the model's training and testing phases. For more implementation details see Sec. 11.

[2]The samples depicted in this caption were selected by the authors. They do not cover all scenarios, but are meant to represent some common scenarios the authors encountered.

4.2. Learned Task Embeddings

To guide the generation process toward the correct task, we learn an embedding vector for each task in the dataset. During training, given a sample from our dataset, we use the task index, i, to fetch the task's embedding vector, v_i, from an embedding table, and optimize it jointly with the model weights. We do so by introducing the task embedding v_i as an additional condition to the U-Net, ϵ_θ. Concretely, we integrate the task embedding into the U-Net via cross-attention interactions, and by adding it to the timestep embeddings. The optimization problem is updated to

$$\min_{\theta, v_1, \ldots, v_k} \mathbb{E}_{\hat{y}, \epsilon, t}\left[\|\epsilon - \epsilon_\theta(z_t, t, E(c_I), c_T, v_i)\|_2^2\right] \quad (2)$$

where k is the total number of tasks in our dataset and $\hat{y} = (c_I, c_T, x, i)$ is a quadruplet of input image, input instruction text, target image, and task index from the dataset.

Task-specific conditioning arises from the observation that models lacking such conditioning can become perplexed about the type of edit required, particularly when the instructions are complex or the edit type is ambiguous. For instance, as visualized in Fig. 4, (1) a model without task conditioning might perform a global edit when a texture edit is required, (2) it might opt for segmentation when a global edit is necessary, and (3) it could implement a style edit in situations where a local edit would fit better.

In the inference stage, we predict the task index. Specifically, we fine-tune a Flan-T5-XL model to identify the task at hand given the input instruction.

4.3. Task Inversion

To enable few-shot learning of new tasks without losing the general abilities of Emu Edit, we propose a method for adapting the model without changing the U-Net weights. Given a few examples of a new task, we learn a new task embedding, v_{new}. We freeze the model weights, and adapt it to the task only through the task embedding. Thus, to fit a new task embedding we solve the following optimization problem:

$$\min_{v_{\text{new}}} \mathbb{E}_{y, \epsilon, t}\left[\|\epsilon - \epsilon_\theta(z_t, t, E(c_I), c_T, v_{\text{new}})\|_2^2\right] \quad (3)$$

where v_{new} is the learned task embedding. Note that during task inversion y is a triplet belonging to the new task.

The model can then be employed for the new task by conditioning it on the learned task embedding, and it can still handle its original tasks by relying on the initial task embeddings. In Sec. 5.5, we demonstrate that our model effectively generalizes to novel tasks using this method.

4.4. Sequential Edit Thresholding

We notice that applying the model repeatedly, in multi-turn editing scenarios, aggregates reconstruction and numerical errors, which translate to noticeable artifacts. To mitigate this, we add a per-pixel thresholding step after each edit-turn. At each step s, we use the pixel value in the output image, c_I^{s+1}, only if its alteration surpasses a specific threshold. Otherwise, we keep the pixel value from the input image, c_I^s. Specifically, given an edit turn s, we compute the absolute difference image $d = \|c_I^{s+1} - c_I^s\|_1$ over the RGB channel, and apply the following thresholding:

$$c_I^{s+1} = \begin{cases} c_I^s & \text{if } \bar{d} < \alpha, \\ c_I^{s+1} & \text{otherwise.} \end{cases} \quad (4)$$

where, $\bar{d}$ is obtained after passing d through a low pass filter, in order to smooth the transition between previous and current pixels. In practice, we choose $\alpha = 0.03$. Please refer to Sec. 10.6 for a qualitative comparison. In Fig. 2 we show examples of multi-turn editing.

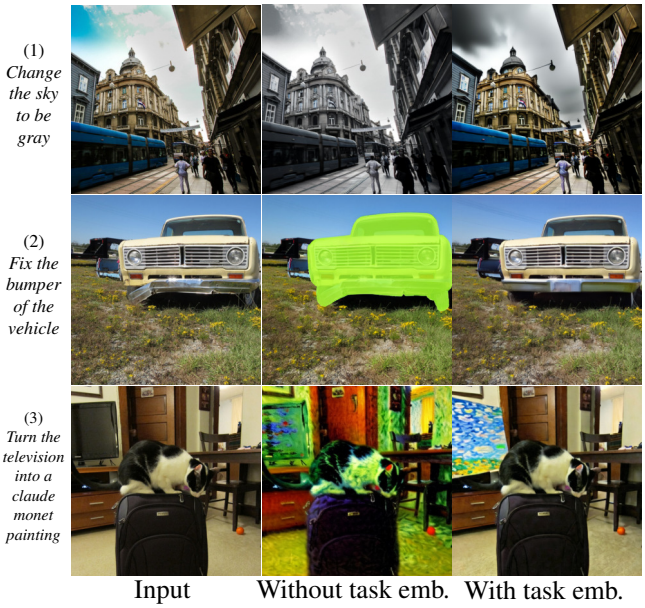


Figure 4. **Task embeddings.** Model trained without task embeddings may get confused about the edit type when the instructions are complicated or there is ambiguity regarding the edit type: (1) Global edit (instead of Texture), (2) Segmentation (instead of Global), (3) Style edit (instead of Local).

5. Experiments

Our experiments evaluate the ability of Emu Edit to follow user instructions faithfully and preserve the visual fidelity of the original image. First, we evaluate the performance of our approach on instruction-based image editing tasks. Second, we conduct a comprehensive ablation study to assess the effectiveness of our different contributions. Specifically, we ablate the contribution of the computer vision tasks to the model performance on image editing tasks, the importance of learned task embeddings, and the effect of multi-task learning on instruction-based image editing. Further

Table 2. Comparison with image-editing baselines evaluated on Emu Edit test set and MagicBrush test set. For each benchmark we report CLIP, L1, DINO metrics and human ratings. Human evaluation shows the percentage of raters that prefer the results of Emu Edit.

	Emu Edit Test set							MagicBrush Test Set						
Method	$\text{CLIP}_{dir}\uparrow$	$\text{CLIP}_{im}\uparrow$	$\text{CLIP}_{out}\uparrow$	L1$\downarrow$	DINO$\uparrow$	Text align.	Image faith.	$\text{CLIP}_{dir}\uparrow$	$\text{CLIP}_{im}\uparrow$	$\text{CLIP}_{out}\uparrow$	L1$\downarrow$	DINO$\uparrow$	Text align.	Image faith.
InstructPix2Pix [2]	0.078	0.834	0.219	0.121	0.762	77.33	76.71	0.115	0.837	0.245	0.093	0.767	71.79	71.60
MagicBrush [29]	0.090	0.838	0.222	0.100	0.776	74.50	74.10	0.123	0.883	0.261	0.058	0.871	59.54	60.39
PnP [25]	0.028	0.521	0.089	0.304	0.153	98.95	99.00	0.025	0.568	0.101	0.289	0.220	97.24	96.96
Null-Text Inv. [15]	0.101	0.761	**0.236**	**0.075**	0.678	81.63	85.47	0.121	0.752	**0.263**	0.077	0.664	76.54	85.66
Our	**0.109**	**0.859**	0.231	0.094	**0.819**	–	–	**0.135**	**0.897**	0.261	**0.052**	**0.879**	–	–

ablation on our data generation pipeline can be found in Appendix 8. Finally, we demonstrate our model's ability to learn new tasks via few-shot learning.

5.1. Measures

We employ two main measures in our evaluation: edit text alignment and image faithfulness. Specifically, for each pair of input image and editing instruction, we use the following automatic metrics: (i) CLIP [18] text-image direction similarity (CLIP_{dir}) – measuring agreement between change in captions and the change in images, (ii) CLIP image similarity (CLIP_{img}) – measuring change between edited and input image, (iii) CLIP output similarity (CLIP_{out}) – measuring edited image similarity with output caption, (iv) L1 pixel-distance between input and edit image, and (v) DINO [4] similarity between the DINO embeddings of input and edited images. With the exception of the L1 distance, where lower values indicate better performance, higher values in all other measures signify better results. A low L1 distance translates to small changes in image's pixel values. A high DINO and CLIP_{img} similarity score, suggests semantic similarity between the images. For region-based edits, high image similarity scores indicate the edits were precise. For free-form edits, high similarity scores indicate image structure preservation. CLIP_{dir} and CLIP_{out} measure how well the model followed the instruction.

In addition, we asked human raters to evaluate the text alignment and image faithfulness. In each evaluation scenario, raters are presented with two modified images alongside the original input image and instruction, and are presented with two questions: (i) Image Faithfulness: which image better preserves elements in the input image, and (ii) Text Alignment: which image best follows the instruction.

5.2. Evaluation

Throughout the paper, we report results on the MagicBrush test set [29] and the Emu Edit benchmark. In the following section, we describe our motivation for creating this benchmark, and detail its curation process. To date, there are two main benchmarks for evaluating instruction-based image editing capabilities. First, the InstructPix2Pix benchmark [2], which is intrinsically biased due to its reliance on *generated* Stable Diffusion [21] input images, and GPT-3 [3] *generated* instructions. Consequently, it is unclear whether its results will truly mirror the performance on *real* input images, with *genuine* user instructions.

Unlike InstructPix2Pix, the second benchmark, MagicBrush [29], uses a diverse set of authentic input images from the MS-COCO benchmark [5, 13], and annotator-defined instructions. Nonetheless, this dataset also suffers from inherent bias. During data collection, annotators were directed to use the DALLE-2 image editing platform [17] to generate the edited images. Thus, this benchmark is biased towards editing instructions that the DALLE-2 editor can successfully follow, which may compromise both its diversity and complexity.

Emu Edit Benchmark. To collect a dataset with reduced bias and of higher diversity, we take a different approach. We first define seven different categories of potential image editing operations: background alteration (Background), comprehensive image changes (Global), style alteration (Style), object removal (Remove), object addition (Add), localized modifications (Local), and color/texture alterations (Texture). Then, we utilize the diverse set of input images from the MagicBrush benchmark [29], and for each editing operation, we task crowd workers to devise relevant, creative, and challenging instructions. Moreover, to increase the quality of the collected examples, we apply a post-verification stage, in which crowd workers filter examples with irrelevant instructions. Finally, to support evaluation for methods that require input and output captions [9, 25], we additionally collect an input caption and output caption. When doing so, we ask annotators to ensure that the captions capture both important elements in the image, and elements that should change based on the instruction. See Sec. 9 for examples of our benchmark, which we publicly release to support better evaluation of instruction-based image editing models, and more details on the benchmark curation process.

5.3. Baseline Comparisons

We compare our model against two instruction-based image editing baseline models: InstructPix2Pix [2], and MagicBrush [29], which is a variant of InstructPix2Pix that

was fine-tuned on the MagicBrush dataset. Additionally, we compare our model against two text-based image editing methods: PNP [25] and Null-Text Inversion modification of P2P [9, 15]. Unlike instruction-based models, these works expect image descriptions. Therefore, we provide them with access to the input caption and output caption. Do note, providing these methods with access to the ground-truth captions could potentially offer an edge over instruction-based models, since the automatic metrics also rely on these captions. Tab. 2 shows our results versus the baselines. The findings indicate that human raters consistently prefer Emu Edit over all baselines by a large margin. Furthermore, apart from Null-Text Inversion, which as explained above, utilizes the ground-truth captions during inference, our approach outperforms the existing baselines on the automatic metrics. We provide qualitative comparisons in Fig. 3, Fig.13-14, and Fig.17-18. For performance on vision tasks, see Sec. 10.1.

5.4. Ablations

Computer Vision Tasks Enhance Image Editing Tasks. Here we demonstrate the importance of the vision tasks to Emu Edit performance on image editing tasks. For this, we trained two additional models on all tasks except: (i) detect and segment tasks, and (ii) image-to-image translation tasks. As we show in Tab. 4, adding the detection and segmentation tasks improves the model performance in region-based editing tasks. Additionally, we observe that image-to-image translation tasks improve the performance in free-form editing tasks. We hypothesize that the recognition tasks improve the model's recognition capabilities, leading to more accurate and precise localized modifications. Similarly, image-to-image tasks assist the model in understanding the entire image structure, thereby enhancing its capabilities for global operations.

Contribution of Learned Task Embeddings. We compare three variants of Emu Edit: (i) conditioned on the ground-truth task embedding, (ii) conditioned on the task embedding, as predicted by the task predictor described in Sec. 4.2, and (iii) without conditioning on the task type.[3] Tab. 3 shows the results on the validation set of our benchmark. As can be seen, conditioning on the task type boosts the model's performance. Furthermore, our task predictor closes the gap with the ground-truth conditioned model. Qualitatively, we observe that without conditioning on the task type the model may perform the wrong editing operation (Fig. 4). In Fig. 9, we demonstrate the effect of manipulating the task while keeping the instruction and input image fixed. As can be seen, changing the task embedding directly influences the task executed by the model.

Multi-Task versus Expert Models. We hypothesize that training a single model on a diverse range of tasks, encompassing both image editing and computer vision, leads to enhanced performance in each individual task, outperforming models that are specifically trained for a single task. To validate this hypothesis, we train an expert model for each task, and compare its performance to ours on that particular task. In this ablation, we train the models for half the steps of our complete model. Results provided in Tab. 7 show that the multi-task model is superior to all experts models.

Influence of Number of Tasks. Here, we ablate the number of tasks participating in the multi-task training scheme. In Fig. 8 we report the average CLIP_{dir} on the Style and Texture tasks when iteratively excluding other tasks, and training a model on the new tasks list. As can be seen, augmenting the model with additional tasks leads to improved performance, even in tasks which are not directly associated with the added ones. For instance, the Background task enhances the model's performance on the Texture and Style tasks.

[3] (iii) was trained without learned task embeddings.

Table 3. Learned task embeddings ablation on our validation set. We compare variations of Emu Edit: without task type condition, with predicted task type, and with ground-truth task type.

Method	$\text{CLIP}_{dir}\uparrow$	$\text{CLIP}_{im}\uparrow$	$\text{CLIP}_{out}\uparrow$	L1$\downarrow$	DINO$\uparrow$
w/o task emb.	0.104	0.843	0.227	0.109	0.792
with pred. task	0.117	0.850	0.231	0.103	0.809
with gt task	0.119	0.852	0.231	0.100	0.811

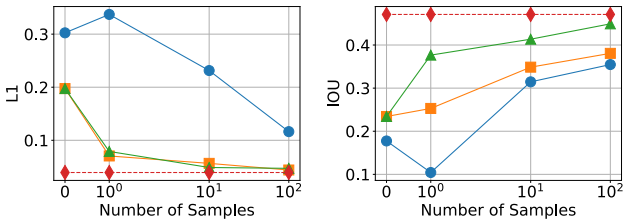
Table 4. Contribution of computer vision tasks. Human evaluation is shown as a percentage of majority votes in favor of our model.

	Region-Based		Free-form	
Method	Text align.	Image faith.	Text align.	Image faith.
without detect/segment	**60.0**	**60.2**	52.3	51.5
without im2im translation	50.2	49.0	**58.0**	**60.1**

5.5. Few-Shot Learning of New Tasks

Finally, we evaluate our model's ability to generalize by testing it in a few-shot scenario with previously unseen tasks. We test its generalization performance across the following tasks: (i) super-resolution (x4), (ii) object contour detection, (iii) mask-based inpainting, and (iv) a composite task formed by combining two tasks from our dataset: add and detect. For each task, we assess the model's performance when trained with 0, 1, and 100 examples, with the 0-example case being equivalent to a zero-shot setting. We compare three baselines: (i) Scratch – Emu Edit initialized with Emu's weights, trained on the examples, (ii) Task Inversion – Emu Edit with task inversion (Sec. 4.3), and (iii) Finetune – Emu Edit where we finetune all of the model's weights on the examples. As an upper-bound expert, we train the first baseline on 100,000 examples. Note that, the "Task Inversion" and "Finetune" baselines were trained on

Figure 5. Generations on unseen tasks with task inversion. (i) composition of add and detect tasks, (ii) object contour detection.



(a) Inpainting (b) Add + Detect

Figure 6. Few-shot performance for different tasks over 1, 10, and 100 samples. Each line represents a different training setting: Emu finetune (Blue, ◯), Emu Edit finetune (Orange, □), task inversion (Green, △), all compared to an upper-bound expert trained on 100k samples (Red dashed line, ♢).

the multi-task dataset whereas the "Scratch" baseline was not. As can be seen in Fig. 6, fine-tuning with a single example is enough to significantly enhance the performance, while training from scratch results in overfitting. Moreover, utilizing 100 samples nearly achieves expert-level performance, implying that the model can effectively generalize to novel tasks. Additionally, we observe that task inversion is comparable to a full finetune. This suggests that the model already possesses all the necessary information and can simply be "queried" with a new task embedding to produce the desired output. For performance and generation examples on additional tasks see Figs. 5, 10 and 11.

6. Conclusion

Emu Edit presents a step change in instructable image editing capabilities, primarily due to its unique training on both recognition and generation tasks. This dual-focus approach significantly enhances the model's comprehension of natural language instructions, enabling it to accurately execute a wide array of editing operations. Its ability to generalize to new tasks like image inpainting and super-resolution with minimal examples further demonstrates its versatility and advanced understanding. Additionally, our framework has the potential for further integration with a multimodal LLM in future work. This enhancement could be especially useful for editing tasks that necessitate more intricate reasoning from the input image, like counting objects or undertaking complex, highly detailed tasks.

Acknowledgements

We extend our gratitude to the following people for their contributions (alphabetical order): Andrew Brown, Ankit Ramchandani, Guan Pang, Ishan Misra, Mannat Singh, Ning Zhang, Parveen Krishnan, Peizhao Zhang, Peter Vajda, Rohit Girdhar, Roshan Sumbaly, Tong Xiao, Vladan Petrovic, Xide Xia.

References

[1] Omri Avrahami, Dani Lischinski, and Ohad Fried. Blended diffusion for text-driven editing of natural images. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 18208–18218, 2022. 3

[2] Tim Brooks, Aleksander Holynski, and Alexei A Efros. Instructpix2pix: Learning to follow image editing instructions. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 18392–18402, 2023. 1, 3, 4, 5, 7

[3] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, T. J. Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeff Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. *ArXiv*, abs/2005.14165, 2020. 3, 7

[4] Mathilde Caron, Hugo Touvron, Ishan Misra, Hervé Jégou, Julien Mairal, Piotr Bojanowski, and Armand Joulin. Emerging properties in self-supervised vision transformers. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 9650–9660, 2021. 7

[5] Xinlei Chen, Hao Fang, Tsung-Yi Lin, Ramakrishna Vedantam, Saurabh Gupta, Piotr Dollár, and C. Lawrence Zitnick. Microsoft coco captions: Data collection and evaluation server. *ArXiv*, abs/1504.00325, 2015. 7

[6] Xiaoliang Dai, Ji Hou, Chih-Yao Ma, Sam Tsai, Jialiang Wang, Rui Wang, Peizhao Zhang, Simon Vandenhende, Xiaofang Wang, Abhimanyu Dubey, et al. Emu: Enhancing image generation models using photogenic needles in a haystack. *arXiv preprint arXiv:2309.15807*, 2023. 4, 5

[7] Yuning Du, Chenxia Li, Ruoyu Guo, Xiaoting Yin, Weiwei Liu, Jun Zhou, Yifan Bai, Zilin Yu, Yehua Yang, Qingqing Dang, et al. Pp-ocr: A practical ultra lightweight ocr system. *arXiv preprint arXiv:2009.09941*, 2020. 2

[8] Oran Gafni, Adam Polyak, Oron Ashual, Shelly Sheynin, Devi Parikh, and Yaniv Taigman. Make-a-scene: Scene-based text-to-image generation with human priors. In *European Conference on Computer Vision*, pages 89–106. Springer, 2022. 2

[9] Amir Hertz, Ron Mokady, Jay M. Tenenbaum, Kfir Aberman, Yael Pritch, and Daniel Cohen-Or. Prompt-to-prompt image editing with cross attention control. *ArXiv*, abs/2208.01626, 2022. 3, 7, 8

[10] Bahjat Kawar et al. Imagic: Text-based real image editing with diffusion models. *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6007–6017, 2022. 3

[11] Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C. Berg, Wan-Yen Lo, Piotr Dollár, and Ross B. Girshick. Segment anything. *ArXiv*, abs/2304.02643, 2023. 1

[12] Shanchuan Lin, Bingchen Liu, Jiashi Li, and Xiao Yang. Common diffusion noise schedules and sample steps are flawed. *arXiv preprint arXiv:2305.08891*, 2023. 5

[13] Tsung-Yi Lin, Michael Maire, Serge J. Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft coco: Common objects in context. In *ECCV*, 2014. 7, 3

[14] Siyi Liu, Zhaoyang Zeng, Tianhe Ren, Feng Li, Hao Zhang, Jie Yang, Chun yue Li, Jianwei Yang, Hang Su, Jun-Juan Zhu, and Lei Zhang. Grounding dino: Marrying dino with grounded pre-training for open-set object detection. *ArXiv*, abs/2303.05499, 2023. 1

[15] Ron Mokady, Amir Hertz, Kfir Aberman, Yael Pritch, and Daniel Cohen-Or. Null-text inversion for editing real images using guided diffusion models. *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6038–6047, 2022. 3, 7, 8

[16] Pushmeet Kohli Nathan Silberman, Derek Hoiem and Rob Fergus. Indoor segmentation and support inference from rgbd images. In *ECCV*, 2012. 4

[17] OpenAI. Dall-e 2, 2022. https://openai.com/product/dall-e-2. 3, 7

[18] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PMLR, 2021. 1, 5, 7

[19] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67, 2020. 5

[20] Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. Hierarchical text-conditional image generation with clip latents. *arXiv preprint arXiv:2204.06125*, 2022. 2

[21] Robin Rombach, A. Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10674–10685, 2021. 2, 7

[22] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models, 2021. 4

[23] Chitwan Saharia, William Chan, Saurabh Saxena, Lala Li, Jay Whang, Emily L. Denton, Seyed Kamyar Seyed Ghasemipour, Burcu Karagol Ayan, Seyedeh Sara Mahdavi, Raphael Gontijo Lopes, Tim Salimans, Jonathan Ho, David J. Fleet, and Mohammad Norouzi. Photorealistic text-to-image diffusion models with deep language understanding. *ArXiv*, abs/2205.11487, 2022. 2

[24] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023. 3

[25] Narek Tumanyan, Michal Geyer, Shai Bagon, and Tali Dekel. Plug-and-play diffusion features for text-driven image-to-image translation. *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1921–1930, 2022. 3, 7, 8, 2

[26] Bram Wallace, Akash Gokul, and Nikhil Vijay Naik. Edict: Exact diffusion inversion via coupled transformations. *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 22532–22541, 2022. 3

[27] Su Wang et al. Imagen editor and editbench: Advancing and evaluating text-guided image inpainting. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 18359–18369, 2023. 3

[28] Shaoan Xie, Zhifei Zhang, Zhe Lin, Tobias Hinz, and Kun Zhang. Smartbrush: Text and shape guided object inpainting with diffusion model. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 22428–22437, 2023. 3

[29] Kai Zhang, Lingbo Mo, Wenhu Chen, Huan Sun, and Yu Su. Magicbrush: A manually annotated dataset for instruction-guided image editing. *arXiv preprint arXiv:2306.10012*, 2023. 1, 3, 7

[30] Lvmin Zhang, Anyi Rao, and Maneesh Agrawala. Adding conditional control to text-to-image diffusion models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 3836–3847, 2023. 2

[31] Bolei Zhou, Hang Zhao, Xavier Puig, Sanja Fidler, Adela Barriuso, and Antonio Torralba. Scene parsing through ade20k dataset. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 633–641, 2017. 3

[32] Bolei Zhou, Hang Zhao, Xavier Puig, Tete Xiao, Sanja Fidler, Adela Barriuso, and Antonio Torralba. Semantic understanding of scenes through the ade20k dataset. *International Journal of Computer Vision*, 127:302–321, 2019. 3

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.

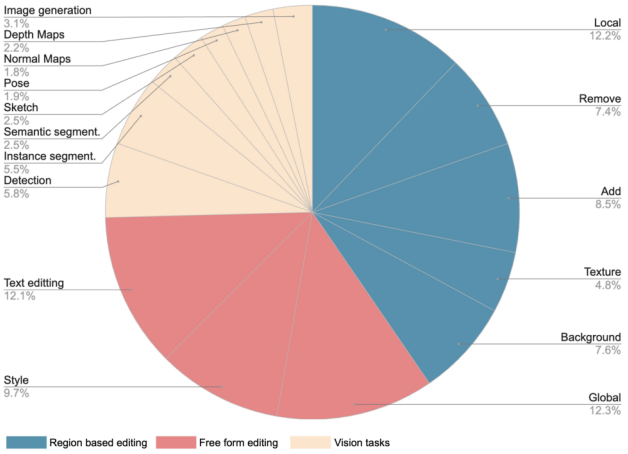


Figure 7. Distribution of the tasks in our training dataset.

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. 15-16 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. 3.3, 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 on all of the tokens. 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. 3.3.

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

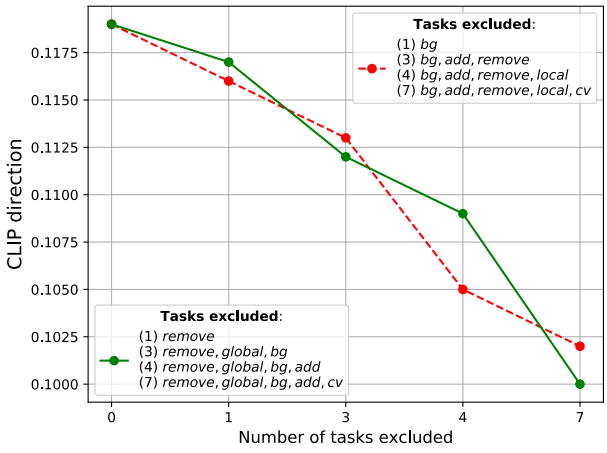


Figure 8. Ablation on the model performance (CLIP$_{dir}$) on Style and Texture tasks as we progressively exclude tasks that don't fall within these categories.

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$_{dir}$↑	CLIP$_{img}$↑	CLIP$_{out}$↑	L1↓	DINO↑	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↑	Semantic Segmentation↑	Depth Estimation↓
Emu Edit	61.467	50.028	0.246

10.2. 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.

10.3. Influence of Number of Tasks

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

10.4. Few-Shot Learning of New Tasks

Fig. 11 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.

10.5. Qualitative Comparisons with Existing Approaches

Figs. 13 and 14 shows qualitative comparisons with baselines on generated samples. In addition, in Figs. 17 and 18 we present qualitative comparisons of our model with baselines on Emu Edit test set.

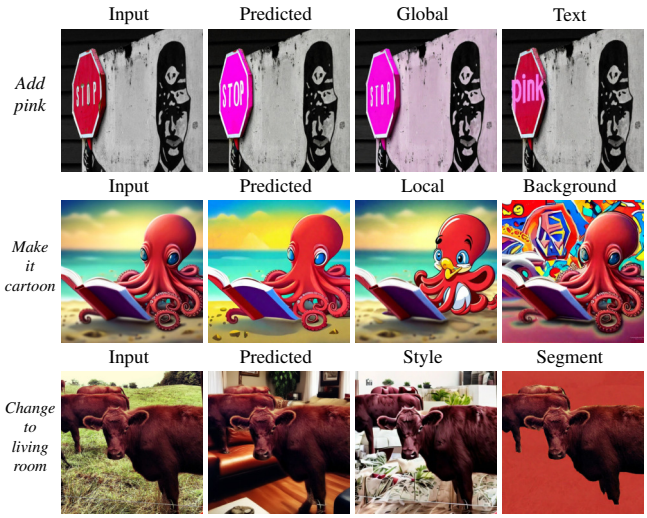
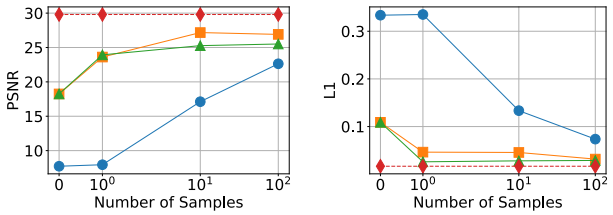


Figure 9. Controlling the Task Embedding. For each sample, we present the edited image using the task predicted by the task predictor. In addition, we present the edited image generated using the same input image and instruction, but with different task embeddings. For instance, in the first row we generate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

10.6. Qualitative Comparison of Sequential Edit Thresholding

In Fig. 12, 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



(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: Emu finetune (Blue, ◯), Emu Edit finetune (Orange, □), task inversion (Green, △), all compared to an upper-bound expert trained on 100k samples (Red dashed line, ♢).

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. The training spans 48,000 steps.

Figure 11. 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.

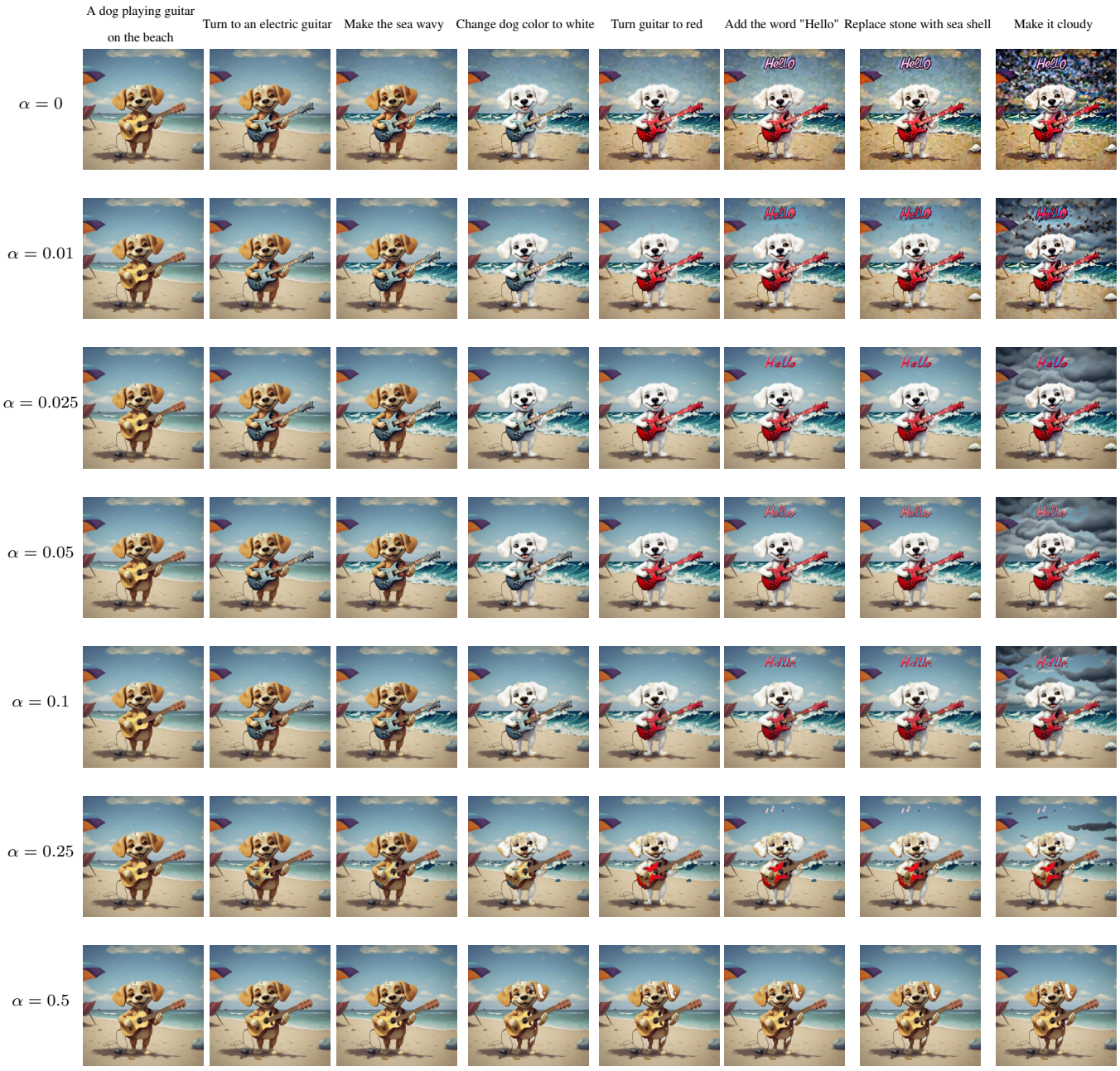


Figure 12. Effect of Sequential Edit Thresholding during sequential edits (from left to right) with different α values.

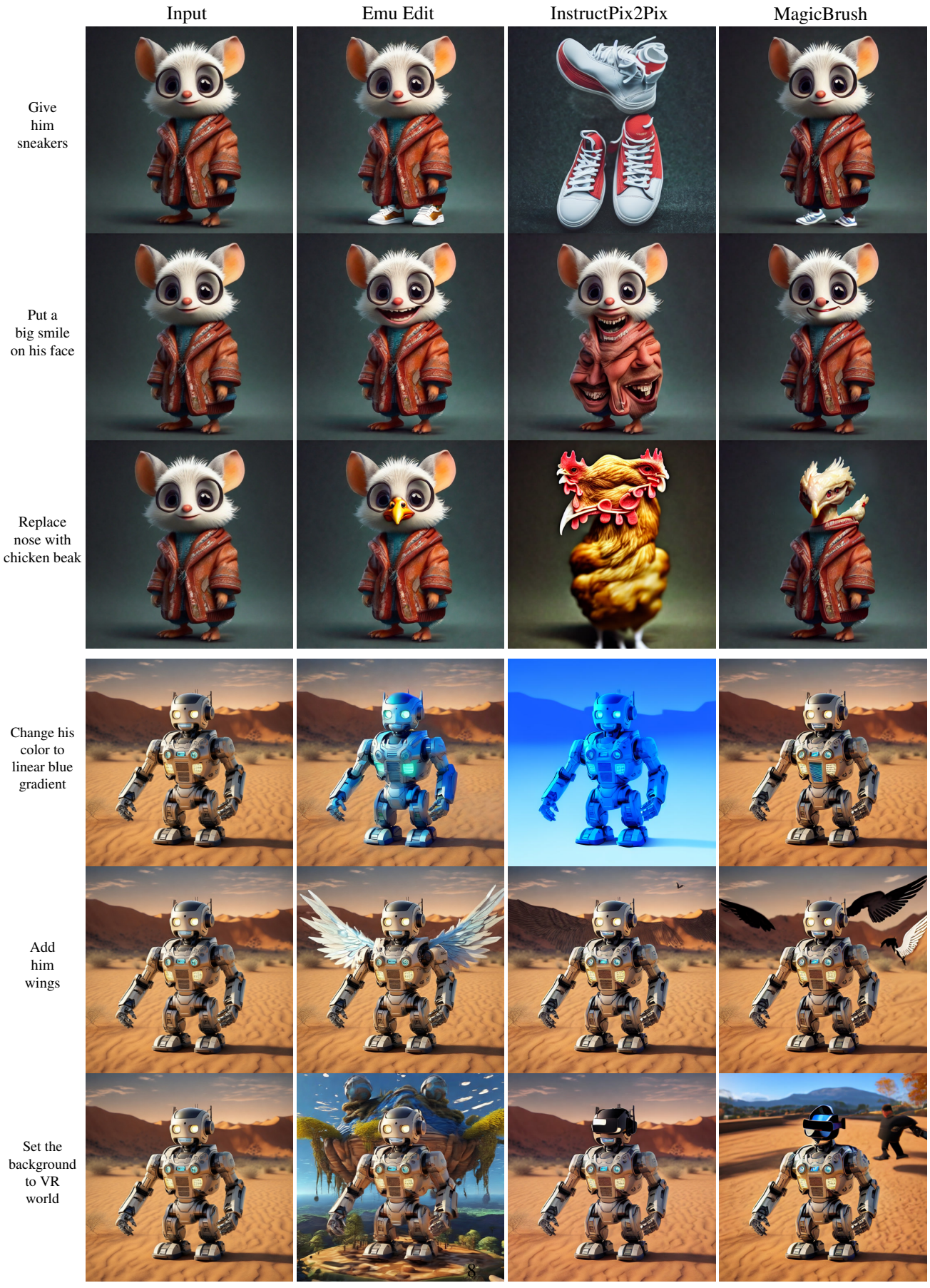


Figure 13. Qualitative comparison with baselines.

Figure 14. Qualitative comparison with baselines.

```
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 15. 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 16. Examples of prompts for Add task.

Figure 17. Qualitative comparison of our model to baselines on Emu Edit Test Set.

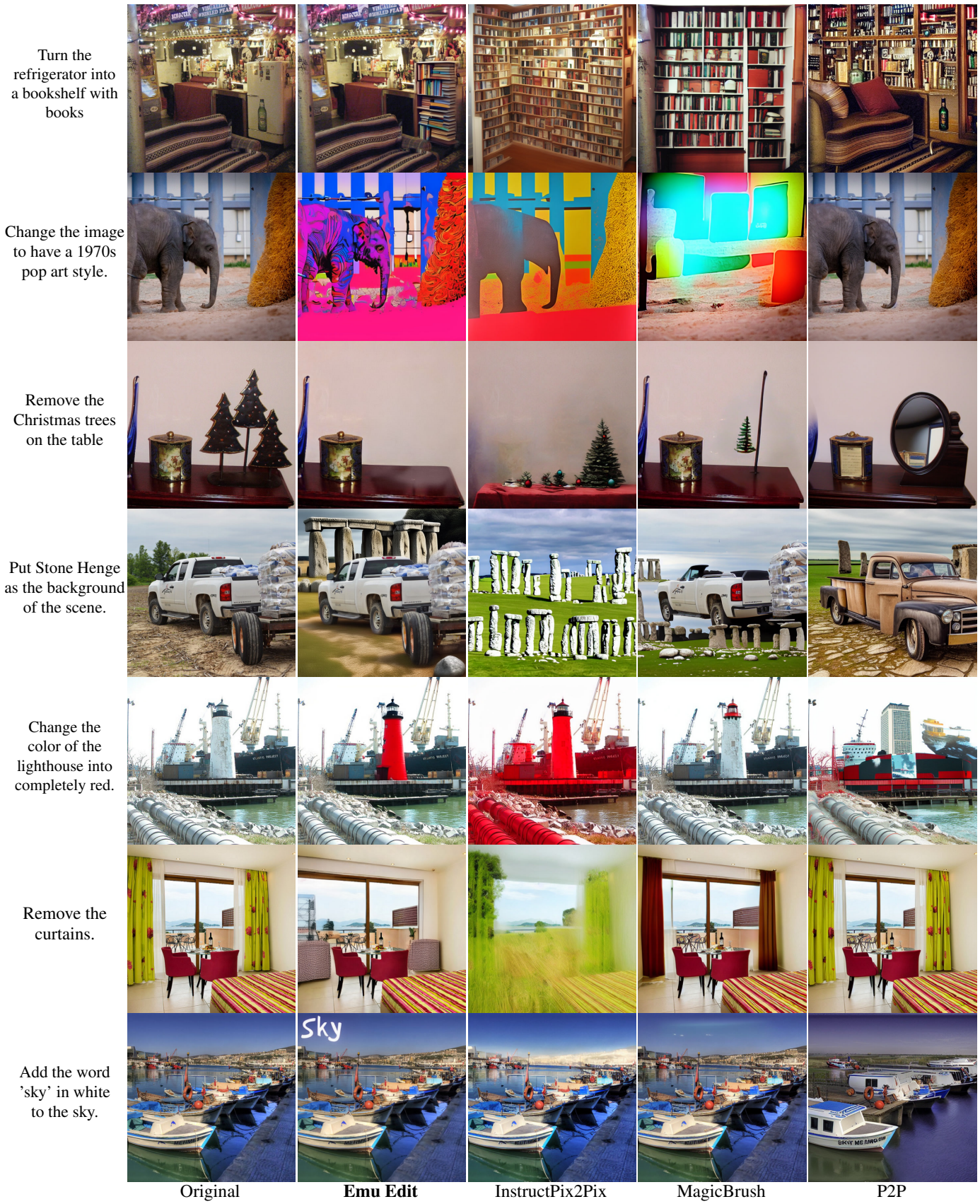


Figure 18. Qualitative comparison of our model to baselines on Emu Edit Test Set.