Appendix

A. Data Collection and User Study

In the evaluation steps, we collect real-world images with instructions using Amazon Mechanical Turk (Mturk) [4]. We randomly collect 200 real-world images. Then we ask Mturk annotators to write five instructions for each image, and encourage them to have wild imaginations and diversify the instruction types. We encourage annotators to not be limited to making the image realistic. For example, annotators can write "add a horse in the sky". A screenshot of the interface is illustrated in Fig. 13. We analyze the top five verbs and nouns in the evaluation dataset. It is shown in Fig. 15(a) that the verbs "add", "change", "make", "remove" and "put" make up around 85% of all verbs, which means that the editing instruction verbs have a long-tail distribution. In contrast, the distribution of nouns in Fig. 15(b) is close to uniform, where the top five nouns represent only around 20% of all nouns.

In user studies, we use Mturk to ask annotators to evaluate edited images. A screenshot of the interface is shown in Fig. 14. The annotators are provided with the original image, two edited images, and the editing instruction. They are asked to select the better edited image. The third option indicates that the edited images are equally good or equally bad. We ask three annotators to label one data sample, and use the majority votes to determine the results. We shuffle the edited images to avoid choosing the left image over the right and vice versa.

B. Implementation Details

B.1. Instructional Supervised Training

We use pre-trained stable diffusion models as the initial checkpoint to start instructional supervised training. We train HIVE on 40GB NVIDIA A100 GPUs for 500 epochs. We use the learning rate of 10^{-4} and the image size of 256. In the inference, we use 512 as the default image resolution.

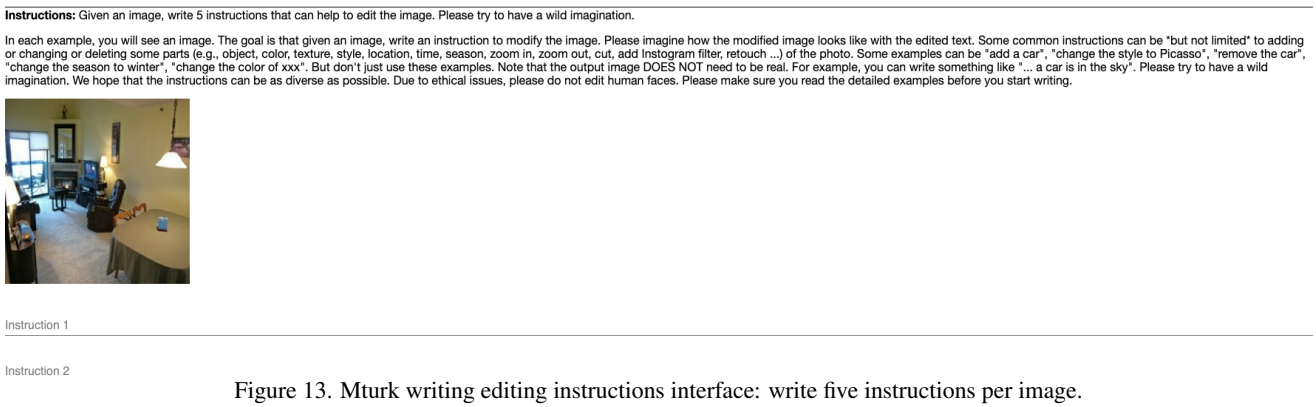


Figure 13. Mturk writing editing instructions interface: write five instructions per image.

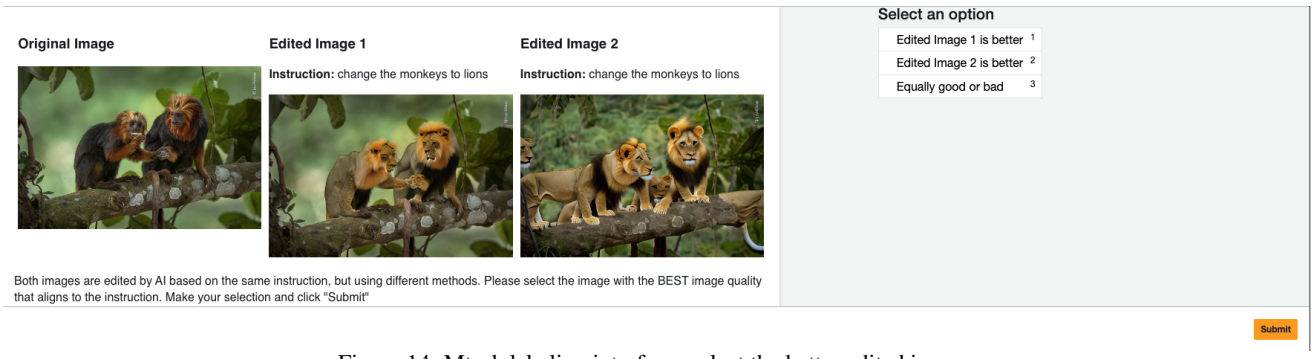


Figure 14. Mturk labeling interface: select the better edited image.

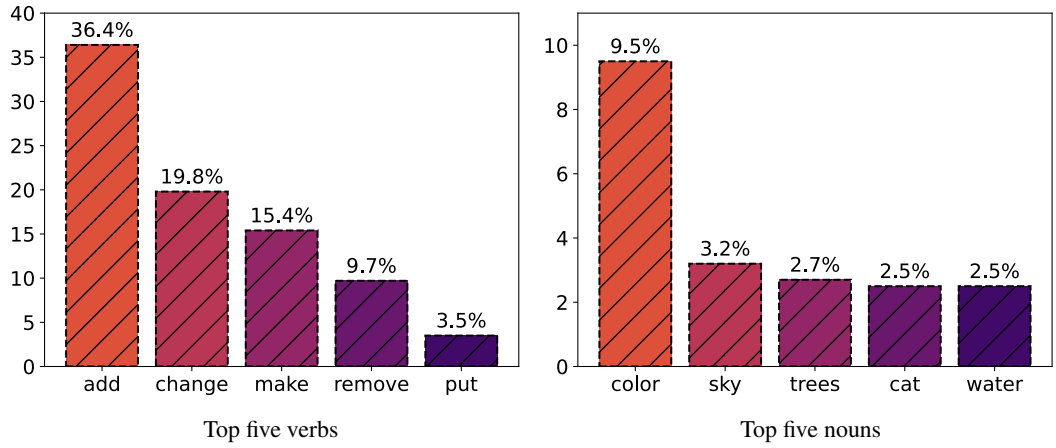


Figure 15. Top five verbs and nouns in the evaluation dataset.

B.2. Human Feedback Rewards Learning

As shown in Fig. 3, the reward model takes in an input image c_I, a text instruction c_E, and an edited image $\tilde{x}$ and outputs a scalar value. Inspired by the recent work on the vision-language model, especially BLIP [28], we employ a visual transformer [13] as our image encoder and an image-grounded text encoder as the multimodal encoder for images and text. Finally, we set a linear layer on top of the image-grounded text encoder to map the multimodal embedding to a scalar value.

(1) **Visual transformer**. We encode both the input image c_I and edited image $\tilde{x}$ with the same visual transformer. Then we obtain the joint image embedding by concatenating the two image embeddings $vit(c_I)$, $vit(\tilde{x})$.

[4]https://www.mturk.com

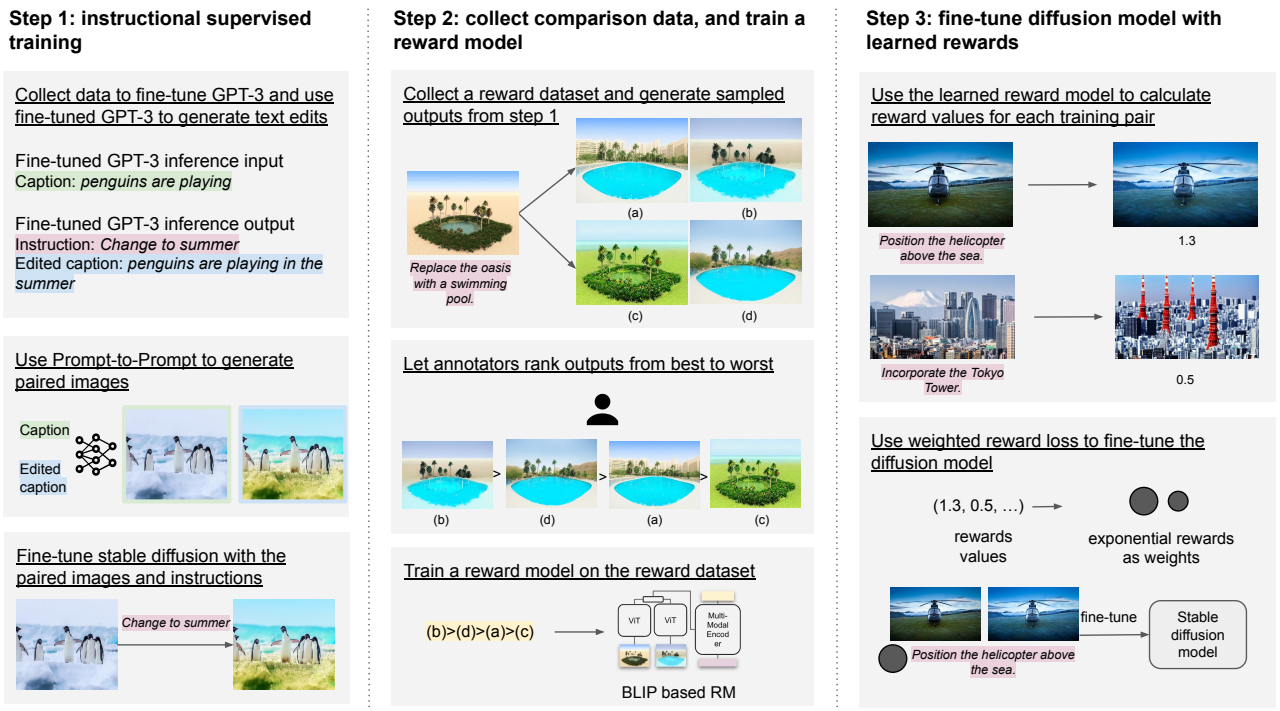


Figure 16. Overall architecture of HIVE. Different from Fig. 2, in the third step, we use weighted reward loss instead of condition reward loss to fine-tune the diffusion model.

(2) **Image-grounded text encoder**. The image-grounded text encoder is a multimodal encoder that inserts one additional cross-attention layer between the self-attention layer and the feed-forward network for each transformer block of BERT [11]. The additional cross-attention layer incorporates visual information into the text model. The output embedding of the image-grounded text encoder is used as the multimodal representation of the $(c_I, c_E, \tilde{x})$ triplet.

We gather a dataset comprising 3,634 images for the purpose of ranking. For each image, we generate five variant edited images, and ask an annotator to rank images from best to worst. Additionally, we ask annotators to indicate if any of the following scenarios apply: (1) all edited images are edited but none of them follow the instruction; (2) all edited images are visually the same as the original image; (3) all images are edited beyond the scope of instruction; (4) edited images have harmful content containing sex, violence, porn, etc; and (5) all edited images look similar to each other. We compare training reward models by filtering some/all of these options.

We note that a considerable portion of the collected data falls under at least one of the aforementioned categories, indicating that even for humans, ranking these images is challenging. As a result, we only use the data that did not include any non-rankable options in the reward model training. From a pool of 1,412 images, we select 1,285 for the training set, while the remaining images were used for the validation set. The reward model is trained on a dataset of comparisons between multiple model outputs on the same input. Each comparison sample contains an input image, an instruction, five edited versions of the image, and the corresponding rankings. We divide the dataset into training and validation sets based on the distribution of the corresponding instructions.

We apply the method in Sec. 3.3 on the reward data to develop a reward model. We initialize the reward model from the pre-trained BLIP, which was trained on paired images and captions using three objectives: image-text contrastive learning, image-text matching, and masked language modeling. Although there is a domain gap between BLIP's pre-training data and our reward data, where the captions in BLIP's data describe a single image, and the instructions in our data refer to the difference between image pairs. We hypothesized that leveraging the learned alignment between text and image in BLIP could enhance the reward model's ability to comprehend the relationship between the instruction and the image pairs.

The reward model is trained using 4 A100 GPUs for 10 epochs, employing a learning rate of 10^{-4} and weight decay of 0.05. The image encoder's and multimodal encoder's last layer outputs are utilized as image and multimodal representations, respectively. The encoders' final layer is the only fine-tuned component.

We use the trained reward model to generate a reward score on our training data. We perform two experiments. The first experiment takes the exponential rewards as weights and fine-tunes the diffusion model with **weighted reward loss** as described in Sec. 3.4. See Fig. 16 for the visualization of the method. The second experiment transforms the rewards to text prompts and fine-tunes the diffusion model with the **condition reward loss** as described in Sec. 3.4. The method is introduced in Fig. 2. We compare those two experiment settings, and results can be found in Sec. D.3.

C. Reward Maximization for Diffusion-Based Generative Models

C.1. Discussion on On-Policy based Reward Maximization for Diffusion Models

Directly adapting on-policy RL methods to the current training pipeline might be computationally expensive, but we do not conclude that sampling-based approaches are not doable for diffusion models. We consider developing more scalable sampling-based methods as future work.

We start the sampling methods derivation with the following objective:

$$J(\theta) := \max_{\pi_\theta} \mathbb{E}_{c\sim p_c}\left[\mathbb{E}_{\tilde{\boldsymbol{x}}\sim\pi_\theta(\cdot|c)}\left[\mathcal{R}_\phi(\tilde{\boldsymbol{x}}, c)\right] - \eta \mathrm{KL}[p_{\mathcal{D}}(\tilde{\boldsymbol{x}}|c)||\pi_\theta(\tilde{\boldsymbol{x}}|c)]\right], \tag{3}$$

where $p_c(c)p_{\mathcal{D}}(\tilde{\boldsymbol{x}}|c)$ is the joint distribution of the condition and edited images pair, and π_θ denotes the policy or the diffusion model we want to optimize. Note that $p_{\mathcal{D}}(\tilde{\boldsymbol{x}}|c)$ and $\pi(\tilde{\boldsymbol{x}}|c)$ are swaped compared with the objective in Eq. (1). The second term in Eq. (3), is the *KL Minimization* formula for maximum likelihood estimation, equivalent to the loss of diffusion models. We represent the policy π_θ via the *reverse process* of a conditional diffusion model:

$$\pi_\theta(\tilde{\boldsymbol{x}}|c) := p_\theta(\tilde{\boldsymbol{x}}^{0:T}|\, c) = p_0(\tilde{\boldsymbol{x}}^T)\prod_{t=1}^{T} p_\theta(\tilde{\boldsymbol{x}}^{t-1}|\tilde{\boldsymbol{x}}^t; c)\,,$$

where $p_0(\tilde{\boldsymbol{x}}^T) := \mathcal{N}(\tilde{\boldsymbol{x}}^T, \mathbf{0}; \mathbf{I})$, and $p_\theta(\tilde{\boldsymbol{x}}^{t-1}|\tilde{\boldsymbol{x}}^t; c) := \mathcal{N}(\tilde{\boldsymbol{x}}^t|\mu_\theta(\tilde{\boldsymbol{x}}_t, t), \sigma_t^2)\mathrm{I}$ is a Gaussian distribution, whose parameters are defined by score function ϵ_θ and stepsize of noise scalings. So we can get a edited image sample $\tilde{\boldsymbol{x}}^0$ by running a reverse

diffusion chain:

$$\tilde{\boldsymbol{x}}^{t-1}|\tilde{\boldsymbol{x}}^t = \frac{1}{\sqrt{\alpha_t}}\left(\tilde{\boldsymbol{x}}^t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon_\theta(\tilde{\boldsymbol{x}}^t, c, t)\right) + \sigma_t \boldsymbol{z}_t, \;\; \boldsymbol{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}), \text{for } \; t = T, \dots, 1\,,$$

and $\tilde{\boldsymbol{x}}^T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$.

As a result, the reverse diffusion process can be viewed as a black box function defined by ϵ_θ and noises $\boldsymbol{\epsilon} := (\boldsymbol{z}_T, \dots, \boldsymbol{z}_1, \tilde{\boldsymbol{x}}^T)$, which we can view as a *shared parameter network with noises*. And for each layer, we can view the parameter is the score function ϵ_θ. Define the network as

$$\tilde{\boldsymbol{x}}^0 := f(c, \boldsymbol{\epsilon}; \theta)\,, \;\; \boldsymbol{\epsilon} \sim p_{\text{noise}}(\cdot), c \sim p_c(\cdot)\,,$$

where we can rewrite the first term as

$$\mathbb{E}_{c\sim\mathcal{D}, \boldsymbol{\epsilon}\sim p_{\text{noise}}(\cdot)}[\mathcal{R}_\phi(f(c, \boldsymbol{\epsilon}; \theta), c)]\,,$$

and we can optimize the parameter θ with path gradient if $\mathcal{R}_\cdot$ is differentiable with path gradient. Similarly, suppose we want to optimize the first term via PPO. In that case, the main technical difficulty is to estimate $\nabla_\theta \log \pi_\theta(\tilde{\boldsymbol{x}}|c)$, which can be estimated with the following derivation:

$$\nabla_\theta \log \pi_\theta(\tilde{\boldsymbol{x}}|c) = \nabla_\theta \log p_\theta(\tilde{\boldsymbol{x}}^{0:T}|\, c) = \sum_{t=1}^{T} \nabla_\theta \log p_\theta(\tilde{\boldsymbol{x}}^{t-1}|\tilde{\boldsymbol{x}}^t; c)\,.$$

Note that for both the end-to-end path gradient method and PPO we require to sample the reverse chain from $\tilde{\boldsymbol{x}}^T$ to $\tilde{\boldsymbol{x}}^0$, thus we can estimate $\nabla_\theta \log \pi(\tilde{\boldsymbol{x}}|c)$ using the empirical samples $\tilde{\boldsymbol{x}}^{0:T}$.

For the above two methods, to perform one step policy gradient update, we need to run the whole reverse chain to get an edited image sample $\tilde{\boldsymbol{x}}^0$ to estimate the parameter gradient for the first term. As a result, the computational cost is the number of diffusion steps more extensive than the supervised fine-tuning cost. Now we need more than two days to fine-tune the stable diffusion model, so for standard LDM, where the number of steps is 1000, we can not finish the training within an acceptable training time. Even if we can use some fast sampling methods such as DDIM or variance preserve (VP) based noise scaling, the diffusion steps are still more than 5 or 10. Further, we haven't seen any previous work using such noise scaling to fine-tune stable diffusion. As a result, we think naive sampling methods might have high risk to obtain similar performance, compared with our current offline RL based approaches.

C.2. Derivation for Eq. (2)

Take a functional view of Eq. (2), and differentiate $J(\rho)$ w.r.t ρ, we get

$$\frac{\partial J(\rho)}{\partial \rho} = \mathcal{R}_\phi(\tilde{\boldsymbol{x}}|c) - \eta\left(\log \rho(\tilde{\boldsymbol{x}}|c) + 1 - \log p(\tilde{\boldsymbol{x}}|c)\right)\,.$$

Setting $\frac{\partial J(\rho)}{\partial \rho} = 0$ gives us

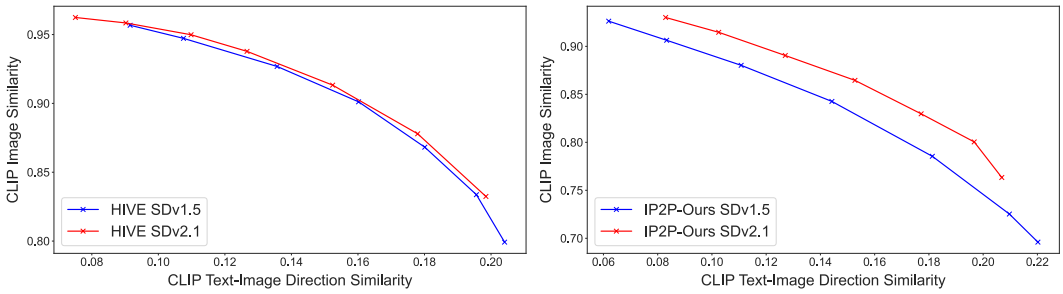
$$\log \rho(\tilde{\boldsymbol{x}}|c) = \frac{1}{\eta}\mathcal{R}_\phi(\tilde{\boldsymbol{x}}|c) + \log p(\tilde{\boldsymbol{x}}|c) - 1\,,$$
$$\rho(\tilde{\boldsymbol{x}}|c) \propto p(\tilde{\boldsymbol{x}}|c)\exp\left(\mathcal{R}_\phi(\tilde{\boldsymbol{x}}, c)/\eta\right)\,.$$

Thus we can get the optimal $\rho^*(\tilde{\boldsymbol{x}}|c)$.

D. Additional Ablation Study

D.1. SD v1.5 and v2.1.

In Sec. 4.2, we upgrade the backbone of stable diffusion from v1.5 to v2.1, where OpenCLIP text encoder [49] replaces the CLIP text encoder [42]. In this section, we demonstrate the quantitative consistency plot in Fig. 17(a) on the synthetic evaluation dataset, which shows similar conclusions to the user study in Fig. 11(a). We compare IP2P-Ours v1.5 with v2.1 as well. An interesting observation is that we train IP2P-Ours with SD v2.1 and show in Fig. 17(b) that its improvement over SD v1.5 is larger than HIVE in Fig. 17(a).



IP2P-Ours with SD v1.5 and v2.1 InstructPix2Pix with SD v1.5 and v2.1

Figure 17. HIVE and IP2P-Ours with SD v1.5 and v2.1.

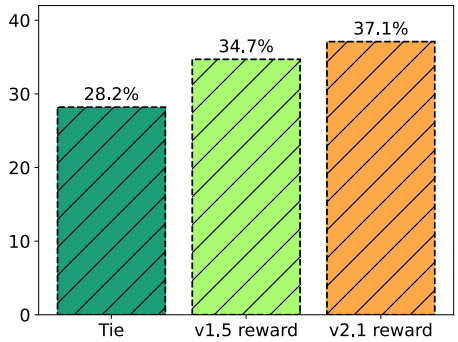


Figure 18. SD v1.5 trained vs. SD v2.1 trained reward model

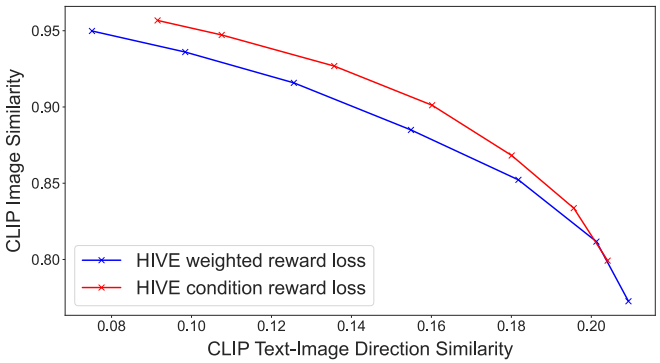


Figure 19. HIVE with weighted reward loss and conditional reward loss.

D.2. Model Adaptation

We demonstrate that HIVE is able to adapt the reward model that is trained on a different backbone from the backbone in Step. 3. We use the SD v1.5 generated data to train the reward model, and process the rest steps using SD v2.1. We report user study results in Fig. 18. It is observed that the users vote similarly between the reward models that are trained on two SD backbones. In other words, the reward model is able to adapt from one backbone to another.

D.3. Weighted Reward and Conditional Reward Losses

We compare the weighted reward loss and conditional reward loss on the synthetic evaluation dataset. As shown in Fig. 19, the performances of these two losses are close to each other, while the conditional reward loss is slightly better. Therefore we adopt the conditional reward loss in all our experiments.

D.4. Training with Less Data

We analyze the effect of the training data size. We compare HIVE with SD v1.5 at four training dataset size ratios: 100%, 50%, 30% and 10%. As shown in Fig. 20, significantly decreasing the size of the dataset, *e.g.* 10% data, leads to worse

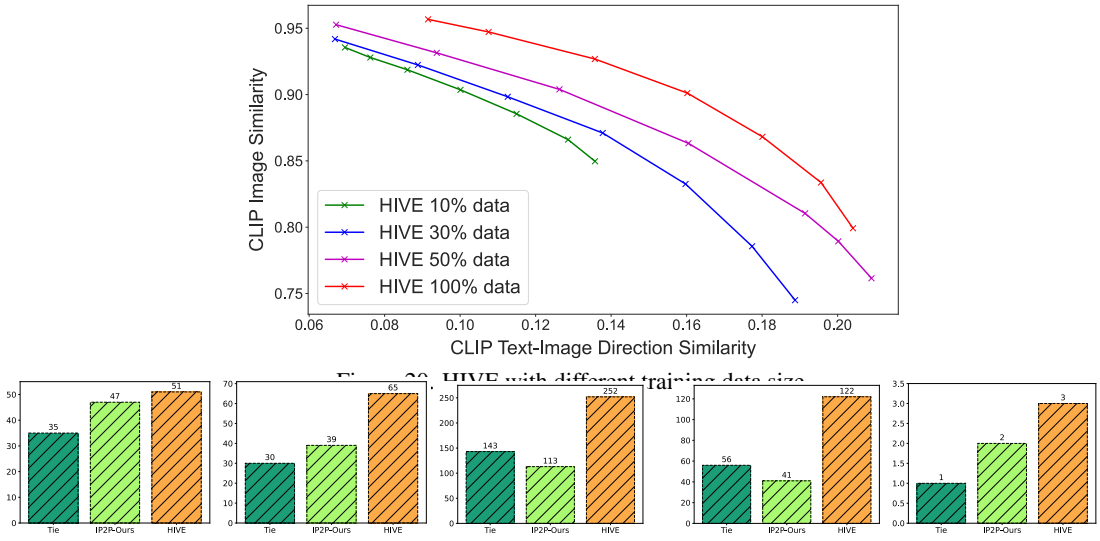


Figure 21. Subcategory analysis between IP2P and HIVE.

ability to perform large image edits. On the other hand, reasonable decreasing dataset size can result in a similar yet slightly worse performance *e.g*. 50% data.

D.5. Subcategory Analysis

We classify the editing into the following sub-categories: changing the global style, adjust attributes for the main object, add/remove objects, manipulate objects, and other challenging cases such as zooming and camera view changes. We use ChatGPT [5] to determine which sub-category the instruction belongs to. Specifically, the numbers of instructions in each sub-category are as follows: changing global style (133), adjust attributes for the main object (134), add/remove objects (508), manipulate objects (219), and others (6). We analyze user study results for each sub-category. It is shown in Fig. 21 that the most improvement comes from the sub-categories "Add/remove objects" and "Manipulate objects".

D.6. Additional Visualized Results

We illustrate additional visualized results in Fig. 22, 23, 24, 25, 26, where each row illustrates three instructional editing examples.

[5] https://chat.openai.com/

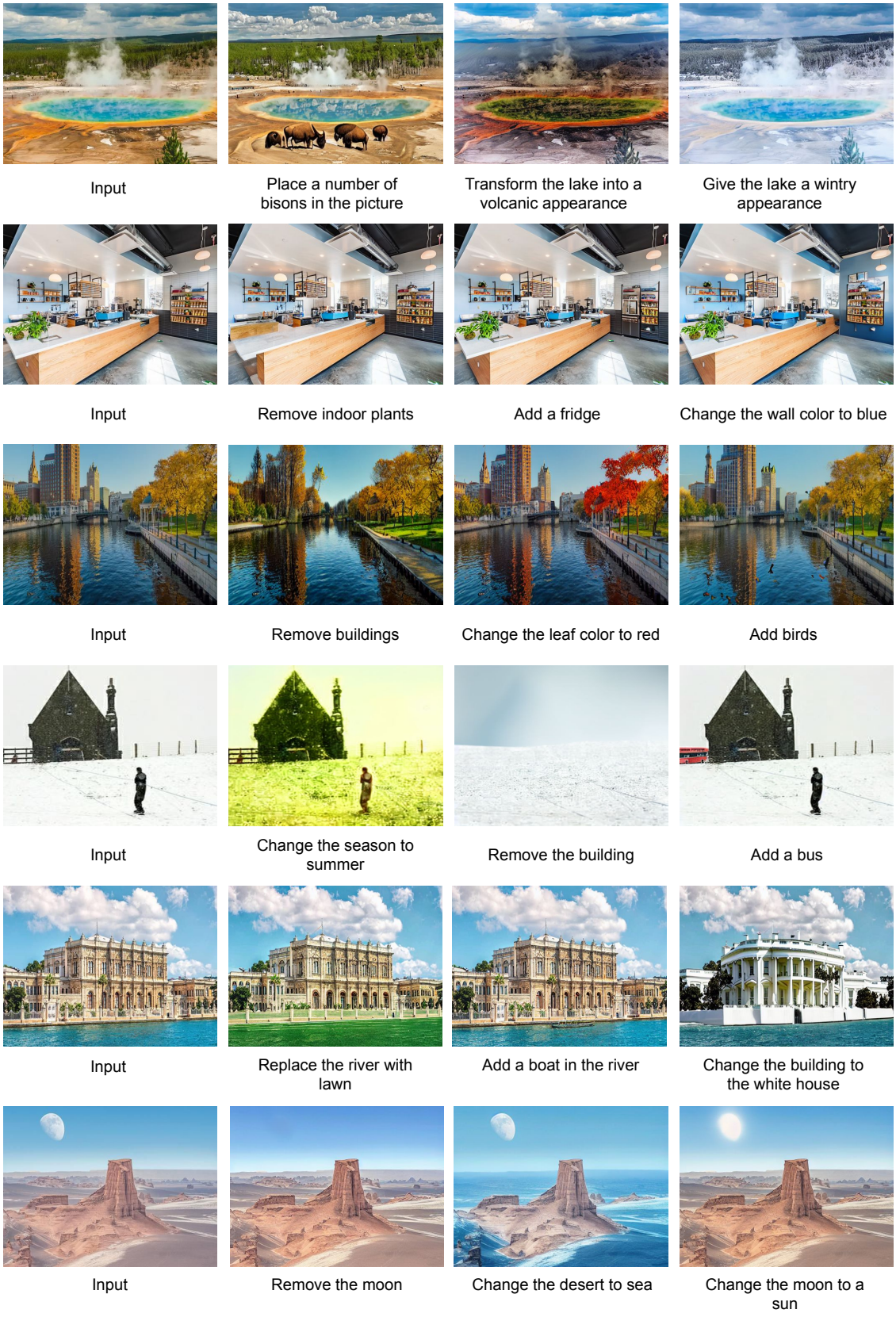


Figure 22. Additional editing results.

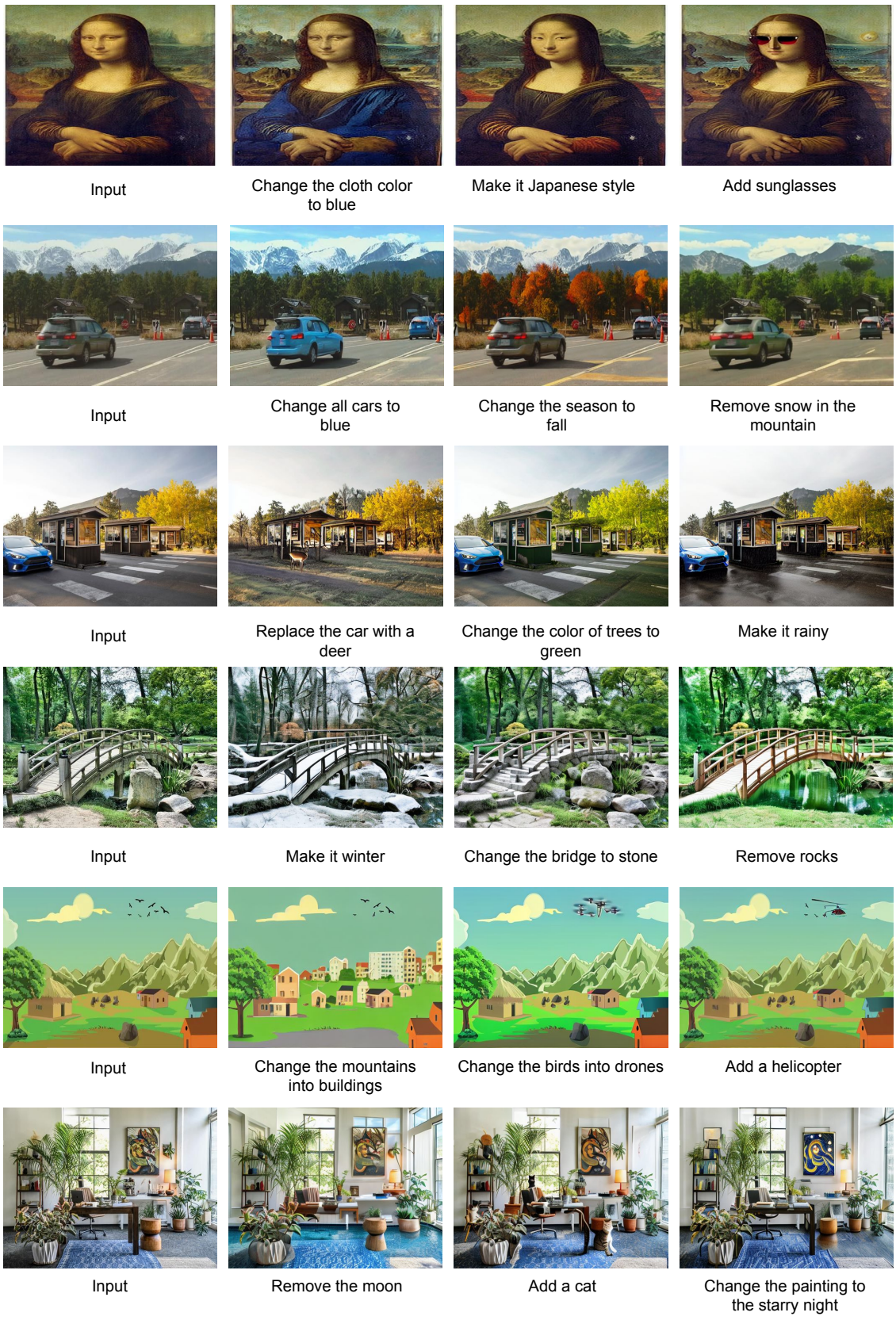


Figure 23. Additional editing results.

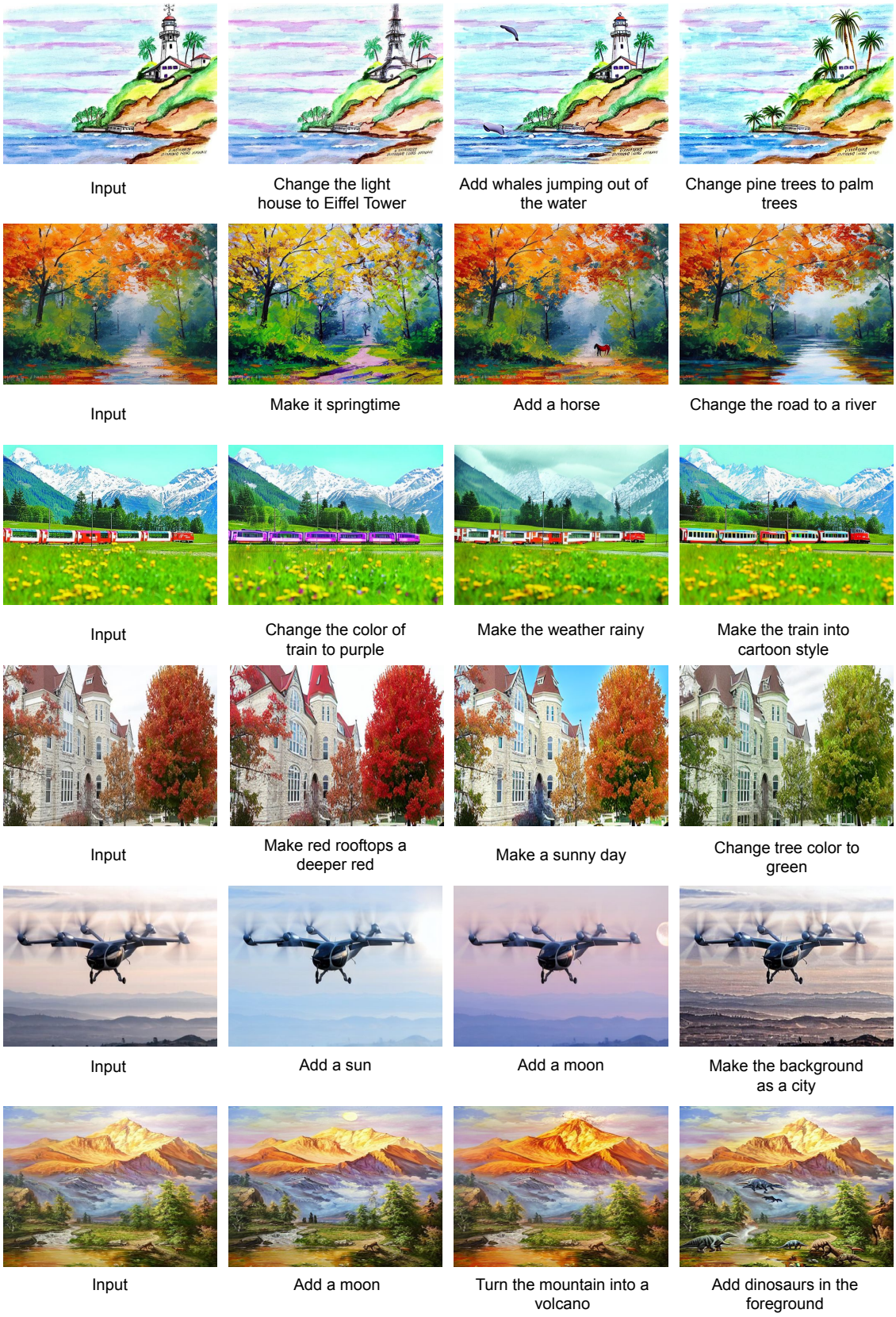


Figure 24. Additional editing results.

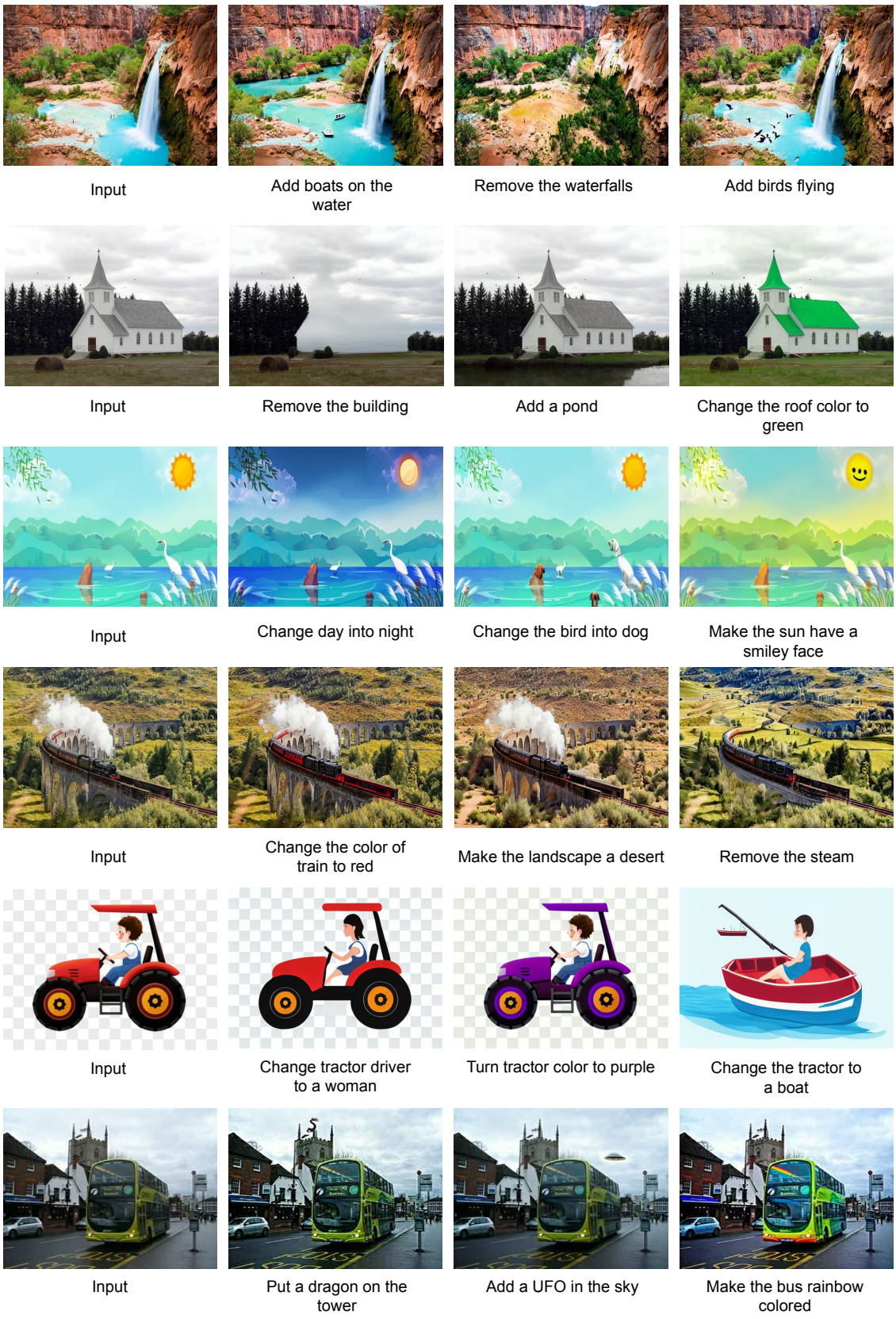


Figure 25. Additional editing results.

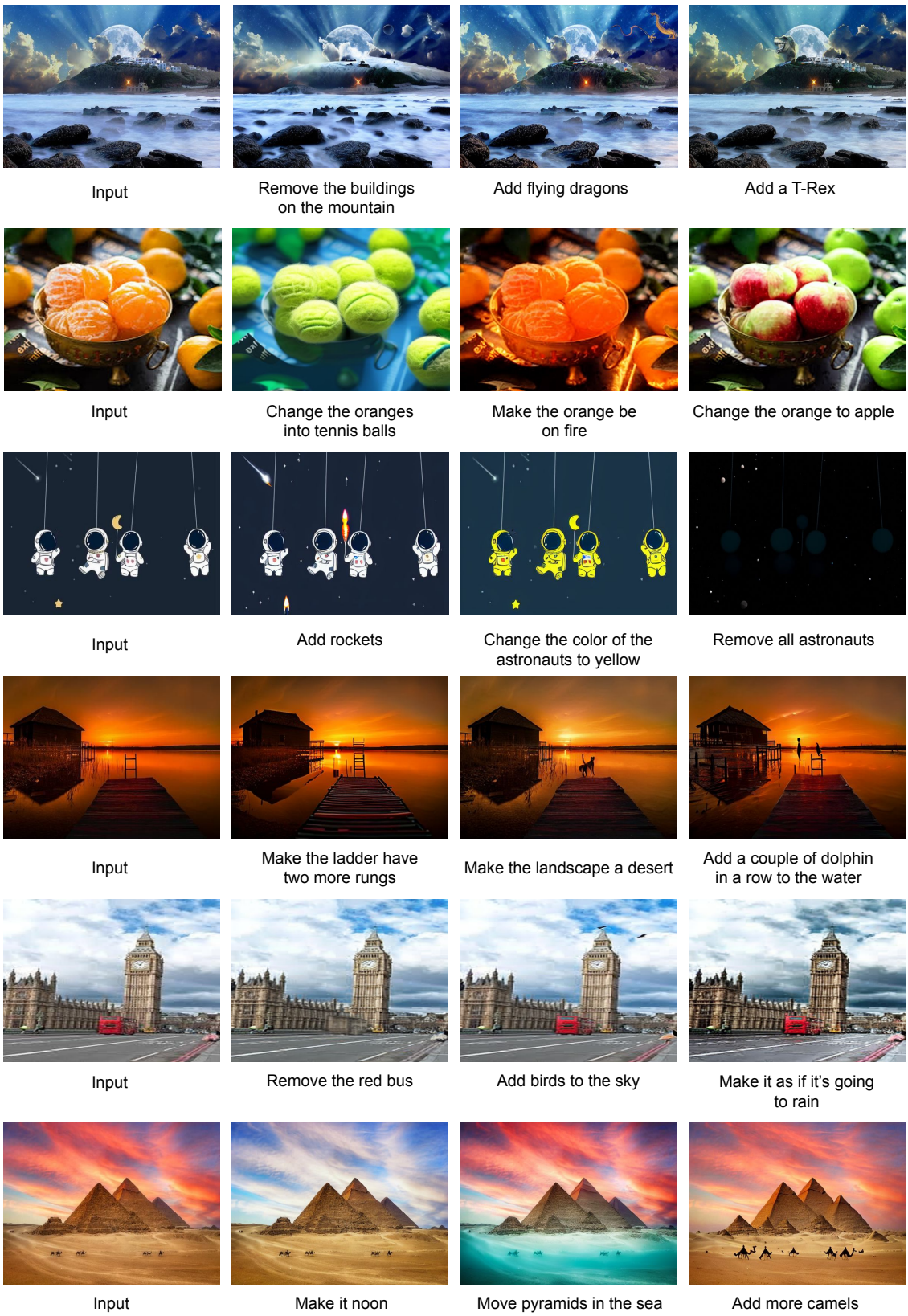


Figure 26. Additional editing results.