LUSD: Localized Update Score Distillation for Text-Guided Image Editing

Supplementary Material

6. Implementation Details

6.1. Identifying noun tokens

As mentioned in Section 3.3, we extract cross-attention maps for all noun tokens related to an edit, which can be inferred by comparing the source and target prompts. We assume that the target prompt is a modified version of the source prompt that either (1) expands on the source prompt or (2) alters specific details within it. Such modifications can appear in various forms, for example:

- a waterfall *with a small boat floating near it*.
- a girl *wearing glasses* sitting in front of a mirror.
- *a bird on* a roof.
- a cup of ("coffee" → *"matcha"*).

We refer to the modified portion as the *differing substring*, which represents the edit. To identify the differing substring, we first remove the longest common suffix and prefix from both prompts, then extract nouns from the remaining target prompt using Part-of-Speech (POS) tags[2]. If the last word of the substring is not (1) a noun, (2) an article, or (3) a preposition, we expand the substring by appending additional words from the target prompt until a noun is included. This step ensures that the extracted segment captures complete noun phrases.

This simple rule-based approach relies on the accuracy of the POS tagger and may not work for all prompt pairs. However, we employ this algorithm to ensure a consistent methodology for both qualitative and quantitative comparisons. In practice, the differing substring can be specified by the user.

6.2. LUSD algorithm

The pseudocode of our LUSD described in Section 3 is given in Algorithm 1 and 2. Our implementation uses $N = 300$, $\eta_0 = 0.01$, $\alpha = 0.1$, $\lambda = 0.02$, $lr = 2000$, and a reverse sigmoid schedule γ.

Algorithm 1: Image Editing with LUSD

Input: $\mathbf{z}^{\text{src}}$: latent code of input image
$y^{\text{src}}, y^{\text{tgt}}$: source and target prompts
lr, λ, N, η_0: hyperparameters
Output: Edited image

$\mathbf{z} \leftarrow \mathbf{z}^{\text{src}}$
for $k \leftarrow 1$ ***to*** N **do**
  $\eta \leftarrow \eta_0$
  $t \sim \mathcal{U}(50, 950)$
  while *True* **do**
    $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
    $\mathbf{z}_t \leftarrow \sqrt{\alpha_t}\mathbf{z} + \sqrt{1-\alpha_t}\epsilon$
    $\epsilon^{\text{tgt}}, \epsilon^{\text{src}} \leftarrow \epsilon_\phi(\mathbf{z}_t, t, (y^{\text{tgt}}, y^{\text{src}}))$
    $\nabla_{\mathbf{z}}\mathcal{L}_{\text{SBP}} \leftarrow \epsilon^{\text{tgt}} - \epsilon^{\text{src}}$
    if $SD(\nabla_{\mathbf{z}}\mathcal{L}_{SBP}) \geq \eta$ **then**
      $\hat{\mathbf{M}}_k \leftarrow$ `AttentionMask` $(\epsilon_\phi, \mathbf{E}, k, \alpha)$ `// Algorithm 2`
      $\nabla_{\mathbf{z}}\mathcal{L}_{\text{SBP-reg}} \leftarrow (1-\lambda)(\hat{\mathbf{M}}_k \odot \nabla_{\mathbf{z}}\mathcal{L}_{\text{SBP}}) + \lambda(\mathbf{z} - \mathbf{z}^{\text{src}})$
      $\nabla_{\mathbf{z}}\mathcal{L}_{\text{LUSD}} \leftarrow \gamma \frac{\nabla_{\mathbf{z}}\mathcal{L}_{\text{SBP-reg}}}{\text{SD}(\nabla_{\mathbf{z}}\mathcal{L}_{\text{SBP-reg}})}$
      $\mathbf{z} \leftarrow \mathbf{z} - lr \cdot \nabla_{\mathbf{z}}\mathcal{L}_{\text{LUSD}}$
      break
    else
      $\eta \leftarrow 0.99\eta$
return `Decode` $(\mathbf{z})$

7. Study on Moving Average in Attention Mask

As discussed in Section 3.3, spatial regularization is introduced to modulate SBP gradients, which may be averaged out over multiple optimization steps (see Figure 4). By estimating the editing region using attention features, our method produces more localized masks than the naive SBP gradients, even without using a moving average (see Figure 11). Nonetheless, we observe that attention masks with a moving average consistently outperform those without it across all metrics (see Table 5).

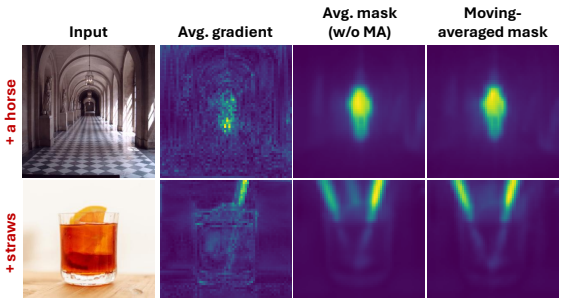


Figure 11. Attention masks are more localized than SBP gradients.

8. Effects of Hyperparameters

This section discusses how hyperparameters influence background preservation, gradient filtering, and detail editing. Our default configuration of the regularizer (λ), filtering threshold (η_0), and timestep range ($t_{\min}, t_{\max}$) aims to

[2]We use Natural Language Toolkit's `nltk.tag.pos_tag` and select tokens tagged as `NN` or `NNS`.

Algorithm 2: AttentionMask

Input: ϵ_ϕ: diffusion model
$\mathbf{E}$: Set of target noun tokens
k: Current optimization step
α: Moving average parameter
Output: Attention-based mask $\hat{\mathbf{M}}_k$

for $l \leftarrow 1$ ***to*** L **do**
  $\mathbf{A}_S^{l,t} \leftarrow$ `get_self`(ϵ_ϕ, l)
  $\mathbf{A}_C^{l,t,\mathbf{e}} \leftarrow$ `get_cross`$(\epsilon_\phi, l, \mathbf{e}), \forall \mathbf{e} \in \mathbf{E}$
$\mathbf{A}_S^t \leftarrow \frac{1}{L}\sum_{l=1}^{L} \mathbf{A}_S^{l,t}$
$\mathbf{A}_C^{t,\mathbf{e}} \leftarrow \frac{1}{L}\sum_{l=1}^{L} \mathbf{A}_C^{l,t,\mathbf{e}}, \forall \mathbf{e} \in \mathbf{E}$
$\hat{\mathbf{A}}_C^t \leftarrow \mathbf{A}_S^t \cdot \left(\frac{1}{|\mathbf{E}|}\sum_{\mathbf{e}\in\mathbf{E}} \mathbf{A}_C^{t,\mathbf{e}}\right)$
$\mathbf{M} \leftarrow \frac{\hat{\mathbf{A}}_C^t - \min(\hat{\mathbf{A}}_C^t)}{\max(\hat{\mathbf{A}}_C^t) - \min(\hat{\mathbf{A}}_C^t)}$
if $k = 1$ **then**
  $\mathbf{M}_k \leftarrow \mathbf{M}$
else
  $\mathbf{M}_k \leftarrow (1-\alpha)\mathbf{M}_{k-1} + \alpha\mathbf{M}$
$\beta \leftarrow k/N$
$\hat{\mathbf{M}}_k \leftarrow \beta\mathbf{M}_k + (1-\beta)\mathbb{1}$
return $\hat{\mathbf{M}}_k$

Moving average	CLIP-T ↑	CLIP-AUC ↑	L1* ↓	CLIP-I* ↑
Without	0.286	0.071	0.0148	0.1921
With (Ours)	0.287	0.074	0.0146	0.1923

Table 5. Applying moving average when computing attention mask yields better results on MagicBrush across all metrics.

ensure the right extent of image modification, robustness against bad gradients from uncommon concepts, and the ability to alter both low- and high-frequency image features.

Regularizer (λ). The regularizer, as used in Equation 3, is crucial for preserving the background during edits. Without the regularizer ($\lambda = 0$), the method modifies the entire image to match the prompt. Conversely, increasing λ limits the extent of the edited region. An overly high λ can prematurely eliminate essential visual cues before larger objects form during the optimization process and thus worsen the quality of the results. Figure 12 illustrates how varying λ affects outcomes.

Filtering threshold (η_0). The filtering threshold η_0 helps prevent edit reversion caused by applying *bad* gradients (Section 3.4). Its necessity varies based on input concepts due to the differing prior knowledge encoded in Stable Diffusion [37]. For instance, less recognizable concepts like "Marengo" (the war horse of Napoleon) has higher chances of encountering bad gradients compared to more common ones like "Eevee" (the Pokémon), necessitating a higher η_0. The right value of η_0 also depends on the input image and the composition of the input prompt. For instance, a prompt such as "Game of Thrones dragon" would already yield a high editing success rate without gradient filtering ($\eta_0 = 0$) because it includes the common term "dragon." Effects of various η_0 values are shown in Figure 13. Lastly, the value of η_0 affects our method's speed because a higher η_0 requires more optimization time as more gradients are filtered.

Timestep range ($t_{\text{min}}, t_{\text{max}}$). The default configuration samples diffusion timesteps $t \sim U(t_{\text{min}}, t_{\text{max}})$, with $t_{\text{min}} = 50$ and $t_{\text{max}} = 950$. Lower timesteps allow the method to better resolve high-frequency details such as texture, which is essential for tasks like transforming "wildflower" into "roses" (Figure 14). Higher timesteps, by contrast, focus on low-frequency details like color, which is crucial for edits such as altering "coffee" to "matcha" (Figure 15).

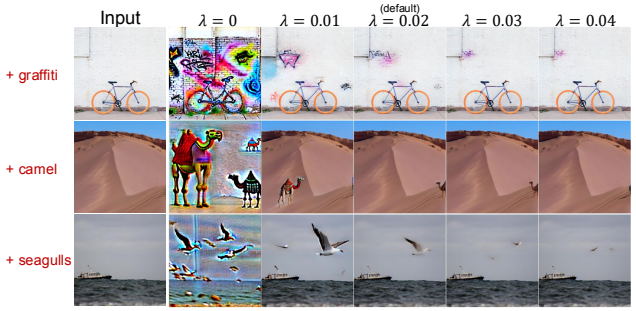


Figure 12. Regularizer λ is necessary for background preservation; however, a higher λ may restrict the size of the edited region.

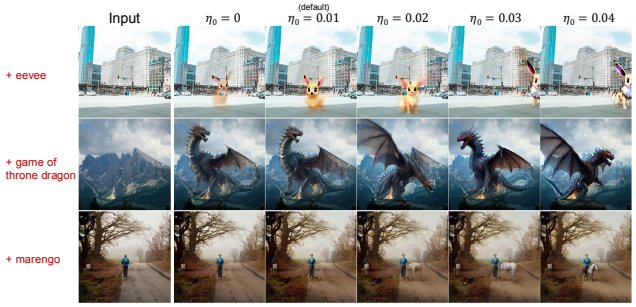


Figure 13. Higher filtering threshold (η_0) mitigates the *bad* gradient issue with less known concepts such as "Marengo" (the war horse of Napoleon), albeit requiring more optimization time.

Figure 14. Low timestep range's lower bound (t_{min}) is necessary for editing high-frequency details, such as the texture of "roses."

Figure 15. High timestep range's upper bound ($t_{\max}$) is necessary for editing low-frequency details, such as the color of "matcha."

9. Additional Experimental Details

9.1. MagicBrush classification

In Table 3 of the main paper, we report scores for examples from the MagicBrush test set [49] involving object insertion. To identify these examples, we first compile a list of keywords for each editing task. For object insertion, we use *add*, *put*, and *let there be*. For other tasks, we use *remove*, *erase*, *delete*, *replace*, *swap*, *make*, *change*, *turn*, *smaller*, *bigger*, *larger*, *smile*, *cry*, and *look*. Instructions containing these keywords are automatically categorized accordingly, and the rest of the instructions, approximately 35%, are classified manually by the authors.

9.2. Human evaluation

As mentioned in Section 4.1, the user study evaluated 200 samples from the MagicBrush test set. Of these, 100 were randomly selected from object insertion tasks and the rest from other tasks. We compared our method against five state-of-the-art competitors in a one-on-one setup, which results in $200 \times 5 = 1000$ sample-competitor pairs.

Each worker was presented with multiple sample-competitor pairs. For each pair, they saw the input image, the edit instruction, the target caption, and the outputs of our method and the competitor. The worker was not informed of the task type the sample belongs to, and the outputs were presented side-by-side in a randomized order to prevent positional bias. They were asked to choose between our method and the competitor as the better method based on 4 criteria: (1) background preservation, (2) prompt fidelity, (3) quality of edited elements, and (4) overall preference. The user interface and detailed instructions are shown in Figure 16.

A total of 350 unique workers participated via Amazon Mechanical Turk[3]. We designed the study so that five different workers evaluated each sample-competitor pair. For each sample, the better method in a one-on-one comparison was determined by majority vote (i.e., at least 3 out of 5 workers selected it). As discussed in Section 4.1, our method outperforms all state-of-the-art approaches when considering all tasks.

Table 6 presents the scores separately for object insertion (Add) and other tasks (Other). Our method achieves higher overall preference scores in both task categories, except when compared to CDS in the "Other" category. Upon examination, we found that this outcome stems from examples involving complex edits where both methods struggle to match the target prompt, such as those illustrated in Figure 17. In such cases, our method attempts modifications, sometimes introducing slight visual artifacts or corrupted elements. In contrast, CDS makes almost no changes to the input image. While this conservative behavior in CDS does not adhere to the target prompt, it avoids introducing errors, leading to higher scores in these specific cases.

9.3. Inherent biases in commonly used background preservation metrics

Metrics commonly used to assess background preservation in previous works [39, 49] are L1, CLIP-I, and DINO, all computed on the MagicBrush test set [49]. L1 is defined as the L_1-norm between the edited and reference images, while CLIP-I measures the cosine similarity between their CLIP embeddings. Similarly, DINO computes the cosine similarity between their DINO [10] embeddings, making it highly correlated with CLIP-I.

In MagicBrush [49], the reference, or "ground-truth" images, were created by workers on Amazon Mechanical Turk with a mask-based inpainting model DALL-E 2[4]. While this process yields high-quality edited images verified by humans, each input has only one "correct" ground-truth image. As a result, the metrics may penalize good results where changes are made in a perfectly valid location but not in the single ground-truth location specified by workers during dataset creation.

We illustrate this on the MagicBrush test set. Specifically, we compute pixel-wise differences between the input and ground-truth reference image to infer a ground-truth edit region $\mathbf{B}_1$. We then use Paint-by-Example [47] to insert *same-sized objects* (sourced from Unsplash) into both $\mathbf{B}_1$ and other plausible regions $\mathbf{B}_2$. As shown in Table 7, these metrics disfavor editing made in $\mathbf{B}_2$.

Moreover, these metrics can produce misleading rankings by favoring unchanged outputs over valid edits that deviate from the ground truth (see Table 8). Additionally, as DINO is trained via self-supervised training to capture differences between objects of the same class, the DINO metric may penalize valid edits that produce the correct object but with an appearance different from the one in the ground-truth image.

For these reasons, we exclude these metrics from Table 2, propose four new metrics (Section 4.2) and assess visual quality with a user study (Section 4.1).

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

[4] https://openai.com/index/dall-e-2/

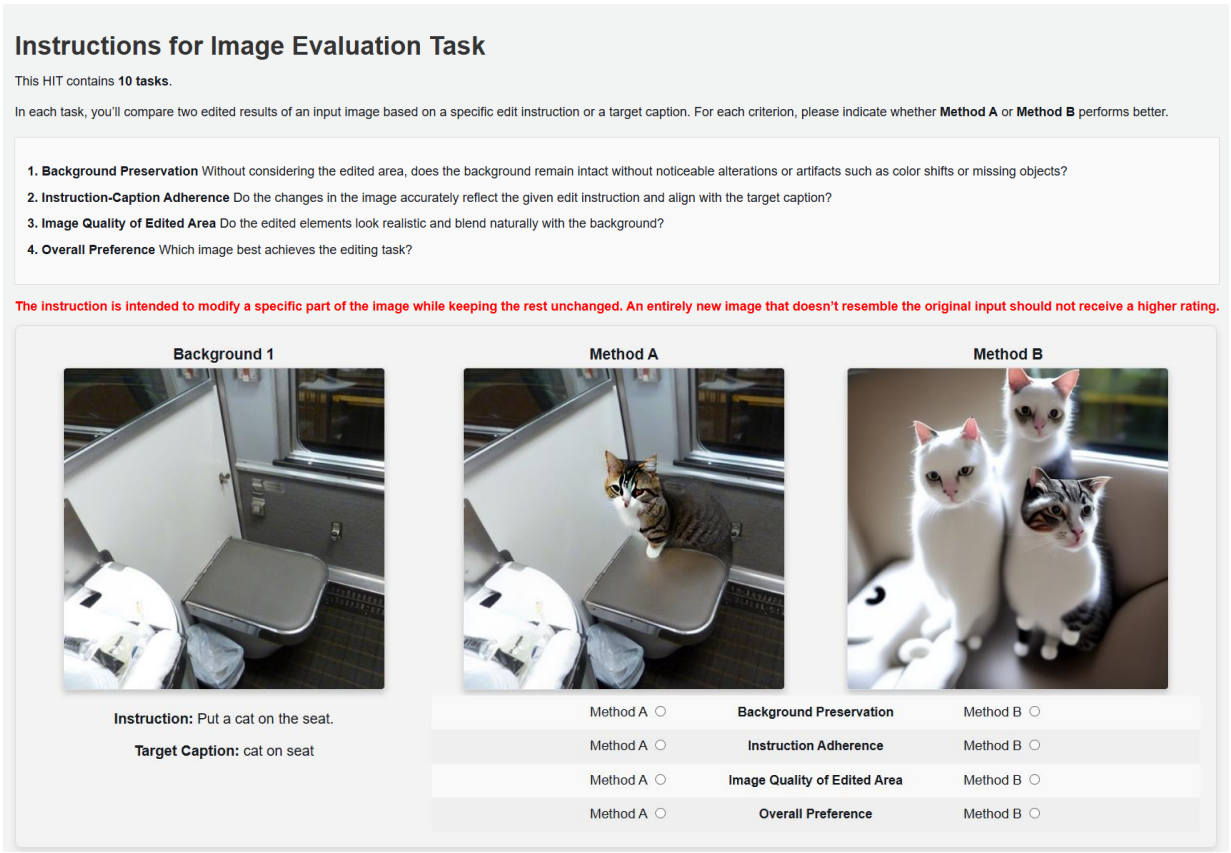


Figure 16. User study interface.

Method	Background		Prompt		Quality		Overall	
	Add	Other	Add	Other	Add	Other	Add	Other
InstructPix2Pix [6]	30.0%	37.0%	39.0%	41.0%	35.0%	38.0%	36.0%	36.0%
HIVE [51]	42.0%	52.0%	34.0%	47.0%	42.0%	48.0%	34.0%	44.0%
LEDITS++ [5]	31.0%	40.0%	27.0%	39.0%	27.0%	47.0%	29.0%	41.0%
DDS [16]	39.0%	48.0%	31.0%	43.0%	34.0%	42.0%	35.0%	42.0%
CDS [30]	28.0%	61.0%	25.0%	55.0%	31.0%	55.0%	29.0%	55.0%

Table 6. Percentage of times users preferred other methods over ours in 1-on-1 comparisons. We present the scores separately for samples involving object insertion (Add) and other tasks (Other). Please refer to Section 4.1.

Locations	L1 ↓	CLIP-I ↑	DINO ↑
Same ($\mathbf{B}_1$)	0.048	0.911	0.876
Different ($\mathbf{B}_2$)	0.057	0.899	0.839
p-value	2.33e-9	1.88e-2	1.28e-3

Table 7. Editing the same region as the reference images yields statistically better scores ($N = 70$). We restrict our test to cases where the ground-truth region $\mathbf{B}_1$ is sufficiently small, allowing us to select a non-overlapping region of the same size $\mathbf{B}_2$ for inpainting using Paint-by-Example.

10. Additional Qualitative Results

10.1. Benchmark dataset

This section provides qualitative results for the experiment on MagicBrush test set [49] in Section 4 of the main paper. We show editing results from our LUSD method alongside other state-of-the-art approaches in Figures 24 and 25. While our method may occasionally make incorrect edits (e.g., the bottom two examples in Figure 24) due to the inherently limited language understanding of Stable Diffusion, it generally offers a good balance between prompt fidelity and background preservation.

Additionally, Figure 19 compares the performance of our

Method	L1 ↓	CLIP-I ↑	DINO ↑
Do Nothing	0.037	0.943	0.917
InstructPix2Pix [6]	0.147	0.782	0.607
HIVE [51]	0.090	0.893	0.824
LEDITS++ [5]	0.097	0.864	0.775
DDS [16]	0.066	0.920	0.886
CDS [30]	0.061	0.931	0.902
SBP [26]	0.095	0.825	0.752
Ours	0.063	0.900	0.853

Table 8. The best and second-best scores are color-coded. We observe that the commonly used L1, CLIP-I, and DINO metrics for this task are biased toward unchanged results, with a method that does nothing to the input (Do Nothing) ranking first across the board. As a result, comparisons based on these scores can be misleading. We discuss this limitation in Section 9.3 and propose less biased evaluation in Section 4.

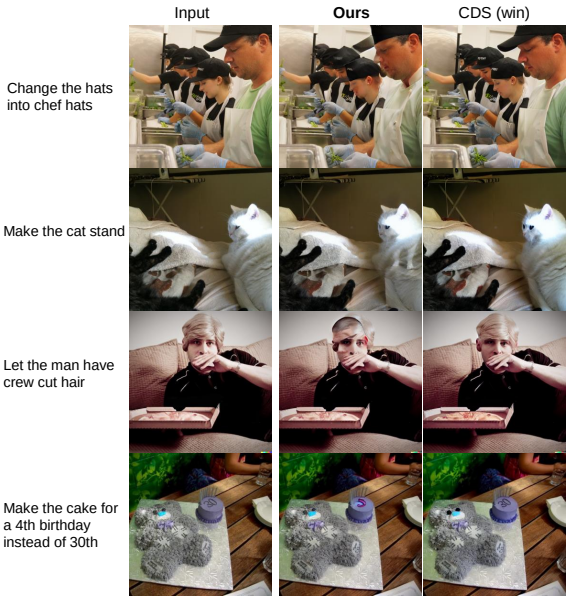


Figure 17. For complex edits, both CDS and our method fail to match the target prompt. However, CDS typically returns almost unchanged results, whereas our method may introduce artifacts.

full method against its ablated versions. Excluding spatial regularization results in entirely new images. Not annealing normalized gradients magnitude via γ produces visual artifacts due to unstable optimization. Without gradient filtering and normalization, our method often struggles to insert objects correctly or produces incomplete additions.

10.2. In-the-wild images

As images in MagicBrush [49] are curated from MS COCO dataset [25] only, we present additional qualitative results for diverse images under CC4.0 license from Unsplash[5] and other websites, using multiple random seeds in Figures 26 to 28. We input source prompts and target prompts directly into LEDITS++ [5], DDS [16], CDS [30], and our method. For InstructPix2Pix [7] and HIVE [51], we use edit instructions generated by ChatGPT, as these models are trained on edit instructions. To generate these inputs, given a source prompt and a target prompt from MagicBrush, we ask chatgpt to generate an edit instruction, prepping it with a short prompt that contains a couple of examples of desired text transformation. Note that this approach is similar to the procedure used in MagicBrush, where a global description (i.e., a target prompt) is inferred from a source prompt and an edit instruction using ChatGPT.

Unlike other methods, which require adjusting hyperparameters for each image to achieve good editing results, LUSD achieves competitive performance—or even better in challenging cases involving object insertion—using a single configuration. It also works across diverse scenarios, such as adding a Google logo to a t-shirt, adding a party hat to a cat, and replacing meatballs with chrome balls. Refer to Appendix 12 for hyperparameter tuning grids.

10.3. Comparison with object insertion works

Our work focuses on stabilizing score distillation, which enables general image editing using diffusion priors. This differs from object insertion techniques that specifically tackle object insertion with supervised fine-tuning on datasets such as Paint-by-Inpaint [45] and Diffree [52]. While supervised approaches generally perform better for common objects (e.g., curtain, apple, turtle), they can produce qualitatively worse results for objects outside their training classes (e.g., dragon, Pikachu, Minion), as shown in Figure 22. Interestingly, even fine-tuned models exhibit the minimal-effort issue (Section 11), albeit to a lesser extent (e.g., sunglasses on a statue, candle). Bridging the gap between these two approaches remains an interesting research direction.

10.4. Comparison with rectified flow models

In Section 4, we limit our comparison to methods applicable to Stable Diffusion [37] and those fine-tuned on it to ensure a fair evaluation, as models vary in their prior knowledge and language understanding. Nonetheless, we also include comparisons with RF-Inversion [38] and RF-Edit [42], both zero-shot methods designed for rectified flow models. For implementation, we use FLUX.1-dev [6] with Diffusers' implementation for RF-Inversion and the official implementation for RF-Edit. Following the paper's recommendation, we set the inversion prompt in RF-Inversion to an empty string and limit the number of feature-sharing steps in RF-edit to 5, with other hyperparameters set to default values.

[5] https://unsplash.com/

[6] https://huggingface.co/black-forest-labs/FLUX.1-dev

Note that the number of parameters in Stable Diffusion and FLUX.1-dev are 1.3 billion and 12 billion, respectively.

As shown in Table 9, our method outperforms RF-Edit and is competitive with RF-Inversion in CLIP-T on the MagicBrush [49] test set. However, RF-Inversion outperforms our method in CLIP-AUC. This improvement can be due to RF-Inversion and RF-Edit's ability to handle more complex edits (making a cat meowing, altering texts, and opening a pizza box) by leveraging the richer prior and better language understanding of the larger FLUX.1-dev (Figure 23). Nonetheless, these methods still struggle with background preservation, which is the central challenge addressed by our work.

Method	**CLIP-T** ↑	**CLIP-AUC** ↑	**L1*** ↓	**CLIP-I*** ↑
RF-Inversion	0.287	0.096	0.026	0.171
RF-Edit	0.279	0.068	0.016	0.182
Ours	0.287	0.074	0.015	0.192

Table 9. Comparison on MagicBrush between rectified-flow-based methods and our method.

11. Additional Failure Cases

Our technique successfully improves the success rate of SDS-based image editing, particularly for object insertion. However, it remains susceptible to *minimal-effort regions*, where the visual cues needed for object formation are already present, leading our method to only add objects there. As shown in Figure 18, these cues can manifest as intensity (e.g., a candle), color (e.g., bread), or shape (e.g., a ship or sunglasses). We observed that such regions are associated with unusually high values in the cross-attention map $\mathbf{A}_C^{l,t,\mathbf{e}}$, averaged across layers l, timesteps t, and target noun tokens $\mathbf{e}$ (see Section 3.3 and Appendix 6.1). Since the magnitude of averaged gradients correlates with the spatial location of these bright spots, SDS-based methods that derive gradient updates directly from model predictions are inherently vulnerable to this issue. To address this spatial bias, a potential solution might be reweighting attention features. This problem is an interesting area for future work.

12. More Comparison with SOTA Image Editing Methods

In Figures 1, 2 and 10 in the main paper, along with Figures 26 to 28 in Appendix 10.2, we present qualitative comparison between our method and various SOTA approaches: CDS [30], DDS [16], LEDITS++ [5], HIVE [51], and InstructPix2Pix [7]. In this section, we provide the hyperparameter tuning grids for all methods in Figures 29 to 42. For each method, we tune the following hyperparameters:

1. **InstructPix2Pix**:
   - text guidance scale $\omega_T \in \{3, 7.5, 10, 15\}$
   - image guidance scale $\omega_I \in \{0.8, 1.0, 1.2, 1.5\}$
2. **HIVE**:
   - text guidance scale $\omega_T \in \{3, 7.5, 10, 15\}$
   - image guidance scale $\omega_I \in \{1.0, 1.5, 1.75, 2.0\}$
3. **DDS** and **CDS**:
   - learning rate $lr \in \{0.05, 0.10, 0.25, 0.50\}$
   - guidance scale $\omega \in \{3, 7.5, 15, 30\}$
4. **LEDITS++**:
   - skip time step $skip\ t \in \{0.0, 0.1, 0.2, 0.4\}$
   - masking threshold $\lambda_{\text{LEDIT}} \in \{0.6, 0.75, 0.8\}$
   - guidance scale $s_e \in \{10, 15\}$

Unlisted hyperparameters are set to their default values.

13. More Comparison with DDS and CDS

In Figure 2 in Section 2, we provide qualitative comparison for object insertion between our approach and existing SDS-based methods: CDS [30] and DDS [16]. We present 20 additional object insertion results in Figures 21. For DDS and CDS, we include results from both their default configurations and a configuration optimized for better object insertion, manually selected based on hyperparameter tuning detailed in Appendix 12. This *object configuration* employs a higher learning rate (0.25 instead of 0.1) and a higher classifier-free guidance value (15 instead of 7.5).

As shown in Figures 21, our method and other SDS-based methods show competitive performance in common scenarios (e.g., adding sunglasses or a hat to a person). However, the default configurations of existing approaches fail to add objects in more challenging cases, such as inserting a horse into a chateau or putting a necktie on a cat. While the *object configuration* alleviates this issue to some extent, it comes at the cost of poorer background preservation, particularly in earlier common scenarios. Additionally, this configuration still fails in certain instances, such as adding a rabbit to a walkway. In contrast, our method produces good results in most cases, albeit with some minor issues with minimal-effort regions (see Appendix 11).

14. Extension to Other Score Distillation

In this work, we introduce attention-based spatial regularization, along with gradient filtering and normalization, to enhance prompt fidelity while preserving the background in SBP [26] for image editing. Nonetheless, our preliminary study suggests that these components can also effectively improve other distillation algorithms, such as DDS [16], as illustrated in Figure 20. This can be done by simply modifying the noise prediction step (line 8-9 in Algorithm 1) to reflect the DDS loss:

$$\nabla_{\mathbf{z}} \mathcal{L}_{\text{DDS}} = \epsilon_\phi(\mathbf{z}_t, y^{\text{tgt}}, t) - \epsilon_\phi(\mathbf{z}_t^{\text{src}}, y^{\text{src}}, t), \tag{6}$$

where $\mathbf{z}_t^{\text{src}}$ denotes a noisy latent code of the original image.

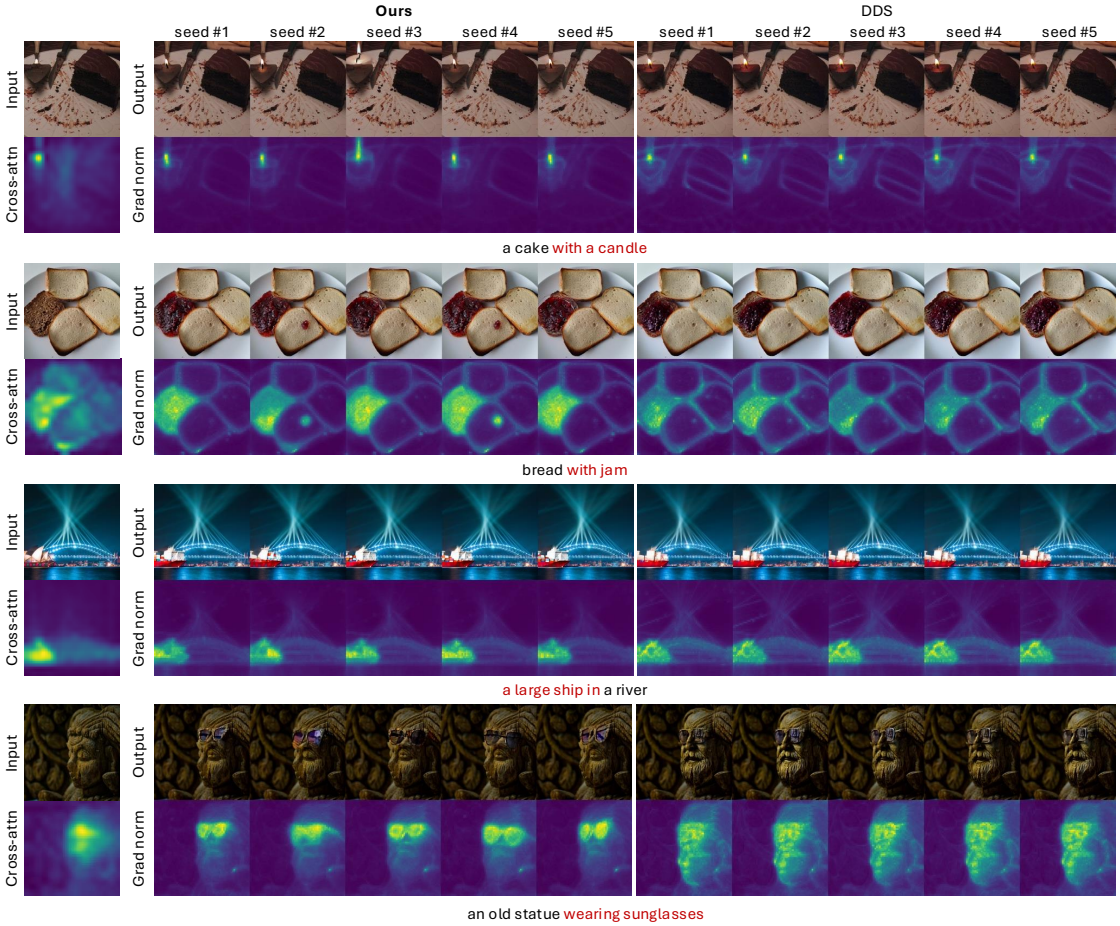


Figure 18. Failure mode: Our method and other SDS-based methods (e.g., DDS [16]) favor minimal-effort regions, where the visual cues needed for object formation are already present. This bias may lead to unnatural object placements or limited diversity in image edits.

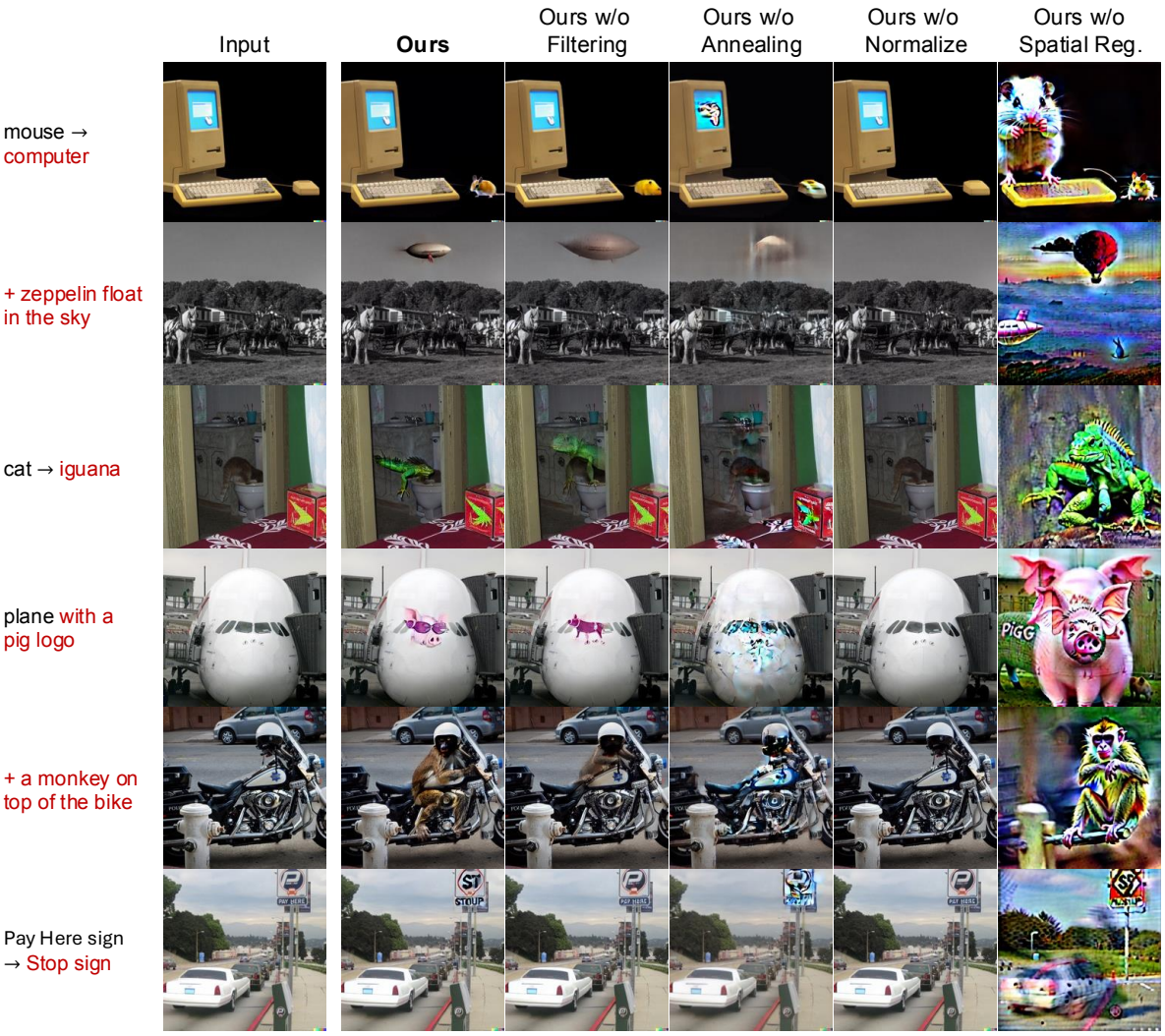


Figure 19. Qualitative results on MagicBrush dataset [49] between our full method and its ablated versions.

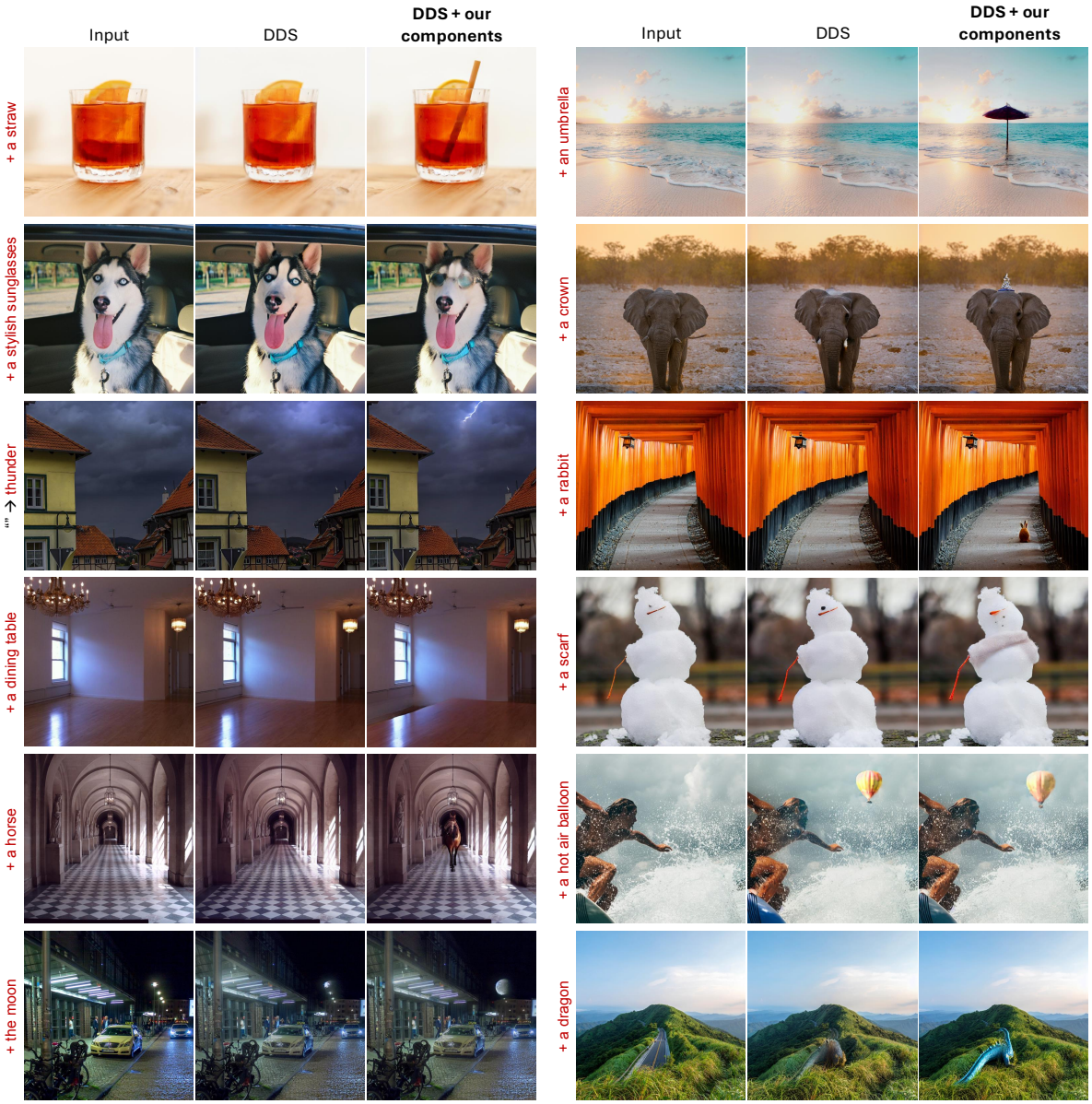


Figure 20. Our regularizer and gradient filtering/normalization help improve DDS's success rate and its background preservation in the default configuration.

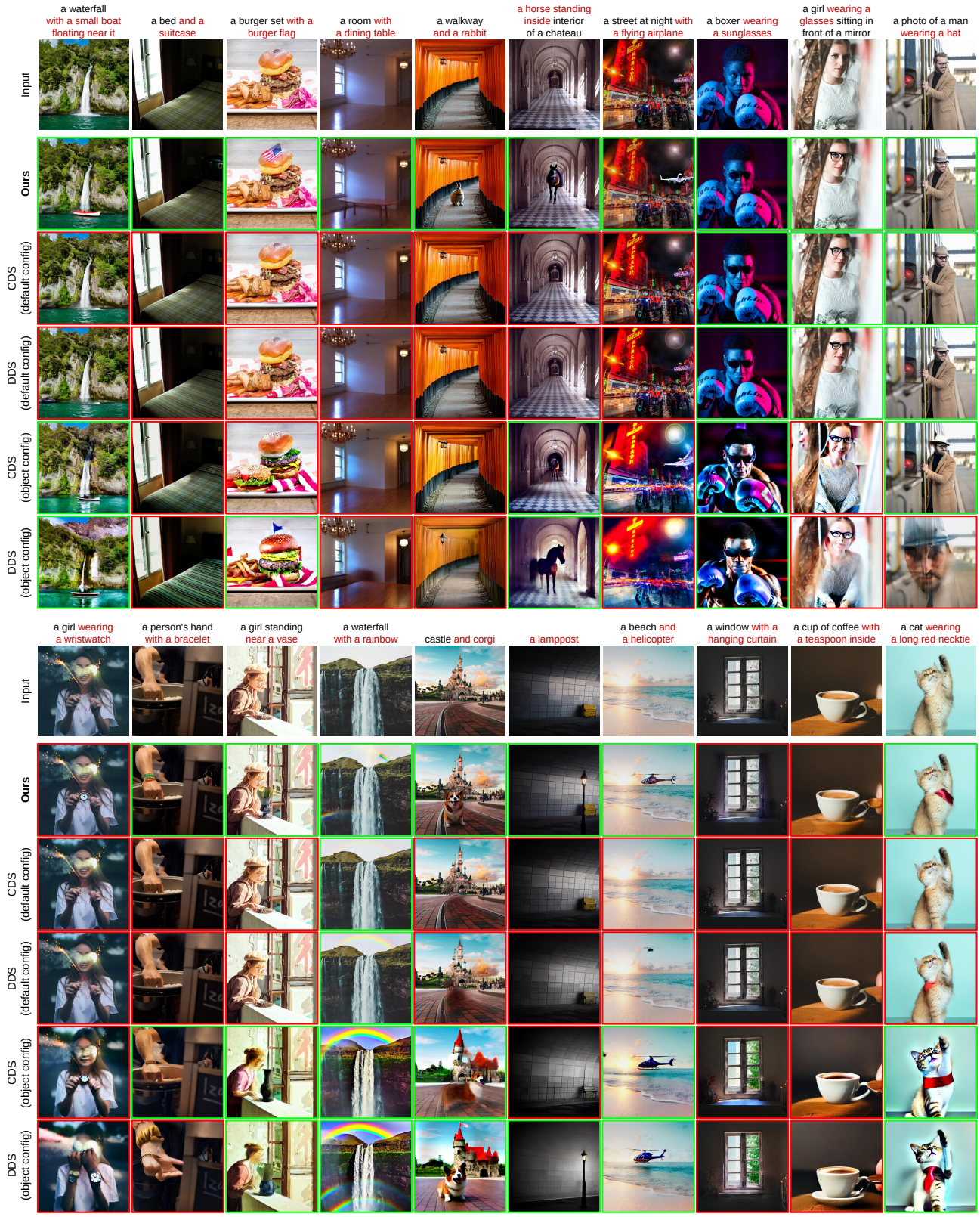


Figure 21. Qualitative results of SDS-based methods for object addition. For CDS [30] and DDS [16], we present results from both the default configuration and an alternative configuration (object config) that encourages object appearance but compromises background preservation. Successful cases are highlighted in green, while failed cases are highlighted in red. Our method has a higher success rate.

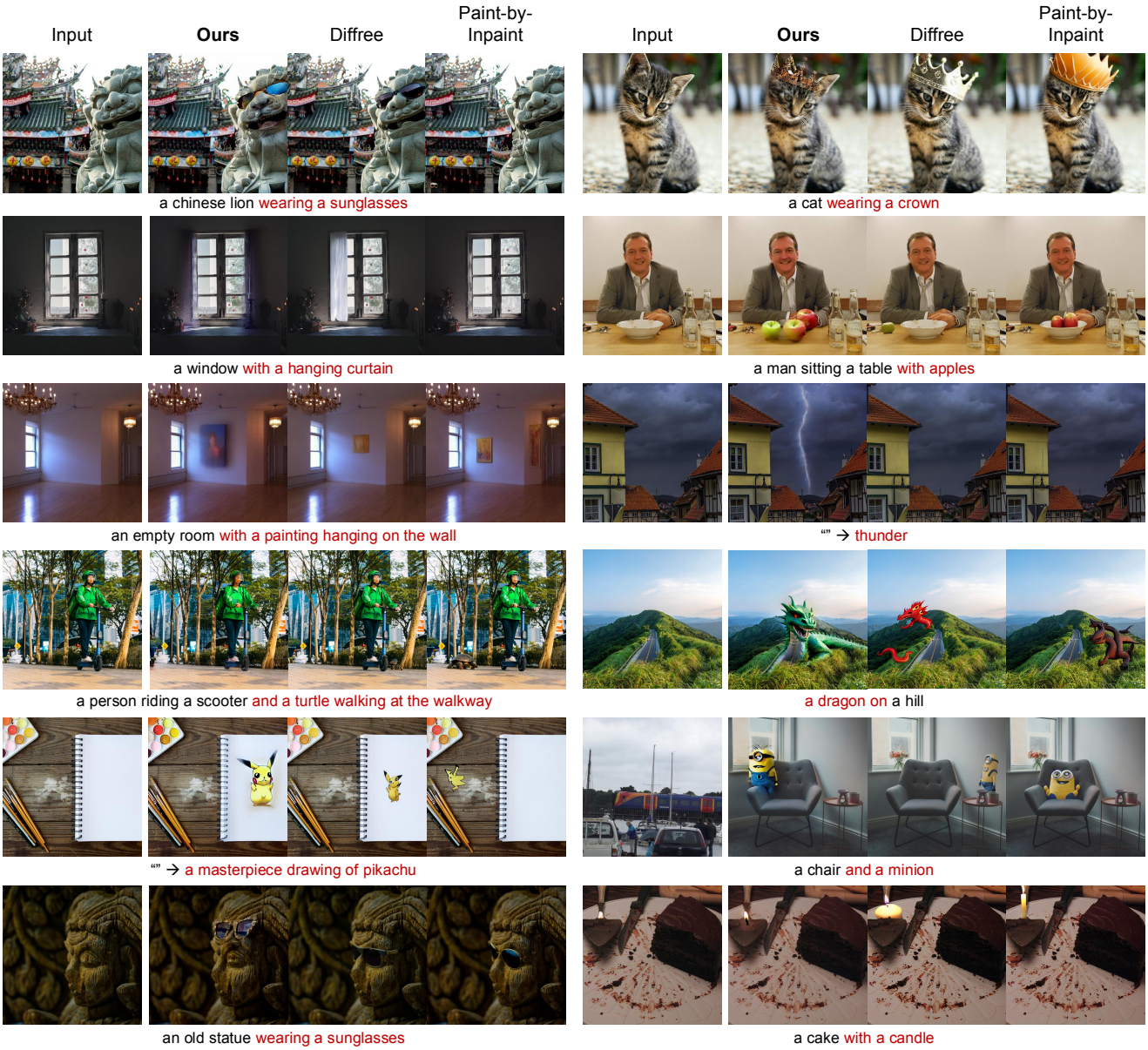


Figure 22. Comparison of our method with other supervised object insertion methods. While task-specific approaches perform better on common objects, they struggle with objects outside their training classes.

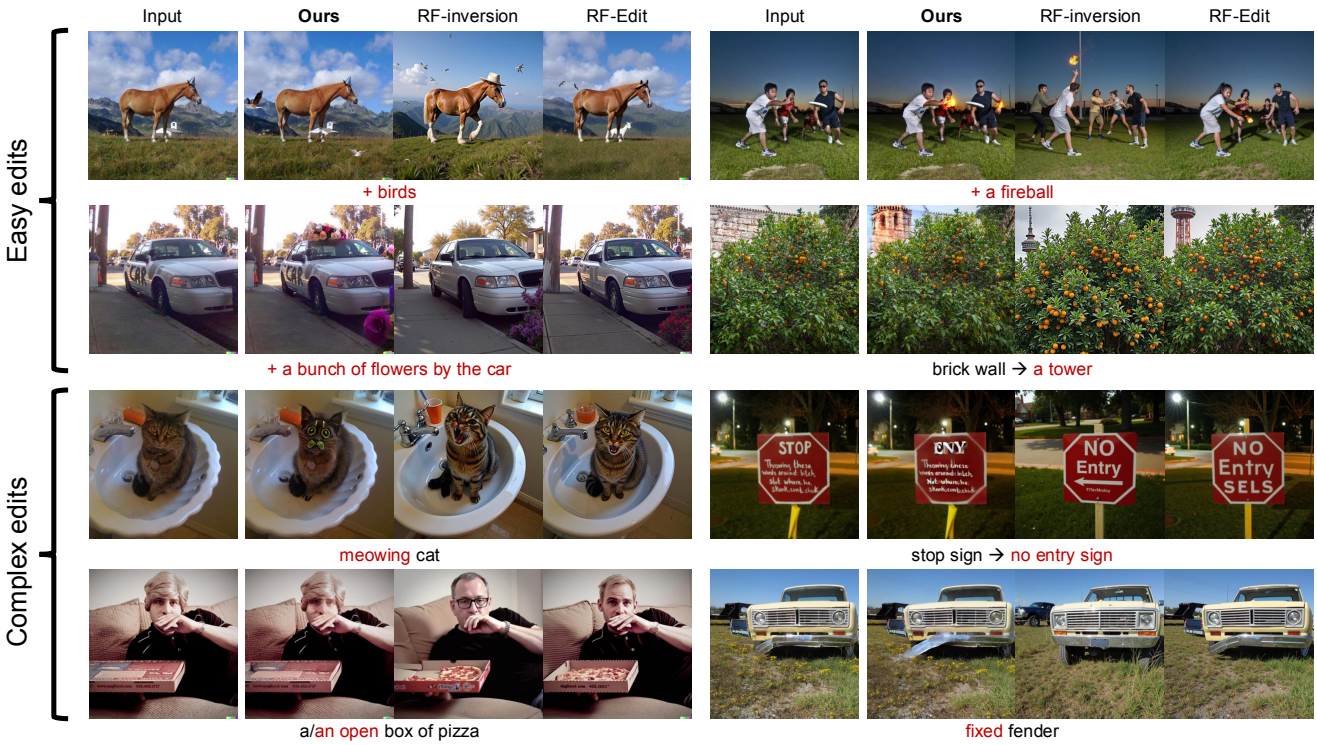


Figure 23. Comparison of our method with zero-shot rectified-flow-based approaches. For simple edits, all methods can follow the text prompt, but our approach better preserves background elements, such as the horse's hat, people's poses, the graffiti on the car, and the distribution of fruits on the plant. However, RF-Inversion and RF-Edit can handle more complex edits by leveraging the richer prior and better language understanding of the larger base model (FLUX.1-dev).

Figure 24. Qualitative results on MagicBrush dataset [49] between our method and other state-of-the-art methods

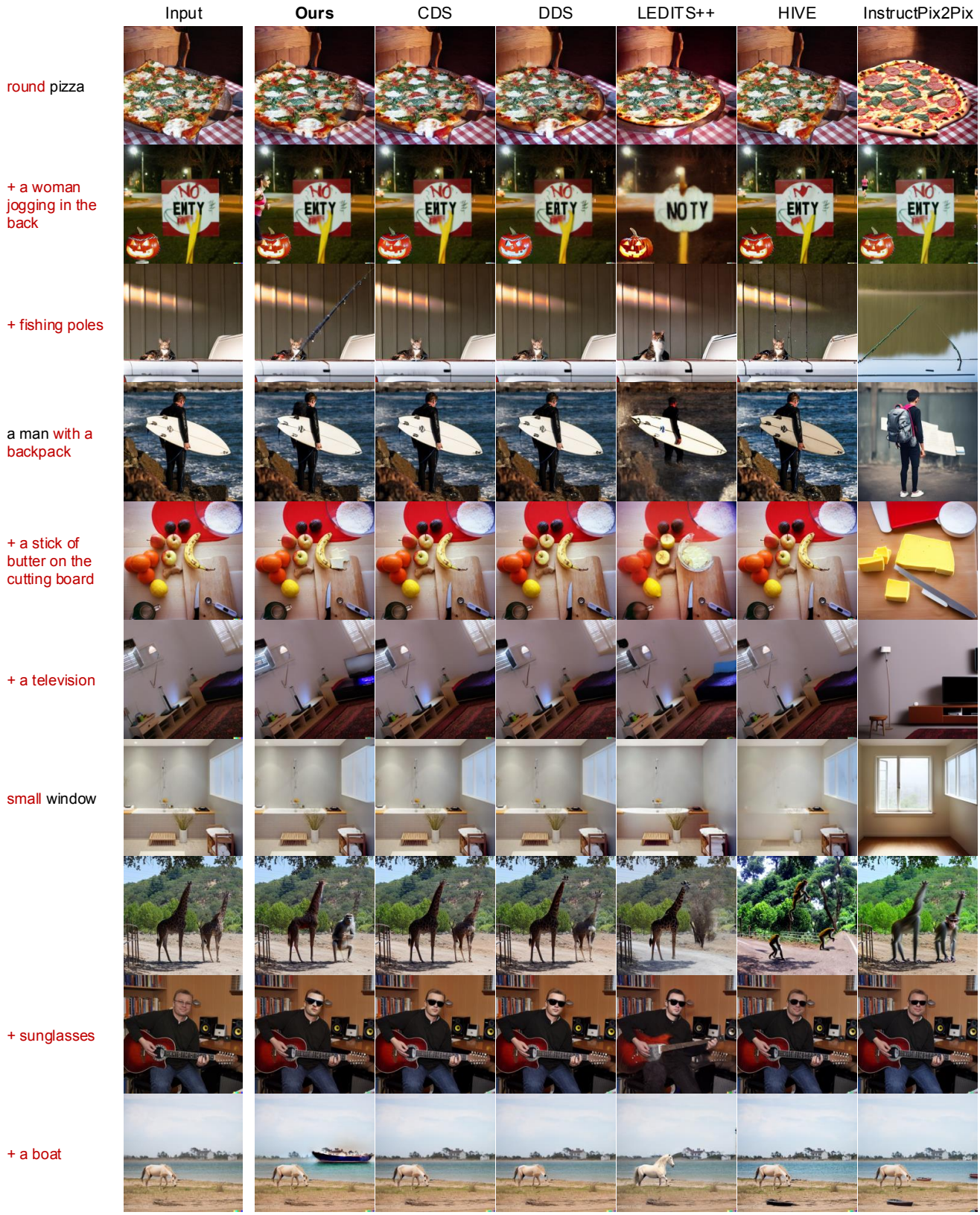


Figure 25. Qualitative results on MagicBrush dataset [49] between our method and other state-of-the-art methods

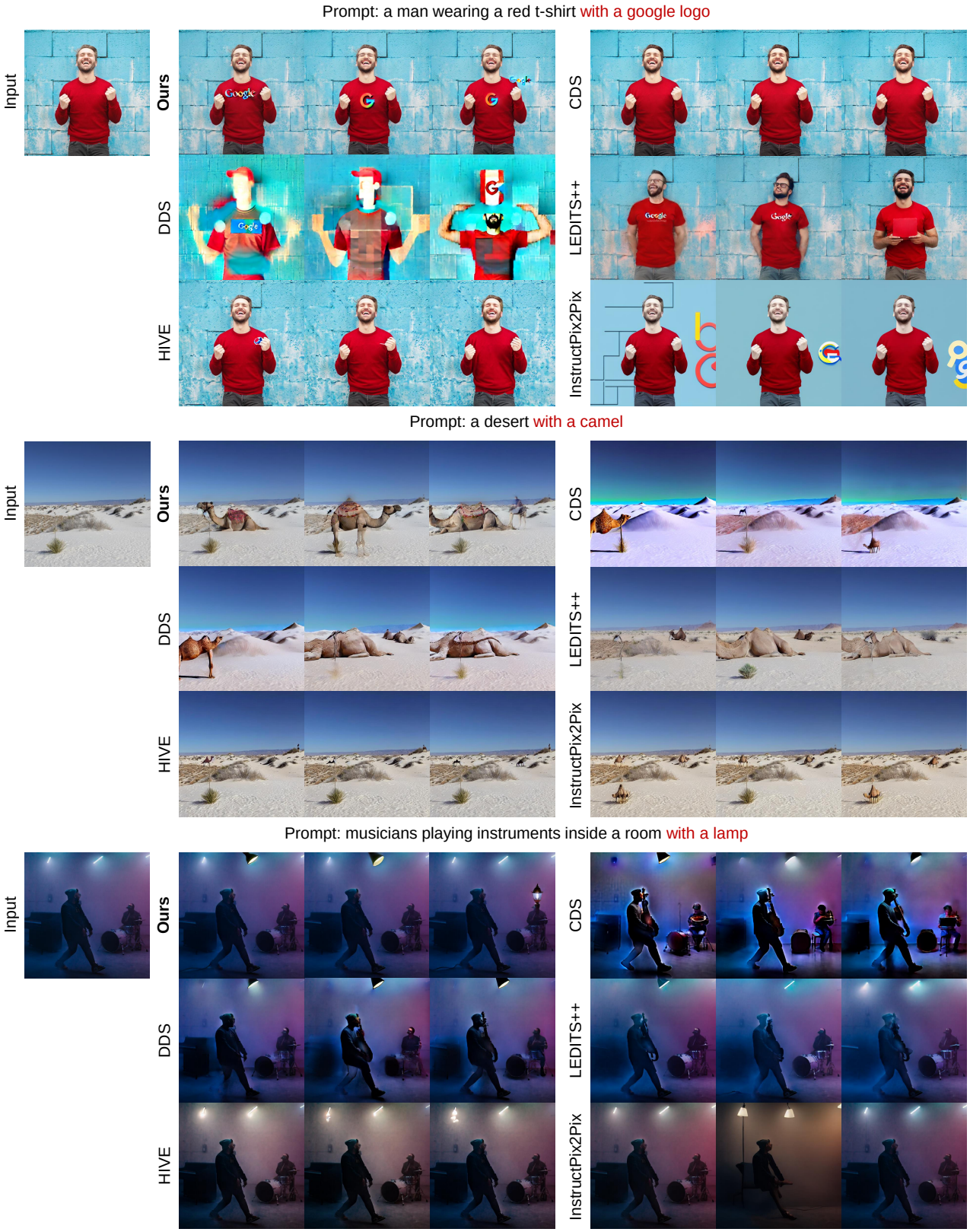


Figure 26. Qualitative results on various in-the-wild editing tasks. We compare the best results (prioritizing object appearance) from state-of-the-art methods, optimized through hyperparameter tuning (see Appendix 12), with our results all generated using a single configuration. For each input image, we show results from 3 different random seeds.

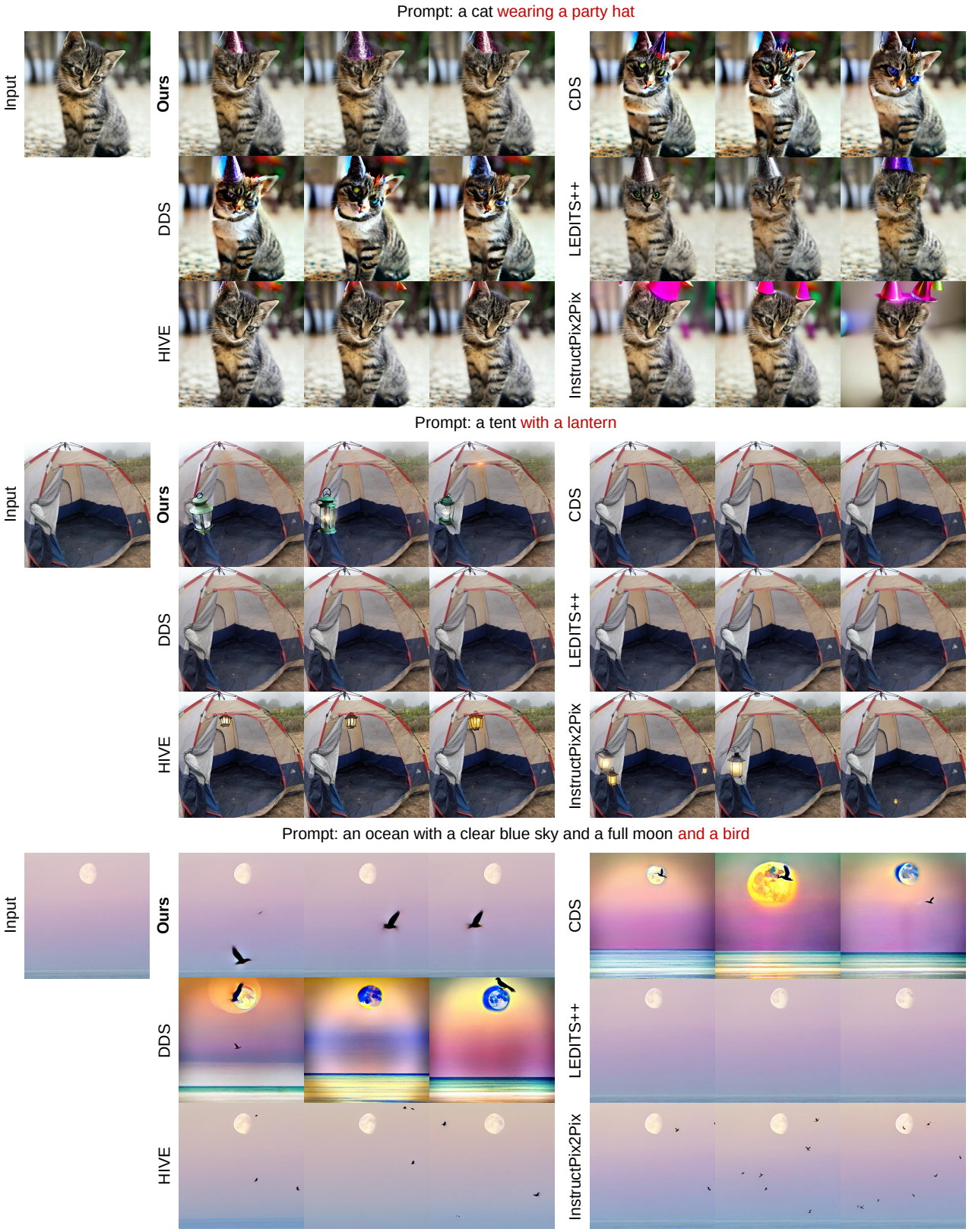


Figure 27. Qualitative results on various in-the-wild editing tasks. We compare the best results (prioritizing object appearance) from state-of-the-art methods, optimized through hyperparameter tuning (see Appendix 12), with our results all generated using a single configuration. For each input image, we show results from 3 different random seeds.

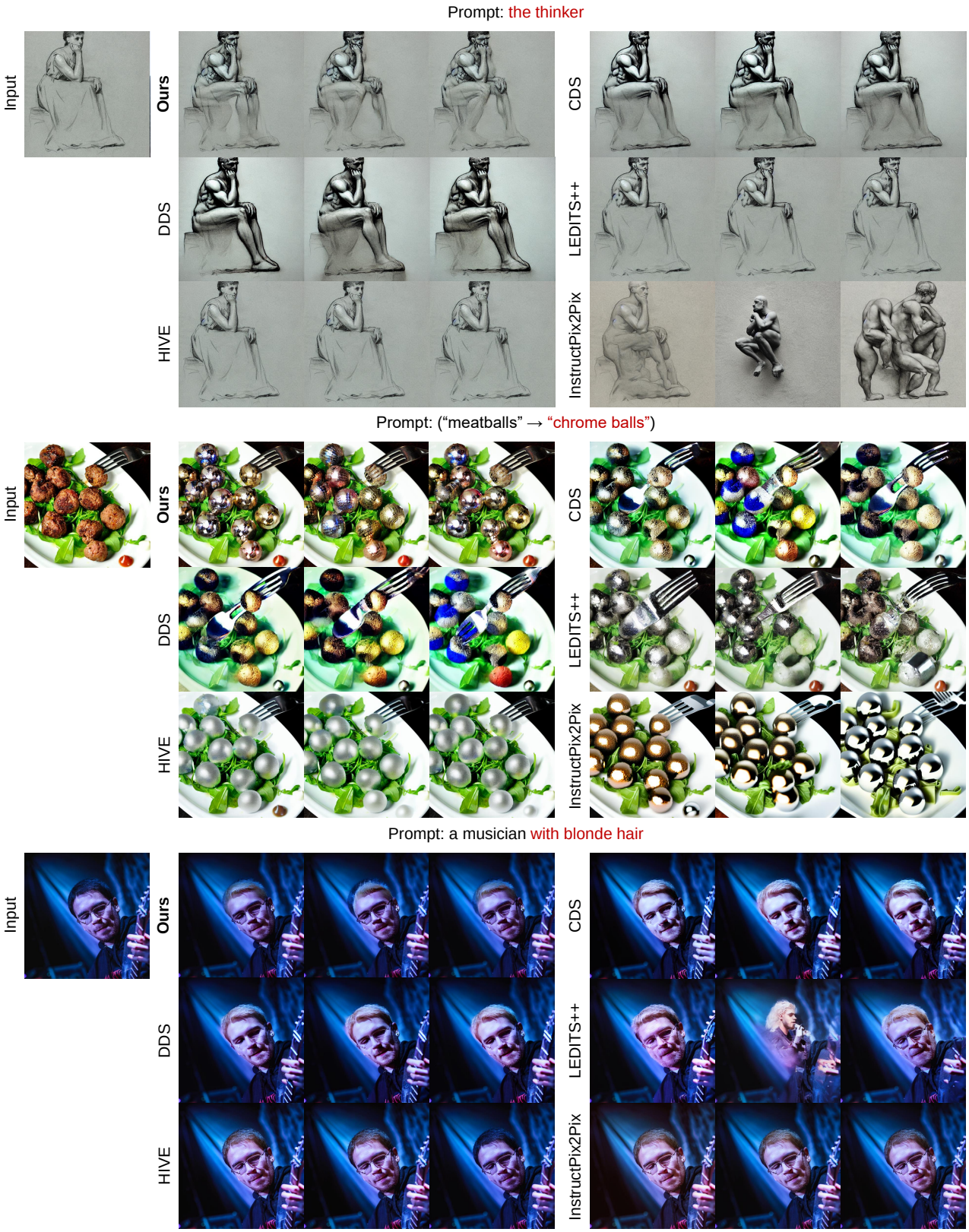
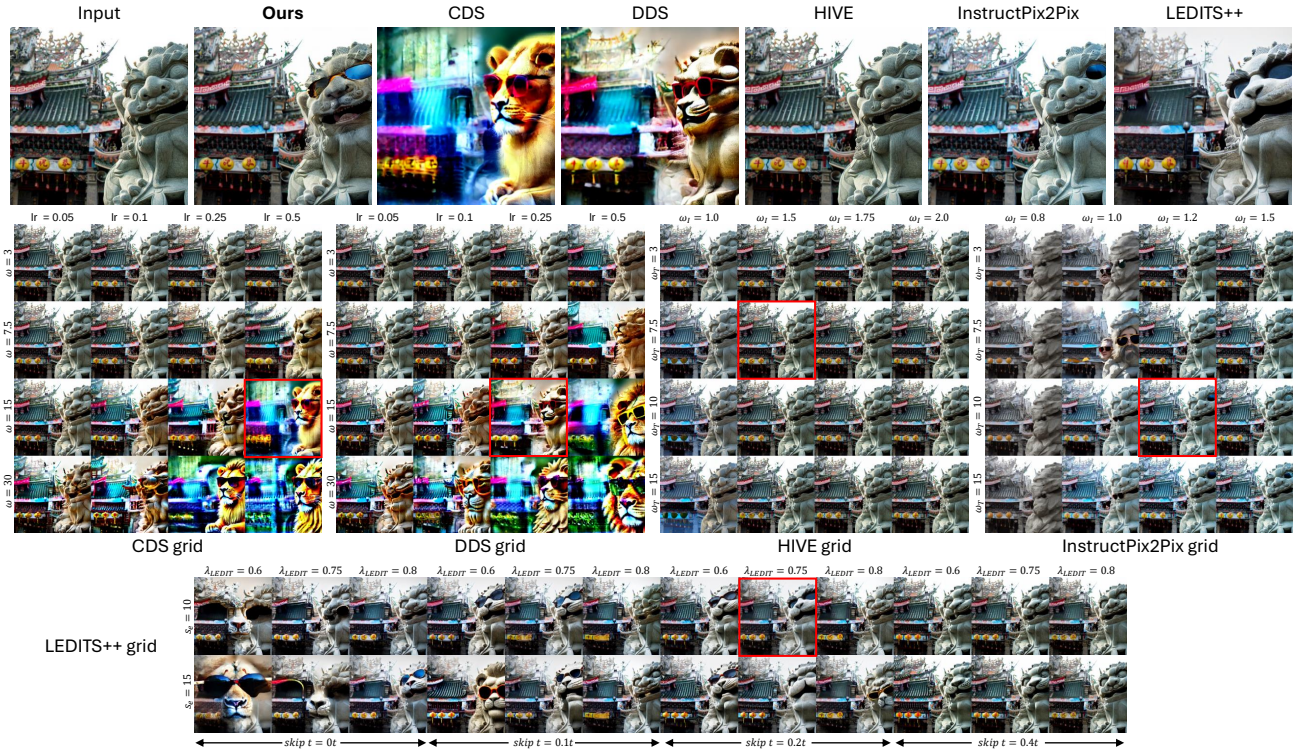
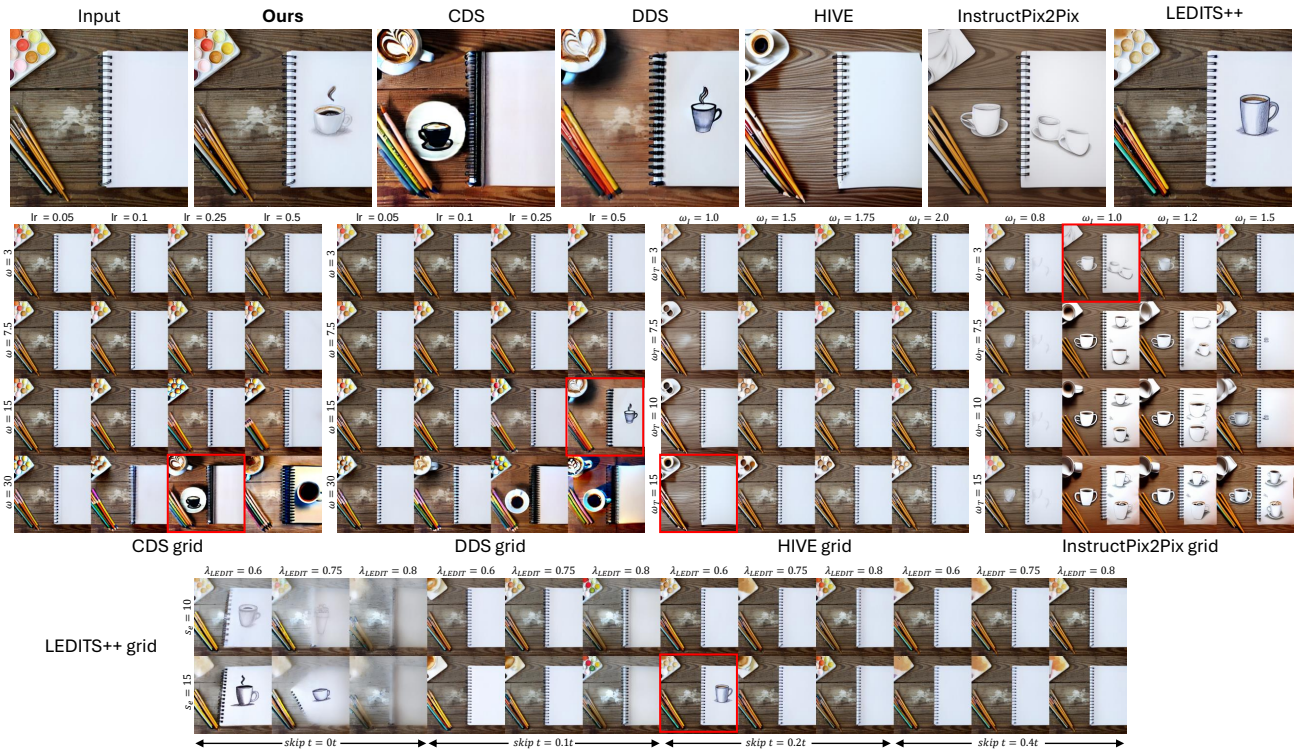


Figure 28. Qualitative results on various in-the-wild editing tasks. We compare the best results (prioritizing object appearance) from state-of-the-art methods, optimized through hyperparameter tuning (see Appendix 12), with our results all generated using a single configuration. For each input image, we show results from 3 different random seeds.

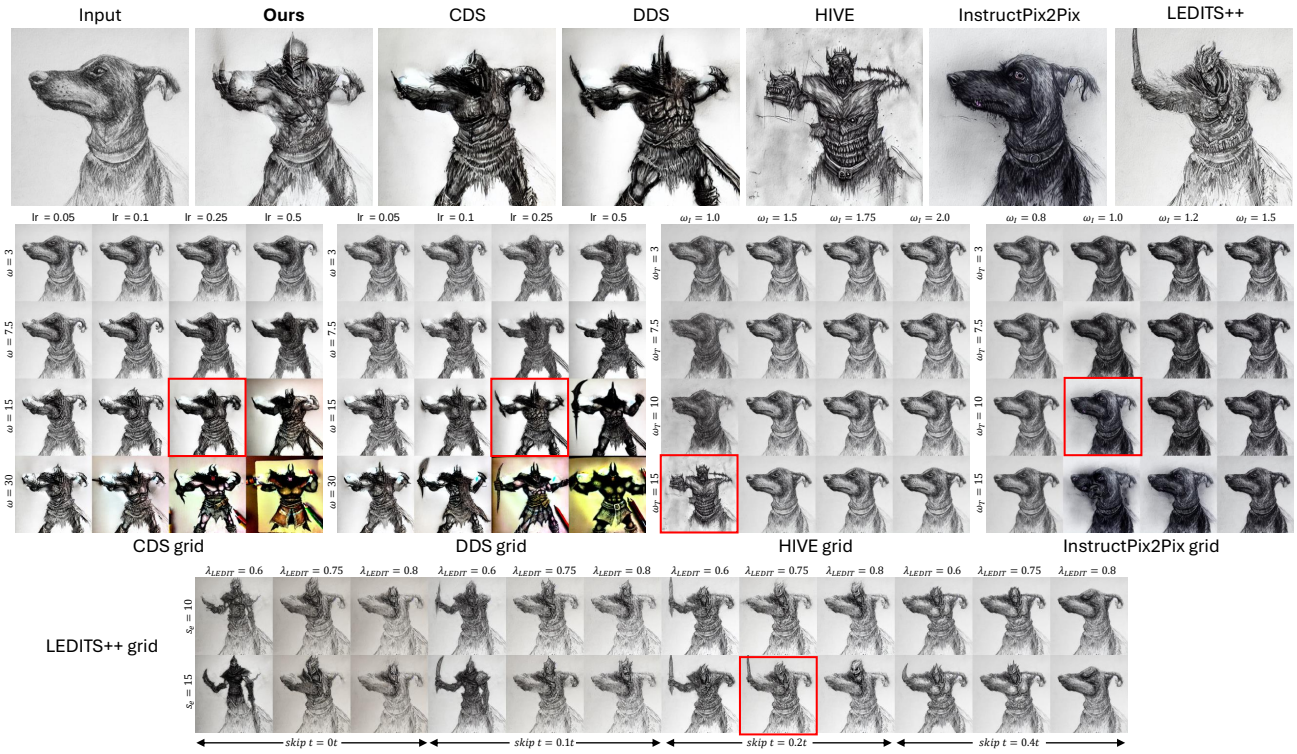


(a) Prompt: a Chinese lion wearing a sunglasses

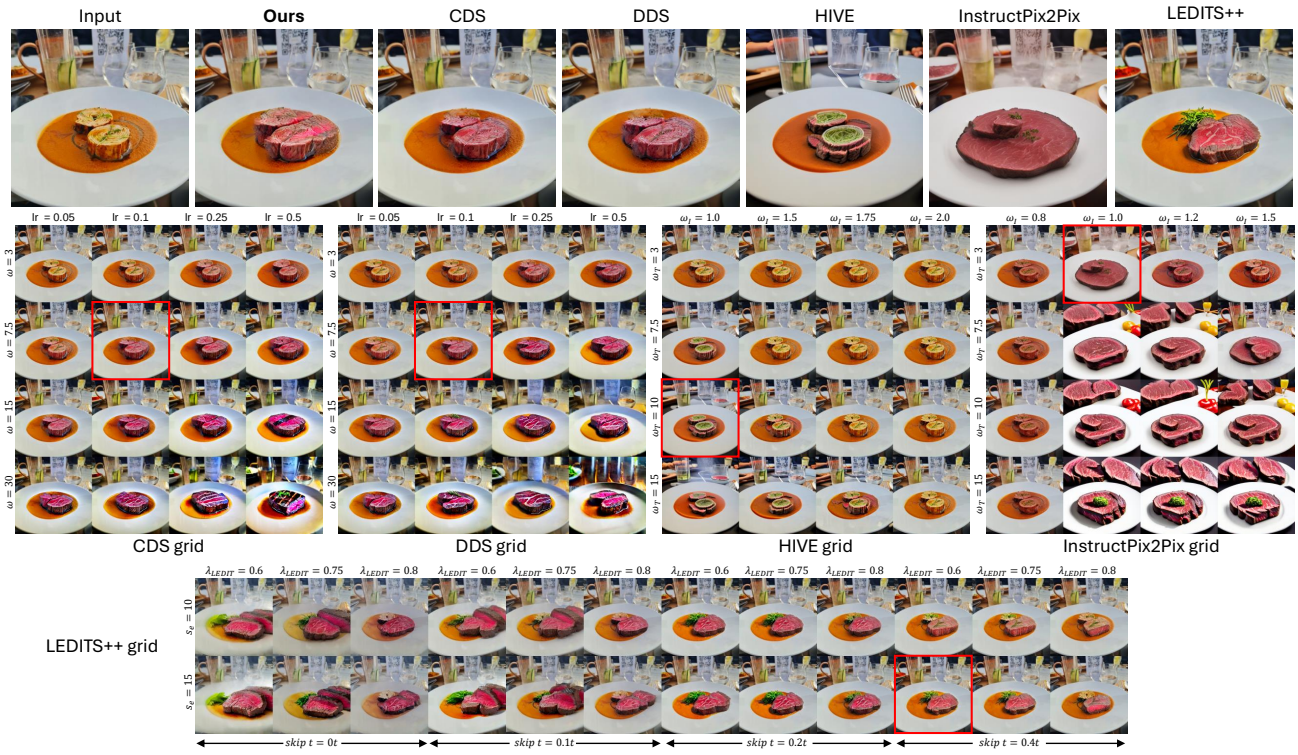


(b) Prompt: a notebook with a drawing of a coffee cup

Figure 29. Hypertuning grids on various in-the-wild editing tasks. We compare the best results (prioritizing object appearance) from state-of-the-art methods, optimized through hyperparameter tuning (highlighted by red boxes), with our results all generated using a single configuration. When several configurations perform equally, we choose the one that best preserves the background.

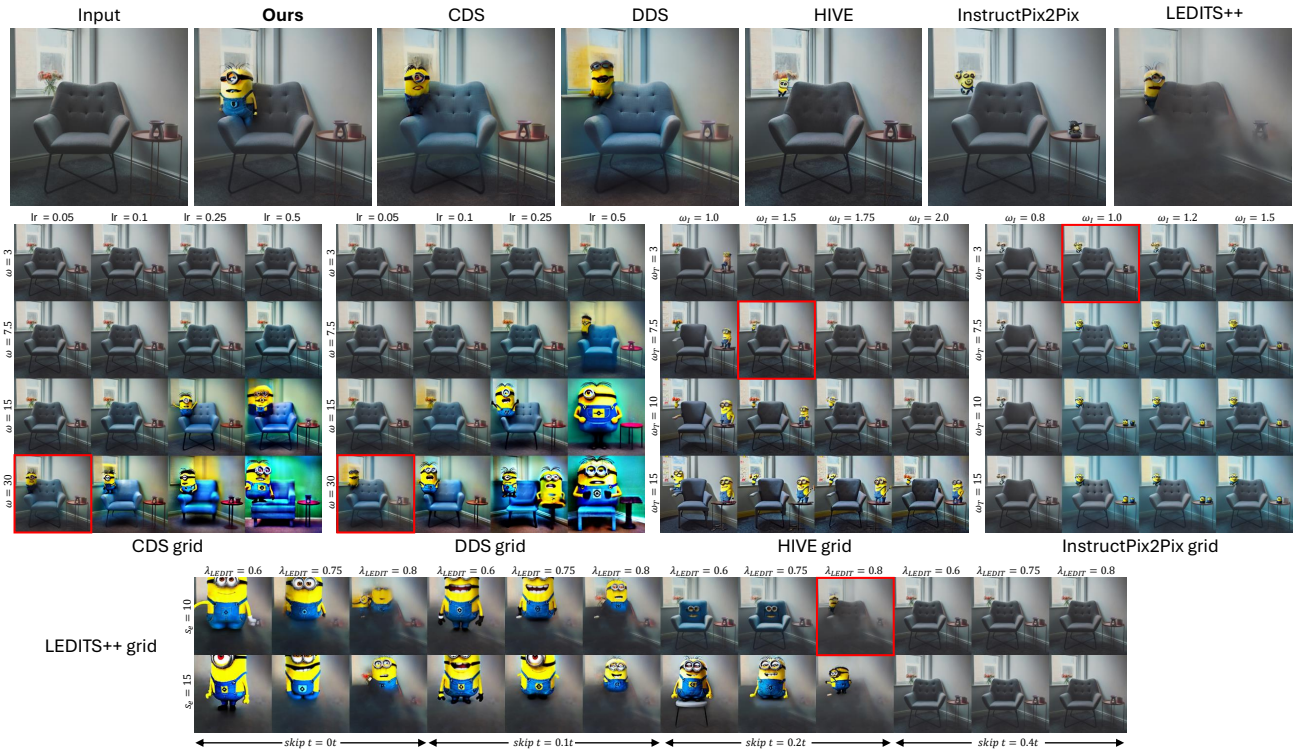


(a) Prompt: a drawing of ("a dog" → "a boss in dark soul")

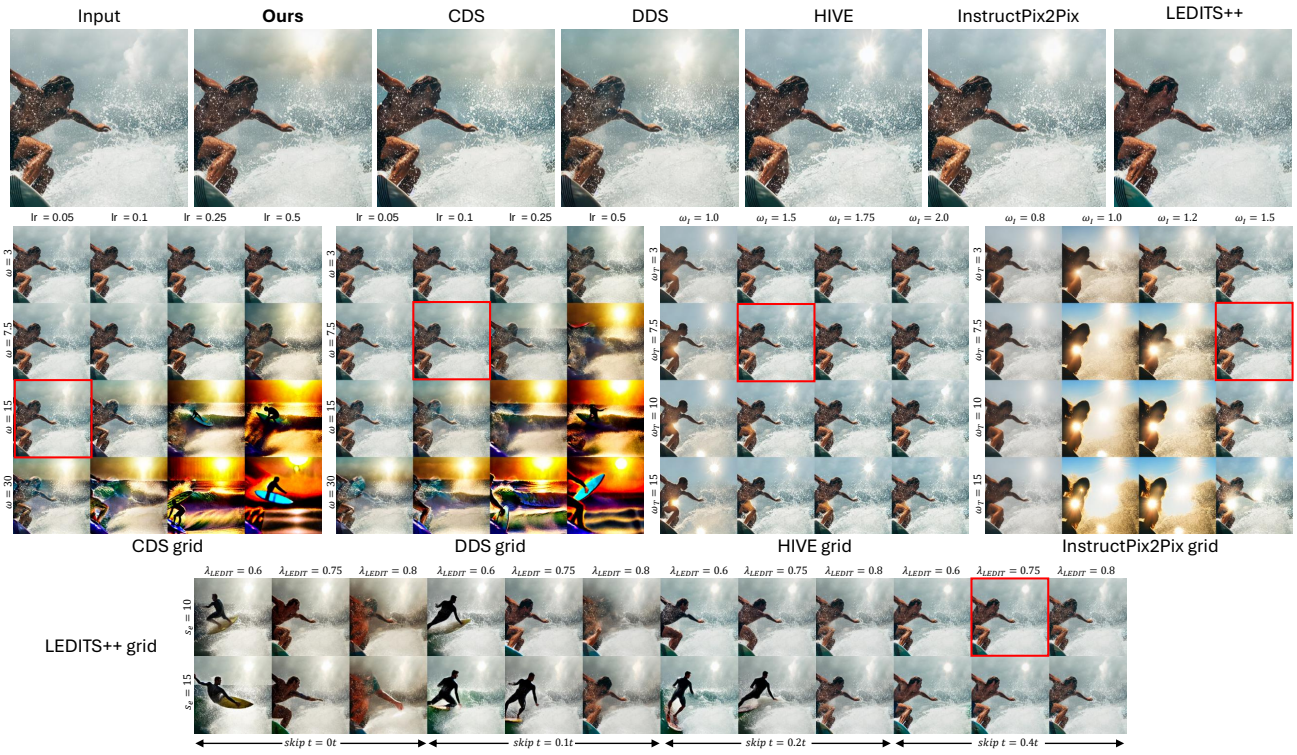


(b) Prompt: ("food" → "wagyu steak")

Figure 30. Hypertuning grids on various in-the-wild editing tasks. We compare the best results (prioritizing object appearance) from state-of-the-art methods, optimized through hyperparameter tuning (highlighted by red boxes), with our results all generated using a single configuration. When several configurations perform equally, we choose the one that best preserves the background.

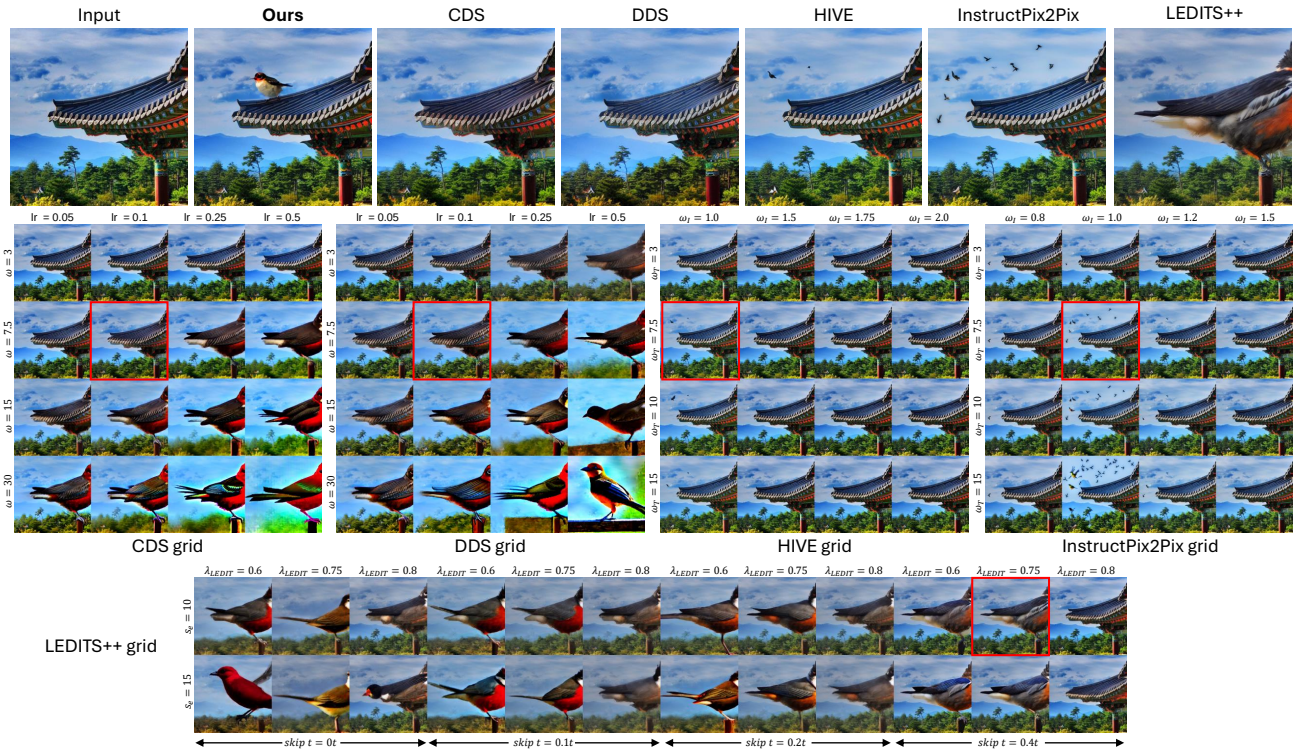


(a) Prompt: a chair and a minion



(b) Prompt: a man surfing and a sun

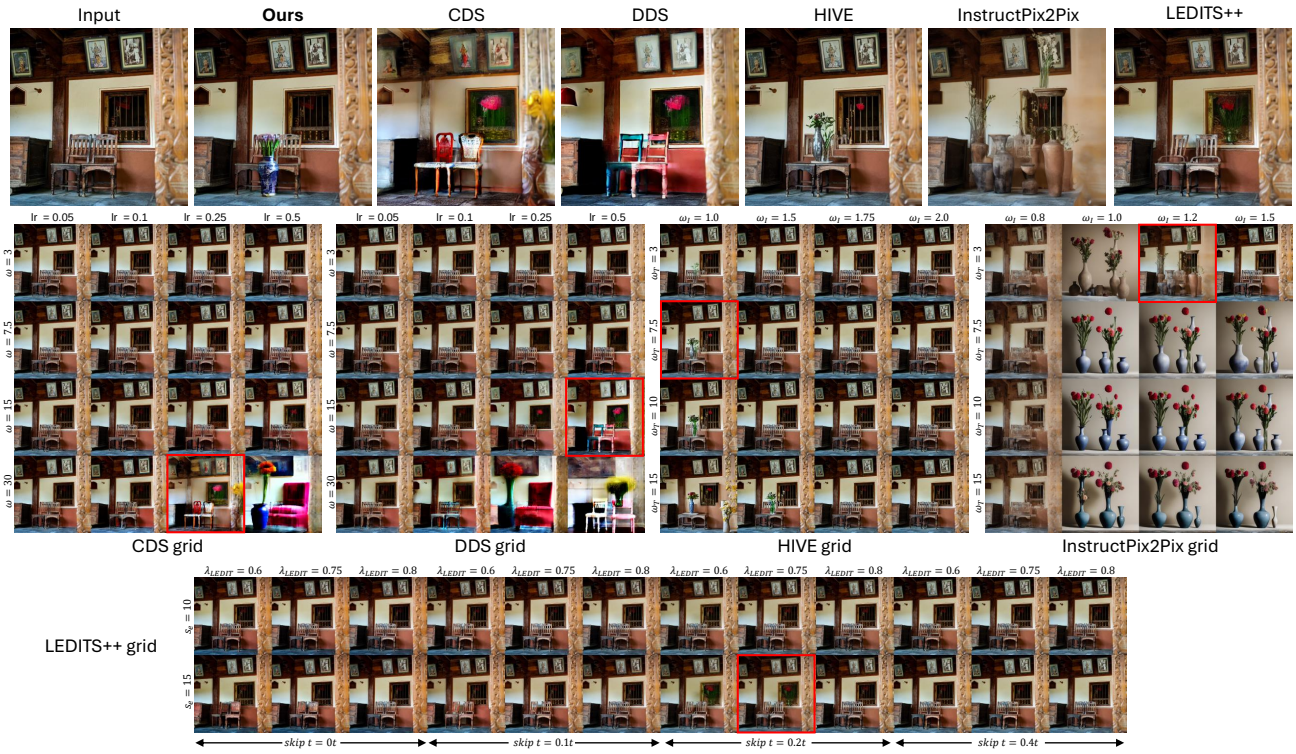
Figure 31. Hypertuning grids on various in-the-wild editing tasks. We compare the best results (prioritizing object appearance) from state-of-the-art methods, optimized through hyperparameter tuning (highlighted by red boxes), with our results all generated using a single configuration. When several configurations perform equally, we choose the one that best preserves the background.



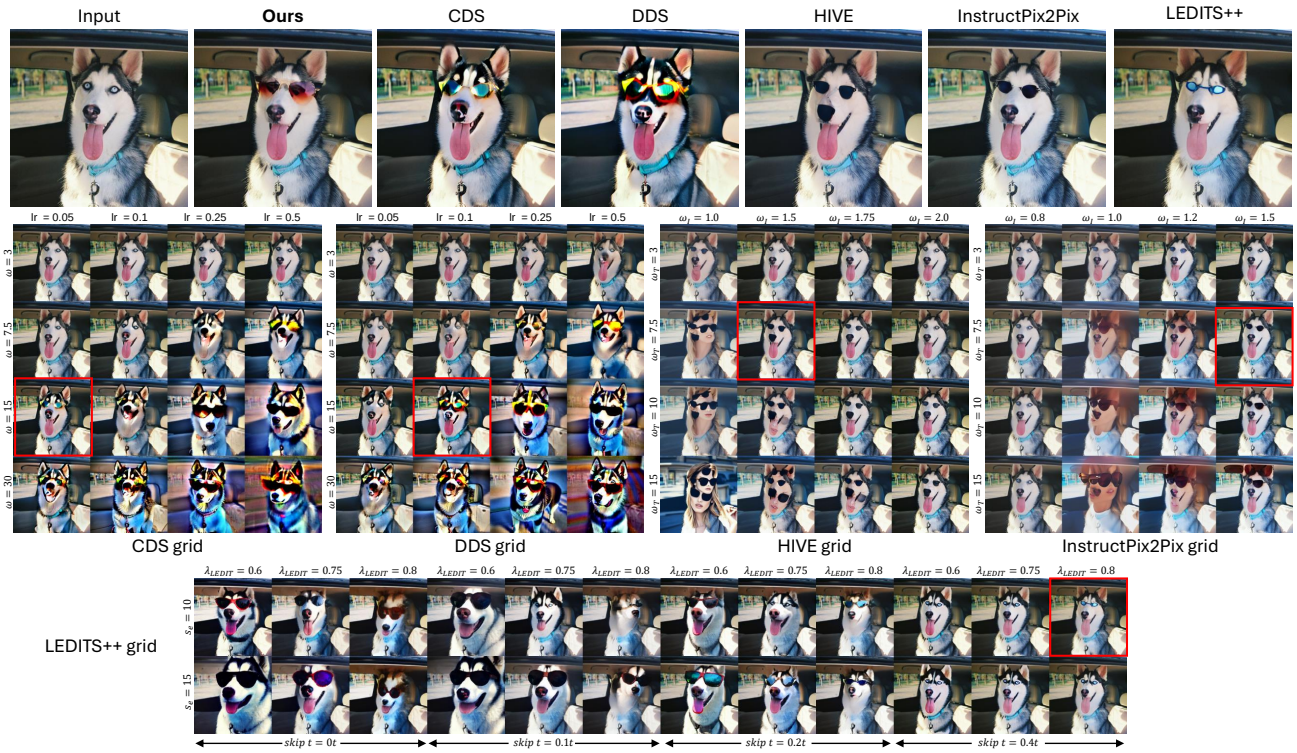
(a) Prompt: a bird on a roof

(b) Prompt: a painting of a dog wearing a blindfold over its eyes

Figure 32. Hypertuning grids on various in-the-wild editing tasks. We compare the best results (prioritizing object appearance) from state-of-the-art methods, optimized through hyperparameter tuning (highlighted by red boxes), with our results all generated using a single configuration. When several configurations perform equally, we choose the one that best preserves the background.

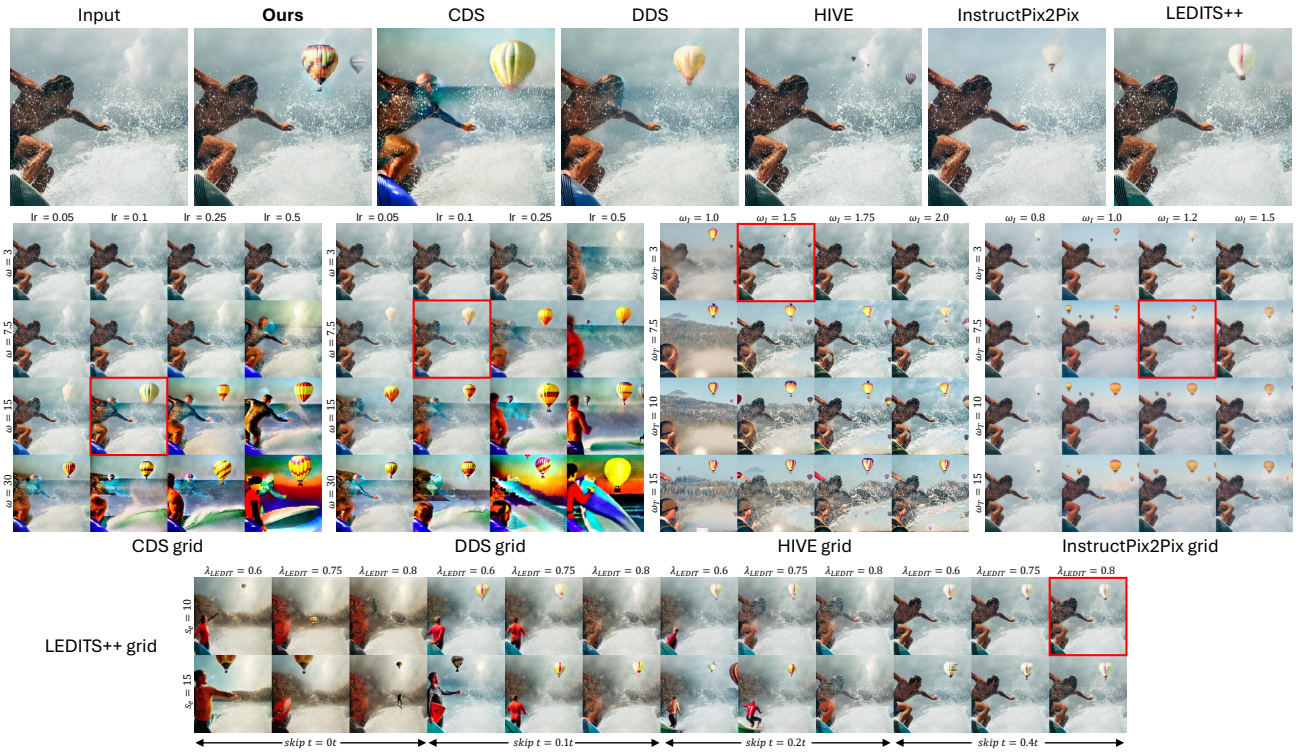


(a) Prompt: two chairs and a vase



(b) Prompt: a siberian husky wearing stylish sunglasses

Figure 33. Hypertuning grids on various in-the-wild editing tasks. We compare the best results (prioritizing object appearance) from state-of-the-art methods, optimized through hyperparameter tuning (highlighted by red boxes), with our results all generated using a single configuration. When several configurations perform equally, we choose the one that best preserves the background.

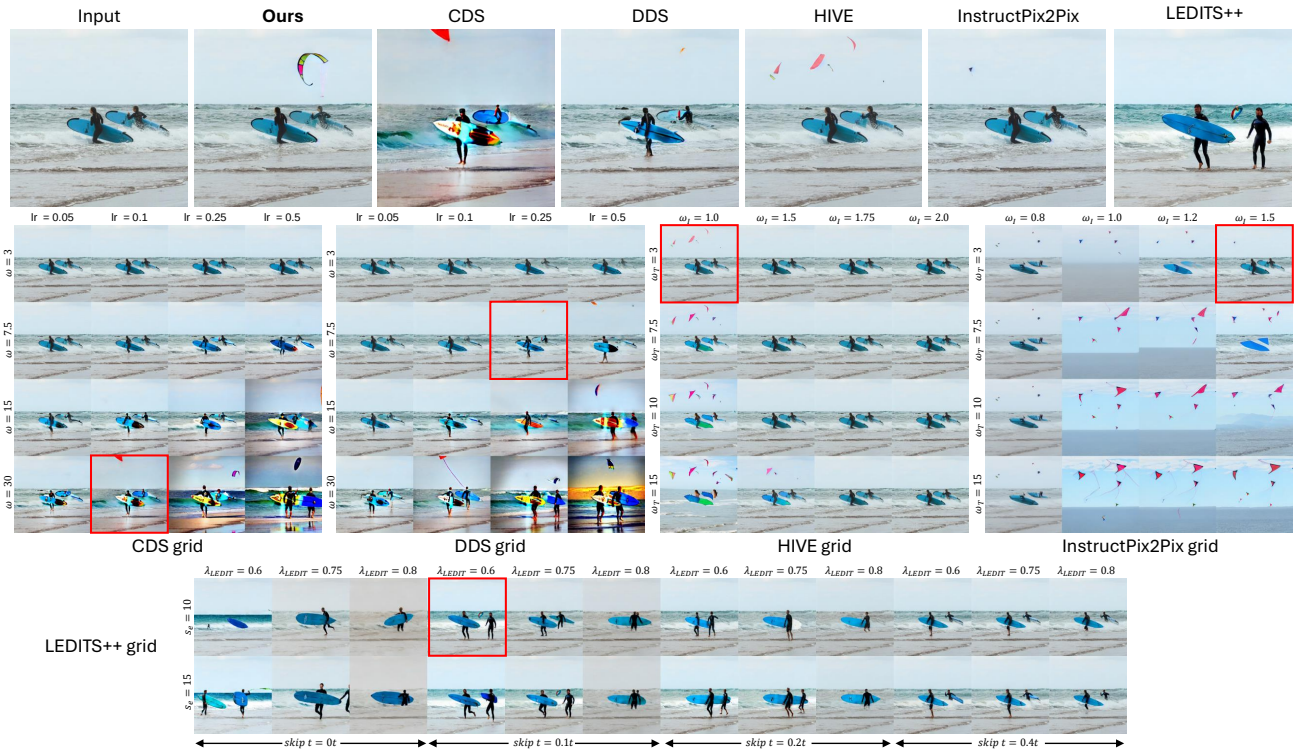


(a) Prompt: a man surfing with a hot air balloon at the background

(b) Prompt: a glass of lemon tea with a straw

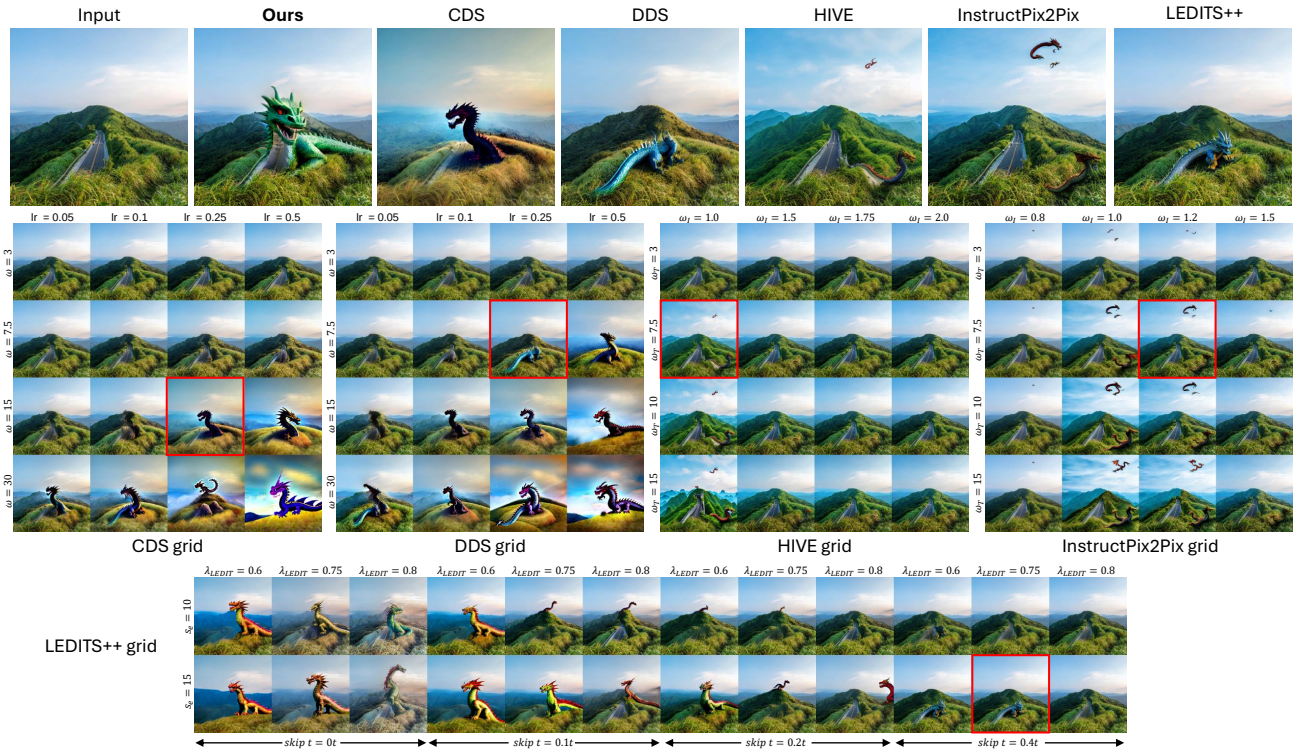
Figure 34. Hypertuning grids on various in-the-wild editing tasks. We compare the best results (prioritizing object appearance) from state-of-the-art methods, optimized through hyperparameter tuning (highlighted by red boxes), with our results all generated using a single configuration. When several configurations perform equally, we choose the one that best preserves the background.

(a) Prompt: a snowman with a scarf

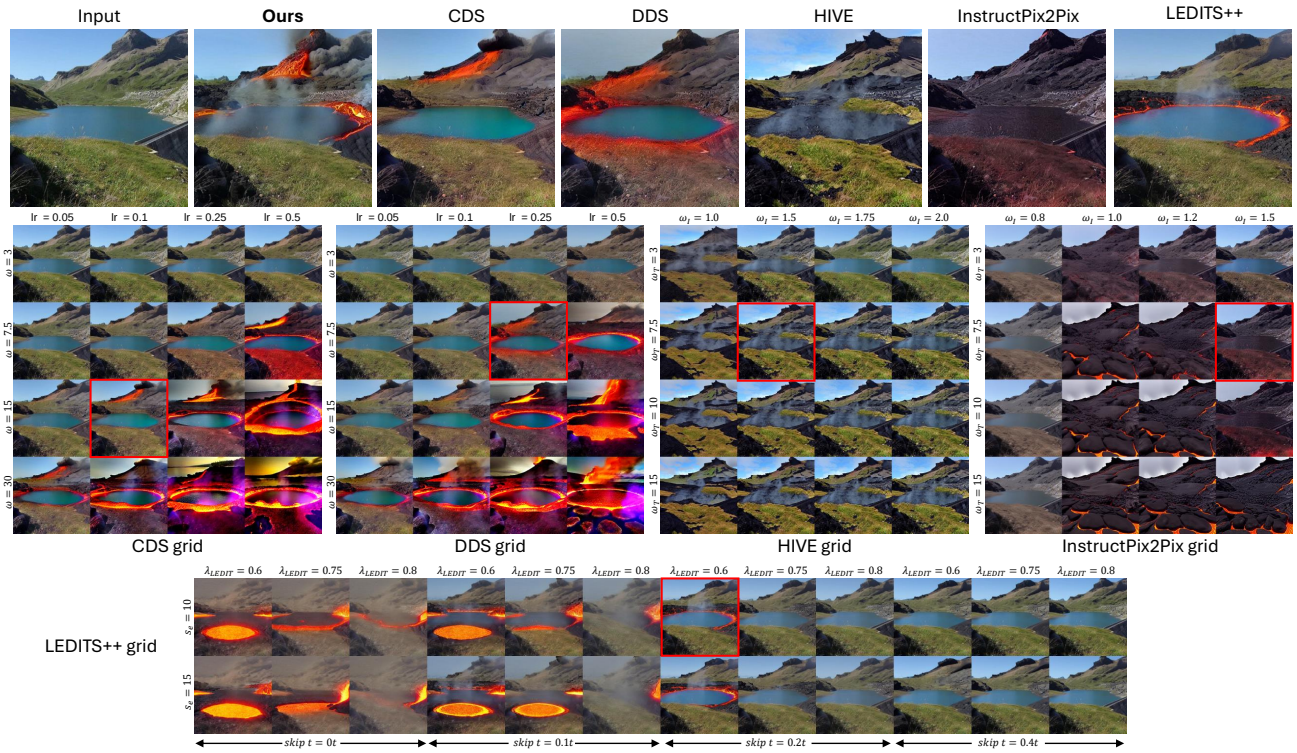


(b) Prompt: two surfers carrying bright blue boards wade into the ocean and a kite in the sky

Figure 35. Hypertuning grids on various in-the-wild editing tasks. We compare the best results (prioritizing object appearance) from state-of-the-art methods, optimized through hyperparameter tuning (highlighted by red boxes), with our results all generated using a single configuration. When several configurations perform equally, we choose the one that best preserves the background.



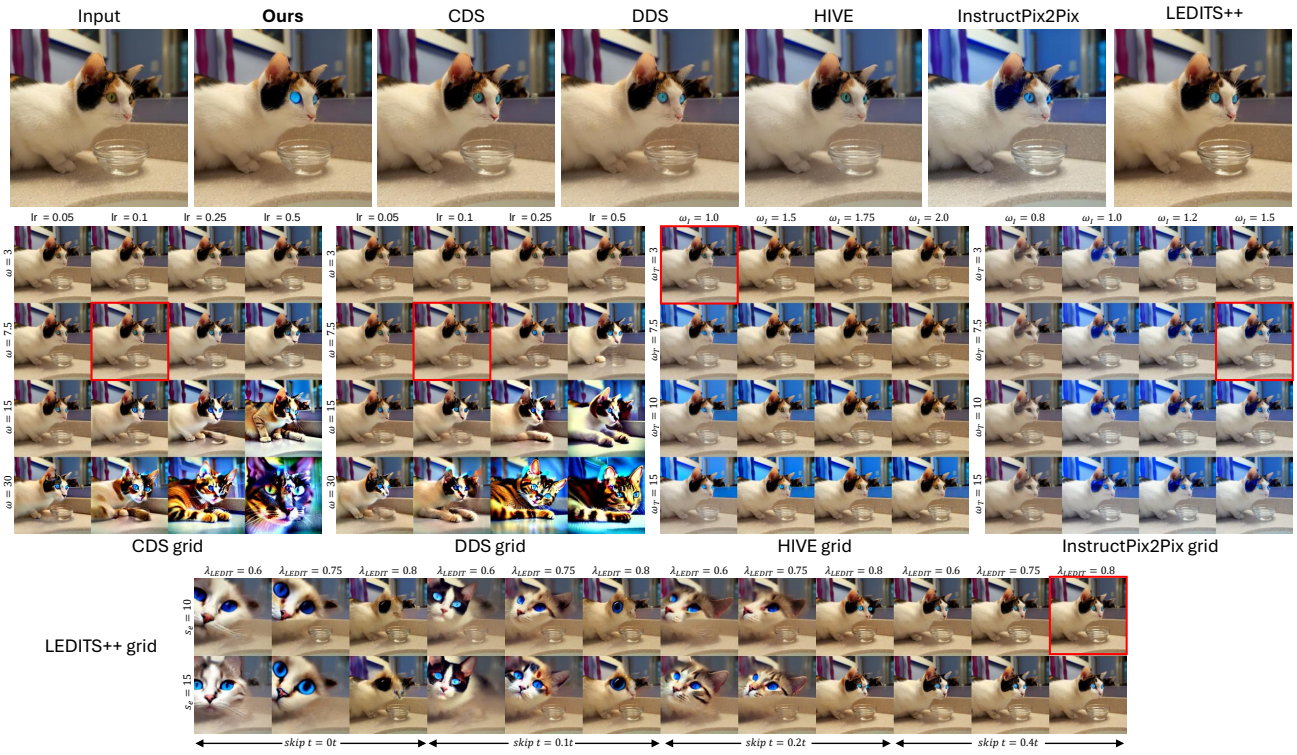
(a) Prompt: a dragon on a hill



(b) Prompt: a lake of lava

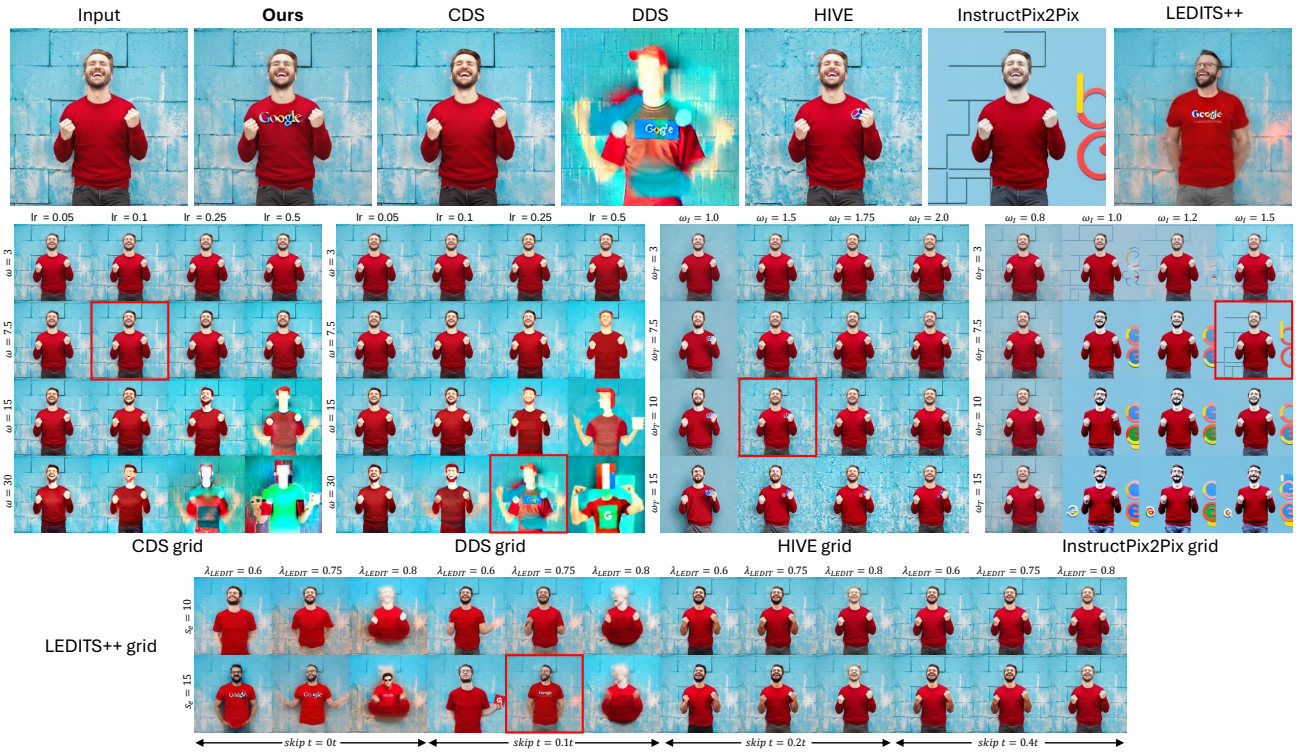
Figure 36. Hypertuning grids on various in-the-wild editing tasks. We compare the best results (prioritizing object appearance) from state-of-the-art methods, optimized through hyperparameter tuning (highlighted by red boxes), with our results all generated using a single configuration. When several configurations perform equally, we choose the one that best preserves the background.

(a) Prompt: seafood pizza

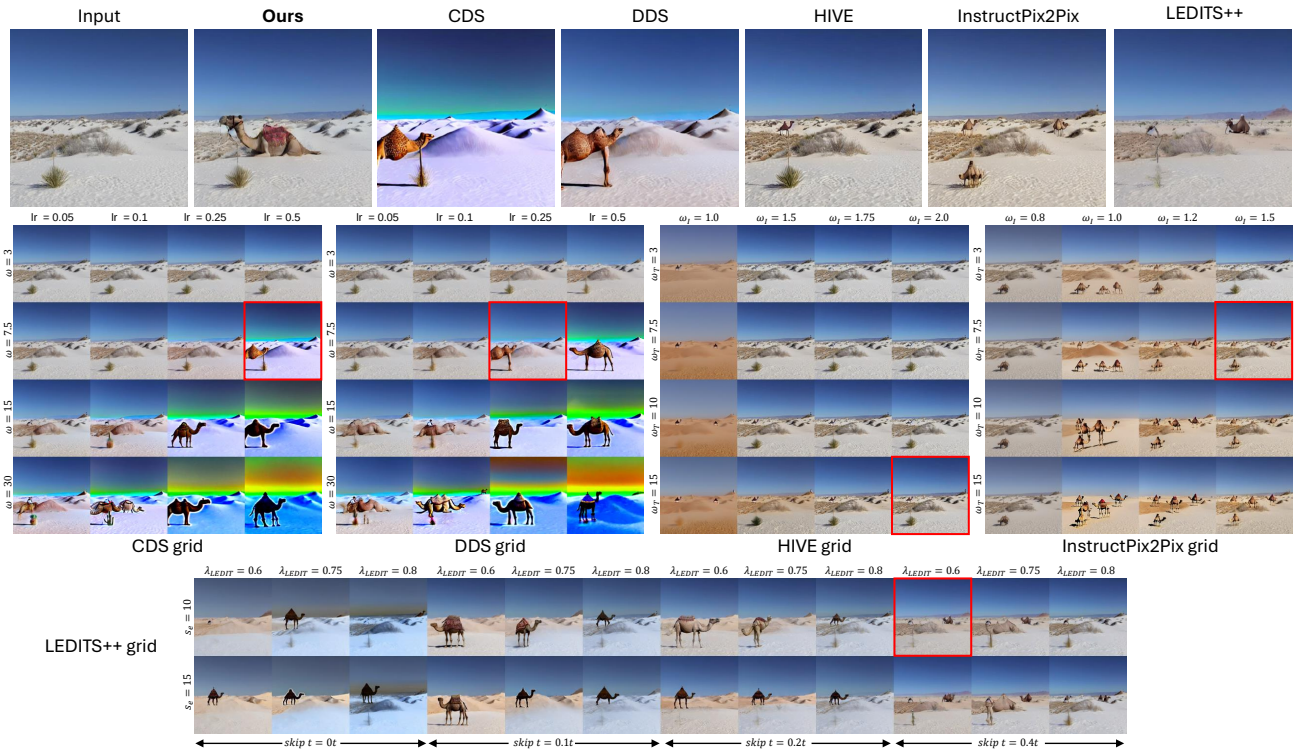


(b) Prompt: a cat with blue eyes

Figure 37. Hypertuning grids on various in-the-wild editing tasks. We compare the best results (prioritizing object appearance) from state-of-the-art methods, optimized through hyperparameter tuning (highlighted by red boxes), with our results all generated using a single configuration. When several configurations perform equally, we choose the one that best preserves the background.

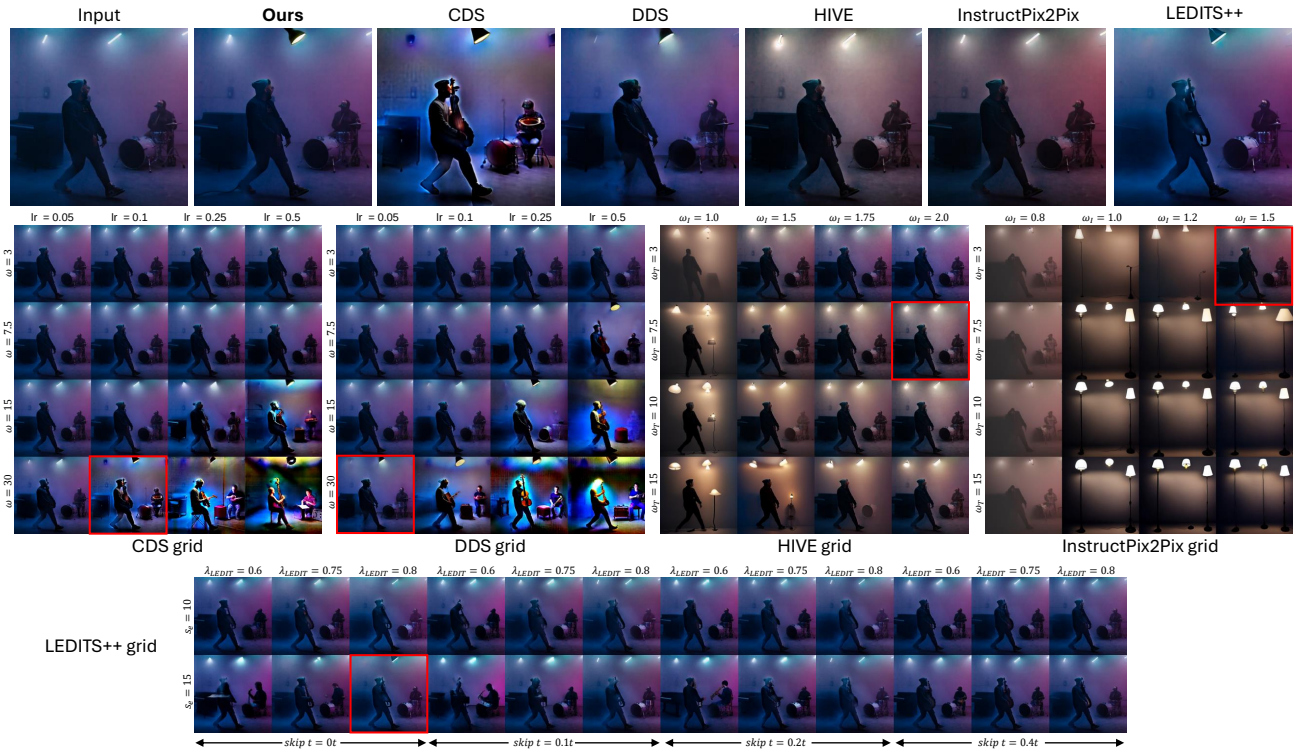


(a) Prompt: a man wearing a red t-shirt with a google logo

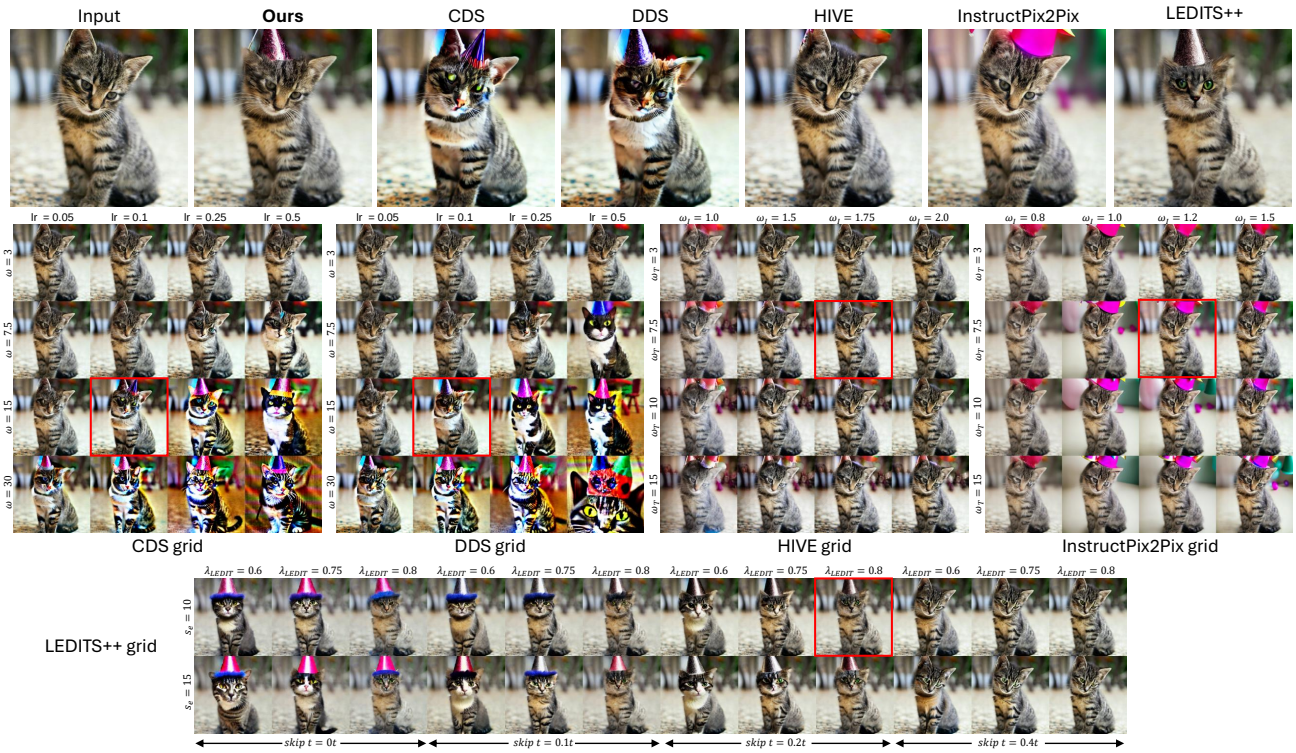


(b) Prompt: a desert with a camel

Figure 38. Hypertuning grids on various in-the-wild editing tasks. We compare the best results (prioritizing object appearance) from state-of-the-art methods, optimized through hyperparameter tuning (highlighted by red boxes), with our results all generated using a single configuration. When several configurations perform equally, we choose the one that best preserves the background.

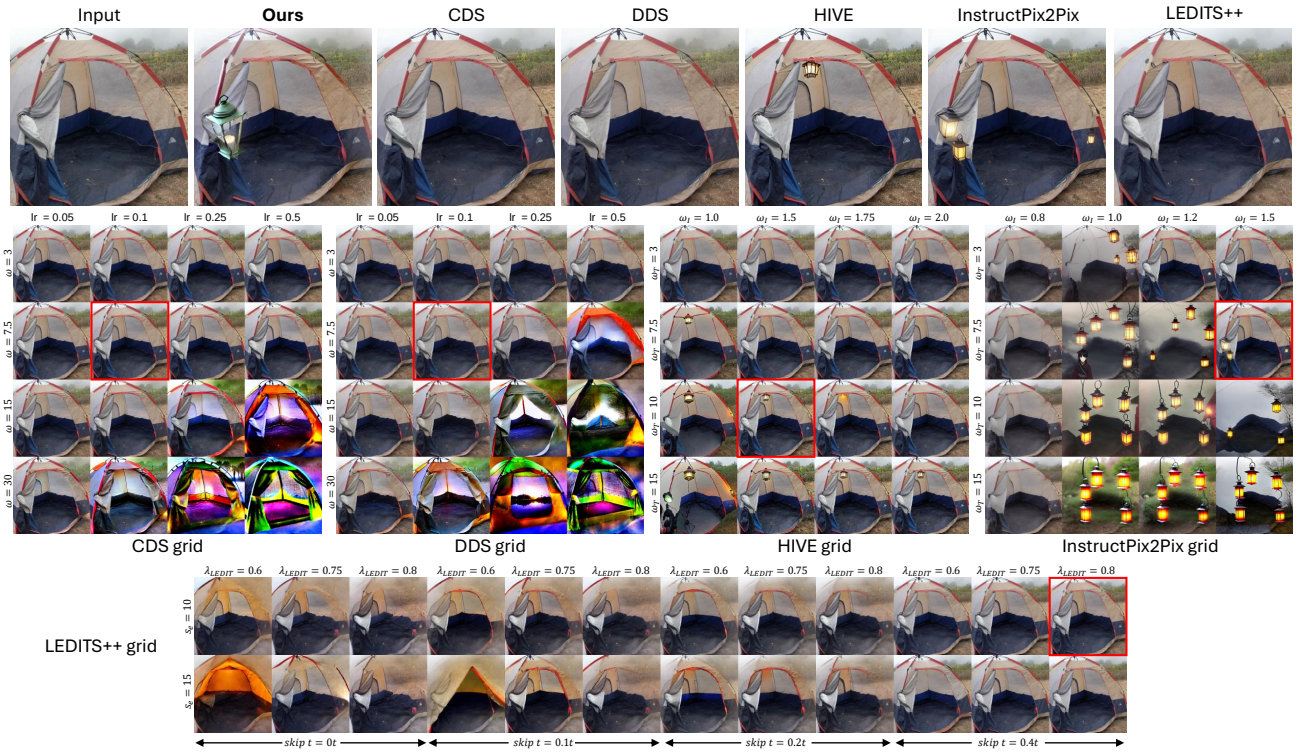


(a) Prompt: musicians playing instruments inside a room with a lamp

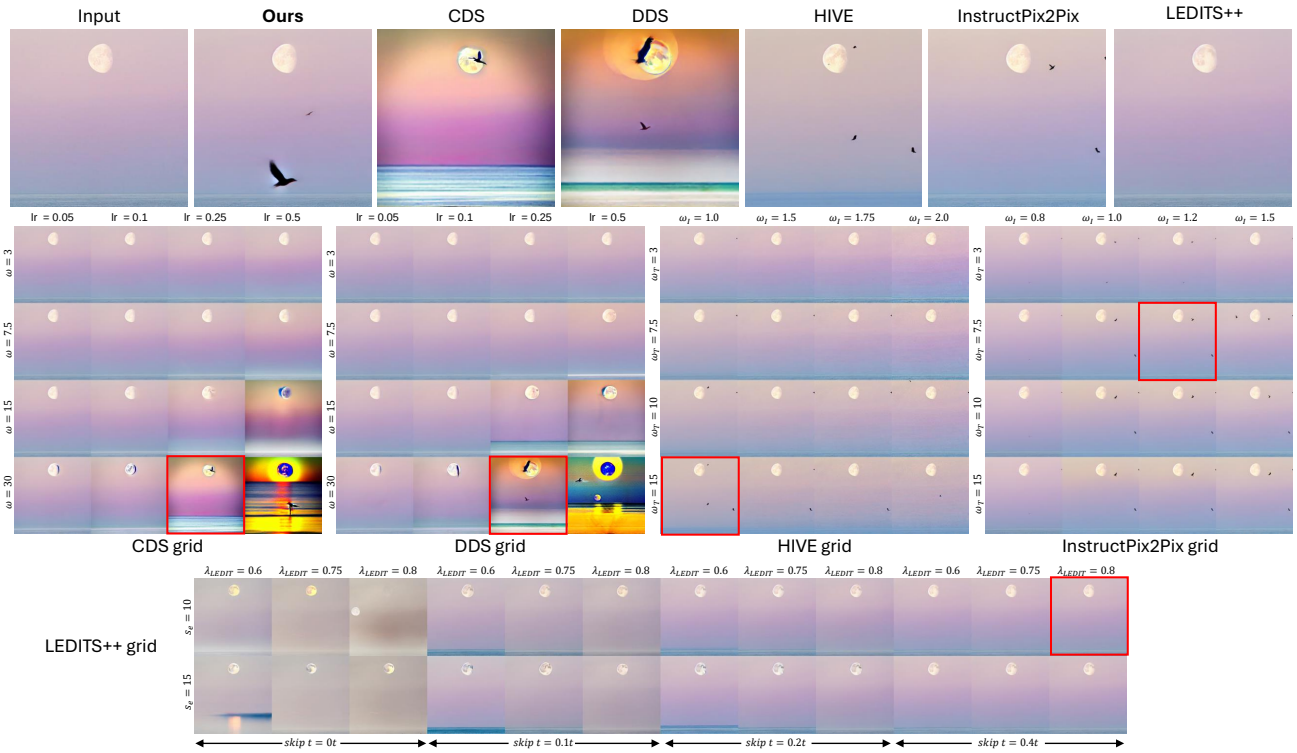


(b) Prompt: a cat wearing a party hat

Figure 39. Hypertuning grids on various in-the-wild editing tasks. We compare the best results (prioritizing object appearance) from state-of-the-art methods, optimized through hyperparameter tuning (highlighted by red boxes), with our results all generated using a single configuration. When several configurations perform equally, we choose the one that best preserves the background.

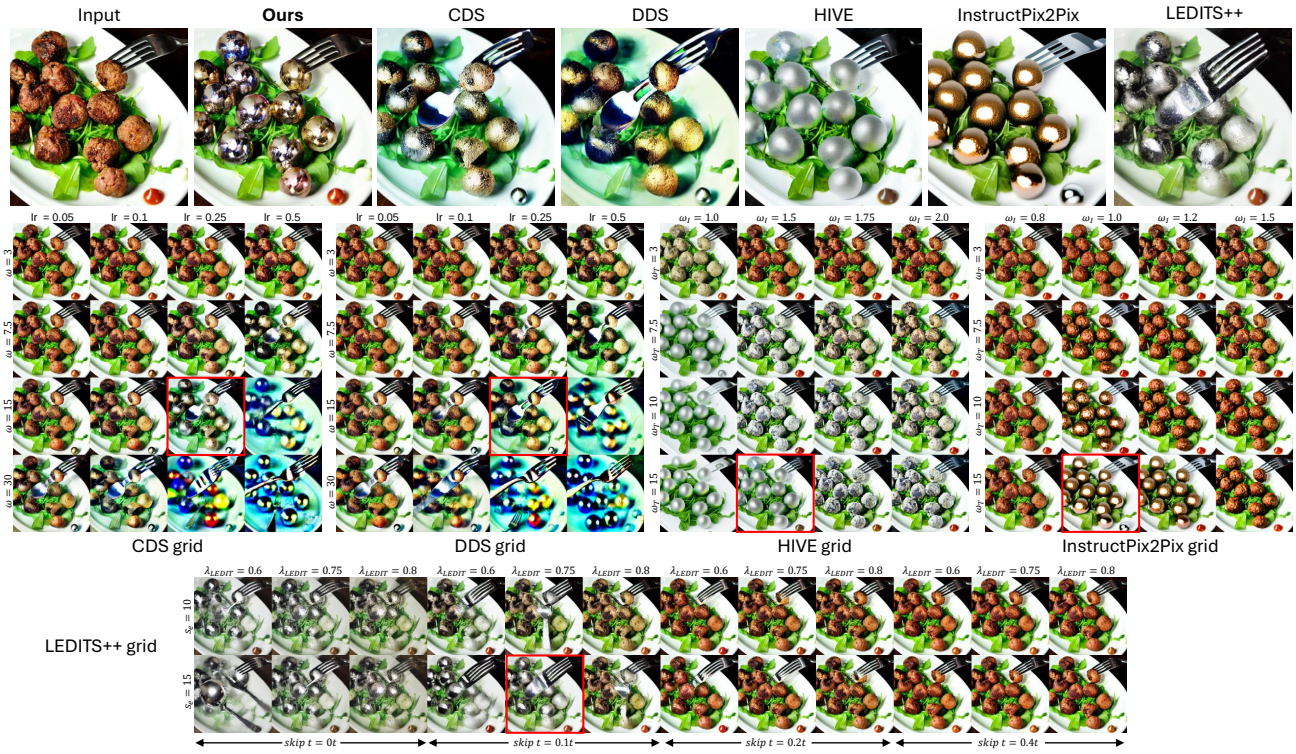


(a) Prompt: a tent with a lantern

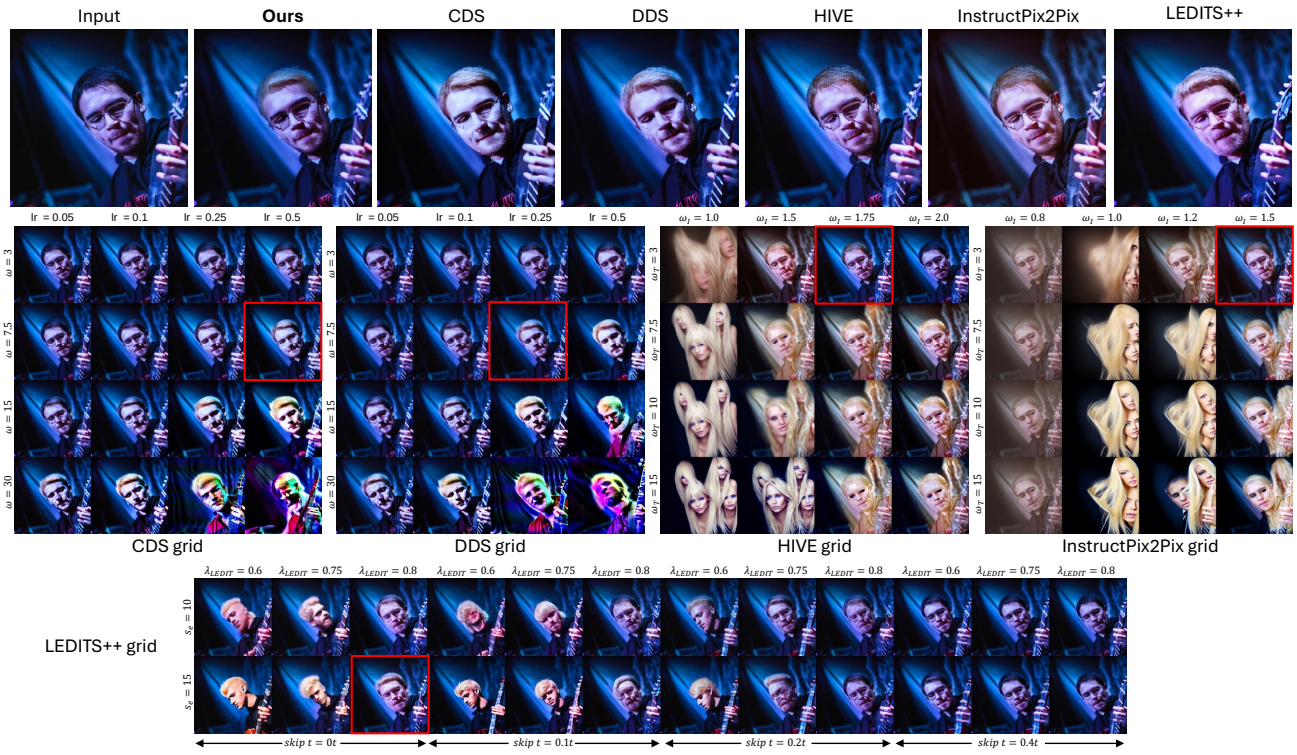


(b) Prompt: an ocean with a clear blue sky and a full moon and a bird

Figure 40. Hypertuning grids on various in-the-wild editing tasks. We compare the best results (prioritizing object appearance) from state-of-the-art methods, optimized through hyperparameter tuning (highlighted by red boxes), with our results all generated using a single configuration. When several configurations perform equally, we choose the one that best preserves the background.

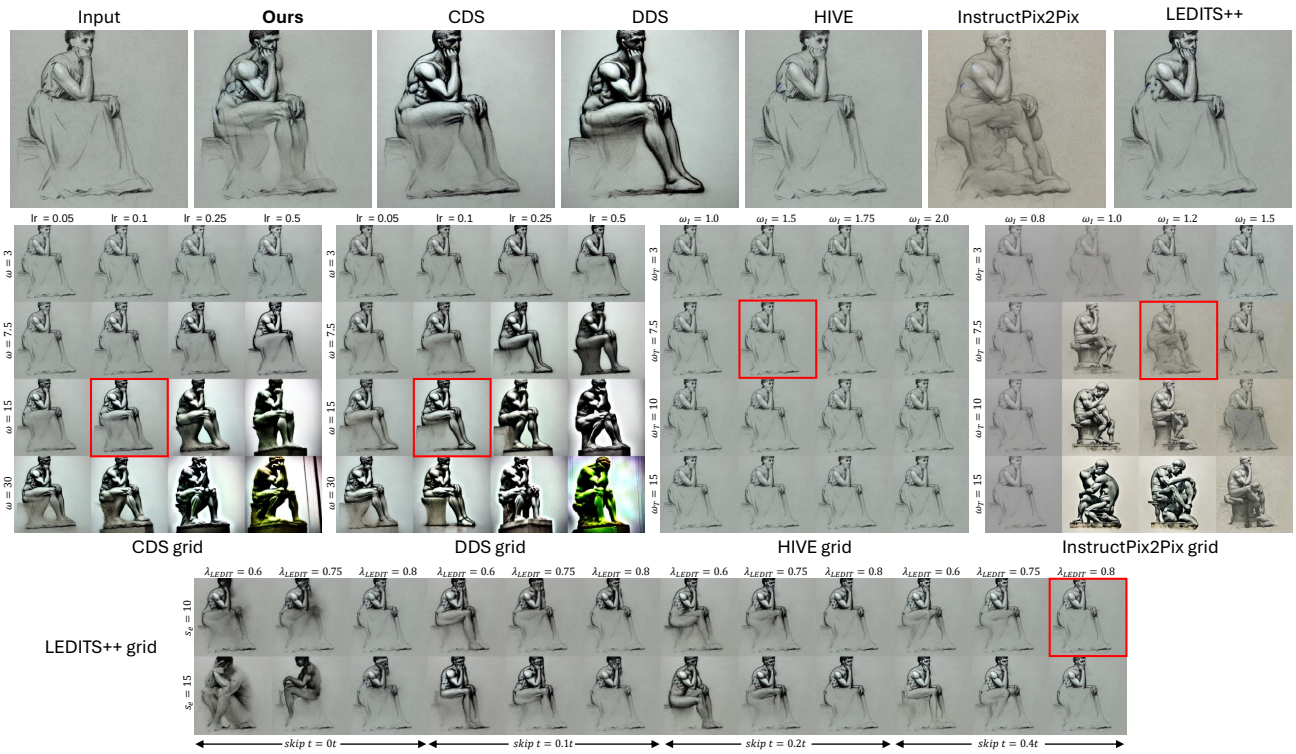


(a) Prompt: ("meatballs" → "chrome balls")



(b) Prompt: a musician with blonde hair

Figure 41. Hypertuning grids on various in-the-wild editing tasks. We compare the best results (prioritizing object appearance) from state-of-the-art methods, optimized through hyperparameter tuning (highlighted by red boxes), with our results all generated using a single configuration. When several configurations perform equally, we choose the one that best preserves the background.



(a) Prompt: (“ ” → “the thinker”)

Figure 42. Hypertuning grids on various in-the-wild editing tasks. We compare the best results (prioritizing object appearance) from state-of-the-art methods, optimized through hyperparameter tuning (highlighted by red boxes), with our results all generated using a single configuration. When several configurations perform equally, we choose the one that best preserves the background.