SILO: Solving Inverse Problems with Latent Operators

Supplementary Material

A. Flaws of bi-domain LDM restoration

The MMSE denoiser employed in the diffusion process of LDMs is designed to denoise latents, not images, while the observation y resides in the measurement space. Thus, the approximation of DPS,

$$\nabla_{x_t} \ln p(y|x_t) \approx -\frac{1}{\sigma_y^2} \nabla_{x_t} \|y - \mathcal{A}(\hat{x}_0^t)\|_2^2, \tag{18}$$

is no longer applicable. As mentioned in Sec. 4, one possible solution is to decode the latents to the pixel (measurement) space during the restoration process, allowing, for example, the use of

$$\nabla_{z_t} \ln p(y|z_t) \approx -\frac{1}{\sigma_y^2} \nabla_{z_t} \|y - \mathcal{A}(\mathcal{D}(\hat{z}_0^t))\|_2^2. \tag{19}$$

We categorize such solutions as part of the "bi-domain" family, as they compute gradients in both the pixel and latent domains. To the best of our knowledge, all existing methods leveraging LDMs for inverse problems (apart from SILO) fall within this category. In this section, we demonstrate the underlying flaws in solutions belonging to this family.

During the training process, the decoder is exposed only to clean latents, z_0. Moreover, we do not require that its Jacobian be informative or well-behaved. This leads to two main problems when using the decoder during restoration. First, the decoder is applied to out-of-distribution (OOD) latents compared to its training data, as the latents in the restoration process, $\hat{z}_0^t$, are MMSE denoised, thus coming from a different distribution. Second, differentiating through the decoder transforms the score-likelihood gradient from the pixel to the latent space, introducing a possibly uninformative Jacobian to the backpropagation process. These two problems are tightly related to each other. An OOD latent leads to an unpredictable Jacobian, further destabilizing the differentiation process of the likelihood.

To demonstrate these problems, we focus on LDPS [45] and PSLD [39] as representatives of the bi-domain family. LDPS (Eq. (9)) is the starting ground for all other methods in this family, and PSLD (Eq. (11)) is an extension of it. As we see in Figs. 14, 15 and 17 to 21, the reconstructions of LDPS and its derivatives often suffer from the presence of "blob" artifacts and noise patterns. To investigate this matter, we record the gradients

$$\nabla_{\hat{z}_0^t} \|y - \mathcal{A}(\mathcal{D}(\hat{z}_0^t))\|_2^2 \tag{20}$$

during the restoration process of LDPS and PSLD. Note the subtle difference between Eq. (20) and Eq. (19); the two differ only in the Jacobian of the denoiser, which is irrelevant to our analysis as it is independent of the decoder.

In Fig. 9, we see that from the early stages of the restoration process, these gradients contain a patch that causes the latent to change in a way that does not correlate with the measurement. This effect prevails throughout the diffusion process, leading to a closely-related defect in the resulting image. Qualitatively, from $t \approx 500$ onward, the gradients exhibit a noise pattern that dominates the signal, leading to a similar noise pattern in the reconstructed image. This behavior is inherent to the use of the decoder in the way practiced by LDPS and PSLD. Adding projections and a regularization could improve the restoration, but only to some extent. For example, PSLD attempts to mitigate this by guiding the latents to areas the encoder-decoder handles better, yet similar artifacts are still presented as seen in Fig. 9. Another example is ReSample, which performs likelihood optimization in pixel space followed by an encoding step in parts of the restoration process to avoid those blob artifacts (Appendix B of [45]).

In summary, differentiating through the decoder might corrupt the information required for faithful reconstruction. This motivates us to avoid using the decoder altogether during the restoration process.

B. Discussion on plug-and-play methods

As explained in Section 1 and in the literature [37, 53, 58, 61], *plug-and-play* (P&P) methods treat separately the likelihood term from the prior term. This separation results in two appealing properties: (i) A P&P method can leverage new and improved priors with the same likelihood term. (ii) A P&P method can solve different degradations by adapting appropriate likelihood terms. Notably, a method is not bound to use an analytic prior or likelihood term and may learn one or both of them. For example, RED [37] demonstrate restoration with both hand-crafted priors and learned ones. Moreover, LGD [47] and DEFT [8] both demonstrate learned likelihood terms.

Following the above discussion, it is natural to call SILO a P&P method. Specifically, we have demonstrated in the main paper how SILO can use different priors with no additional training of the degradation operator, as expected from a P&P method. The unique aspect of SILO is the fact that we adapt the likelihood term to the space of the prior. This stand in contrast with other P&P methods that translate back-and-forth between the likelihood and prior spaces.

It is important to note the fundamental differences between *end-to-end* (E2E) supervised methods and P&P

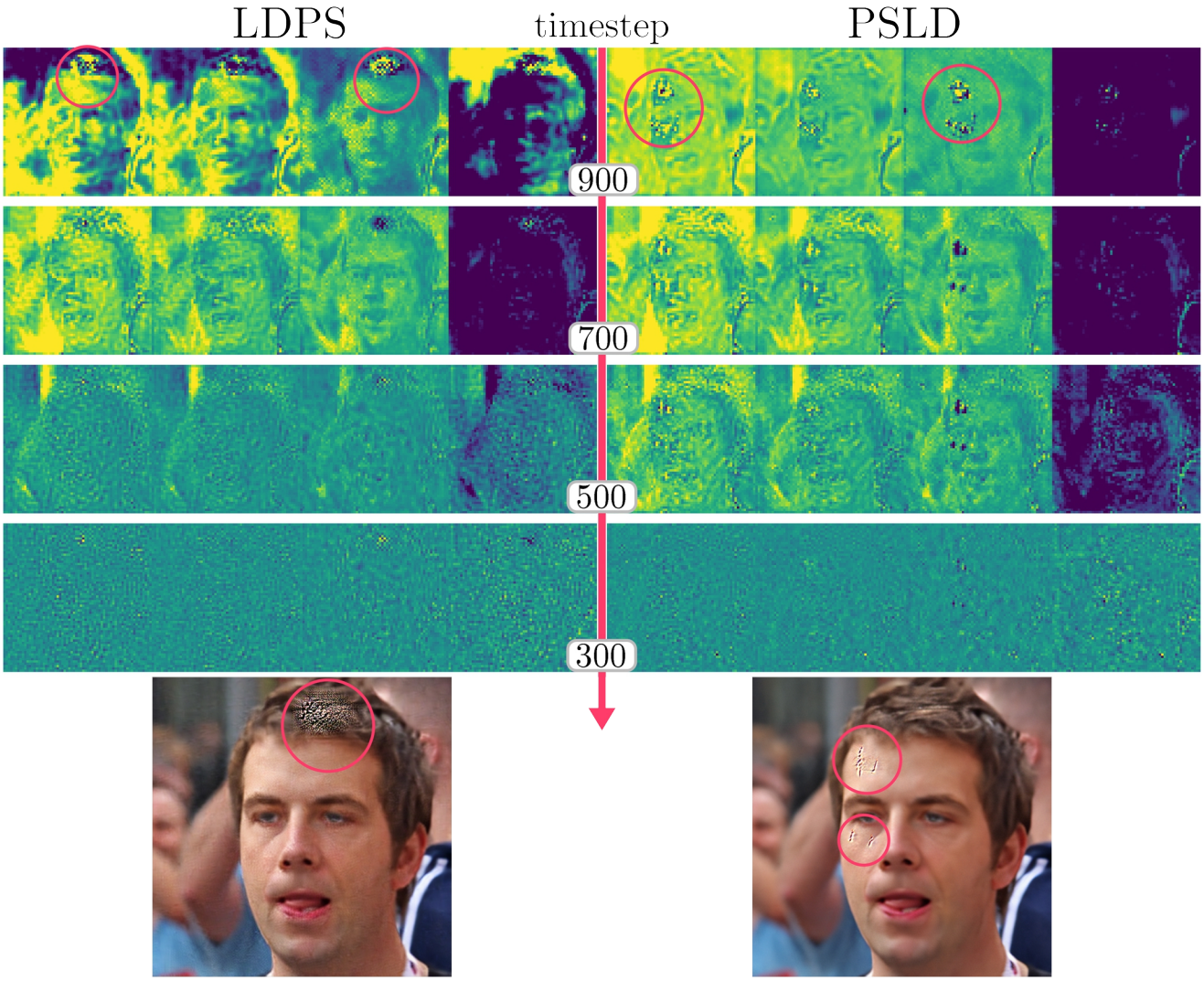

Figure 9. **The decoder's artifacts.** We present the likelihood's gradient through the decoder (Eq. (20)) at different timesteps t. Each gradient is presented as four single-channel images, clipped to $[-3, 3]$ and scaled by 10^2 for better visualization. At the bottom, we present the resulting image of the restoration process. The decoder's Jacobian introduces "blob" artifacts and noise patterns that are present in the final restorations.

methods, and SILO in particular. E2E methods has several defining aspects: (i) An E2E method trains a model to solve a particular task. (ii) The prior of an E2E method is embedded in the model itself, coupled to the task at hand. (iii) Such methods tend to train large models for extended periods of time. All of those aspects stand in contrast with P&P methods which are adaptable to different priors and tasks. Crucially, while E2E methods learn to restore images, SILO only learns to predict how likely a given image is the source of the measurements in hand. In other words, E2E methods inherently solves a generative task while the *learned* part in SILO is discriminative in nature.

C. Additional ablations

C.1. Diverse reconstructions using SILO

Similar to DPS [5], SILO is a stochastic solver of inverse problems. To demonstrate the effect of this stochasticity, we present in Fig. 10 reconstruction examples using SILO for the box inpainting task. We present multiple reconstruction per input, each is the result of a different random seed. We see large variability in the reconstructions, while keeping the consistency intact.

Figure 10. **Diverse reconstructions.** We present two measurements from FFHQ corrupted by a box mask at the top. Below each measurement are several reconstructions using SILO. The reconstructions' hyperparameters are identical over all the images except for the random seed used. We see great variability in the details reconstructed inside the missing box, all while being consistent outside of it.

C.2. Priors and text conditioning

Since the diffusion priors are text-conditioned, we explore the relationship between an informative prompt, which includes details about the clean image, and the method's ability to reconstruct it as a sharp, clean image. In Tab. 2, we compare the quality of reconstructions generated using RV-v5.1 and SD-v1.5. For each model, we use Algorithm 1 with varying prompts and classifier-free guidance (CFG) [19]. We test three types of prompts: a null prompt (an empty string), a generic prompt ("A high quality photo of a face"), and a set of high-quality (HQ) prompts, specific to each image. The HQ prompts are generated using Qwen [55], a vision-language model. Looking at Tab. 2, we observe that better diffusion models, using detailed text conditions and sampling with CFG, can improve the reconstructions' perceptual quality.

Model	CFG	Prompt	PSNR	LPIPS	FID	KID
RV	4	HQ	25.78	0.219	24.43	2.27
		generic	26.06	0.222	26.21	3.96
	1	generic	26.28	0.226	27.10	5.23
		null	26.35	0.230	27.63	5.83
SD	4	HQ	26.08	0.252	29.29	7.65
		generic	26.05	0.246	29.24	7.03
	1	generic	26.13	0.253	30.71	8.47
		null	26.18	0.263	32.55	10.3

Table 2. Comparison of using SILO on FFHQ dataset, for SR $\times 8$, when different models, CFG levels, and text-conditions are used.

C.3. Generalization capability

Following a reviewer's suggestion, we include a cross-domain generalization ablation (settings of inpainting in Tab. 6). The purpose of this ablation is to see if the trained operator is limited to its training data or can generalize to the same degradation on a different dataset. Note that the results in Tab. 8 are made using an operator that is trained on LSDIR, but tested on COCO. In the results of Tab. 3, we observe that the operator can generalize for the degradation of center box inpainting. Predictably, the best results are achieved whenever an operator is tested on images that are more closely related to its training data (*e.g.* by comparing rows 1,3 to each other, and rows 2,4 to each other). However, when testing on FFHQ, the operator that is trained on general images (LSDIR), has a comparable performance to that of an operator trained only on face images (FFHQ). This phenomenon still manifests when testing on COCO with operators that are trained on FFHQ or LSDIR, but to a lesser extent. This result is not surprising, as training on face images is more domain-specific; thus, the operator is not incentivized to generalize, unlike when learning with general images.

Test	Train	PSNR	LPIPS	FID	KID
FFHQ	FFHQ	22.23	0.151	21.04	4.32
	LSDIR	21.95	0.151	21.78	4.46
COCO	FFHQ	17.96	0.249	61.79	11.79
	LSDIR	18.30	0.221	45.59	2.15

Table 3. Ablation of the generalization ability of the operator to different datasets, Appendix C.3

D. Implementation details

D.1. Full algorithm

Our goal is to be able to generate restorations in a plug-and-play manner (for example, like in DPS) in the latent space while maintaining a simple and elegant sampling algorithm. Specifically, we wanted to migrate the entire restoration part to the latent space. In Algorithm 1, we present an abbreviated version of the full algorithm, which is given in Algorithm 2. Notice how the algorithm is essentially DPS, but is performed completely in the latent space. This comes in contrast to the naive adaptation, LDPS (Eq. (9)). To the best of our knowledge, no other plug-and-play method behaves in such way.

Algorithm 2: SILO: Reconstruction Algorithm

Data: measurement y, encoder $\mathcal{E}$, decoder $\mathcal{D}$, latent diffusion model ϵ_θ, trained degradation operator H_θ, text condition $\mathcal{C}$, consistency scale η, noise schedule $\{\beta_t\}_{t=0}^{T}$

Result: A reconstruction $\hat{x}$

$z_T \sim \mathcal{N}(0, I)$;
Encoding: $w = \text{clamp}(\mathcal{E}(y)\text{,-4,4})$;
for $t = T$ *to* 1 **do**
  $\hat{\epsilon} \leftarrow \epsilon_\theta(z_t, t, \mathcal{C})$;
  $\hat{z}_0^t \leftarrow \frac{1}{\sqrt{\bar{\alpha}_t}}\left(z_t - \sqrt{1-\bar{\alpha}_t}\hat{\epsilon}\right)$;
  $n \sim \mathcal{N}(0, I)$;
  $z'_{t-1} \leftarrow \frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t} z_t + \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t}\hat{z}_0^t + \sqrt{\frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\beta_t}\, n$;
  $z_{t-1} \leftarrow z'_{t-1} - \eta \nabla_{z_t} \|w - H_\theta(\hat{z}_0^t)\|_2$;
Decoding: $\hat{x} = \mathcal{D}(z_0)$;
return $\hat{x}$;

D.2. General details

The degradations described in Sec. 5.2 are done using the original code base[4] of DPS [5]. For JPEG, we use the publicly available implementation from kornia[5]. For the diffusion models, we use Stable Diffusion v1.5[6] (denoted as SD or SD-v1.5) and Realistic Vision v5.1[7] (denoted as RV or Rv-v5.1). In order to generate the HQ captions from Appendix C.2, we use Qwen2-VL-7B-Instruct [8].

D.3. Metrics

The metrics we use are divided into 2 groups: distortion and perception metrics. All metrics are calculated using torchmetrics (lightning.ai/docs/torchmetrics).

Perception-Distortion. The perception-distortion tradeoff [3, 15] states that there is an inherent tension between perceptual quality (how natural a reconstruction appears) and distortion (how similar a reconstruction is to the source). Therefore, it is essential to include both types of metrics in evaluations. As our goal is to achieve reconstructions with high perceptual quality, we sacrifice some distortion, particularly in terms of PSNR. As PSNR favors smooth and blurry results, it penalizes plausible and sharp restorations, such as posterior samples. Specifically, MSE-based restoration methods [4, 7, 12, 16, 17, 27, 52] indeed increase data fidelity but lack visual quality. Since LPIPS is considered a distortion metric by the definitions of Blau and Michaeli [3], SILO approaches the tradeoff bounds through SOTA LPIPS, FID, and KID metrics, advancing the FID/KID-LPIPS Pareto-frontier.

Distortion Metrics. Distortion metrics are calculated between two images. **PSNR** evaluates how close a reconstruction, $\hat{x}$, is to the original image, x, in a pixel-wise manner,

$$\text{PSNR}(x, \hat{x}) = 10\log_{10}\left(\frac{2^2}{\text{mean}\left(\|x - \hat{x}\|_2^2\right)}\right). \tag{21}$$

Since the images can have values in the range $[-1, 1]$, 2 is used as the data range to calculate the PSNR. The denominator transfers $\|x - \hat{x}\|_2^2$ to a per-pixel error via the 'mean' operation. **LPIPS** measures the perceptual similarity between two images and can be computed using different neural networks. As recommended by [60], we use AlexNet [25] for evaluation, as it is preferred over VGG [43]. Unlike prior work, which primarily reports LPIPS-VGG, we present LPIPS-Alex in Sec. 5.3 and include both LPIPS-Alex and LPIPS-VGG in Appendix G for completeness.

Perception Metrics. Perception metrics assess the divergence between the distribution of real images, p_x, and the distribution of reconstructed images, $p_{\hat{x}}$. When these distributions are close, it indicates that our algorithm approximately samples from the real image distribution. The perception measures we provide are Fréchet Inception Distance

[4] github.com/DPS2022/diffusion-posterior-sampling
[5] kornia.readthedocs.io/en/latest/enhance.html#kornia.enhance.jpeg_codec_differentiable
[6] huggingface.co/botp/stable-diffusion-v1-5
[7] huggingface.co/stablediffusionapi/realistic-vision-v51
[8] https://huggingface.co/Qwen/Qwen2-VL-7B-Instruct

(**FID**) [18], and Kernel Inception Distance (**KID**) [2] multiplied by a factor of 10^3.

D.4. Hyperparameters of results in paper

In this subsection, we detail the exact hyperparameters used for all results presented in the paper.

- **seed:** The random seed to the process is fixed to $1,000$ throughout the paper unless otherwise mentioned.
- **Model:** Either SD or RV, as defined in Appendix D.
- **idx:** The index of the image from the given dataset, when the count starts at 0.
- **prompt:** If the dataset is FFHQ, then the options are null, general and HQ. They are defined in Sec. 5.3 and the HQ prompts are given in the supplementary material. If the dataset is COCO, we only use a null prompt. One can use "A high quality photo" as a general prompt, but we did not experiment with it.
- **CFG:** Classifier-free guidance [19] allows for reconstruction that adhere more to the given prompt. We use either 1 (equal to not performing CFG) or 4.
- $\boldsymbol{\mathcal{A}}$ **:** The degradation operator, as defined in Sec. 5.2. In the appendix, we use the following abbreviations: GB for Gaussian blur, IP for inpaining, JP for JPEG, SR8 for Super-resolution $\times 8$, and SR4 for Super-resolution $\times 4$.
- $\boldsymbol{\sigma_y}$ **:** The amount of noise added to create the measurements was either 0.01 or 0.03 in all experiments.
- $\boldsymbol{\eta}$ **:** This parameter determines the scale (*i.e.*, step size) of the guidance term. Higher scales result in more consistent reconstructions at the expense of perceptual quality. We set $\eta = 0.5$ for all tasks, except for inpainting, where $\eta = 1$ is used.

For all the results of SILO (denoted as "Ours") in Figs. 6 and 7 we use a CFG of 1, $\sigma_y = 0.01$.

E. Comparison to other methods

In this section, we describe how SILO was compared to ReSample, PSLD, GML, and LDPS. SILO utilizes SD to generate reconstructions at a resolution of 512×512. PSLD, GML, and LDPS natively support this resolution and diffusion prior, as implemented in the PSLD GitHub repository [9]. Resample use the LDM-VQ4, trained on FFHQ [10] as the diffusion prior, which generates images of size 256×256. Hence, to give a fair comparison to ReSample, we had to adapt their publicly available code.

PSLD. We made no modifications to the PSLD code, except for adapting the data-loading process to enable sampling from the COCO dataset. The hyperparameters used were identical to those provided in the official repository for each task. For tasks not explicitly included (JPEG and SR $\times 8$), we applied the same hyperparameters as those used for the SR $\times 4$ task.

GML-DPS. We used the same implementation as for PSLD, replacing the PSLD step (Eq. (11)) with the GML step (Eq. (10)). The hyperparameters were identical to those used for PSLD.

LDPS. We used the same implementation as for PSLD, except that the PSLD step (Eq. (11)) was omitted entirely. This removes the associated computational requirements. The hyperparameters remained the same as those used for PSLD.

ReSample. As mentioned earlier, the reported results for ReSample are based on 256×256 reconstructions, generated using a different diffusion prior than SD, which was trained specifically on face images. To implement ReSample-SD, we started with the publicly available ReSample codebase [11]. We replaced the LDM-VQ4 denoiser with the SD one, updated the Autoencoder to match the one used for SD (consistent with all other methods), and adjusted the data-loading process to handle larger images. The hyperparameters used were identical to those in the original codebase. We acknowledge that the results of ReSample-SD differ from the reported results in ReSample, particularly for box inpainting. This discrepancy could stem from changes in the diffusion prior and image size, as well as suboptimal hyperparameters (due to these changes). The authors of the original paper were contacted to discuss the discrepancies we encountered. We should note that reconstruction time remains a significant factor – ReSample is notably slower than SILO, which achieves speedups of $10\times$ and $18\times$ for SR $\times 8$ and JPEG tasks, respectively.

F. Emulating the operator

The training scheme for the degradation operator is illustrated in Fig. 11, and described in Sec. 4. As a reminder, SILO does not require a specific type of network. The only requirement is to train it using Eq. (17). To demonstrate this, we used the network suggested in Readout Guidance [32], and a simple CNN. Training procedures for both setups are presented.

When training a Readout Guidance (RG) [32] operator H_θ, the inputs are features extracted from the denoising network. We followed the same settings as described in RG [32]. The learning rate was set to 2×10^{-4}, using the AdamW optimizer [12] [31]. Training was conducted on a single NVIDIA L40S card with a batch size of 16, for 7.5×10^4

[9] github.com/LituRout/PSLD

[10] github.com/CompVis/latent-diffusion/tree/main?tab=readme-ov-file#model-zoo

[11] github.com/soominkwon/resample

[12] pytorch.org/docs/stable/generated/torch.optim.AdamW

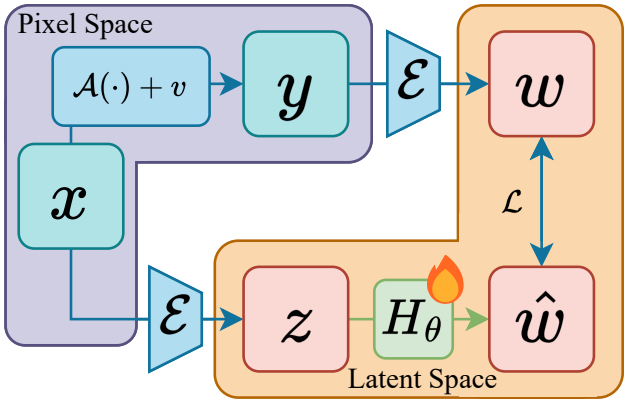

Figure 11. **Training scheme of the latent operator, $\mathbf{H}_\theta$.** In training, gradients flow from $\mathcal{L}$ to update the parameters of H_θ. Note that no gradients pass through the pixel space. H_θ learns to mimic the effect of the degradation operator in the latent space, allowing us to use SILO to solve inverse problems using LDMs.

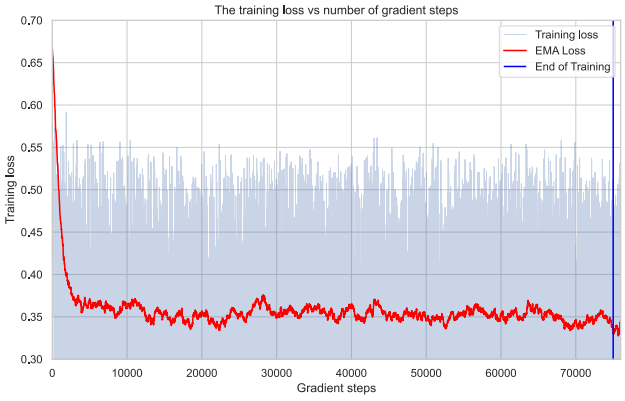

Figure 12. Training loss when learning a RG H_θ to mimic the Super-resolution $\times 8$ operator.

steps, taking approximately 14 hours. Training loss *vs*. the number of optimization steps for the Super-resolution $\times 8$ operator is shown in Fig. 12. We did not optimize the training process at all.

The CNN-based H_θ takes a latent z as input and produces $\hat{w}$ as output. The training scheme for the CNN is depicted in Fig. 11. The learning rate was set to 10^{-3}, using the AdamW optimizer. Training was conducted on a single NVIDIA L40S card with a batch size of 16, for 10^5 steps, requiring approximately 14 hours. Training loss *vs*. the number of optimization steps for the Super-resolution $\times 8$ operator is shown in Fig. 13. We did not optimize the training process or network architecture at all.

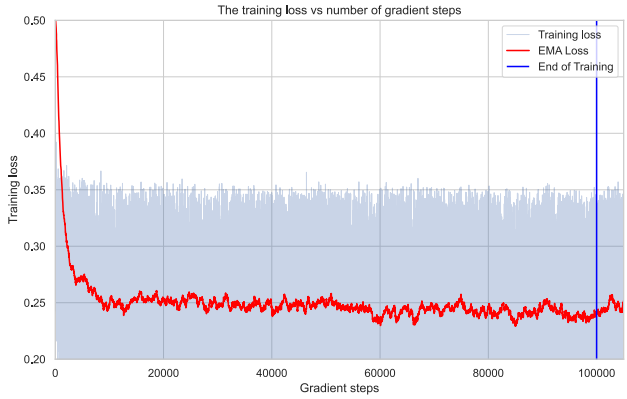

Figure 13. Training loss when learning a CNN H_θ to mimic the Super-resolution $\times 8$ operator.

G. Additional results

We reprint the results in Figs. 2, 6 and 7 in Tabs. 4 to 8, including PSNR and LPIPS-VGG for completeness. In Figs. 14, 15 and 17 to 21, we provide additional reconstructions examples of SILO, ReSample, PSLD, GML, and LDPS.

Additionally, following a reviewer's request, we added results of random box inpainting to this section. For this degradation, the operator gets the inpainting mask as a condition (see published code). The results in Tab. 9 and Fig. 16 were obtained for the same settings of inpainting as in Tab. 6.

Method	Time [sec]	PSNR	LPIPS-A	LPIPS-V	FID	KID
Ours (RV)	138	25.52	0.203	0.326	25.48	4.21
Ours (SD)	133	25.40	0.212	0.341	27.30	6.02
ReSample	2438	25.69	0.456	0.493	39.71	20.17
PSLD	cannot compute for nonlinear $\mathcal{A}$					
GML-DPS	402	27.60	0.268	0.373	33.69	7.54
LDPS	412	24.53	0.373	0.445	53.21	17.71

Table 4. Comparison of inverse problem solvers using latent diffusion on the FFHQ dataset. In this table, the time values are for the JPEG decompression task.

Method	PSNR	LPIPS-A	LPIPS-V	FID	KID
Ours (RV)	25.27	0.252	0.357	30.28	6.04
Ours (SD)	25.21	0.279	0.377	32.77	8.00
ReSample	16.18	0.724	0.740	235.8	253.1
PSLD	24.37	0.359	0.493	61.99	33.23
GML-DPS	26.19	0.354	0.425	41.06	13.69
LDPS	26.20	0.359	0.423	39.61	12.80

Table 5. Comparison of inverse problem solvers using LDMs on the FFHQ dataset, for SR $\times 8$ with $\sigma_y = 0.03$.

		Super-Resolution ×8					Inpainting				
Method	Time [sec]	PSNR	LPIPS-A	LPIPS-V	FID	KID	PSNR	LPIPS-A	LPIPS-V	FID	KID
Ours (RV)	149	26.28	0.226	0.327	27.10	5.23	22.51	0.139	0.239	18.98	1.80
Ours (SD)	148	26.13	0.253	0.344	30.71	8.47	22.23	0.151	0.258	21.04	4.32
ReSample	1418	22.80	0.575	0.603	131.75	118.57	16.91	0.273	0.359	146.08	119.34
PSLD	390	25.08	0.320	0.419	41.58	14.90	20.58	0.357	0.445	50.84	17.23
GML-DPS	389	27.01	0.327	0.399	38.71	12.99	20.64	0.356	0.443	49.89	16.54
LDPS	331	26.89	0.343	0.404	38.50	12.94	20.58	0.368	0.440	49.56	16.02

Table 6. Comparison of inverse problem solvers using latent diffusion on the FFHQ dataset.

		Gaussian blur					Super-Resolution ×4				
Method	Time [sec]	PSNR	LPIPS-A	LPIPS-V	FID	KID	PSNR	LPIPS-A	LPIPS-V	FID	KID
Ours (RV)	149	26.70	0.222	0.311	28.34	8.21	27.03	0.182	0.291	23.82	5.10
Ours (SD)	148	26.55	0.236	0.327	30.33	9.68	26.95	0.200	0.306	26.51	7.34
ReSample	1418	27.92	0.253	0.411	29.61	10.74	24.62	0.433	0.504	45.02	25.50
PSLD	390	28.63	0.288	0.372	38.44	12.23	28.23	0.249	0.355	29.63	10.11
GML-DPS	389	28.74	0.309	0.359	42.68	16.58	29.34	0.247	0.335	30.71	9.05
LDPS	331	28.00	0.327	0.378	47.38	19.95	29.06	0.281	0.362	34.44	11.69

Table 7. Comparison of inverse problem solvers using latent diffusion on the FFHQ dataset.

Method	PSNR	LPIPS-A	LPIPS-V	FID	KID
Ours (RV)	18.51	0.214	0.286	48.96	3.74
Ours (SD)	18.30	0.221	0.302	45.59	2.15
ReSample	16.53	0.297	0.368	104.37	54.16
PSLD	18.24	0.454	0.513	90.38	24.40
GML-DPS	18.25	0.453	0.513	88.16	21.99
LDPS	18.26	0.474	0.513	92.67	24.98

Table 8. Comparison of inverse problem solvers using latent diffusion on 1,000 images from the COCO dataset with inpainting.

Method	PSNR	LPIPS-A	LPIPS-V	FID	KID
Ours (RV)	22.52	0.146	0.243	17.46	1.02
Ours (SD)	21.98	0.165	0.267	19.43	2.34
PSLD	21.26	0.338	0.430	43.95	14.38

Table 9. Comparison of inverse problem solvers using latent diffusion on the FFHQ dataset, for the random box inpainting task

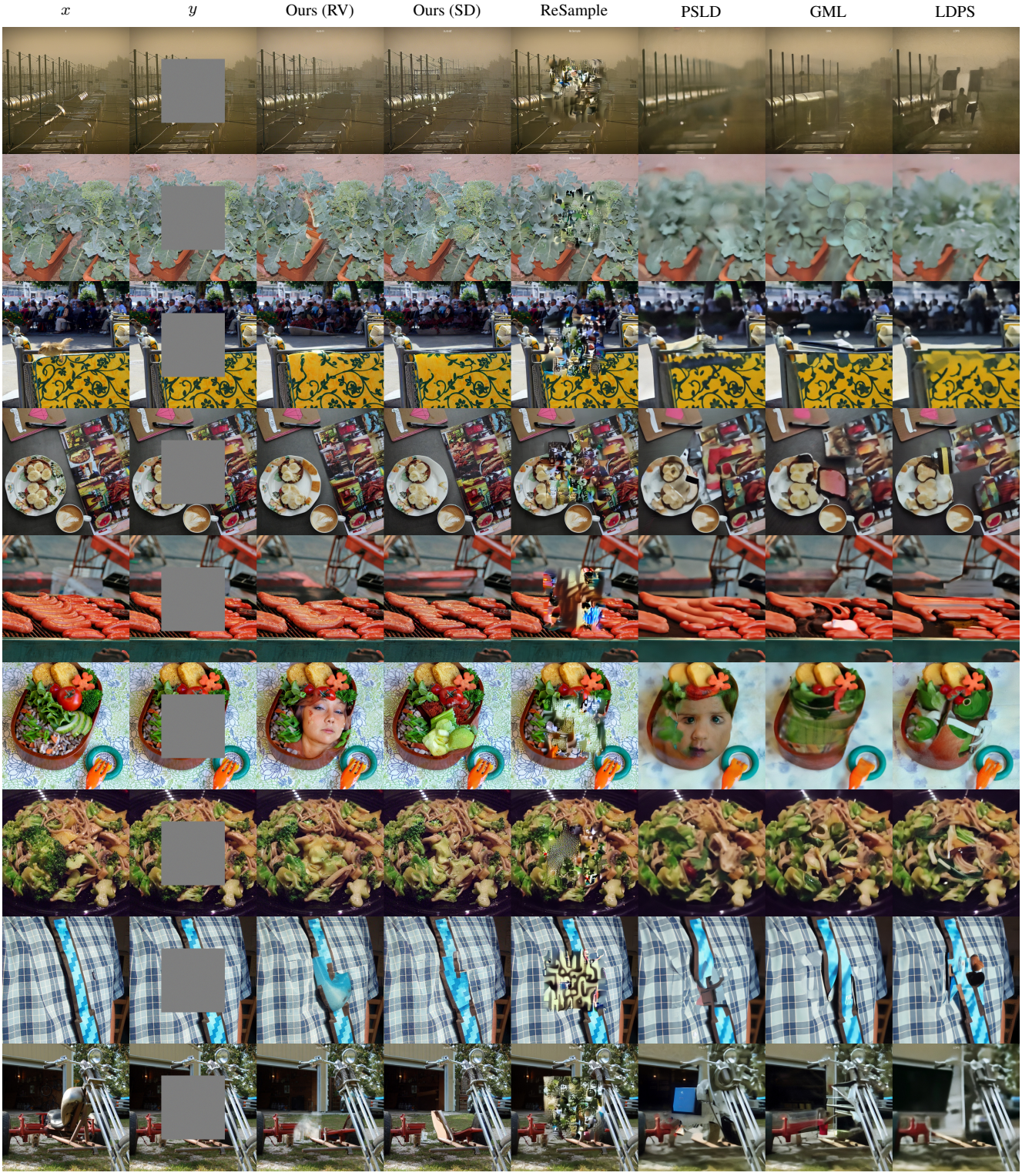

Figure 14. Box inpainting with $\sigma_y = 0.01$, COCO dataset. Additional results.

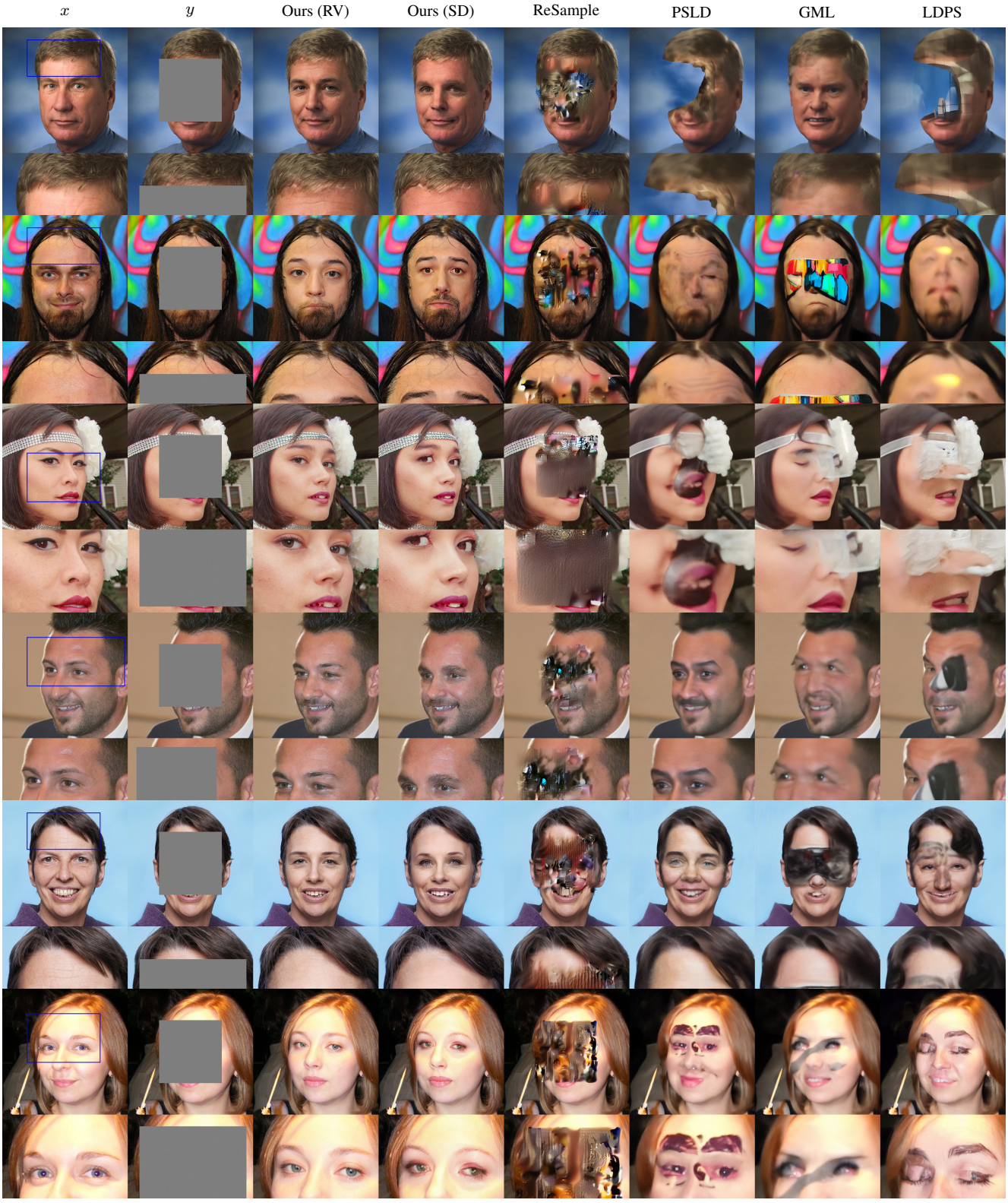

Figure 15. Box inpainting with $\sigma_y = 0.01$, FFHQ dataset. Additional results.

Figure 16. Random box inpainting with $\sigma_y = 0.01$, FFHQ dataset.

Figure 17. SR$\times$4 with $\sigma_y = 0.01$, FFHQ dataset. Additional results.

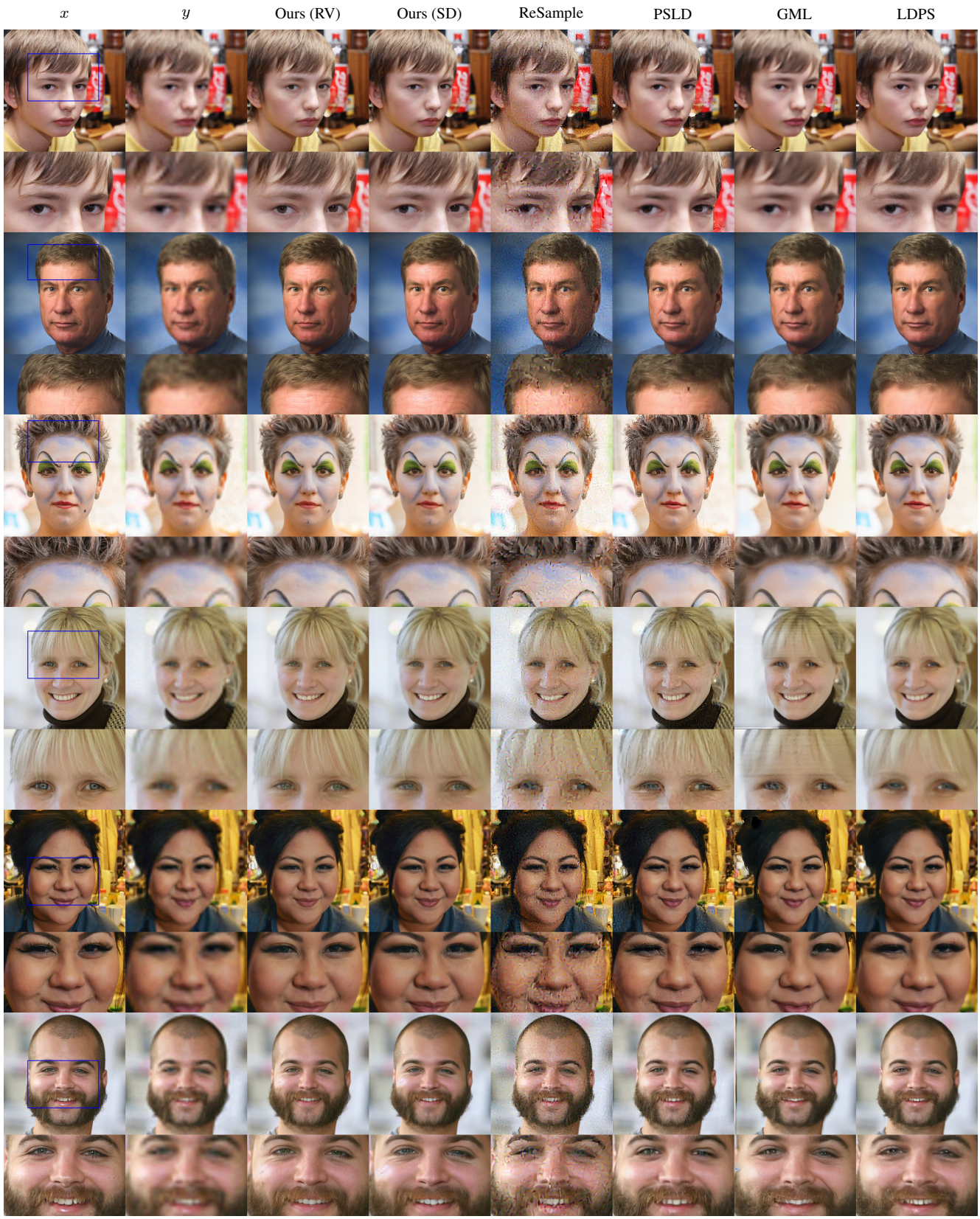

Figure 18. SR×8, FFHQ dataset. Additional results.

Figure 19. JPEG with $\sigma_y = 0.01$, FFHQ dataset. Additional results.

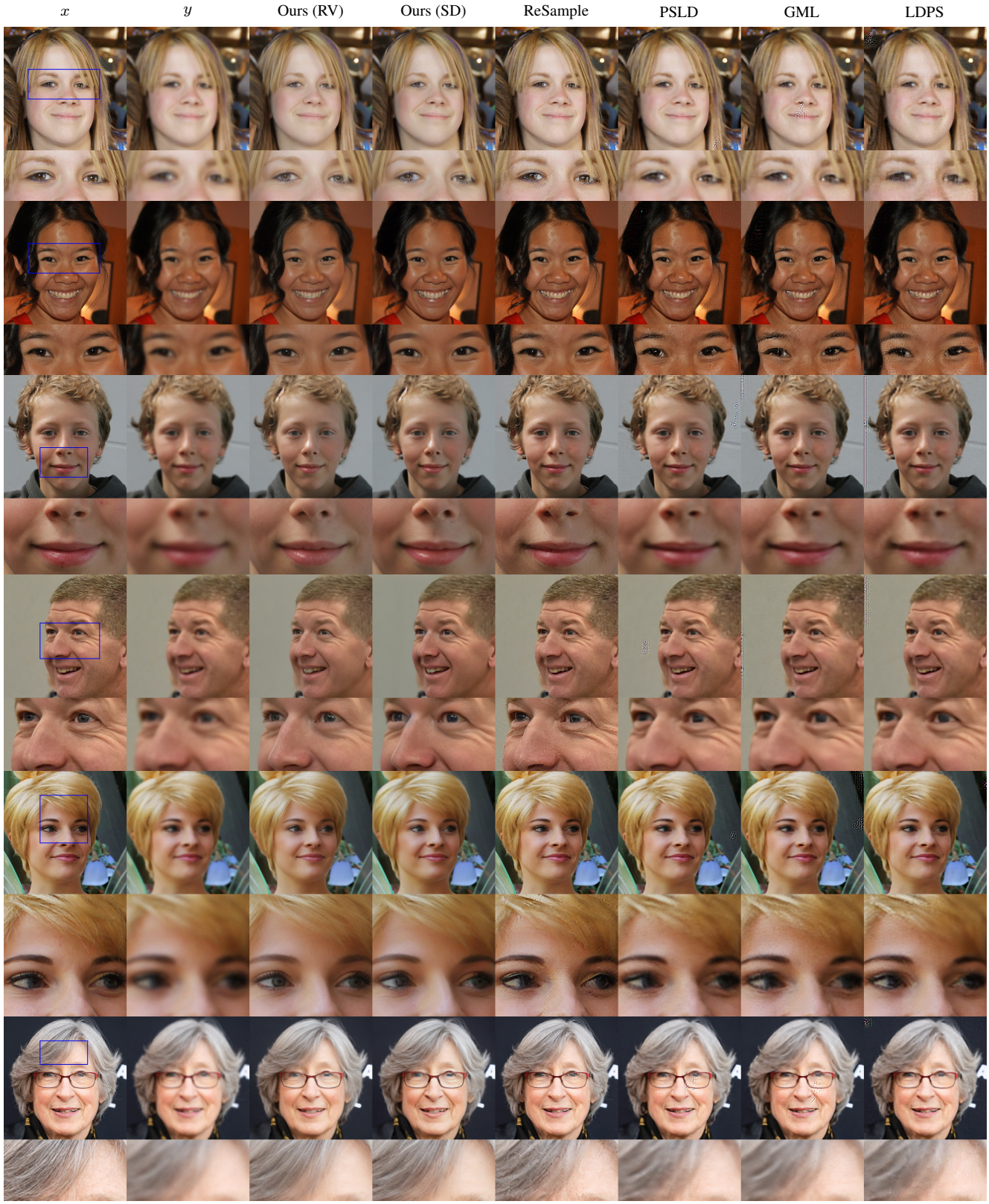

Figure 20. Gaussian blur with $\sigma_y = 0.01$, FFHQ dataset. Additional results.

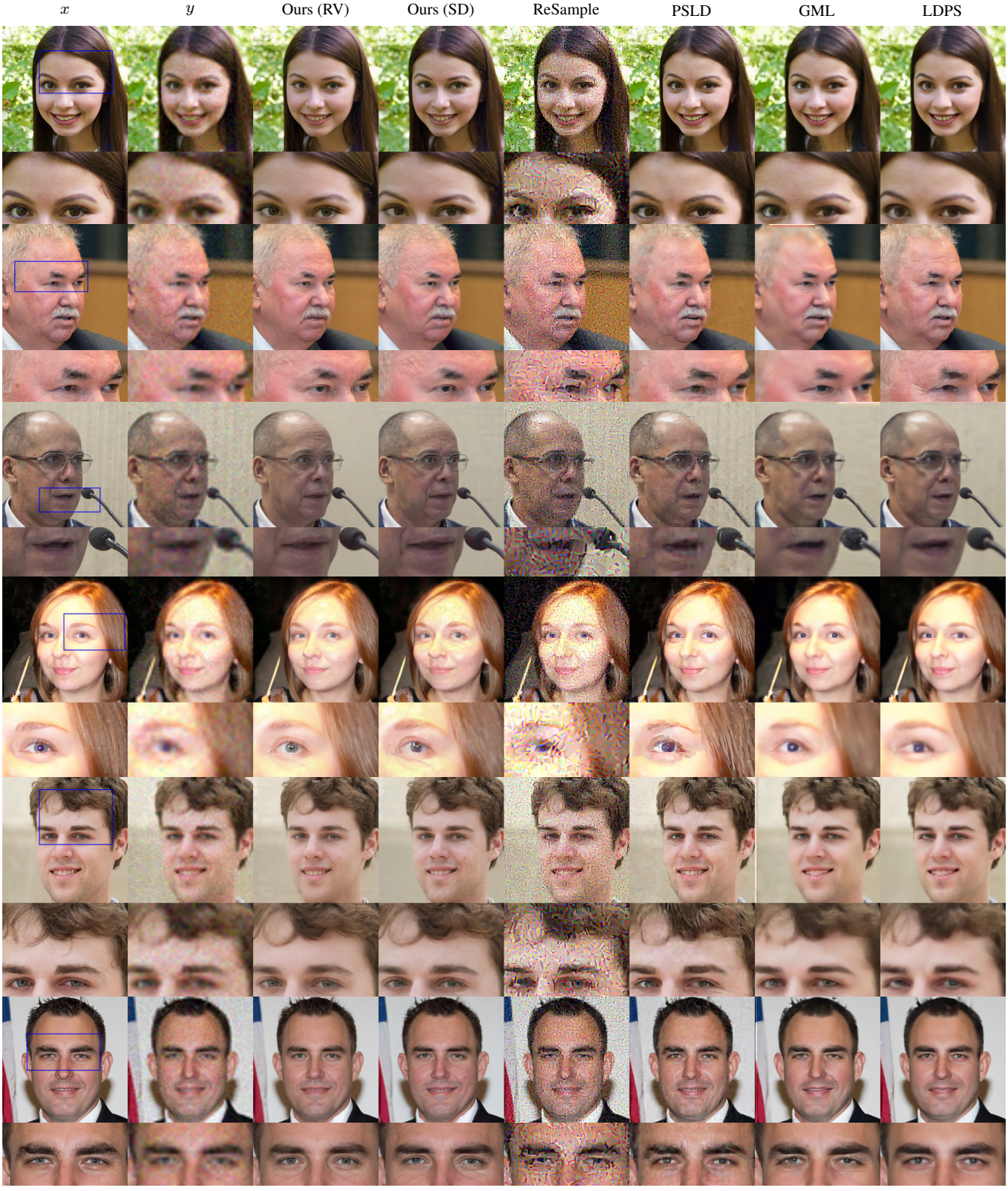

Figure 21. SR×8 with $\sigma_y = 0.03$, FFHQ dataset. Additional results.