Supplementary Material

The Stable Signature: Rooting Watermarks in Latent Diffusion Models

A. Implementation Details & Parameters

A.1. Details on the watermark encoder/extractor

Architectures of the watermark encoder/extractor. We keep the same architecture as in HiDDeN [108], which is a simple convolutional encoder and extractor. The encoder consist of 4 Conv-BN-ReLU blocks, with 64 output filters, 3×3 kernels, stride 1 and padding 1. The extractor has 7 blocks, followed by a block with k output filters (k being the number of bits to hide), an average pooling layer, and a $k \times k$ linear layer. For more details, we refer the reader to the original paper [108].

Optimization. We train on the MS-COCO dataset [47], with 256×256 images. The number of bits is $k = 48$, and the scaling factor is $\alpha = 0.3$. The optimization is carried out for 300 epochs on 8 GPUs, with the Lamb optimizer [95] (it takes around a day). The learning rate follows a cosine annealing schedule with 5 epochs of linear warmup to 10^{-2}, and decays to 10^{-6}. The batch size per GPU is 64.

Attack simulation layer. The attack layer produces edited versions of the watermarked image to improve robustness to image processing. It takes as input the image output by the watermark encoder $x_w = \mathcal{W}_E(x_o)$ and outputs a new image x' that is fed to the decoder $\mathcal{W}$. This layer is made of cropping, resizing, or identity chosen at random in our experiments, unless otherwise stated. The parameter for the crop or resize is set to 0.3 or 0.7 with equal probability. This is followed by a JPEG compression with probability 0.5. The parameter for the compression is set to 50 or 80 with equal probability. This last layer is not differentiable, therefore we back-propagate only through the difference between the uncompressed and compressed images: $x' = x_{\text{aug}} + \text{nograd}(x_{\text{aug,JPEG}} - x_{\text{aug}})$ [101].

Whitening. At the end of the training, we whiten the output of the watermark extractor to make the hard thresholded bits independently and identically Bernoulli distributed on vanilla images (so that the assumption of 3.1 holds better, see App. B.5). We perform the PCA of the output of the watermark extractor on a set of 10k vanilla images, and get the mean μ and eigendecomposition of the covariance matrix $\Sigma = U\Lambda U^T$. The whitening is applied with a linear layer with bias $-\Lambda^{-1/2}U^T\mu$ and weight $\Lambda^{-1/2}U^T$, appended to the extractor.

A.2. Image transformations

We evaluate the robustness of the watermark to a set of transformations in sections 5, 6 and B.2. They simulate image processing steps that are commonly used in image editing software. We illustrate them in Figure 9. For crop and resize, the parameter is the ratio of the new area to the original area. For rotation, the parameter is the angle in degrees. For JPEG compression, the parameter is the quality factor (in general 90% or higher is considered high quality, 80%-90% is medium, and 70%-80% is low). For brightness, contrast, saturation, and sharpness, the parameter is the default factor used in the PIL and Torchvision [53] libraries. The text overlay is made through the AugLy library [60], and adds a text at a random position in the image. The combined transformation is a combination of a crop 0.5, a brightness change 1.5, and a JPEG 80 compression.

A.3. Generative tasks

Text-to-image. In text-to-image generation, the diffusion process is guided by a text prompt. We follow the standard protocol in the literature [65, 66, 69, 74] and evaluate the generation on prompts from the validation set of MS-COCO [47]. To do so, we first retrieve all the captions from the validation set, keep only the first one for each image, and select the first 1000 or 5000 captions depending on the evaluation protocol. We use guidance scale 3.0 and 50 diffusion steps. If not specified, the generation is done for 5000 images. The FID is computed over the validation set of MS-COCO, resized to 512×512.

Image edition. DiffEdit [13] takes as input an image, a text describing the image and a novel description that the edited image should match. First, a mask is computed to identify which regions of the image should be edited. Then, mask-based generation is performed in the latent space, before converting the output back to RGB space with the image decoder. We use the default parameters used in the original paper, with an encoding ratio of 90%, and compute a set of 5000 images from the COCO dataset, edited with the same prompts as the paper [13]. The FID is computed over the validation set of MS-COCO, resized to 512×512.

Inpainting. We follow the protocol of LaMa [82], and generate 5000 masks with the "thick" setting, at resolution 512×512, each mask covering $1 - 50\%$ of the initial image (with an average of 27%). For the diffusion-based inpainting, we use the inference-time algorithm presented in [81], also used in Glide [57], which corrects intermediate esti-

Figure 9. Illustration of all transformations evaluated in sections 5 and 6.

mations of the final generated image with the ground truth pixel values outside the inpainting mask. For latent diffusion models, the same algorithm can be applied in latent space, by encoding the image to be inpainted and downsampling the inpainting mask. In this case, we consider 2 different variations: (1) inpainting is performed in the latent space and the final image is obtained by simply decoding the latent image; and (2) the same procedure is applied, but after decoding, ground truth pixel values from outside the inpainting mask are copy-pasted from the original image. The latter allows to keep the rest of the image perfectly identical to the original one, at the cost of introducing copy-paste artifacts, visible in the borders. Image quality is measured with an FID score, computed over the validation set of ImageNet [16], resized to 512×512.

Super-resolution. We follow the protocol suggested by Saharia *et al*. [75]. We first resize 5000 random images from the validation set of ImageNet to 128×128 using bicubic interpolation, and upscale them to 512×512. The FID is computed over the validation set of ImageNet, cropped and resized to 512×512.

A.4. Watermarking methods

For Dct-Dwt, we use the implementation of https://github.com/ShieldMnt/invisible-watermark (the one used in Stable Diffusion). For SSL Watermark [25] and FNNS [42] the watermark is embedded by optimizing the image, such that the output of a pre-trained model is close to the given key (like in adversarial examples [30]). The difference between the two is that in SSL Watermark we use a model pre-trained with DINO [9], while FNNS uses a watermark or stenography model. For SSL Watermark we use the default pre-trained model of the original paper. For FNNS we use the HiDDeN extractor used in all our experiments, and not SteganoGan [103] as in the original paper, because we want to extract watermarks from images of different sizes. We use the image optimization scheme of Active Indexing [24], *i.e*. we optimize the distortion image for 10 iterations, and modulate it with a perceptual just noticeable difference (JND) mask. This avoids visible artifacts and gives a PSNR comparable with our method ($\approx$ 30dB). For HiDDeN, we use the watermark encoder and extractor from our pre-training phase, but the extractor is not whitened and we modulate the encoder output with the same JND mask. Note that in all cases we watermark images one by one for simplicity. In practice the watermarking could be done by batch, which would be more efficient.

A.5. Attacks

Watermark removal. The perceptual auto-encoders aim to create compressed latent representations of images. We select 2 state-of-the-art auto-encoders from the CompressAI library zoo [6]: the factorized prior model [4] and the anchor model variant [11]. We also select the auto-encoders from Esser *et al*. [20] and Rombach *et al*. [68]. For all models, we use different compression factors to observe the trade-off between quality degradation and removal robustness. For `bmshj2018`: 1, 4 and 8, for `cheng2020`: 1, 3 and 6, for `esser2021`: VQ-4, 8 and 16, for `rombach2022` KL-4, 8, 16 and 32 (KL-8 being the one used by SD v1.4). We generate 1k images from text prompts with our LDM watermarked with a 48-bits key. We then try to remove the watermark using the auto-encoders, and compute the bit accuracy on the extracted watermark. The PSNR is computed between the original image and the reconstructed one, which explains why the PSNR does not exceed 30dB (since the watermarked image already has a PNSR of 30dB). If we compared between the watermarked image and the image reconstructed by the auto-encoder instead, the curves would show the same trend but the PSNR would be 2-3 points higher.

Watermark removal (white-box). In the white-box case, we assume have access to the extractor model. The adversarial attack is performed by optimizing the image in the same manner as [25]. The objective is a MSE loss between the output of the extractor and a random binary message fixed beforehand. The attack is performed for 10 iterations with the Adam optimizer [41] with learning rate 0.1.

Watermark removal (network-level). We use the same fine-tuning procedure as in Sec. 4.2. This is done for different numbers of steps, namely 100, 200, and every multiple of 200 up to 1600. The bit accuracy and the reported PSNR are computed on 1k images of the validation set of COCO, for the auto-encoding task.

Model collusion. The goal is to observe the decoded watermarks on the generation when 2 models are averaged together. We fine-tune the LDM decoder for 10 different 48-bits keys (representing 10 Bobs). We then randomly sample a pair of Bobs and average the 2 models, with which we generate 100 images. We then extract the watermark from the generated images and compare them to the 2 original keys. We repeat this experiment 10 times, meaning that we observe $10 \times 100 \times 48 = 48000$ decoded bits.

In the inline figure, the rightmost skewed normal is fitted with the Scipy library and the corresponding parameters are $a : 6.96, e : 0.06, w : 0.38$. This done over all bits where Bobs both have a 1. The same observation holds when there is no collusion, with approximately the same parameters. When the bit is not the same between Bobs, we denote by $m_1^{(i)}$ the random variable representing the output of the extractor in the case where the generative model only comes from Bob$^{(i)}$, and by m_2 the random variable representing the output of the extractor in the case where the generative model comes from the average of the two Bobs. Then in our model $m_2 = 0.5 \cdot (m_1^{(i)} + m_1^{(j)})$, and the pdf of m_2 is the convolution of the pdf of $m_1^{(i)}$ and the pdf of $m_1^{(j)}$, rescaled in the x axis because of the factor 0.5.

B. Additional Experiments

B.1. Perceptual loss

The perceptual loss of (4) affects the image quality. Figure 10 shows how the parameter λ_i affects the image quality. For high values, the image quality is very good. For low values, artifacts mainly appear in textured area of the image. It is interesting to note that this begins to be problematic only for low PSNR values (around 25 dB).

Figure 10 shows an example of a watermarked image for different perceptual losses: Watson-VGG [15], Watson-DFT [15], LPIPS [105], MSE, and LPIPS+MSE. We set the weight λ_i of the perceptual loss so that the watermark performance is approximately the same for all types of loss, and such that the degradation of the image quality is strong enough to be seen. Overall, we observe that the Watson-VGG loss gave the most eye-pleasing results, closely followed by the LPIPS. When using the MSE, images are blurry and artifacts appear more easily, even though the PSNR is higher.

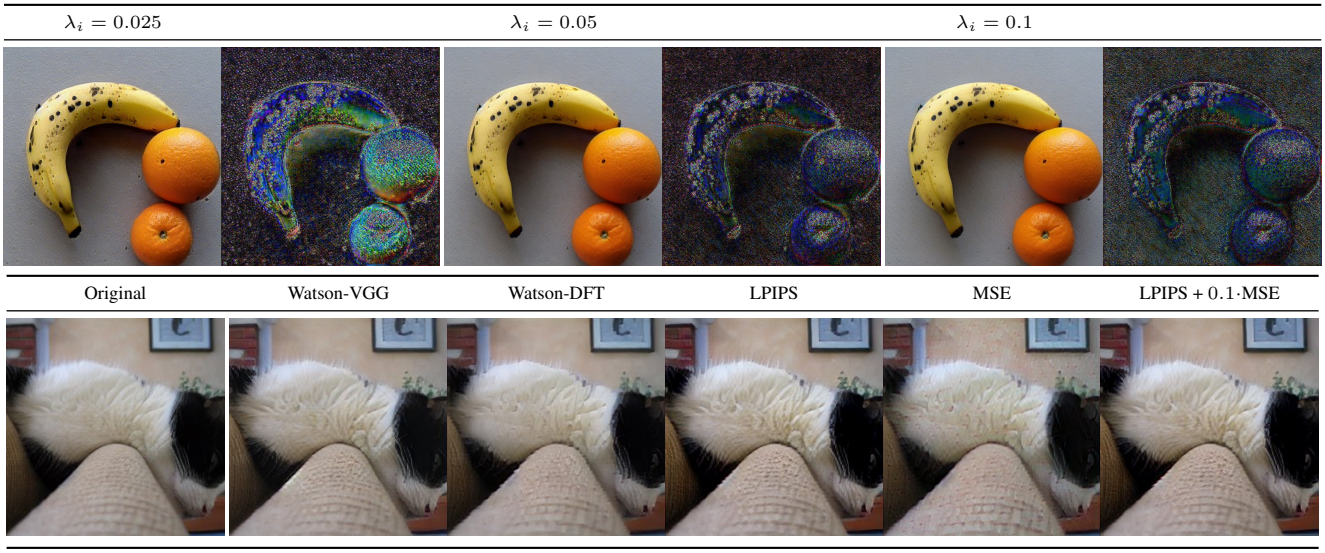

Figure 10. Qualitative influence of the perceptual loss during LDM fine-tuning. (Top): we show images generated with the LDM auto-encoder fine-tuned with different λ_i, and the pixel-wise difference ($\times 10$) with regards to the image obtained with the original model. PSNR are 24dB, 26dB, 28dB from left to right. (Bottom): we change the perceptual loss and fix λ_i to have approximately the same bit accuracy of 0.95 on the "combined" augmentation.

Table 5. Watermark robustness on different tasks and image transformations applied before decoding. We report the bit accuracy, averaged over 10×1k images generated with 10 different keys. The combined transformation is a combination Crop 50%, Brightness 1.5 and JPEG 80. More detail on the evaluation is available in the supplement A.3.

Task		Image transformation									
		None	Crop 0.1	JPEG 50	Resi. 0.7	Bright. 2.0	Cont. 2.0	Sat. 2.0	Sharp. 2.0	Text over.	Comb.
Text-to-Image	LDM [68]	0.99	0.95	0.88	0.91	0.97	0.98	0.99	0.99	0.99	0.92
Image Edition	DiffEdit [13]	0.99	0.95	0.90	0.91	0.98	0.98	0.99	0.99	0.99	0.94
Inpainting - Full	Glide [57]	0.99	0.97	0.88	0.90	0.98	0.99	0.99	1.00	0.99	0.93
- Mask only		0.89	0.76	0.73	0.77	0.84	0.86	0.89	0.91	0.89	0.78
Super-Resolution	LDM [68]	0.98	0.93	0.86	0.85	0.96	0.96	0.97	0.98	0.98	0.92

B.2. Additional results on watermarks robustness

In Table 5, we report the same table as in Table 1 that evaluates the watermark robustness in bit accuracy on different tasks, with additional image transformations. They are detailed and illustrated in App. A.3. As a reminder, the watermark is a 48-bit binary key. It is robust to a wide range transformations, and most often yields above 0.9 bit accuracy. The resize and JPEG 50 transformations seems to be the most challenging ones, and sometimes get bellow 0.9. Note that the crop location is not important but the visual content of the crop is, *e.g.* there is no way to decode the watermark on crops of blue sky (this is the reason we only show center crop).

B.3. Additional network level attacks

Tab. 6 reports robustness of the watermarks to different quantization and pruning levels for the LDM decoder. Quantization is performed naively, by rounding the weights to the closest quantized value in the min-max range of every weight matrix. Pruning is done using PyTorch [61] pruning API, with the L1 norm as criterion. We observe that the network generation quality degrades faster than WM robustness. To reduce bit accuracy lower than 98%, quantization degrades the PSNR <25dB, and pruning <20dB.

Table 6. Bit accuracy after network attacks, observed over 10×1k images generated from text prompts.

Quantization (8-bits)	0.99	Pruning L1 (30%)	0.99
Quantization (4-bits)	0.99	Pruning L1 (60%)	0.95

B.4. Scaling factor at pre-training.

The watermark encoder does not need to be perceptually good and it is beneficial to degrade image quality during pre-training. In the following, ablations are conducted on a shorter schedule of 50 epochs, on 128×128 images and 16-bits messages. In Table 7, we train watermark encoders/extractors for different scaling factor α (see Sec. 4.1), and observe that α strongly affects the bit accuracy of the method. When it is too high, the LDM needs to generate low quality images for the same performance because the distortions seen at pre-training by the extractor are too strong. When it is too low, they are not strong enough for the watermarks to be robust: the LDM will learn how to generate watermarked images, but the extractor won't be able to extract them on edited images.

B.5. Are the decoded bits i.i.d. Bernoulli random variables?

The FPR and the p-value (2) are computed with the assumption that, for vanilla images (not watermarked), the bits output by the watermark decoder $\mathcal{W}$ are independent

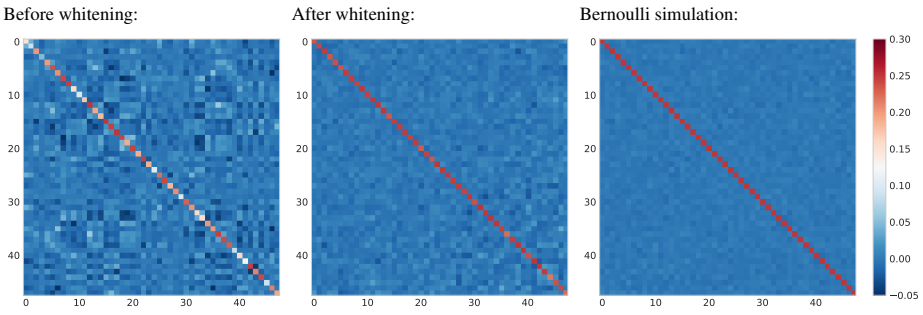

Figure 11. Covariance matrices of the bits output by the watermark decoder $\mathcal{W}$ before and after whitening.

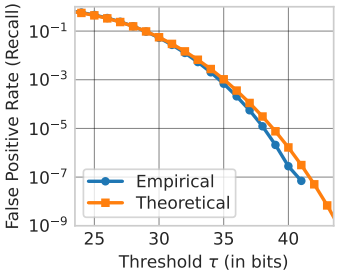

Figure 12. FPR Empirical check.

Table 7. Influence of the (discarded) watermark encoder perceptual quality. $P_{1,2}$ stands for Phase 1 or 2.

Scaling factor α	0.8	0.4	0.2	0.1	0.05
(P_1) - PSNR ↑	16.1	21.8	27.2	33.5	39.3
(P_2) - PSNR ↑	27.9	30.5	**30.8**	28.8	27.8
(P_1) - Bit acc. ↑ on 'none'	1.00	1.00	0.86	0.72	0.62
(P_2) - Bit acc. ↑ on 'none'	0.98	**0.98**	0.91	0.90	0.96
(P_2) - Bit acc. ↑ on 'comb.'	**0.86**	0.73	0.82	0.81	0.69

and identically distributed (i.i.d.) Bernoulli random variables with parameter 0.5. This assumption is not true in practice, even when we tried using regularizing losses in the training at phase one [5, 72]. This is why we whiten the output at the end of the pre-training.

Figure 11 shows the covariance matrix of the hard bits output by $\mathcal{W}$ before and after whitening. They are computed over 5k vanilla images, generated with our LDM at resolution 512×512 (as a reminder the whitening is performed on 1k vanilla images from COCO at 256×256). We compare them to the covariance matrix of a Bernoulli simulation, where we simulate $5k$ random messages of 48 Bernoulli variables. We observe the strong influence of the whitening on the covariance matrix, although it still differs a little from the Bernoulli simulation. We also compute the bit-wise mean and observe that for un-whitened output bits, some bits are very biased. For instance, before whitening, one bit had an average value of 0.95 (meaning that it almost always outputs 1). After whitening, the maximum average value of a bit is 0.68. For the sake of comparison, the maximum average value of a bit in the Bernoulli simulation was 0.52. It seems to indicate that the distribution of the generated images are different than the one of vanilla images, and that it impacts the output bits. Therefore, the bits are not perfectly i.i.d. Bernoulli random variables. We however found they are close enough for the theoretical FPR computation to match the empirical one (see next section) – which was what we wanted to achieve.

B.6. Empirical check of the FPR

In Figure 3, we plotted the TPR against a theoretical value for the FPR, with the i.i.d. Bernoulli assumption. The FPR was computed theoretically with (2). Here, we empirically check on smaller values of the FPR (up to 10^{-7}) that the empirical FPR matches the theoretical one (higher values would be too computationally costly). To do so, we use the 1.4 million vanilla images from the training set of ImageNet resized and cropped to 512×512, and perform the watermark extraction with $\mathcal{W}$. We then fix 10 random 48-bits key $m^{(1)}, \cdots, m^{(10)}$, and, for each image, we compute the number of matching bits $d(m', m^{(i)})$ between the extracted message m' and the key $m^{(i)}$, and flag the image if $d(m', m^{(i)}) \geq \tau$.

Figure 12 plots the FPR averaged over the 10 keys, as a function of the threshold τ. We compare it to the theoretical one obtained with (2). As it can be seen, they match almost perfectly for high FPR values. For lower ones ($< 10^{-6}$), the theoretical FPR is slightly higher than the empirical one. This is a good thing since it means that if we fixed the FPR at a certain value, we would observe a lower one in practice.

C. Additional Qualitative Results

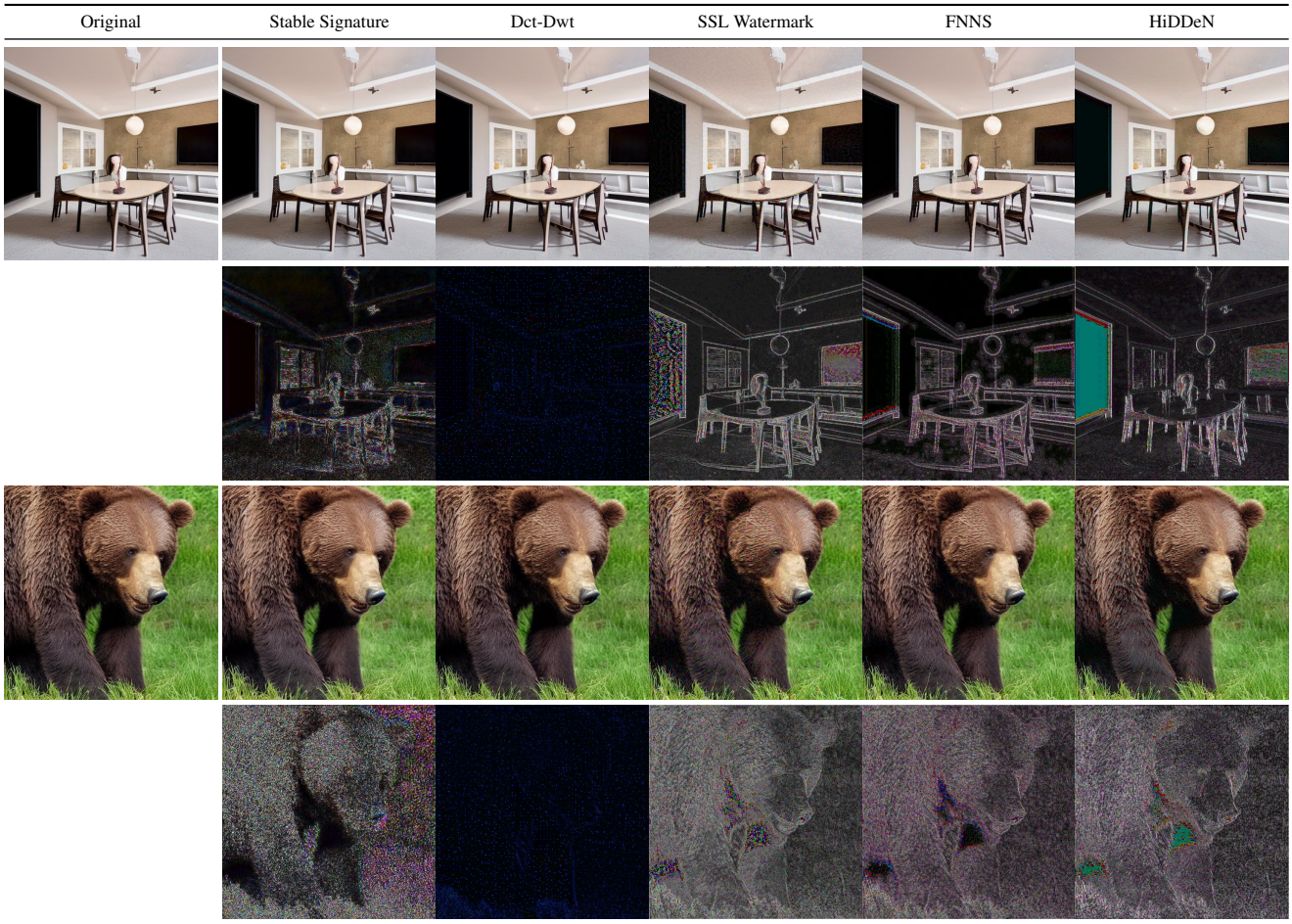

Figure 13. Qualitative results for different watermarking methods on generated images at resolution 512.

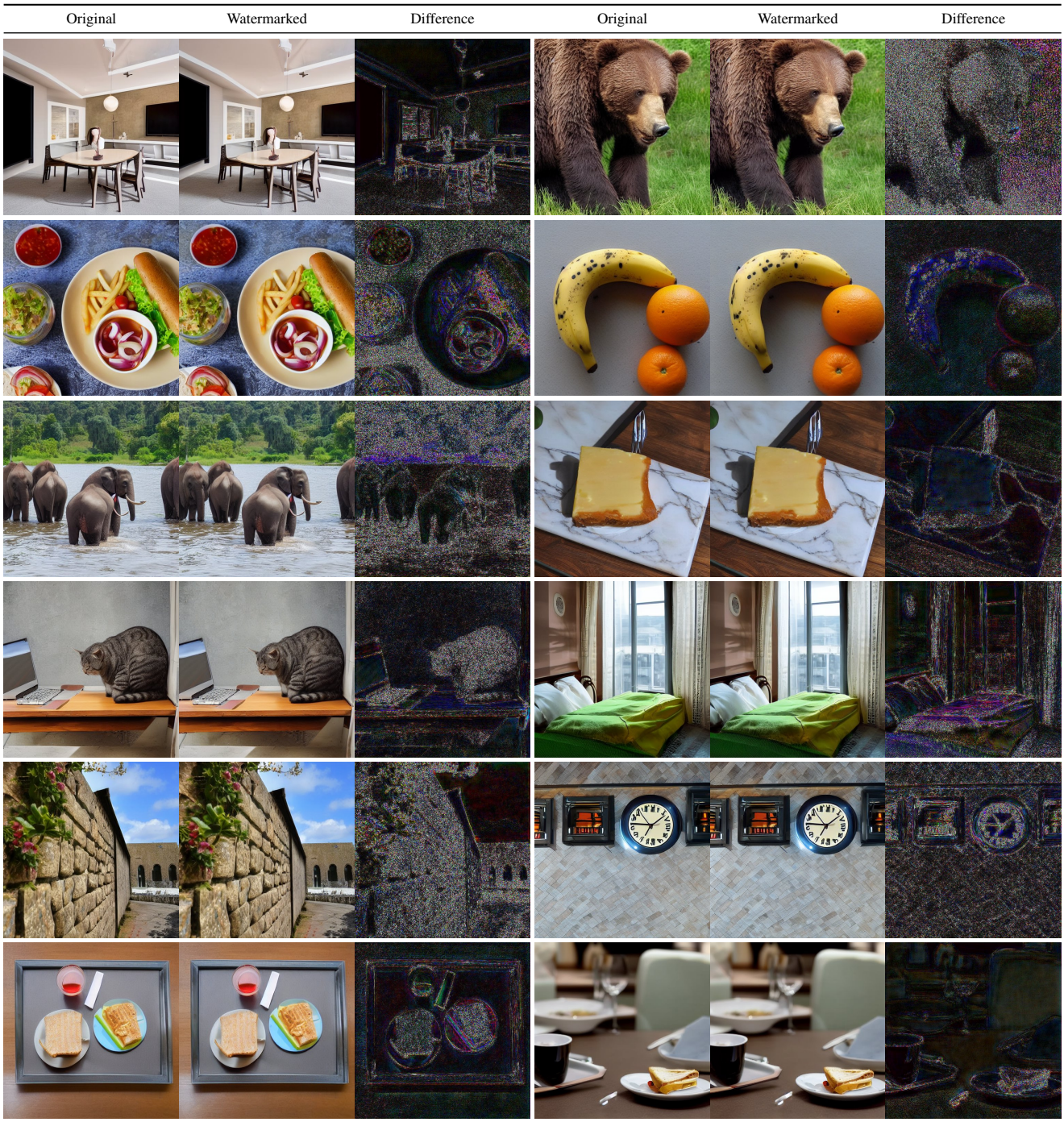

Figure 14. Qualitative results on prompts of the validation set of MS-COCO, at resolution 512 and for a 48-bits signature. Images are generated from the same latents, with original or watermarked generative models.

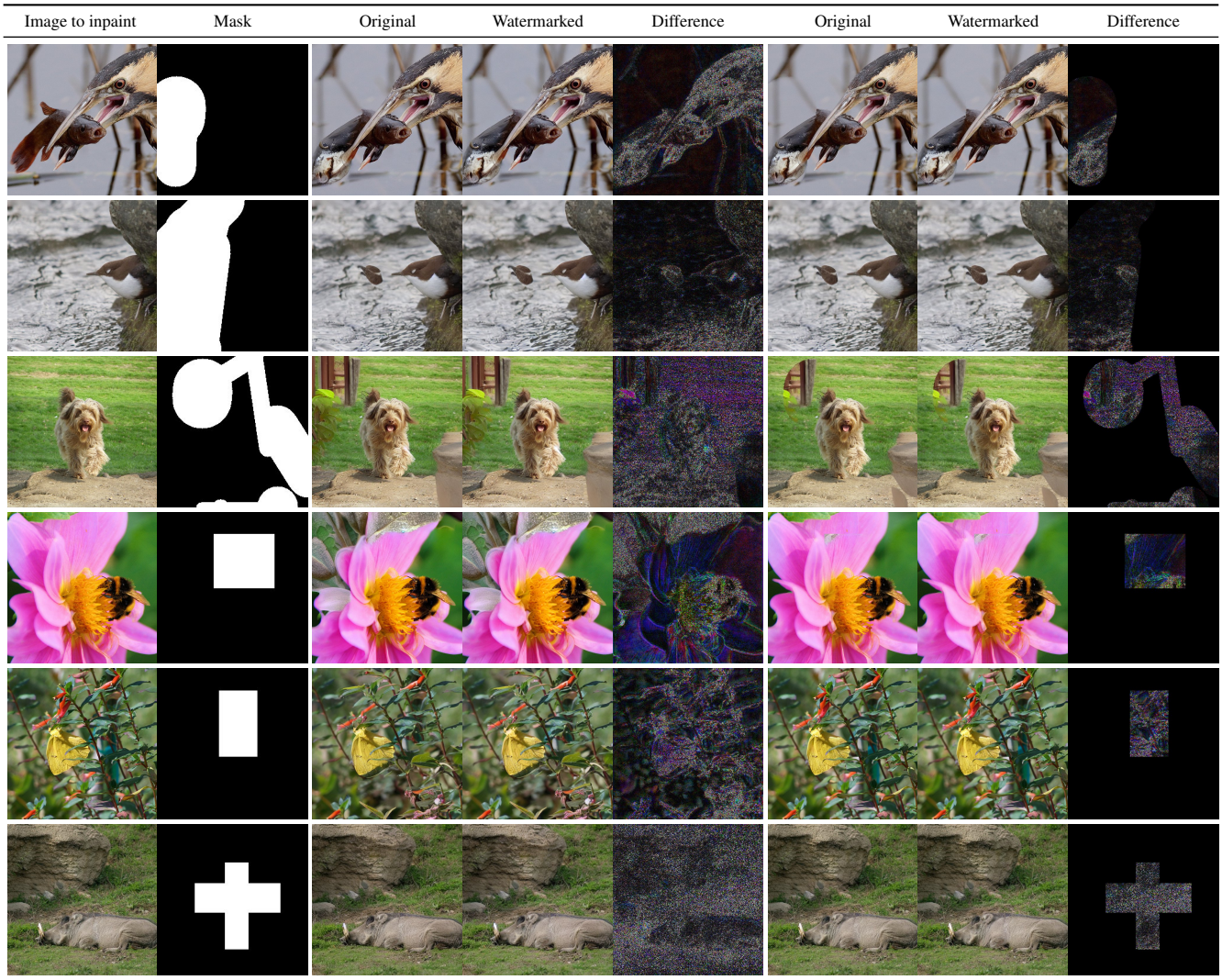

Figure 15. Qualitative results for inpainting on ImageNet, with masks created from LaMa protocol [82], with original or watermarked generative models. We consider 2 scenarios: (middle) the full image is modified to fill the masked area, (rigtht) only the masked area is filled. Since our model is not fine-tuned for inpainting, the last scenario introduces copy-paste artifacts. From a watermarking point of view, it is also the more interesting, since the watermark signal is only present in the masked area (and erased wherever the image to inpaint is copied). Even in this case, the watermark extractor achieves bit accuracy significantly higher than random.

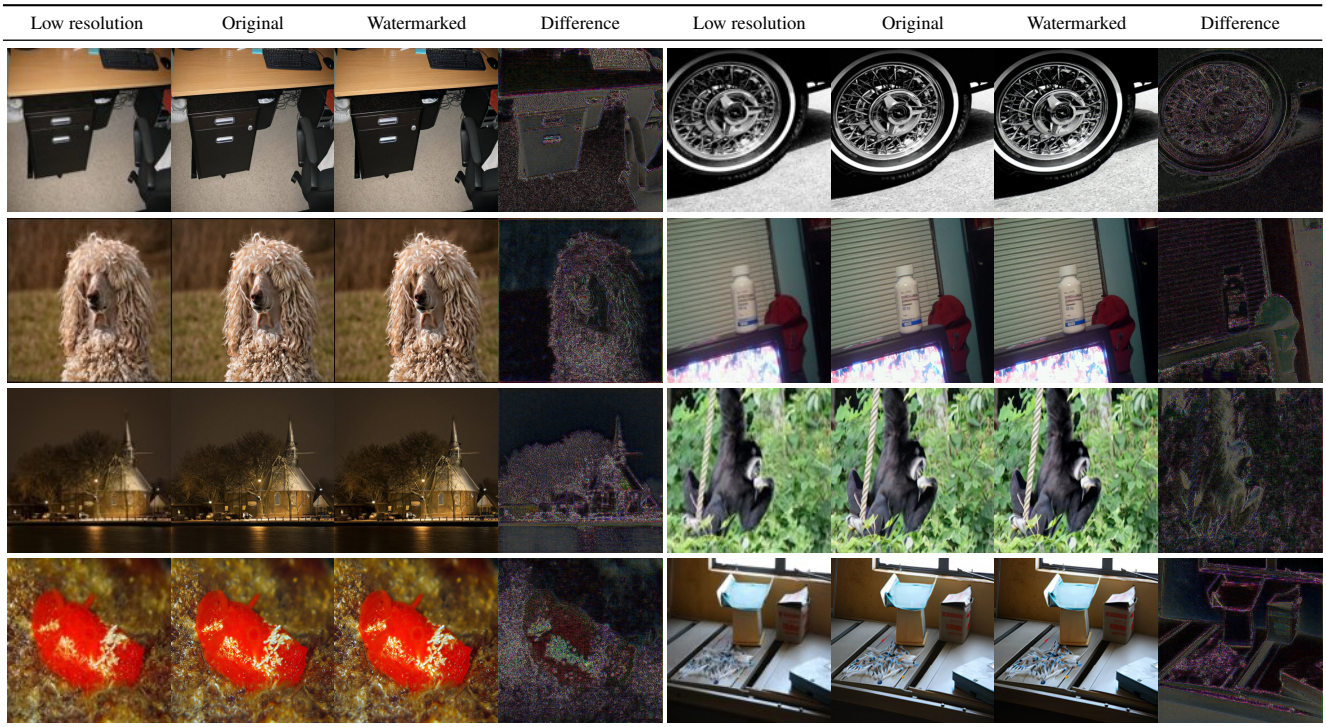

Figure 16. Qualitative results for super-resolution on ImageNet, with original and watermarked generative models. Low resolution images are 128×128, and upscaled to 512×512 with an upscaling factor $f = 4$.