

## A Additional Algorithms

We detail the functions used in Algorithm 1, Algorithm 2 and Algorithm 3 here:

- Bias: Addition of the bias  $\delta$  to the current logit  $l_c$  based on the green list  $G$ .
- Sample: Use the probabilities in  $p_c$  to sample the returned bit, either 0 or 1.
- lookup: Perform the BSQ lookup [42] to return the quantized states.
- interpolate: Scaling to target resolution.
- Count: Action performed for the statistical test, which counts the number of bits which are part of the green list.

We present the pseudo code of the novel Bit-Flipper attack in Algorithm 3. To apply the attack, we have to first encode the image in to bits (line 1). During deconstructing the old watermarked image, we construct a new attacked image. We recursively iterate through all scales  $K$ , retrieve the bit representations of the respective scale (line 4) and count how many bits are part of the green list and the red list (line 5). Then we define the probability of flipping a bit based on the green fraction and the flip factor  $\phi$ . Next, we change the bits which are part of the green list based on the random probability  $p$  (line 8, 9). We follow the standard encoding and decoding process, to iterate through all resolutions of the image (line 10-13). Finally, we return the decoded version of the attacked image.

---

### Algorithm 3 Bit-Flipper Attack

---

**Inputs:** Image  $im$ , green list  $G$ , red list  $R$ , image decoder  $\mathcal{D}$ .

**Hyperparameters:** Flip factor  $\phi$ , steps  $K$  (number of resolutions), resolutions  $(h_i, w_i)_{i=1}^K$ , the number of bits for resolution  $i$  is  $t_i$ ,  $n$  - the length of the bit vector,  $|s|$  the total number of bit sequences.

```

 $e = \mathcal{E}(im)$ 
for  $i = 1, \dots, K$  do
     $u_i = \mathcal{Q}(\text{interpolate}(e, h_i, w_i))$ 
     $C_i = \text{Count}((b_1, \dots, b_{t_i}), G)$ 
     $p = (\frac{C_i}{|s|} - 0.5) * \phi$ 
    for  $j = n, \dots, t_i$  do
        if  $(b_{j-n}, \dots, b_j) \in G \wedge \text{random}() < p$  then
             $b_j = \neg b_j$  {Flip bit with probability  $p$  if sequence is in green list}
     $z_i = \text{lookup}(u_i)$ 
     $z_i = \text{interpolate}(z_i, h_K, w_K)$ 
     $e = e - \phi_i(z_i)$ 
\phi_i(z_i)

```

**Return:** attacked image  $\mathcal{D}(img)$

---

## B Green and Red List Selection

**Selection of Red and Green Lists.** The re-encoding loss presented in Section 3.1 also motivates the choice of a small length of the green list  $n$ , as larger consecutive bit sequences are more likely to be interrupted. Additionally, this auto-encoder has the property of re-encoding images towards an equal frequency of 0's and 1's, so it is not efficiently possible to bias each bit towards either 0 or 1, making the choice of  $n = 1$  unpractical (see Table 6). Thus we leverage the second smallest possible choice,  $n = 2$ , resulting in 6 possible choices to separate  $G$  and  $R$  by keeping  $|G| = |R|$ .

As we have the constraint to not bias towards 0 or 1, as this type of pattern is more likely removed by the auto-encoder, and given the constraint shown that the same prefix must not be in the same list twice as discussed in Section 3.2 the most effective  $G$  for BitMark are  $G = \{01, 10\}$  and  $G = \{00, 11\}$ . We choose  $G = \{01, 10\}$  throughout all experiments.

**Changing the Length of the Bit Sequence.** We additionally ablate the size of the bit sequence in Table 6. For each sequence size  $n$ , if both lists should have the same size, we choose  $\frac{2^n}{2}$  out of  $2^n$  possibilities without replacement, so there are  $\binom{2^n}{2}$  possibilities of constructing the  $G$  and  $R$ . For the 2-bit sequences, flipping the green and red list has only limited impact on the final z-score, thus

for the 3-bit sequences we only analyze all unique choices, *i.e.*, we do not interchange green and red list, leaving us with 35 distinct possible green and red lists. For  $n = 3$ , two  $G$  have a similar  $z$  compared to the best 2-bit sequences. These choices have the same properties as discussed above, namely (1) they equally often bias towards 0 and 1 and (2) none of the sequences share the same prefix, *i.e.*, the biasing is always effective.

Table 6: **Green and Red list selection.** We analyze all possible green and red lists for  $n = 1$ ,  $n = 2$  and  $n = 3$  with  $\delta = 2$  and 1,000 images. We also report the fraction of 1's that are in the bit representation during generation of the image (**1s Gen**) and after re-encoding the image using the image encoder (**1s Enc**).

Green List $G$	Red List $R$	Average Z-Score	1s Gen	1s Enc
$\{0\}$	$\{1\}$	6.18	31.50%	49.47%
$\{1\}$	$\{0\}$	2.39	64.52%	50.21%
$\{11, 00\}$	$\{01, 10\}$	91.35	49.74%	49.98%
$\{01, 10\}$	$\{00, 11\}$	90.33	49.97%	49.99%
$\{00, 10\}$	$\{01, 11\}$	6.52	31.85%	49.44%
$\{11, 01\}$	$\{00, 10\}$	2.56	63.70%	50.22%
$\{00, 01\}$	$\{10, 11\}$	0.25	49.92%	49.98%
$\{11, 10\}$	$\{00, 01\}$	-0.25	49.92%	49.98%
$\{000, 011, 100, 111\}$	$\{001, 010, 101, 110\}$	88.00	49.75%	49.98%
$\{000, 011, 110, 111\}$	$\{001, 010, 100, 101\}$	87.91	49.75%	49.98%
$\{000, 011, 101, 111\}$	$\{001, 010, 100, 110\}$	67.90	49.75%	49.98%
$\{000, 100, 110, 111\}$	$\{001, 010, 011, 101\}$	44.95	40.92%	49.52%
$\{000, 001, 011, 111\}$	$\{010, 100, 101, 110\}$	32.91	58.19%	50.24%
$\{000, 001, 010, 101\}$	$\{011, 100, 110, 111\}$	32.85	44.09%	49.68%
$\{000, 010, 011, 111\}$	$\{001, 100, 101, 110\}$	29.04	40.92%	49.52%
$\{000, 100, 101, 111\}$	$\{001, 010, 011, 110\}$	29.04	40.92%	49.52%
$\{000, 101, 110, 111\}$	$\{001, 010, 011, 100\}$	26.43	40.92%	49.52%
$\{000, 011, 100, 110\}$	$\{001, 010, 101, 111\}$	20.11	49.75%	49.98%
$\{000, 001, 011, 110\}$	$\{010, 100, 101, 111\}$	10.03	58.19%	50.24%
$\{000, 001, 011, 100\}$	$\{010, 101, 110, 111\}$	8.72	58.19%	50.24%
$\{000, 001, 010, 100\}$	$\{011, 101, 110, 111\}$	6.48	44.09%	49.68%
$\{000, 010, 100, 101\}$	$\{001, 011, 110, 111\}$	6.20	33.07%	49.48%
$\{000, 010, 100, 110\}$	$\{001, 011, 101, 111\}$	5.99	33.07%	49.48%
$\{000, 010, 011, 100\}$	$\{001, 101, 110, 111\}$	5.55	40.92%	49.52%
$\{000, 001, 010, 110\}$	$\{011, 100, 101, 111\}$	3.74	44.09%	49.68%
$\{000, 010, 101, 110\}$	$\{001, 011, 100, 111\}$	3.21	33.07%	49.48%
$\{000, 010, 011, 110\}$	$\{001, 100, 101, 111\}$	2.93	40.92%	49.52%
$\{000, 100, 101, 110\}$	$\{001, 010, 011, 111\}$	2.93	40.92%	49.52%
$\{000, 010, 100, 111\}$	$\{001, 011, 101, 110\}$	2.14	33.07%	49.48%
$\{000, 001, 100, 101\}$	$\{010, 011, 110, 111\}$	0.25	49.92%	49.98%
$\{000, 001, 010, 011\}$	$\{100, 101, 110, 111\}$	0.25	49.92%	49.98%
$\{000, 011, 100, 101\}$	$\{001, 010, 110, 111\}$	0.11	49.75%	49.98%
$\{000, 001, 101, 110\}$	$\{010, 011, 100, 111\}$	0.07	49.92%	49.98%
$\{000, 011, 101, 110\}$	$\{001, 010, 100, 111\}$	0.02	49.75%	49.98%
$\{000, 001, 100, 110\}$	$\{010, 011, 101, 111\}$	-0.05	49.92%	49.98%
$\{000, 001, 101, 111\}$	$\{010, 011, 100, 110\}$	-0.26	49.92%	49.98%
$\{000, 001, 100, 111\}$	$\{010, 011, 101, 110\}$	-0.37	49.92%	49.98%
$\{000, 001, 110, 111\}$	$\{010, 011, 100, 101\}$	-0.55	49.92%	49.98%
$\{000, 010, 101, 111\}$	$\{001, 011, 100, 110\}$	-0.65	33.07%	49.48%
$\{000, 010, 110, 111\}$	$\{001, 011, 100, 101\}$	-0.85	33.07%	49.48%
$\{000, 001, 011, 101\}$	$\{010, 100, 110, 111\}$	-1.45	58.19%	50.24%
$\{000, 010, 011, 101\}$	$\{001, 100, 110, 111\}$	-12.98	40.92%	49.52%
$\{000, 001, 010, 111\}$	$\{011, 100, 101, 110\}$	-13.39	44.09%	49.68%

**Consistency in Longer Sequences.** We analyze the overlap of bit sequences with growing size in Figure 3 which shows that with increased length of the sequence, the consistency of the pattern



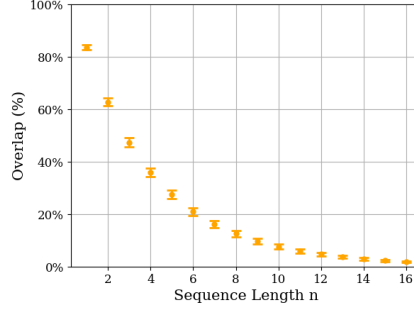


Figure 3: **Longer sequences are less consistent after re-encoding.** We analyze the consistency of sequences of length  $n$  after re-encoding over the whole image, following the setup of Figure 1.

decreases. This has a direct impact on the choice of  $n$ , as it shows that larger  $n$  lead to less overlap, decreasing the impact of BitMark which is already noticeable for  $n = 3$  (see Table 6).

## C Experimental Setup

### C.1 Hyperparameters

For the experiments on radioactivity, we chose 1,000 prompts from the MS-COCO 2014 validation dataset, generated the respective number of images with the  $M_1$  Infinity model and then fine-tuned the second model  $M_2$  for 5 epochs with a learning rate of  $10^{-4}$  and a batch-size of 4 (batch-size of 1 for Infinity-2B).

Since VAR [32] and RAR [37] are class-conditional models for the ImageNet1k dataset, we first generated ImageNet fakes using Infinity-2B with the ImageNet class name and "A photo of {class name}" as prompt. We used the same hyperparameters such as batch-size, number of epochs and learning rate consistent with the experiments on Infinity-2B and Stable Diffusion 2.1.

### C.2 Environment

Our experiments are performed on Ubuntu 22.04, with Intel(R) Xeon(R) Gold 6330 CPU and NVIDIA A40 Graphics Card with 40 GB of memory.

## D Natural Images

Table 7: **Detection on non-watermarked samples.** Mean (std) of  $z$  and green fractions for 1,000 non-watermarked samples.

Dataset	$z$	Green Fraction
Imagenet1k	0.6311 (1.966)	0.5005 (0.0017)
MS-COCO2014	0.6773 (1.735)	0.5006 (0.0015)
MS-COCO2014 (Infinity-2B generated)	0.3620 (1.641)	0.5003 (0.0014)

To ensure the validity of our hypothesis we compute the average number of members of the green list and  $z$  for natural images from different datasets, as well as images generated for the non-watermarked Infinity model. Table 7 shows that indeed, the green fraction for non-watermarked images is 50%.

## E Additional Experiments

### E.1 Detailed Analysis of BitMark

Next, we analyze the low-level behavior of BitMark during image generation. Table 9 depicts information about the generation of images with Infinity, averaged over 10,000 images. It shows that

the entropy is low in earlier scales, meaning the bits are chosen with high certainty as these depend more the input prompt, but large on later scales, which means sampling either of the possible values for the bit can lead to different, but both high-quality content. This insight aligns with the re-encoding loss, which keeps more bits consistent on earlier scales.

Table 10 depicts the low-level information of applying BitMark with  $\delta = 2$ . Applying BitMark changes up to 37% of bits per scale, yet the re-encoding loss only increases by 5% compared to the non watermarked image. Table 8 further shows the correlation between different analyzed factors. As expected, there is a high correlation between changed bits and entropy, as due to the soft-biasing we mostly change bits which have low entropy.

Table 8: **Correlations of different factors when applying BitMark on all scales within the Infinity generation process.** Entropy, Green Fraction, Changed Bits & Re-Encoding Loss are relative values. The Re-Encoding Loss displays the number of bits changed during re-encoding, whereas Changed Bits display the number of bits changed during the generation process. 10,000 images,  $\delta = 2$ .

	Entropy	Changed Bits	$z$	Green Fraction
Changed Bits	0.999	-	-	-
$z$	0.706	0.689	-	-
Green Fraction	0.918	0.917	0.642	-
Re-Encoding Loss	0.926	0.931	0.475	0.751

Table 9: **Statistics regarding the infinity generation process for 10,000 images.** Entropy, Green Fraction & Re-Encoding Loss are relative values. The Re-Encoding Loss displays the number of bits changed during re-encoding. We report the mean (std) for all values.

Scale	Entropy	$z$	Green Fraction	Re-Encoding Loss	Number of Tokens (Bits)
1	0.051 (0.037)	0.01 (0.876)	0.501 (0.079)	0.068 (0.063)	1 (32)
2	0.108 (0.044)	-0.195 (1.114)	0.491 (0.049)	0.132 (0.063)	4 (128)
3	0.133 (0.034)	-0.463 (1.226)	0.49 (0.027)	0.145 (0.041)	16 (512)
4	0.178 (0.029)	-0.522 (1.217)	0.492 (0.018)	0.2 (0.031)	36 (1,152)
5	0.202 (0.024)	-0.558 (1.199)	0.494 (0.013)	0.226 (0.024)	64 (2,048)
6	0.215 (0.021)	-0.663 (1.229)	0.495 (0.009)	0.247 (0.02)	144 (4,608)
7	0.223 (0.019)	-0.5 (1.266)	0.497 (0.007)	0.239 (0.017)	256 (8,192)
8	0.223 (0.015)	-0.349 (1.187)	0.498 (0.005)	0.277 (0.016)	400 (12,800)
9	0.226 (0.014)	-0.122 (1.173)	0.5 (0.004)	0.279 (0.014)	576 (18,432)
10	0.243 (0.014)	0.157 (1.165)	0.5 (0.003)	0.264 (0.012)	1,024 (32,768)
11	0.237 (0.012)	0.227 (1.161)	0.501 (0.003)	0.319 (0.011)	1,600 (51,200)
12	0.233 (0.012)	0.329 (1.198)	0.501 (0.002)	0.328 (0.01)	2,304 (73,728)
13	0.234 (0.014)	0.462 (1.173)	0.501 (0.002)	0.212 (0.012)	4,096 (131,072)
1-13	0.193 (0.012)	0.232 (1.644)	0.497 (0.008)	0.266 (0.009)	10,521 (336,372)

**Watermarking in Early vs Late Scales.** We provide both qualitative and quantitative analysis for the scales in the Infinity architecture. We observe that the model produces high-level information in the image during the initial scales, forming a coarse shape of the main visual components, such as the object and the background (see Figure 4). In larger scales, the model predicts the details of the image (see Figure 5). As shown in Table 11, earlier and late scales contribute differently to the robustness of BitMark. Earlier scales are more robust against like CtrlRegen, as CtrlRegen does not change the content of the image itself.

## E.2 Image Quality

We analyze the impact of applying BitMark with different  $\delta$  on the image quality. Figure 6 shows that applying BitMark with  $\delta \leq 3$  keeps the semantic structure and quality of the image the same, while changing small features, *e.g.*, the color of the train in the first row. For larger watermarking strengths, the image quality starts to deteriorate, where artifacts become visible in the final image.

## E.3 Timing

We present the average elapsed time (in seconds) for the generation of an image with Infinity, when the watermark is applied to all scales (resolutions) and when using the standard generation process

Table 10: **Statistics regarding the infinity generation process with BitMark applied on all scales for 10,000 images,  $\delta = 2$ .** Entropy, Green Fraction, Re-Encoding Loss & Changed Bits are relative values. The Re-Encoding Loss displays the number of bits changed during re-encoding, whereas Changed Bits display the number of bits changed during the generation process. We report the mean (std) for all values.

Scale	Entropy	$z$	Green Fraction	Re-Encoding Loss	Changed Bits
1	0.051 (0.037)	0.218 (0.897)	0.52 (0.081)	0.088 (0.076)	0.09 (0.077)
2	0.108 (0.043)	0.854 (1.105)	0.538 (0.049)	0.181 (0.079)	0.179 (0.075)
3	0.132 (0.032)	2.37 (1.452)	0.552 (0.032)	0.2 (0.053)	0.208 (0.05)
4	0.175 (0.028)	3.965 (1.613)	0.558 (0.024)	0.26 (0.037)	0.279 (0.042)
5	0.2 (0.024)	6.252 (1.812)	0.569 (0.02)	0.286 (0.029)	0.313 (0.035)
6	0.213 (0.021)	10.118 (2.304)	0.575 (0.017)	0.305 (0.022)	0.329 (0.029)
7	0.224 (0.019)	15.701 (2.832)	0.587 (0.016)	0.3 (0.02)	0.346 (0.027)
8	0.224 (0.016)	16.247 (2.788)	0.572 (0.012)	0.329 (0.016)	0.345 (0.022)
9	0.228 (0.015)	20.876 (3.341)	0.577 (0.012)	0.333 (0.013)	0.351 (0.02)
10	0.244 (0.015)	32.837 (5.104)	0.591 (0.014)	0.319 (0.011)	0.373 (0.019)
11	0.237 (0.013)	29.535 (5.06)	0.565 (0.011)	0.361 (0.009)	0.363 (0.017)
12	0.242 (0.013)	34.973 (6.681)	0.564 (0.012)	0.37 (0.008)	0.368 (0.017)
13	0.247 (0.014)	64.333 (11.823)	0.589 (0.016)	0.256 (0.015)	0.374 (0.019)
1-13	0.194 (0.012)	90.784 (14.852)	0.566 (0.013)	0.312 (0.009)	0.366 (0.017)

Table 11: **Robustness of BitMark applied to different scales with  $\delta = 2$ .** We report the TPR@1%FPR (%) for the different attacks on 5,000 watermarked images.

Scales	Conventional Attacks						Reconstruction Attacks		
	None	Noise	Blur	Color	Crop	Rotation	JPEG	SD2.1-VAE	CtrlRegen+
1-10	100.0	97.9	98.7	97.6	61.6	15.0	100.0	100.0	82.1
11-13	100.0	96.7	98.6	99.5	20.0	9.8	100.0	100.0	15.5
1-13	100.0	99.6	99.9	99.8	98.8	20.1	100.0	100.0	91.6

(without a watermark). For hardware information see Appendix C.2. Table 12 shows that applying BitMark results in negligible overhead to image generation and allows for a fast detection in under 0.5 seconds.

Table 12: **Timing results for image processing and detection per image.** We report the mean (std) for 1,000 images.

Setup	Seconds/Image
BitMark	2.5817 (0.0063)
No watermark	2.3196 (0.0043)
Detection	0.4491 (0.0046)

#### E.4 Radioactivity

Table 13: **We analyze the radioactivity of BitMark for class-conditional autoregressive models.** We report the TPR@1%FPR (%).

Type of $M_1$	Type of $M_2$	Output of $M_1$	Output of $M_2$
Infinity-2B	VAR-16	100.0	24.2
Infinity-2B	VAR-20	100.0	25.8
Infinity-2B	VAR-24	100.0	25.7
Infinity-2B	VAR-30	100.0	25.6
Infinity-2B	RAR-B	100.0	4.3
Infinity-2B	RAR-L	100.0	3.3
Infinity-2B	RAR-XL	100.0	3.9
Infinity-2B	RAR-XXL	100.0	4.1

Table 13 reports the radioactivity of BitMark for the class-conditional autoregressive models, namely, VAR [32], which operates in the multi-scale manner similarly to Infinity, and RAR [37], which generates tokens only on a single scale in an autoregressive fashion. We find that BitMark exhibits detectable traces of radioactivity for the class-to-image autoregressive models with

966 TPR  $\gg 1\%$  @ FPR = 1%, with slightly lower traces for RAR (probably due to its single scale genera-  
 967 tion) than VAR. A potential cause of the overall relatively lower radioactivity for VAR and RAR is  
 968 stemming from the class-conditional setting, instead of the text-to-image as in Infinity.

## 969 E.5 Rotation

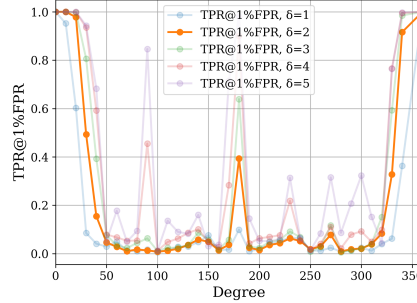


Figure 7: **BitMark is robust against rotation within  $\pm 30$  degrees.** The TPR@1%FPR for different  $\delta$  and rotation degrees.

970 We ablate the robustness of BitMark against different degrees of rotation and different watermarking  
 971 strengths  $\delta$ . We find that with  $\delta = 2$  BitMark is robust for rotation degrees  $\pm 30$ . Additionally we  
 972 find an increase in TPR@1%FPR for a rotation of  $180^\circ$ .

## 973 F Analysis for the Watermarks in the Sand Attack

974 As shown by the qualitative analysis in Figure 8, the watermarks in the sand attack causes a significant  
 975 loss of details and produces many perceptible artifacts, limiting the applicability of this attack.

## 976 G Analysis for the Bit-Flipper Attack

977 Figure 9 shows different flipping strengths  $\phi$  for  $\delta = 2$  on 1,000 watermarked samples. The most  
 978 effective  $\phi$  is  $\phi = 2.2$  with a TPR@1%FPR below 50% (see Table 14). The decrease in image quality  
 979 for Bit-Flipper for different  $\phi$  is shown in Figure 9, as well a more detailed analysis of  $\phi = 2.2$  on  
 980 BitMark with a  $\delta = 2$  is displayed in Figure 10.

Table 14: **Applying Bit-Flipper with different factors  $\phi$  on watermarked images with  $\delta = 2$ .** We report the TPR@1%FPR for 1,000 Images.

$\phi$	TPR@1%FPR (%)	$z$
1.8	0.878	10.06 (4.24)
2.0	0.845	9.17 (4.16)
2.2	0.472	5.52 (5.07)
2.4	0.595	8.06 (7.28)
2.6	0.52	6.97 (6.96)

## 981 H Limitations and Broader Impact

982 **Limitations.** While BitMark are robust against moderate rotations (99.9% TPR@1%FPR within  
 983 rotation angles  $-20^\circ$  to  $20^\circ$ ), the detection accuracy degrades with larger rotation angles, dropping to  
 984 56.8% TPR@1%FPR for  $\pm 50^\circ$  rotations. Although open-source methods have a strong capability for  
 985 rotation correction, this can be a limitation for real-world scenarios. This rotation sensitivity, however,  
 986 is an inherent issue in Infinity’s data augmentation pipeline, which makes the autoencoder less robust  
 987 to large rotations. Moreover, the soft biasing parameter  $\delta$  requires tuning, as values exceeding  $\delta = 3$

lead to noticeable quality degradation (FID increases from 29.61 to 127.44 when  $\delta$  increases from 3 to 5), constraining the maximum achievable watermark strength. Although we evaluate the detection BitMark with a commonly adopted metric, *i.e.*, true positive rate at very low false positive rates (1%), it could potentially become problematic in extreme-scale applications involving trillions of samples, where even small false positive rates might impact training data quality.

**Broader Impact.** Image generative models have the capabilities of generating photorealistic images, which are near imperceptible to human generated content. BitMark provides a method to watermark the content, without harming image generation capabilities of the model and allows for provenance tracking of generated images for models hosted under a private API. It supports the defense against model collapse, when training a model on data generated by itself.

## I Additional Related Work

**LLM Watermarking.** Here, we review more related work regarding the LLM watermarking, which we build upon in our method. Beyond the soft biasing approach proposed by KGW [14], SynthID-Text [5] introduces tournament sampling during the model inference to enhance watermark detectability. This sampling strategy modifies the token sampling process to embed watermark signals. However, tournament sampling is only applicable to autoregressive models with large vocabulary sizes, making it incompatible with the binary bit sampling process used in Infinity. There are also works that incorporate watermarks by training or finetuning the models. For example, REMARK-LLM trains a learning-based encoder for watermark insertion and a decoder for watermark extraction, while using a reparameterization module to bridge the gap between the token distribution and the watermark message [39]. Moreover, potential malicious transformations are incorporated into the training phase to enhance the robustness of the watermarking. Although effective, these training-based methods require substantial computational resources and time. Therefore, we focus on inference-time watermarking methods that can be applied to Infinity without any retraining.

Sander et al. [28] investigate the *radioactivity* of LLM watermarking, which is the ability of watermarks to transfer across different model architectures. They design a specific protocol for amplifying the watermark signals after transferring across models. The radioactivity of watermarks also falls into the scope of our work, as our proposed watermark exhibits significant radioactivity across various model architectures. Regarding the undetectability of the watermark, recent studies show that the biases between watermarked and non-watermarked outputs from the LLM are identifiable with carefully designed prompts [17]. As a specialized application of LLM watermarking, Li et al. [15] design a watermarking scheme tailored for quantized LLMs. The proposed watermarks are only detectable on quantized models while remaining invisible for full-precision models.

**Model Collapse.** With the fraction of artificial generated content within the World Wide Web increasing, Shumailov et al. [31] investigate the phenomenon of model collapsing, which is defined as the decrease of generation quality when models are recursively trained on (partly) generated data. This decrease in generation quality is mainly characterized in overestimating probable events and underestimating low-probability events, as well as recursively adding noise (*i.e.*, errors) which is generated during earlier model generations. Hence, this generated data is polluting future datasets, decreasing the potential quality of successor models and therefore highlighting the necessity to distinguish human- from artificial-generated data. BitMark mitigates model collapsing by robustly watermarking generated images to exclude said marked images from future datasets.



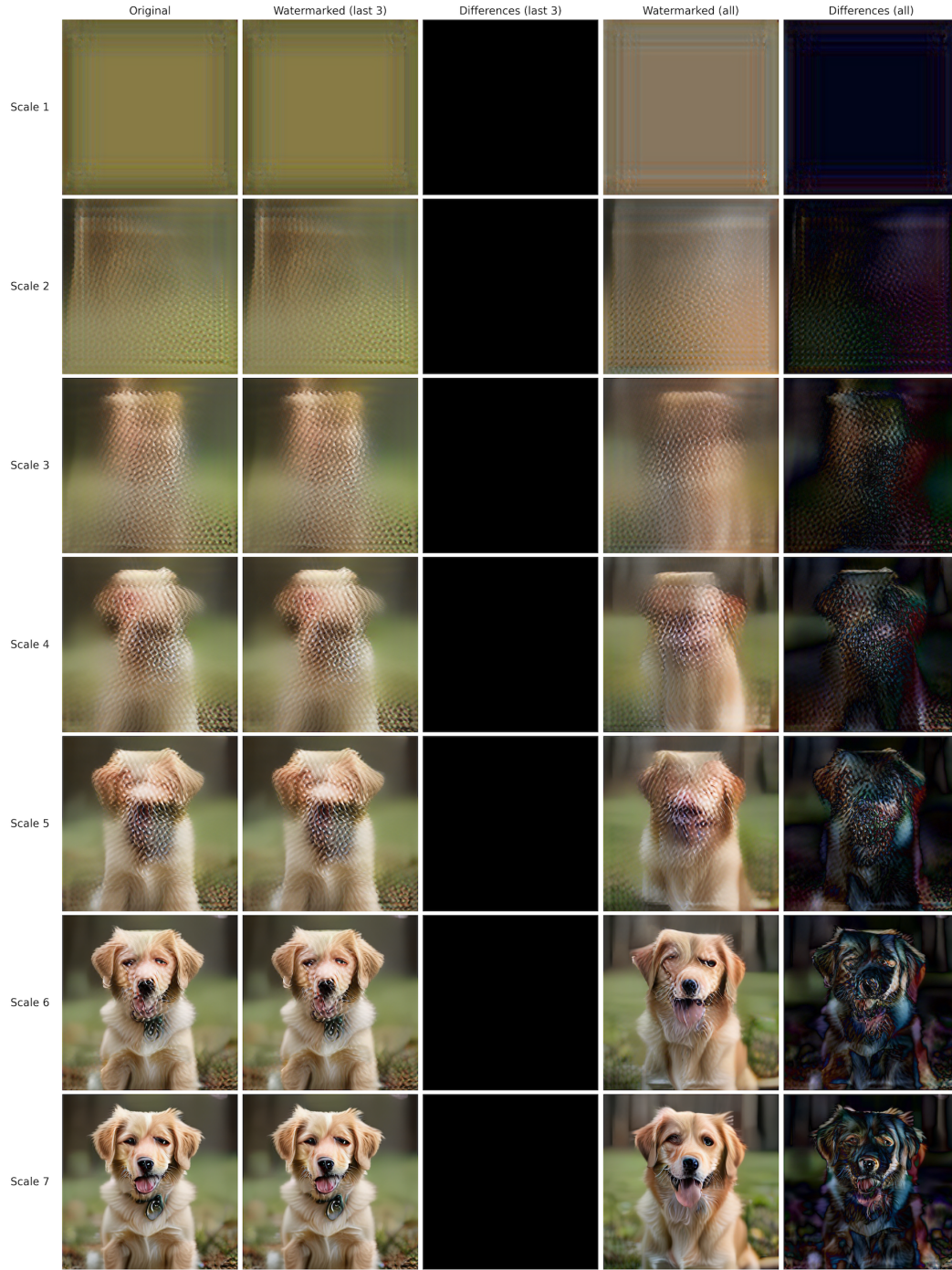


Figure 4: **Visualization of different images on different scales.** Here, the image on scale  $i$  refers to the cumulative image obtained by adding up 1 to  $i$  scales. We visualize five sets of images: **1) Original** images without watermarking, **2) Watermarked (last 3)**, where the watermarks are generated on scale 11 to 13, **3) Differences (last 3)**, which is the difference between the original and watermarked images (last 3), **4) Watermarked (all)**, where the watermarks are generated on all scales, **5) Differences (all)**, which is the difference between the original and watermarked images (all). Here, we visualize scales 1-7.



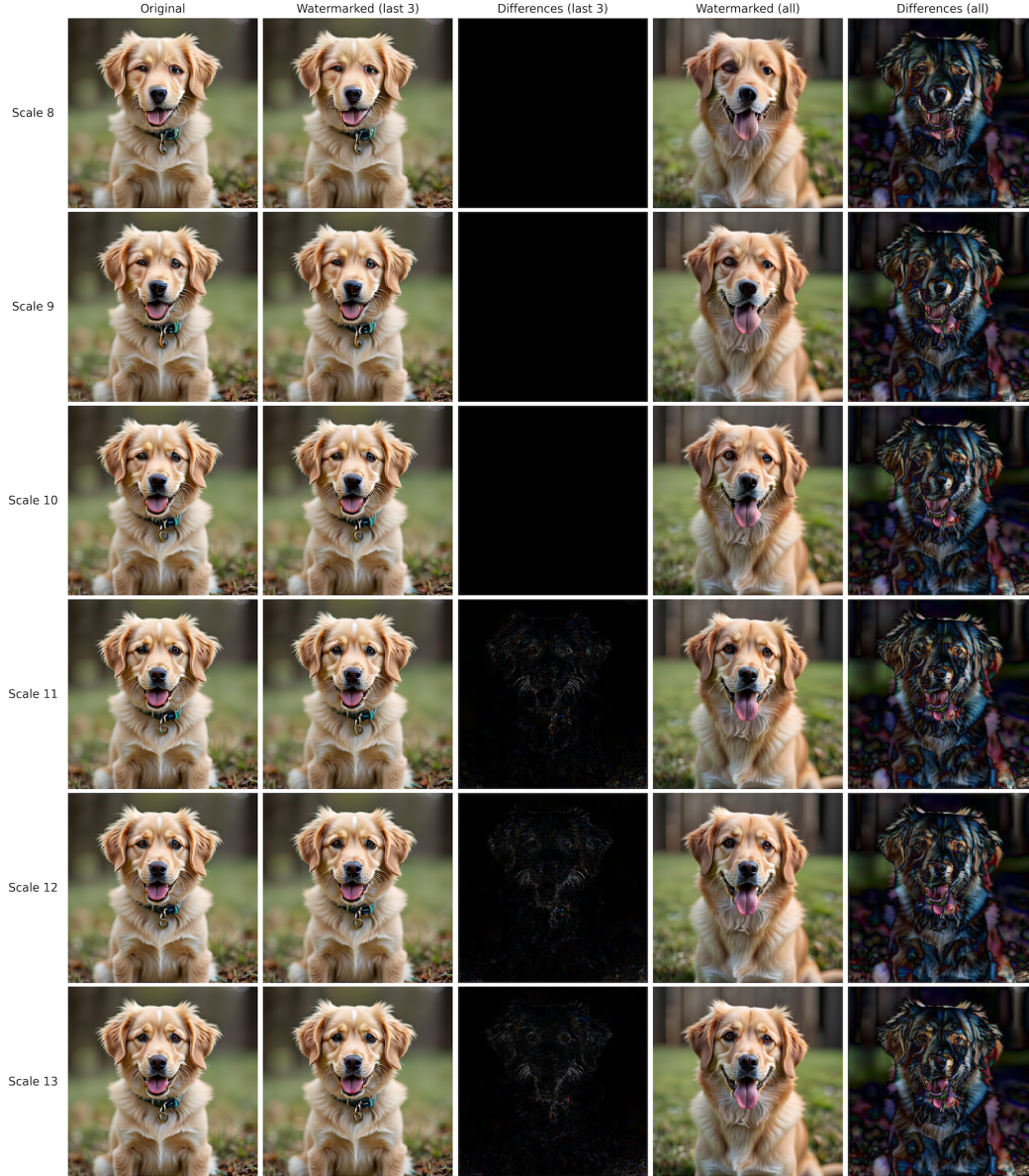


Figure 5: **Visualization of different images on different scales.** Here, the image on scale  $i$  refers to the cumulative image obtained by adding up 1 to  $i$  scales. We visualize five sets of images: **1) Original** images without watermarking, **2) Watermarked (last 3)**, where the watermarks are generated on scale 11 to 13, **3) Differences (last 3)**, which is the difference between the original and watermarked images (last 3), **4) Watermarked (all)**, where the watermarks are generated on all scales, **5) Differences (all)**, which is the difference between the original and watermarked images (all). Here, we visualize scales 8-13.

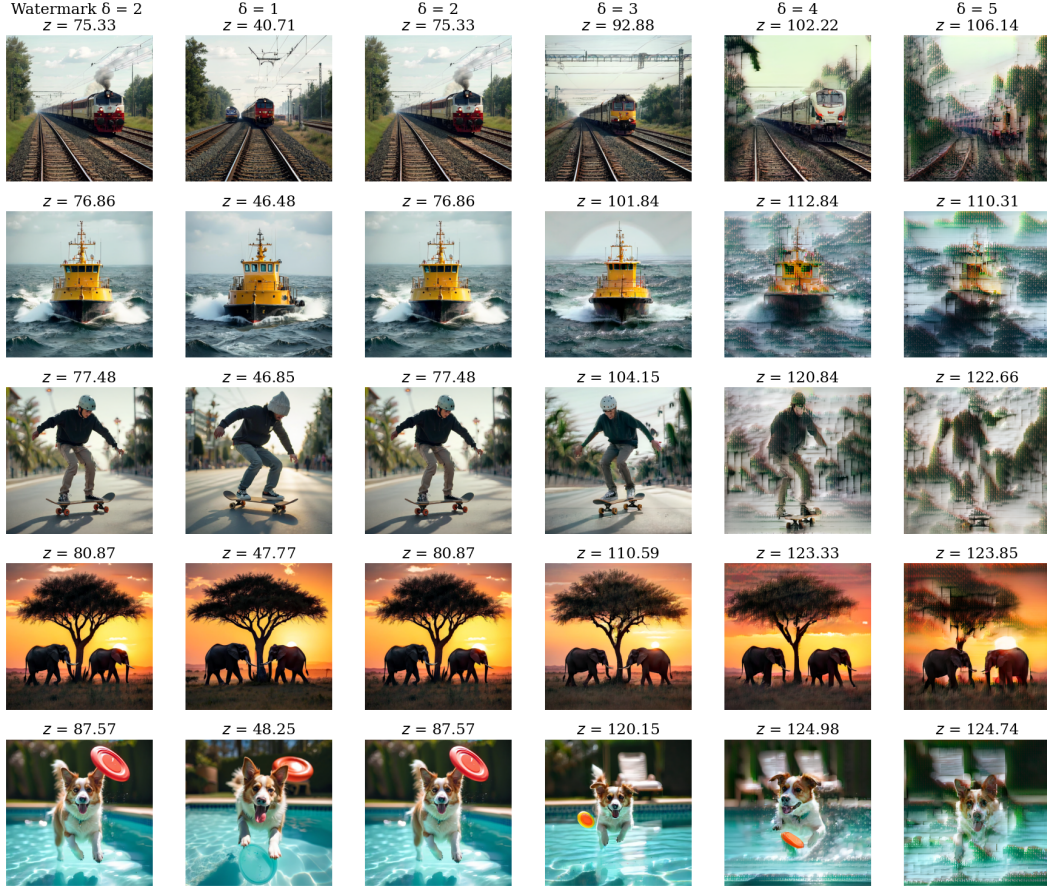


Figure 6: **Artifacts start to form in high detailed images (third row) already with  $\delta = 3$ .** Qualitative analysis of the impact of the choice of watermarking strength ( $\delta$ ) on the perceived image quality.



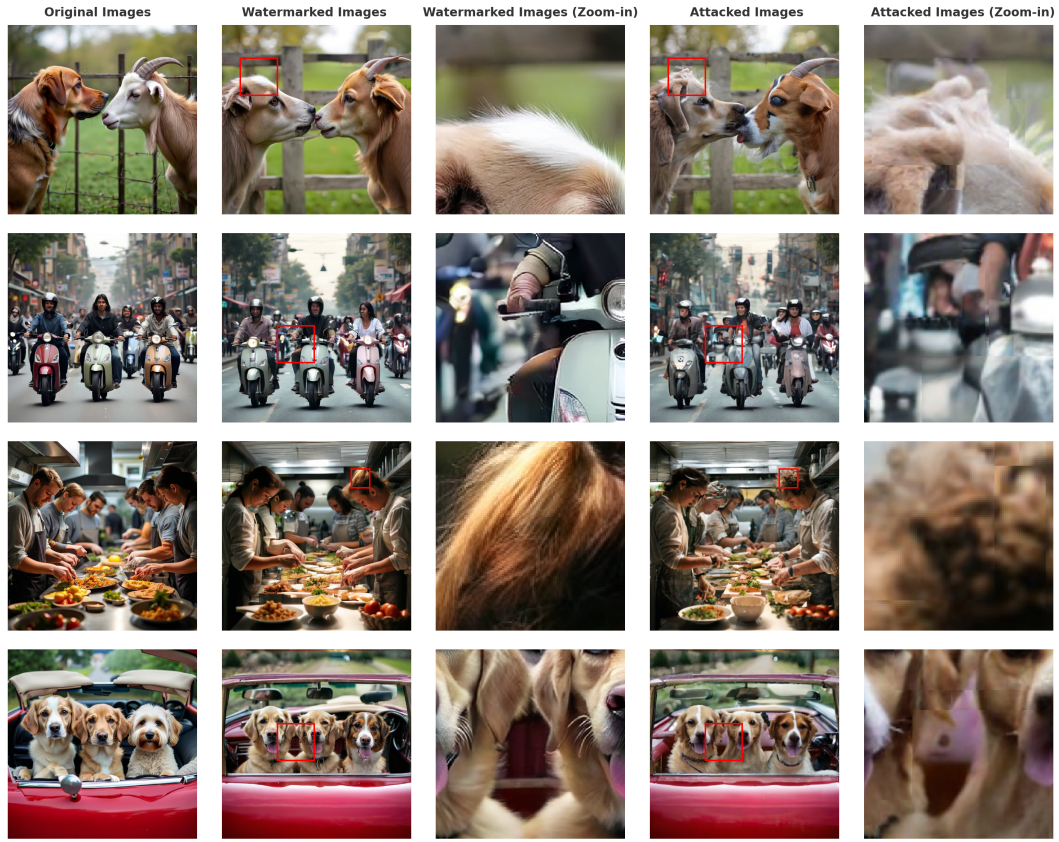


Figure 8: **Watermarks in the Sand has non-negligible impact on image quality.** Qualitative analysis of the impact of the watermarks in the sand. Here, we zoom in on some regions in the image to demonstrate the quality loss in details.

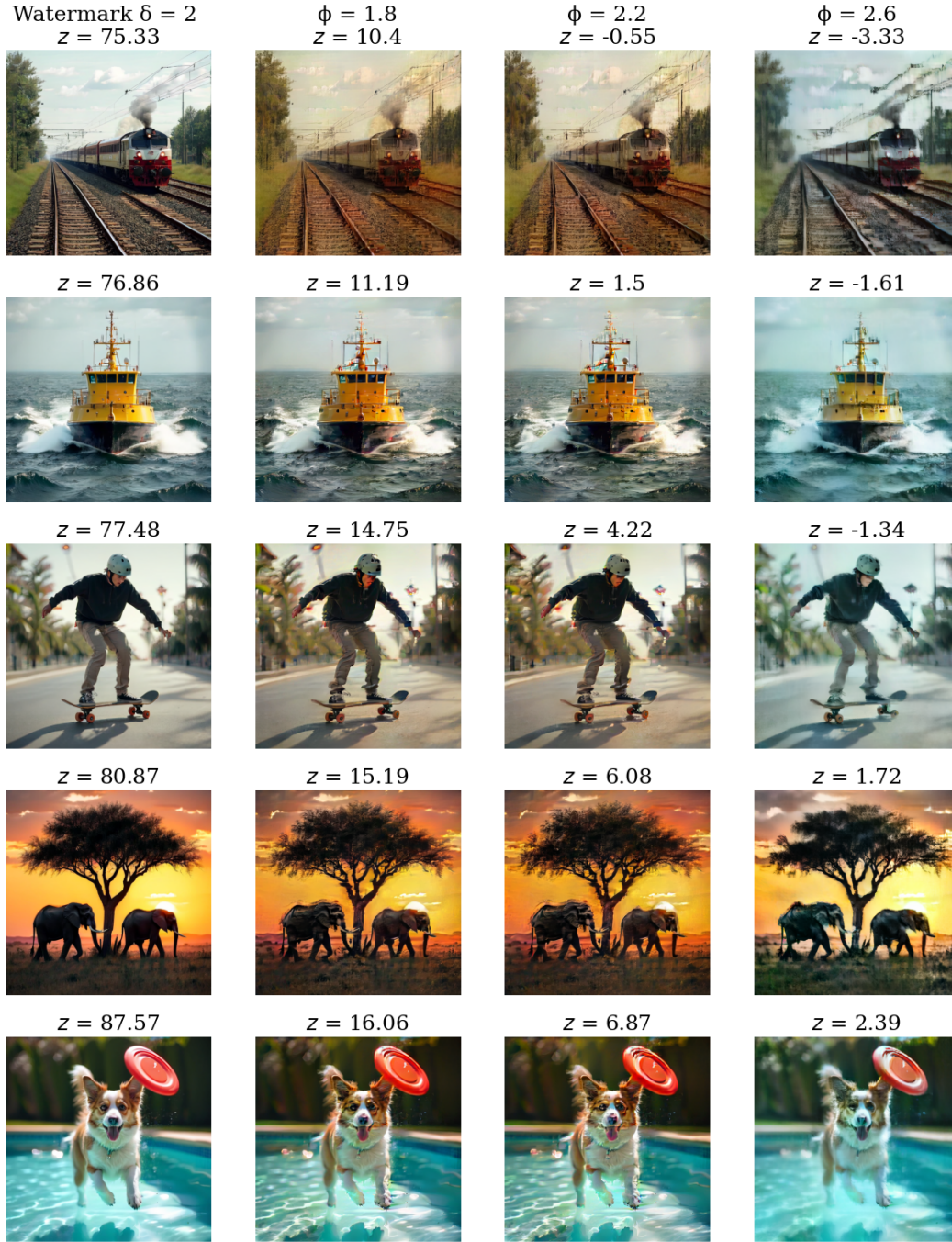


Figure 9: A higher flip factor leads to worse image quality but better watermark removal. Qualitative analysis of the impact of the Bit-Flipper with different  $\phi$ .



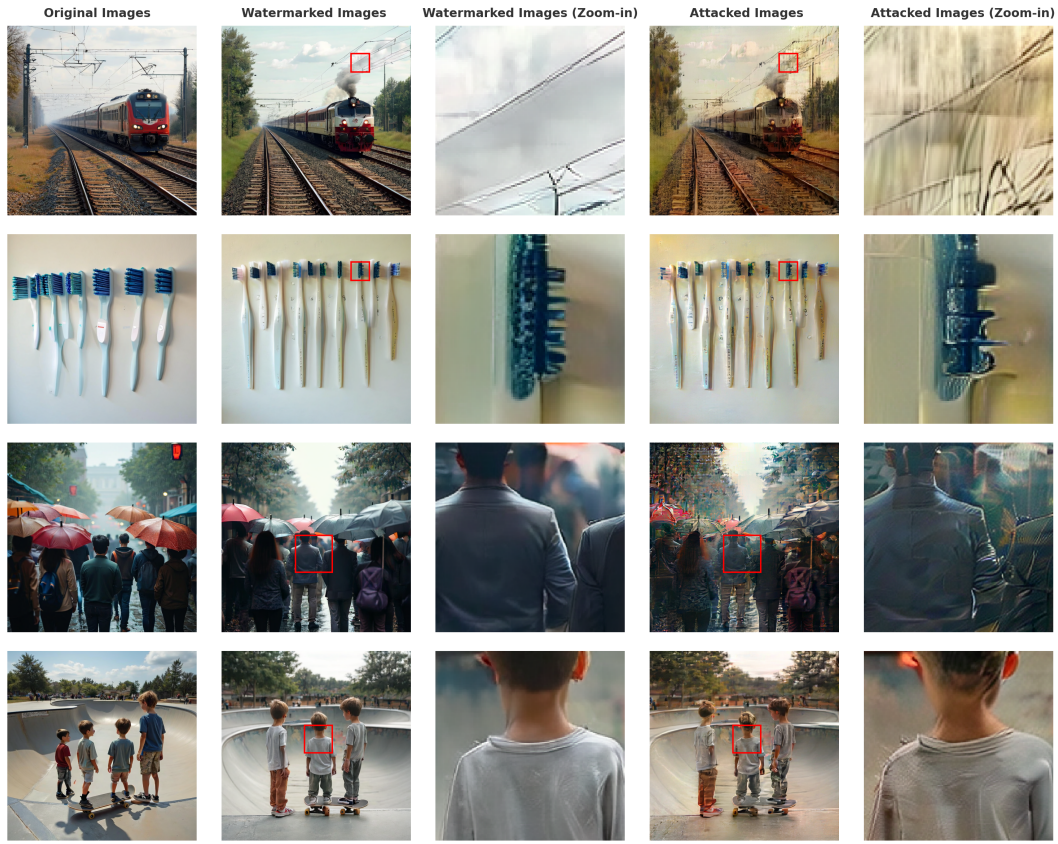


Figure 10: **The Bit-Flipper has strong impact on the image quality.** Here, we zoom in on some regions in the image to demonstrate the quality loss in details.  $\phi = 2.2$ ,  $\delta = 2$ .