Gaussian Shannon: High-Precision Diffusion Model Watermarking Based on Communication

Supplementary Material

1. Details of Gaussian Shannon

1.1. Diffusion Models

DDPM. Denoising Diffusion Probabilistic Models are based on a forward diffusion process that gradually adds Gaussian noise to data and a reverse process that learns to remove it. In the forward process, a clean sample x_0 from the data distribution $q(x_0)$ is corrupted over T time steps according to

$$q(x_t|x_{t-1}) = \mathcal{N}(x_t; \sqrt{1-\beta_t}, x_{t-1}, \beta_t I),$$

where $\{\beta\}_{t=1}^{T}$ are small positive constants controlling noise magnitude. The marginal distribution can be expressed as

$$q(x_t|x_0) = \mathcal{N}(x_t; \sqrt{\bar{\alpha}_t}x_0, (1-\bar{\alpha}_t)I),$$

where $\bar{\alpha}_t = \prod_{s=1}^{t}(1-\beta_s)$. The reverse process aims to learn $p_\theta(x_{t-1}|x_t)$, modeled as

$$p_\theta(x_{t-1}|x_t) = \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \Sigma_\theta(x_t, t)),$$

with the mean predicted using a neural network $\epsilon_\theta(x_t, t)$ that estimates the added noise. The training objective minimizes the mean squared error between true and predicted noise:

$$\mathcal{L} = \mathbb{E}_{t,x_0,\epsilon}\left[||\epsilon - \epsilon_\theta(\sqrt{\bar{\alpha}_t}x_0 + \sqrt{1-\bar{\alpha}_t}\epsilon, t)||^2\right].$$

DDIM. DDIM reformulate the sampling process as a non-Markovian deterministic process that reduces the number of sampling steps. Instead of drawing new noise at each step, DDIM defines a deterministic mapping between x_t and x_{t-1} :

$$x_{t-1} = \sqrt{\bar{\alpha}_{t-1}}x_0 + \sqrt{1-\bar{\alpha}_{t-1}}\epsilon_{\theta(x_t,t)}.$$

where x_0 is estimated as $x_0 = \frac{x_t - \sqrt{1-\bar{\alpha}_t}, \epsilon_\theta(x_t,t)}{\sqrt{\bar{\alpha}_t}}$.

DDIM Inversion. DDIM inversion is the reverse operation of the DDIM sampling process, mapping a real image x_0 to its corresponding latent noise x_t. The DDIM formula originally derives the relationship from x_t to x_{t-1}, which can be transformed to obtain the relationship from x_{t-1} to x_t:

$$x_t = \sqrt{\frac{\bar{\alpha}_t}{\bar{\alpha}_{t-1}}}x_{t-1} + \sqrt{\bar{\alpha}_t}\left(\sqrt{\frac{1}{\bar{\alpha}_t} - 1} - \sqrt{\frac{1}{\bar{\alpha}_{t-1}} - 1}\right)\epsilon_\theta(x_t, t).$$

Initially, we only have x_0, and when calculating each step x_t, we also need to know $\epsilon_\theta(x_t, t)$, but x_t itself is unknown. Considering the assumption that ODE-based processes can be reversed under the constraint of small step sizes, we can use $\epsilon_\theta(x_{t-1}, t)$ to approximate $\epsilon_\theta(x_t, t)$.

1.2. Details of Method

Setup of regular LDPC codes. We use the regular LDPC codes from the pyldpc library for experiments. To generate the parity-check matrix H for sparse representation of regular LDPC codes, three key parameters must be provided: code length n, variable node degree d_v, and check node degree d_c. In our experiments, the information bit length is fixed at 256 bits, the code length varies with the code rate, and d_v and d_c are set to 3 and 4 by default. During decoding, the number of belief propagation iterations for LDPC decoding is fixed at 100.

2. Details of Experiments

2.1. Metrics Details.

BitAcc. The extracted bit sequence from the watermarked image is compared bit-by-bit with the embedded bit sequence. BitAcc(w,w') represents the proportion of matching bits between the bit sequences w and w'.

TPR@10^{-6}FPR. This metric indicates the TPR (True Positive Rate) when the FPR (False Positive Rate) reaches 10^{-6} (i.e., an extremely low false detection rate). In practical watermark detection, a threshold τ corresponding to an FPR of 10^{-6} needs to be calculated in advance. When BitAcc(w,w') exceeds this threshold τ, it is recorded as a true positive. We refer to the method proposed by Yu et al.[9] to calculate τ. They assume that each extracted watermark bit w'_i from non-watermarked images follows a uniform distribution, $w'_i \sim$ Bernoulli(0.5). The FPR is defined as:

$$\text{FPR}(\tau) = \frac{1}{2^k}\sum_{i=\tau+1}^{k}\binom{k}{i} = B_{1/2}(\tau+1, k-\tau).$$

where k is the number of watermark bits, and $B_{1/2}$ is the regularized incomplete beta function.

In user traceability tasks, we need to assign a unique watermark w and a unique key K to each user. For a given image, we first conduct the aforementioned threshold passing test on the extracted watermark. In multi-user scenarios, we need to multiply the original FPR by the number of users to calculate the expected threshold τ:

$$\text{FPR}(\tau, N) = 1 - (1 - \text{FPR}(\tau))^N \approx N \cdot \text{FPR}(\tau).$$

If the bit matching exceeds the threshold, the image is considered generated by our model. Subsequently, the index with the highest matching count will be traced back to the corresponding user.

Noise	Methods						
	DwtDct [4]	DwtDctSvd [4]	Tree-Ring [7]	Stable Signature [5]	GaussianShading[8]	PRCW[6]	**Ours**
None	0.836/0.890/0.887	1.000/1.000/1.000	1.000/1.000/1.000	1.000/1.000/1.000	1.000/1.000/1.000	1.000/1.000/1.000	**1.000/1.000/1.000**
JPEG	0.008/0.006/0.005	0.015/0.017/0.015	0.998/0.999/1.000	0.912/0.709/0.847	1.000/1.000/1.000	0.998/0.999/0.999	**1.000/1.000/1.000**
RandDrop	0.385/0.458/0.481	0.557/0.483/0.451	0.984/0.885/0.915	1.000/1.000/0.999	0.998/0.999/0.999	0.864/0.895/0.873	**1.000/1.000/1.000**
GaussBlur	0.002/0.003/0.002	0.423/0.431/0.426	1.000/0.999/1.000	0.156/0.173/0.171	1.000/1.000/1.000	0.912/0.908/0.901	**1.000/1.000/1.000**
MedianFilter	0.007/0.008/0.007	0.989/0.999/0.999	1.000/1.000/1.000	0.112/0.108/0.112	0.999/1.000/0.999	0.812/0.803/0.819	**1.000/1.000/1.000**
GaussNoise	0.372/0.373/0.376	0.851/0.861/0.879	0.383/0.396/0.394	0.618/0.621/0.634	0.999/1.000/1.000	0.821/0.805/0.811	**1.000/1.000/1.000**
Resize	0.012/0.016/0.017	0.999/0.999/0.998	1.000/1.000/0.999	0.956/0.968/0.964	0.999/1.000/0.999	0.832/0.842/0.837	**1.000/1.000/1.000**
Brightness	0.369/0.354/0.358	0.359/0.354/0.376	0.998/0.996/0.995	0.999/0.999/0.998	0.998/0.994/0.996	0.746/0.586/0.598	**1.000/1.000/1.000**
Average	0.165/0.174/0.178	0.599/0.592/0.592	0.909/0.901/0.899	0.679/0.654/0.675	0.999/0.999/0.999	0.855/0.834/0.834	**1.000/1.000/1.000**

Table 1. Using SDv1.4, v2.0, and v2.1 models respectively with TPR@10^{-6}FPR as the metric, the comparison under different noise levels demonstrates that GaussianShannon achieves the best performance.

Noise	Methods						
	DwtDct [4]	DwtDctSvd [4]	Tree-Ring [7]	Stable Signature [5]	GaussianShading[8]	PRCW[6]	**Ours**
None	0.8106/0.8153/0.8149	0.9996/0.9983/0.9982	-	0.9946/0.9932/0.9923	0.9999/0.9999/0.9999	1.0000/1.0000/1.0000	**1.0000/1.0000/1.0000**
JPEG	0.4156/0.4260/0.4189	0.7053/0.7042/0.7098	-	0.8192/0.8196/0.9153	0.9896/0.9879/0.9904	0.9468/0.9562/0.9566	**0.9998/0.9987/0.9986**
RandDrop	0.6996/0.6815/0.6852	0.6886/0.6859/0.6848	-	0.7325/0.7366/0.7319	0.9623/0.9678/0.9701	0.8973/0.8842/0.8856	**0.9889/0.9918/0.9910**
GaussBlur	0.3696/0.3643/0.3652	0.5986/0.5972/0.5925	-	0.7563/0.7589/0.7512	0.9712/0.9753/0.9768	0.9156/0.9143/0.9131	**0.9912/0.9895/0.9891**
MedianFilter	0.7001/0.6992/0.6998	0.7662/0.7658/0.7394	-	0.7633/0.7698/0.7699	0.9778/0.9776/0.9779	0.9369/0.9278/0.9244	**0.9954/0.9921/0.9927**
GaussNoise	0.6753/0.6749/0.6715	0.6599/0.6542/0.6581	-	0.8326/0.8264/0.8276	0.9732/0.9739/0.9742	0.9254/0.9263/0.9265	**0.9926/0.9923/0.9925**
Resize	0.5896/0.5876/0.5891	0.7031/0.7065/0.7074	-	0.8666/0.8584/0.8563	0.9660/0.9474/9309	0.9668/0.9578/0.9583	**0.9993/0.9992/0.9993**
Brightness	0.5253/0.5264/0.5267	0.7811/0.7050/0.6820	-	0.5789/0.4999/0.4258	0.9611/0.9615/0.9634	0.8722/0.8426/0.8358	**0.9852/0.9846/0.9843**
Average	0.5683/0.5657/0.5652	0.7004/0.6884/0.6820	-	0.7642/0.7528/0.7540	0.9716/0.9702/0.9691	0.9230/0.9156/0.9142	**0.9932/0.9926/0.9925**

Table 2. Using SDv1.4, v2.0, and v2.1 models respectively with BitAcc. as the metric, the comparison under different noise levels demonstrates that GaussianShannon achieves the best performance.

Noise	Methods						
	DwtDct [4]	DwtDctSvd [4]	Tree-Ring [7]	Stable Signature [5]	GaussianShading[8]	PRCW[6]	**Ours**
None	0.047/0.029/0.036	0.416/0.331/0.359	-	0.736/0.725/0.734	0.989/0.989/0.988	1.000/1.000/1.000	**1.000/1.000/1.000**
JPEG	0.004/0.002/0.002	0.156/0.073/0.050	-	0.254/0.263/0.243	0.422/0.431/0.414	0.983/0.986/0.981	**1.000/0.996/0.997**
RandDrop	0.002/0.003/0.002	0.126/0.088/0.052	-	0.094/0.091/0.099	0.256/0.232/0.219	0.803/0.812/0.807	**0.973/0.964/0.962**
GaussBlur	0.001/0.002/0.001	0.129/0.074/0.051	-	0.097/0.096/0.092	0.316/0.304/0.301	0.853/0.854/0.849	**0.976/0.978/0.969**
MedianFilter	0.001/0.000/0.000	0.115/0.092/0.043	-	0.118/0.122/0.134	0.332/0.319/0.314	0.793/0.796/0.797	**0.968/0.969/0.971**
GaussNoise	0.004/0.004/0.003	0.268/0.275/0.184	-	0.156/0.169/0.178	0.298/0.287/0.286	0.862/0.864/0.869	**0.959/0.961/0.958**
Resize	0.003/0.001/0.003	0.082/0.052/0.042	-	0.185/0.187/0.193	0.653/0.649/0.641	0.962/0.975/0.968	**0.946/0.951/0.952**
Brightness	0.005/0.004/0.003	0.069/0.039/0.012	-	0.307/0.325/0.335	0.516/0.487/0.492	0.729/0.502/0.518	**0.954/0.943/0.946**
Average	0.020/0.016/0.014	0.135/0.099/0.062	-	0.173/0.179/0.182	0.399/0.387/0.381	0.855/0.827/0.827	**0.968/0.966/0.965**

Table 3. Using SDv1.4, v2.0, and v2.1 models respectively with TPR@BitAcc.100% as the metric, the comparison under different noise levels demonstrates that GaussianShannon achieves the best performance.

TPR@BitAcc.100%. This metric represents the true positive rate when all extracted bits perfectly match the originally embedded watermark bits.

2.2. Baselines

StableSignature. We followed the official tutorial and fine-tuned the LDM decoder on the filtered COCO dataset using a watermark extractor provided in their repository. Approximately 1000 images were used, with a batch size of 4, 100 training steps and learning rate set to 5×10^{-4}. This extractor model has been trained using image augmentation methods, resulting in better robustness against such attacks at the cost of a slight degradation in image quality.

2.3. Details of Performance Experiments

Different sampling steps and inversion steps. The number of sampling steps during image generation is unknown, and users generally prefer fewer sampling steps for faster generation. The mismatch between the number of steps in generation and inversion can affect performance. Therefore, we evaluated performance under combinations of sampling and inversion steps ranging from 10 to 100. As shown in Tab.4, compared to noise, the impact of step mismatch is relatively small, and we can observe the difference through the voting rate.

Gen. Steps	Inversion Steps			
	10	25	50	100
10	1.000/0.027	1.000/0.024	1.000/0.015	1.000/0.013
25	1.000/0.045	0.999/0.022	1.000/0.015	1.000/0.018
50	1.000/0.037	1.000/0.016	1.000/0.020	1.000/0.020
100	1.000/0.046	1.000/0.021	1.000/0.027	1.000/0.018

Table 4. TPR@BitAcc.100% and voting rate under different step combinations

Performance comparison experimental details Tab.1 Tab.2 and Tab.3 present detailed performance com-

parison experiments, with evaluation metrics including TPR@10^{-6}FPR BitAcc. and TPR@BitAcc.100%. As shown in the table, our method achieves optimal average performance in both noise-free and noisy environments.

2.4. Details of Ablation Study

Node Degree. Using regular LDPC codes requires setting the variable node degree d_v and check node degree d_c. To select appropriate parameters, we tested regular LDPC codes under different code rates. The experimental results are shown in Fig.1. When the information bits are fixed at 256 bits, the (3,4) regular code demonstrates the strongest error tolerance, while lower code rates show performance degradation due to structural limitations of regular codes, requiring us to explore irregular codes.

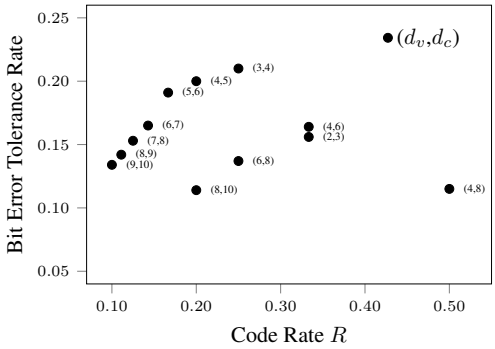

Figure 1. Bit error tolerance rates of regular LDPC codes with different code rates. The parentheses to the right of the data points in the figure indicate different dv and dc combinations, for example (3,4) represents a regular LDPC code with d_v=3, d_c=4 and a code rate of 0.25.

2.5. Details of Advanced Attacks

Compression Attack. For VAE1[3], we use a pre-trained model from the official open-source repository, with the number of filters set to 128. For VAE2[2], we employ Bmshj2018VAE[1] developed by CompressAI[2], with the quality factor set to the minimum value of 1.

Embedding Attack. For embedding attack in the latent space, we use the VAE from the original diffusion model. The pseudocode for the embedding attack is shown in Alg.1. The attack intensity parameters are set as follows: perturbation upper bound $epsilon = 8.0$, number of iterations $num\_iter = 50$, iteration step size $step\_size = 2.0$. The distance metric is set to the maximum absolute difference linf to ensure uniform perturbation.

Algorithm 1 Embedding Space Adversarial Attack

Require: Original image x, encoder E, perturbation bound ε, iterations T, step size α, distance metric d
Ensure: Adversarial example x_{adv}

$\mathbf{z}_{\text{orig}} \leftarrow E(x)$ ▷ Get original latent features
$x_{\text{adv}} \leftarrow x + \mathcal{U}(-\varepsilon, \varepsilon)$ ▷ Add uniform random noise

for $t = 1$ **to** T **do**
  $\mathbf{z}_{\text{adv}} \leftarrow E(x_{\text{adv}})$ ▷ Forward pass through encoder
  $\text{dist} \leftarrow d(\mathbf{z}_{\text{adv}}, \mathbf{z}_{\text{orig}})$ ▷ Compute latent distance
  $\mathcal{L} \leftarrow -\text{dist}$ ▷ Loss: maximize distance
  $\mathbf{g} \leftarrow \nabla_{x_{\text{adv}}} \mathcal{L}$ ▷ Compute gradient
  $x_{\text{adv}} \leftarrow x_{\text{adv}} - \alpha \cdot \text{sign}(\mathbf{g})$ ▷ PGD update step
  $x_{\text{adv}} \leftarrow \text{clip}(x_{\text{adv}}, x - \varepsilon, x + \varepsilon)$ ▷ Project to ε-ball
end for
return x_{adv}

References

[1] Johannes Ballé, David Minnen, Saurabh Singh, Sung Jin Hwang, and Nick Johnston. Variational image compression with a scale hyperprior. In *International Conference on Learning Representations*, 2018. 3

[2] Jean Bégaint, Fabien Racapé, Simon Feltman, and Akshay Pushparaja. Compressai: a pytorch library and evaluation platform for end-to-end compression research. *arXiv preprint arXiv:2011.03029*, 2020. 3

[3] Zhengxue Cheng, Heming Sun, Masaru Takeuchi, and Jiro Katto. Learned image compression with discretized gaussian mixture likelihoods and attention modules. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 7939–7948, 2020. 3

[4] Ingemar Cox, Matthew Miller, Jeffrey Bloom, Jessica Fridrich, and Ton Kalker. *Digital watermarking and steganography*. Morgan kaufmann, 2007. 2

[5] Pierre Fernandez, Guillaume Couairon, Hervé Jégou, Matthijs Douze, and Teddy Furon. The stable signature: Rooting watermarks in latent diffusion models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 22466–22477, 2023. 2

[6] Sam Gunn, Xuandong Zhao, and Dawn Song. An undetectable watermark for generative image models. In *The Thirteenth International Conference on Learning Representations*, 2024. 2

[7] Yuxin Wen, John Kirchenbauer, Jonas Geiping, and Tom Goldstein. Tree-rings watermarks: Invisible fingerprints for diffusion images. In *Advances in Neural Information Processing Systems*, pages 58047–58063. Curran Associates, Inc., 2023. 2

[8] Zijin Yang, Kai Zeng, Kejiang Chen, Han Fang, Weiming Zhang, and Nenghai Yu. Gaussian shading: Provable performance-lossless image watermarking for diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12162–12171, 2024. 2

[9] Ning Yu, Vladislav Skripniuk, Sahar Abdelnabi, and Mario Fritz. Artificial fingerprinting for generative models: Rooting deepfake attribution in training data. In *Proceedings of the IEEE/CVF International conference on computer vision*, pages 14448–14457, 2021. 1