CLaW: Codec-Guided Adaptive Latent Watermarking for Traceable Diffusion Image Generation

Huangchao Song[1] **Lingyun Yu**[2]✉ **Cong Li**[1] **Peiqi Jiang**[1] **Mingqi Fang**[1] **Hongtao Xie**[1]✉
[1]University of Science and Technology of China
[2]Tianjin University
songhuangchao@mail.ustc.edu.cn

Abstract

With the rapid advancement of text-to-image diffusion models, increasingly realistic AI-generated content has raised serious concerns about misuse and copyright infringement. Digital watermarking offers a promising solution by enabling user-level traceability, yet existing methods remain difficult to deploy at scale, hindered by costly model retraining or detection, poor effectiveness–fidelity trade-offs, and limited robustness to image transformations. To address these challenges, we propose **CLaW** (**C**odec-guided **La**tent **W**atermarking), a robust and efficient watermarking framework for traceable diffusion image generation with frozen diffusion backbones and inversion-free detection. Specifically, CLaW first uses a pretrained watermark codec to map each watermark message to a latent residual, enabling low-cost sampling-time injection. Then, the residual is injected within a late denoising window, where image semantics are better preserved and the watermark signal is less perturbed by subsequent denoising updates, yielding a better balance between watermark effectiveness and image fidelity. Furthermore, we introduce a decoder-guided adaptive injection mechanism that uses decoding confidence as feedback to dynamically calibrate watermark strength, reinforcing weak watermark signals and improving robustness under image transformations. Extensive experiments show that CLaW preserves high visual fidelity and achieves state-of-the-art robustness under image transformations, enabling reliable user-level traceability.

1 Introduction

Diffusion models [1, 2] have become a dominant paradigm for high-quality image generation and are now widely deployed through commercial APIs and creative tools, including Stable Diffusion [3, 4, 5], DALL-E [6, 7], Imagen [8], and recent frontier image generators [9, 10, 11]. While these systems make AI-generated content increasingly realistic, they also raise serious concerns about synthetic-media misuse [12, 13], copyright infringement [14, 15], and training-data memorization [16]. As generated images are shared, edited, and redistributed beyond the service boundary, watermarking has become increasingly important for image provenance. It embeds imperceptible yet recoverable signals into generated content, enabling source tracing and ownership protection.

Although watermarking can verify whether an image is AI-generated and identify its source model or service, achieving finer-grained user-level traceability remains a critical objective. We therefore focus on *user-level traceability*, where each registered user is assigned a unique watermark message that can be reliably recovered from redistributed images for attribution. Figure 1 illustrates this process. During generation, the platform embeds a user-specific watermark into the generated image. During detection, the watermark is decoded from a redistributed image for attribution.

✉Corresponding author.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

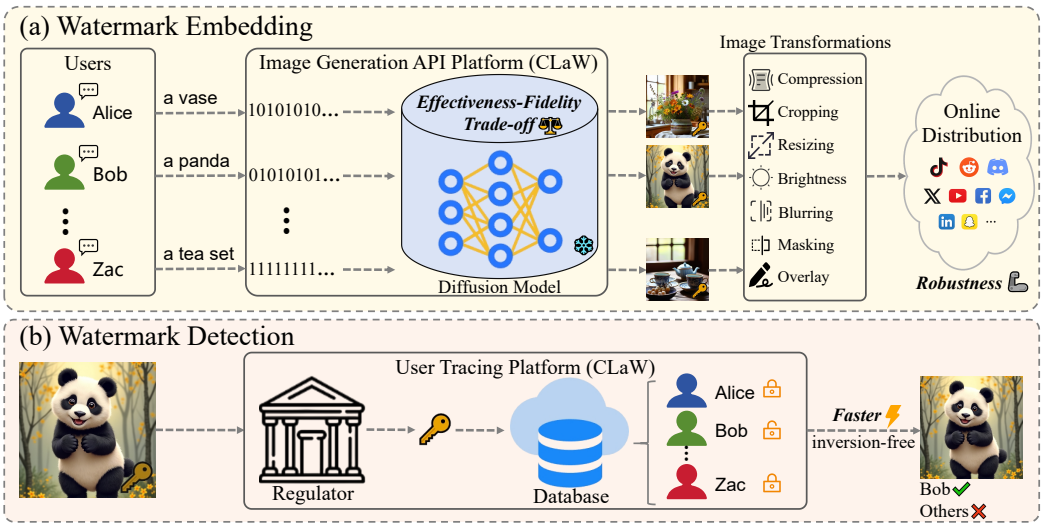


Figure 1: **User-level traceability in image-generation services.** (a) During generation, the platform assigns each user a unique watermark message and embeds it into the generated image, which may undergo common image transformations. (b) During detection, the watermark message is decoded from a suspicious image and matched against a registered-user database for attribution.

However, existing watermarking methods often struggle to satisfy these requirements simultaneously. (1) **High training or detection cost.** Diffusion-native watermarking methods integrate watermarks directly into the generation process. Stable Signature [17] fine-tunes the latent decoder so that generated images carry a recoverable watermark. Subsequent works [18, 19, 20] train watermark-specific modules to better integrate watermarking with diffusion generators. However, these approaches often require retraining the diffusion model. Besides, sampling-time methods [21, 22] avoid retraining but often require diffusion inversion for detection, leading to high test-time cost. (2) **Poor watermark effectiveness–fidelity trade-off.** Methods such as RingID [23] and WIND [24] improve watermark effectiveness by modifying the initial noise. Yet stronger signals alter the denoising trajectory and degrade image quality, while weaker signals fade during denoising, leading to an effectiveness–fidelity imbalance. (3) **Limited robustness under image transformations.** Learned post-generation watermarking methods [25, 26, 27] train an encoder to hide messages in the generated image and a decoder to recover them from the image. Since the watermark message is added after generation rather than embedded during denoising, these methods are often less robust to image transformations.

To address these challenges, we propose CLaW, a robust and efficient framework for traceable diffusion image generation. First, we pretrain a watermark codec that converts each user-specific message into a latent residual and decodes the message from the generated image. The codec is then reused with a frozen diffusion backbone, enabling low-cost sampling-time injection and inversion-free detection. Second, to improve the watermark effectiveness–fidelity trade-off, the codec-derived residual is injected only within a late denoising window. This preserves image semantics while reducing attenuation of the watermark signal along the denoising trajectory. Third, a decoder-guided adaptive injection mechanism is designed to selectively reinforce weak watermark signals during generation. Instead of using a fixed strength for all generations, CLaW uses intermediate decoding confidence as feedback to increase watermark strength when needed, thereby improving robustness.

Extensive experiments on SD3.5-medium and FLUX.1-dev demonstrate the effectiveness of CLaW. CLaW achieves the highest average F1 under image transformations among the methods evaluated on each backbone, reaching **0.9922** and **0.9968**, respectively, while preserving the best PSNR among watermarked methods. In summary, our main contributions are:

- A novel codec-guided latent watermarking framework, called CLaW, is proposed for user-level traceability in diffusion image generation, enabling low-cost sampling-time injection with frozen diffusion backbones and inversion-free detection.
- A late-window latent injection strategy is designed to improve the effectiveness–fidelity trade-off, preserving image semantics while retaining the watermark signal through denoising.

- A decoder-guided adaptive injection strategy is proposed for confidence-aware watermark strength calibration. It uses intermediate decoding confidence as feedback to reinforce weak watermark signals, which improves robustness under image transformations.

2 Related Work

Post-hoc image watermarking. Image watermarking methods can be roughly divided into two groups: transform-based methods and learned image-domain methods. The first group embeds signals into spatial or frequency coefficients and usually targets robustness against compression, filtering, and geometric distortions. Early spread-spectrum and synchronization-based methods established the basic robustness framework [28, 29], and later studies refined practical designs and evaluation protocols for image watermarking under common signal processing operations [30, 31, 32, 33, 34]. The second group learns the embedding and decoding process end-to-end. Generating Steganographic Images [35] and HiDDeN [25] showed that neural encoders and decoders can optimize multi-bit hiding jointly with robustness objectives; StegaStamp [26] targets physical distortions caused by screen-shooting and print-shooting scenarios; RivaGAN [27] adapts attention-based decoding to improve invisible video watermarking; Certified Neural Network Watermarks [36] and supervised GAN watermarking [37] further study certifiable robustness and IP protection under stronger attacks. However, these methods are applied after generation, so the watermark signal remains external to the diffusion process and is vulnerable to image transformations such as compression, editing, and regeneration. This motivates CLaW to embed user-specific signals during denoising, using late-window injection and decoder-guided adaptation to improve robustness.

In-generation diffusion watermarking. Diffusion watermarking methods embed the watermark inside the diffusion pipeline so that it becomes part of the generation process. One branch injects the watermark into the sampling trajectory. Tree-Ring [21] embeds a structured pattern into the initial noise and detects it through DDIM inversion; RingID [23] extends this idea to multi-key identification; Gaussian Shading [22] preserves the sampling distribution while injecting watermarks; PRC [38], WIND [24], SFW [39], and GaussMarker [40] further improve robustness or undetectability through refined noise-space coding and detection designs; SERUM [41] embeds watermark patterns in the initial noise and trains a lightweight detector for inversion-free detection; Shallow Diffuse [42] instead exploits low-dimensional subspaces in the generation process to decouple watermarking from content synthesis. These methods improve robustness by coupling the watermark with the sampling process, but often rely on costly diffusion inversion for detection. Another branch modifies decoder or conditioning components so that watermarked images can be decoded directly in the image domain. Stable Signature [17] fine-tunes the latent decoder to emit a recoverable signature; FSW [18] adds a message matrix to the decoder path to support flexible message updates; WaDiff [19] treats the watermark as an additional conditioning signal for user-specific identification; WMAdapter [20] introduces a plug-in adapter for content-adaptive watermark control. Despite enabling direct image-domain detection, these methods often require retraining or watermark-specific modules, which increases training cost and may shift the original generation distribution. Therefore, CLaW injects codec-derived watermark signals during sampling without modifying the diffusion backbone, reducing training cost, and uses late-window injection to balance watermark effectiveness and image fidelity.

3 Method

This section describes CLaW, a codec-guided latent watermarking method for traceable diffusion generation. Each registered user is assigned a unique binary trace key. We refer to the embedded and recoverable binary representation as the watermark payload. As shown in Figure 2, CLaW first pretrains a secret codec that maps a user-specific payload to a latent residual and decodes the payload from an image (Section 3.1). During generation, CLaW reuses the pretrained encoder to inject the watermark residual in a late denoising window (Section 3.2). The paired decoder further guides adaptive injection by previewing payload recoverability from intermediate noisy latents and increasing the injection strength when the watermark is not yet confidently recoverable (Section 3.3). After generation, CLaW detects the embedded payload from a suspicious image and matches it against a registered user database for attribution (Section 3.4).

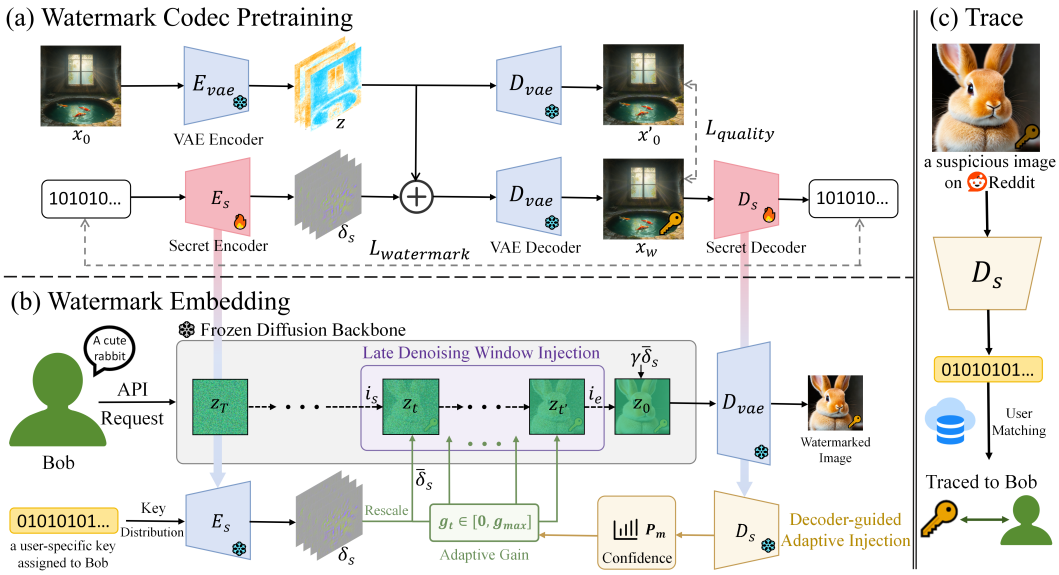


Figure 2: Overview of CLaW. (a) Pretraining a secret encoder E_s and a secret decoder D_s within the latent space of a frozen VAE: E_s converts a user-specific payload into a latent residual, while D_s recovers the payload from the decoded image. (b) During generation, CLaW keeps the diffusion backbone frozen and injects the residual in a late denoising window. The frozen decoder previews intermediate noisy latents and provides feedback to update the adaptive gain, which dynamically calibrates the watermark injection strength. (c) During detection, D_s directly extracts the payload from a suspicious image and matches it against the registered user database for attribution.

3.1 Watermark Codec Pretraining

CLaW first pretrains a watermark codec (E_s, D_s) following the latent-watermarking paradigm of AquaLoRA [43]. The codec contains a secret encoder and decoder. Given a clean image $\boldsymbol{x}_0$ and a payload $\boldsymbol{b} = (b_1, \ldots, b_K) \in \{0,1\}^K$, the encoder maps the payload to a latent residual $\delta_s = E_s(\boldsymbol{b})$. We encode the image as $\boldsymbol{z} = E_{\text{vae}}(\boldsymbol{x}_0)$ and define its VAE reconstruction as $\boldsymbol{x}'_0 = D_{\text{vae}}(\boldsymbol{z})$. The watermark residual δ_s is then embedded into the clean latent to obtain a watermarked latent $\boldsymbol{z} + \delta_s$, which is decoded into the watermarked image: $\boldsymbol{x}_w = D_{\text{vae}}(\boldsymbol{z} + \delta_s)$. The codec is trained to recover the payload after an augmentation layer $\mathcal{A}$, where $\sigma(\cdot)$ denotes the logistic sigmoid function that maps decoder logits to bit-wise probabilities. The watermark loss is defined as:

$$\mathcal{L}_{\text{WM}} = -\frac{1}{2K} \sum_{k=1}^{K} \left[\log \sigma(s_{k,b_k}) + \log\left(1 - \sigma(s_{k,1-b_k})\right) \right], \quad s_k = D_s(\mathcal{A}(\boldsymbol{x}_w))_k. \tag{1}$$

To make the watermark residual visually imperceptible, we pretrain the codec with the watermark loss and two visual regularizers: LPIPS [44] and the Peak Regional Variation Loss (PRVL) proposed in AquaLoRA [43]. We optimize the following objective:

$$\mathcal{L} = \mathcal{L}_{\text{WM}} + \lambda\, \mathcal{L}_{\text{LPIPS}}(\boldsymbol{x}'_0, \boldsymbol{x}_w) + \mu\, \mathcal{L}_{\text{PRVL}}(\boldsymbol{x}'_0, \boldsymbol{x}_w). \tag{2}$$

Once pretrained, E_s and D_s remain fixed throughout CLaW. The frozen encoder converts each user payload into a latent watermark residual that is embedded along the generation trajectory, while the decoder provides feedback for adaptive injection and recovers the payload during detection.

3.2 Late-Denoising Window Watermark Injection

During generation, CLaW requires no additional training and keeps both the diffusion backbone and the pretrained codec frozen. The watermark payload is embedded in latent space by injecting codec residuals into selected noisy latents. Given a prompt c and a payload $\boldsymbol{b}$, the secret encoder maps $\boldsymbol{b}$ to a latent residual $\delta_s = E_s(\boldsymbol{b})$. To align with the scheduler latent scale, we rescale the VAE-space

residual as $\bar{\delta}_s = s_{\text{vae}}\,\delta_s$, where s_{vae} is the VAE scaling factor and $\bar{\delta}_s$ denotes the watermark residual in scheduler units. The VAE shift is not applied to the residual, as the residual is additive. When necessary, we resize δ_s to match the latent resolution using bilinear interpolation before scaling.

We inject the watermark during a late denoising window with flexible injection schedules to balance robustness and image quality. Let N denote the number of inference steps, and let $i = 0, \ldots, N-1$ index the denoising steps. We define the injection window parameterized by two fractions ρ_s and ρ_e with $i_s = \lfloor \rho_s N \rfloor$, $i_e = \lceil \rho_e N \rceil$, and $\mathcal{W} = \{i_s, \ldots, i_e - 1\}$. Watermark injection is applied within the late denoising window $\mathcal{W}$. At each step i, we define the injected residual as:

$$\delta_i(\boldsymbol{b}) = \begin{cases} \alpha\, w_i\, \bar{\delta}_s, & i \in \mathcal{W}, \\ 0, & \text{otherwise.} \end{cases} \tag{3}$$

where α denotes watermark strength, w_i is the normalized schedule weight at step i within the injection window $\mathcal{W}$, and $\bar{\delta}_s$ is the scheduler-space watermark residual.

We inject the watermark by adding the residual defined in Eq. (3) to the latent at each step, i.e., $\tilde{\boldsymbol{z}}^{(i)} = \boldsymbol{z}^{(i)} + \delta_i(\boldsymbol{b})$. The sampler then updates the watermarked latent as:

$$\boldsymbol{z}^{(i+1)} = S_\theta\Big(\tilde{\boldsymbol{z}}^{(i)}, \tau_i, c\Big), \quad i = 0, \ldots, N-1. \tag{4}$$

where S_θ denotes the denoising update at timestep τ_i, and c is the text condition.

The final watermarked image is decoded as $\boldsymbol{x}_w = D_{\text{vae}}(\boldsymbol{z}^{(N)})$. Late injection enables robust payload embedding while preserving generation quality, as watermarks injected at early denoising steps can degrade semantic fidelity and be attenuated by the long denoising trajectory. After the final denoising step, we optionally apply a small terminal anchor $\gamma\bar{\delta}_s$ to the latent, providing a final reinforcement of the watermark signal. We further introduce flexible injection schedules (e.g., flat, linear, cosine) to control the temporal distribution of watermark strength within the injection window. Implementation details of the terminal anchor and injection schedules are provided in Appendix B.1.

3.3 Decoder-Guided Adaptive Watermark Injection

A fixed schedule applies the same base strength αw_i to every prompt and every intermediate state. For user-level traceability, this leads to inefficient signal allocation, assigning unnecessary strength to easy samples while providing insufficient signal for hard cases. Some trajectories become reliably decodable after only a few late-stage injections, whereas others require stronger residual signals to remain recoverable under post-generation perturbations. To address this variability, we augment the fixed rule in Eq. (3) with a decoder-guided feedback mechanism.

We implement the feedback loop illustrated in Figure 2(b). At selected steps, CLaW decodes the current latent using the VAE decoder and uses the pretrained secret decoder to estimate payload decodability. This preview acts purely as a generation-time control signal, with small additional cost and without changing the standard generation pipeline. Formally, at a preview step $i \in \mathcal{W}$, the current latent $\boldsymbol{z}^{(i)}$ is decoded into a preview image $\boldsymbol{x}_{\text{pre}}^{(i)} = D_{\text{vae}}(\boldsymbol{z}^{(i)})$, where scheduler-specific latent rescaling is omitted for notational simplicity. Given the secret decoder D_s, we define the preview payload confidence for the target payload $\boldsymbol{b}$ as:

$$P_m(i) = \frac{1}{K} \sum_{k=1}^{K} \text{softmax}\Big(D_s(\boldsymbol{x}_{\text{pre}}^{(i)})_k\Big)_{b_k}. \tag{5}$$

Here, $D_s(\boldsymbol{x})_k$ denotes the two-class logits for the k-th payload bit, and $P_m(i)$ averages the probability assigned to the target payload $\boldsymbol{b} = (b_1, \ldots, b_K)$. This soft confidence provides the feedback signal for adaptive gain control along the denoising trajectory.

CLaW adapts the injection gain by comparing the current preview confidence $P_m(i)$ with a target confidence τ. Let λ be the gain coefficient, and let $[a]_+ = \max(a, 0)$ denote the positive part operator. To control the overhead of preview decoding, we introduce a preview stride s_{pre}, so that $P_m(i)$ and the gain are updated only at selected steps within the injection window. With i_s denoting the first step in $\mathcal{W}$, these preview-update steps are defined as $\mathcal{P}_{\text{pre}} = \{\, i \in \mathcal{W} \mid (i - i_s) \bmod s_{\text{pre}} = 0 \,\}$. Initialized at the default value $g = 1$, the gain is updated as:

$$g_i = \begin{cases} \text{clip}_{[g_{\min}, g_{\max}]}\big(1 + \lambda[\tau - P_m(i)]_+\big), & i \in \mathcal{P}_{\text{pre}}, \\ g_{i^-}, & i \in \mathcal{W} \setminus \mathcal{P}_{\text{pre}}, \end{cases} \tag{6}$$

Here g_{i^-} denotes the last available gain before step i, which is reused at non-preview steps in $\mathcal{W}$. At preview-update steps, the gain increases only when $P_m(i) < \tau$, with the increase determined by the confidence gap $\tau - P_m(i)$. The clipping range $[g_{\min}, g_{\max}]$ bounds the gain to prevent excessive amplification of the watermark residual. The scheduled residual is then adaptively scaled as:

$$\delta_i^{\mathrm{ada}}(\boldsymbol{b}) = \begin{cases} \alpha\, w_i\, g_i\, \bar{\delta}_s, & i \in \mathcal{W}, \\ 0, & \text{otherwise.} \end{cases} \tag{7}$$

Here, w_i follows the same normalized late-window schedule as in Eq. (3). At each injection step, the watermarked latent is formed as $\tilde{\boldsymbol{z}}^{(i)} = \boldsymbol{z}^{(i)} + \delta_i^{\mathrm{ada}}(\boldsymbol{b})$ and passed to the standard denoising update. Final payload recovery is performed after generation by a single forward pass of the frozen secret decoder on the generated image.

3.4 Watermark Detection and User Attribution

As illustrated in Figure 2(c), CLaW performs user tracing directly from a suspicious image $\bar{\boldsymbol{x}}$. The frozen secret decoder D_s predicts bit-wise logits from which the payload estimate $\hat{\boldsymbol{b}}$ is obtained. For a candidate payload $\boldsymbol{b}$, we define the match count as $M(\boldsymbol{b}, \hat{\boldsymbol{b}}) = \sum_{k=1}^{K} \mathbb{I}[\hat{b}_k = b_k]$ and the bit accuracy as $\mathrm{BitAcc}(\boldsymbol{b}, \hat{\boldsymbol{b}}) = M(\boldsymbol{b}, \hat{\boldsymbol{b}})/K$. Given a registered user set $\mathcal{U}$ with payloads $\{\boldsymbol{b}^{(u)}\}_{u \in \mathcal{U}}$, user attribution is performed by maximizing the payload match count:

$$\hat{u} = \arg\max_{u \in \mathcal{U}} M(\boldsymbol{b}^{(u)}, \hat{\boldsymbol{b}}). \tag{8}$$

The attribution is accepted only if the best match satisfies $M(\boldsymbol{b}^{(\hat{u})}, \hat{\boldsymbol{b}}) \geq \kappa$, where κ is chosen by the binomial false-positive criterion in Appendix Eq. (11) to ensure a low false attribution rate. Unlike binary watermark detection, user tracing requires the decoded payload to match a registered user-specific payload with sufficient accuracy.

4 Experiments

In this section, we present a comprehensive set of experiments to evaluate CLaW as a practical user-level traceability mechanism for diffusion-generated images. We first benchmark its watermarking performance and image quality on SD3.5-medium [45] and FLUX.1-dev [46] (Section 4.1), and then evaluate its robustness to common image transformations and generation-configuration changes (Section 4.2), as well as composite transformations and screen-shooting distortions (Section 4.3). We further analyze user-level attribution in Section 4.4 and module-level ablations in Section 4.5. Additional evaluation results and qualitative examples are provided in Appendix D.

Codec and generation setup. CLaW uses a 48-bit watermark codec pretrained with the objective in Section 3.1. In the main experiments, it injects the watermark in the last 10% of the sampling trajectory with a linear schedule, decoder-guided gain adaptation, and a lightweight terminal anchor. We evaluate all methods on SD3.5-medium and FLUX.1-dev using shared 1,000-image test sets and matched seeds. Additional codec, generation, and evaluation details are provided in Appendix C.

Baselines. We build our comparison protocol on the open-source *MarkDiffusion* framework [47], which provides a unified interface for generative watermarking evaluation. We compare CLaW with diverse representative watermarking baselines: RivaGAN [27], Gaussian Shading [22], RingID [23], PRC [38], WIND [24], GaussMarker [40], and SFW [39]. We also include SERUM [41] in the SD3.5-medium robustness comparison reported in Table 2. A detailed conceptual comparison between CLaW and related watermarking methods is provided in Appendix F.

Evaluation metrics. We evaluate watermarking performance using F1, accuracy, and BitAcc. F1 and accuracy are computed with watermarked images as positives and clean images as negatives, while BitAcc measures correctly recovered payload bits. Robustness is assessed under image transformations. Image quality is measured with CLIP [48], FID [49], PSNR [50], and SSIM [51] following MarkDiffusion [47]. For user-level traceability, trace accuracy is computed by matching decoded payloads to registered payloads under the binomial threshold in Appendix Eq. (11).

Table 1: Main comparison on SD3.5-medium and FLUX.1-dev, evaluated with 1,000 generated images per method and backbone. We report watermark detection accuracy, image fidelity, and runtime. F1 is measured on clean generated images, while F1 (Adv.) reports the average F1 under common post-generation image transformations. Gen. and Det. denote mean seconds per image for watermarked generation and detection, respectively. Runtime standard deviations are reported in Appendix Table 11, and full image-quality results are reported in Appendix Tables 12 and 13.

Method	SD3.5-medium						FLUX.1-dev					
	F1 ↑	F1 (Adv.) ↑	PSNR ↑	FID ↓	Gen. ↓	Det. ↓	F1 ↑	F1 (Adv.) ↑	PSNR ↑	FID ↓	Gen. ↓	Det. ↓
RivaGAN [27]	0.9914	0.9288	40.53	**35.13**	2.56	0.518	**1.0000**	0.9577	40.39	**38.30**	5.12	0.530
Gaussian Shading [22]	**1.0000**	0.8750	9.99	35.57	11.40	1.559	**1.0000**	0.8656	10.85	38.40	14.49	4.703
RingID [23]	0.9935	0.6213	8.56	35.38	2.23	1.589	0.9950	0.4473	11.09	38.74	**4.57**	4.658
PRC [38]	0.9712	0.5642	8.44	35.90	2.70	3.444	0.9975	0.5090	11.06	38.57	4.70	6.836
WIND [24]	0.9901	0.8665	8.31	36.26	2.34	2.241	0.9877	0.8658	10.78	39.31	4.61	4.846
GaussMarker [40]	**1.0000**	0.8744	8.50	36.47	11.27	2.107	**1.0000**	0.8737	11.22	38.78	13.93	4.702
SFW [39]	0.9848	0.6636	12.06	37.12	**1.87**	1.577	0.9950	0.5351	13.80	41.96	4.59	4.696
CLaW (ours)	**1.0000**	**0.9922**	**41.25**	35.70	1.93	**0.0115**	**1.0000**	**0.9968**	**41.06**	38.37	4.73	**0.0104**

Robustness evaluation. We evaluate robustness from two complementary aspects: robustness to image transformations and robustness to generation-configuration changes. For robustness to image transformations, we follow the MarkDiffusion robustness protocol, including JPEG compression, crop-and-resize and Adaptive Noise, and extend the evaluation to composite and repeated transformations, screenshots, and screen-shooting distortions. The standard attack configurations are provided in Table 7. For robustness to generation-configuration changes, we evaluate CLaW across denoising steps, CFG scale, and resolution. Model-specific settings are provided in Appendix C.1; full image-attack and regeneration-attack results are reported in Appendices D.1 and D.2, respectively.

4.1 Benchmarking Watermark Accuracy and Image Quality

Table 1 evaluates watermark accuracy, image fidelity, and runtime for traceability. Since clean-image F1 is largely saturated across methods, the key distinction comes from attacked F1, fidelity, and runtime, which measure robustness after redistribution and practical verification cost.

CLaW achieves the best overall trade-off between watermark reliability and visual fidelity. As shown in Table 1, it attains perfect clean F1 on both diffusion backbones and yields the highest attacked F1 under common image transformations. Notably, CLaW also achieves the highest PSNR among all watermarked methods on both backbones, indicating that its robustness gains do not come at the cost of visible image degradation. This demonstrates a more favorable balance between robustness and image quality than prior baselines and indicates that the robustness gain is not obtained by introducing stronger visible distortions. In contrast, several diffusion-watermarking baselines achieve high clean F1 but substantially degrade PSNR and FID; RivaGAN better preserves image quality, but is less robust under attacks and slower at detection.

Runtime is also important in practice, covering both the generation overhead of watermark embedding and the cost of subsequent detection. CLaW adds only a small generation overhead and uses a single frozen-decoder forward pass for detection, making it substantially faster than baseline detectors. Overall, these results show that CLaW improves robustness and fidelity while retaining practical generation and detection costs.

4.2 Benchmarking Watermark Robustness

To better understand robustness beyond the average score, we further report per-attack results on SD3.5-medium in Table 2. The corresponding FLUX.1-dev results are provided in Appendix Table 9. CLaW remains robust across all perturbations, with every F1 score above 0.96, showing that its robustness is consistent across transformations rather than limited to a few specific attacks.

The per-attack results show that prior methods remain vulnerable to specific transformations. Gaussian Shading, GaussMarker, and WIND are brittle under crop-and-resize despite strong performance under several other corruptions, while RivaGAN is more balanced but still degrades under overlay. In contrast, CLaW maintains strong performance under both geometric and non-geometric transforma-

Table 2: Robustness on SD3.5-medium under common image transformations. F1 and AUC are reported for each transformation, and Avg. denotes the mean over the eight transformations.

Methods	Distortions								
	JPEG	Crop+Resize	Blur	Noise	Brightness	Mask	Overlay	Adaptive Noise	Avg.
F1 ↑									
RivaGAN [27]	0.9189	0.9501	0.9213	0.9148	0.9496	0.9909	0.8318	0.9529	0.9288
Gaussian Shading [22]	**1.0000**	0.0000	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**	0.8750
RingID [23]	0.5380	0.0060	0.6591	0.4639	0.9712	0.9401	0.4962	0.8956	0.6213
PRC [38]	0.7390	0.0000	0.0198	0.3333	0.9224	0.9236	0.7063	0.8694	0.5642
WIND [24]	0.9901	0.0271	0.9886	0.9780	0.9877	0.9877	0.9804	0.9926	0.8665
GaussMarker [40]	**1.0000**	0.0000	0.9950	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**	0.8744
SFW [39]	0.6111	0.0000	0.7896	0.4791	0.9637	0.9610	0.5765	0.9276	0.6636
SERUM [41]	0.9975	0.9224	0.9738	0.9945	0.9965	0.9885	0.9849	0.9995	0.9822
CLaW (ours)	0.9995	**0.9970**	0.9995	0.9924	0.9616	0.9985	0.9970	0.9919	**0.9922**
AUC ↑									
RivaGAN [27]	0.9250	0.9525	0.9270	0.9215	0.9520	0.9910	0.8560	0.9550	0.9350
Gaussian Shading [22]	**1.0000**	0.5000	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**	0.9375
RingID [23]	0.6840	0.5015	0.7140	0.6510	0.9720	0.9435	0.6650	0.9055	0.7546
PRC [38]	0.7930	0.5000	0.5050	0.6000	0.9280	0.9290	0.7730	0.8845	0.7391
WIND [24]	0.9900	0.4970	0.9885	0.9775	0.9875	0.9875	0.9800	0.9925	0.9251
GaussMarker [40]	**1.0000**	0.5000	0.9950	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**	0.9369
SFW [39]	0.7200	0.5000	0.8175	0.6575	0.9650	0.9625	0.7025	0.9325	0.7822
SERUM [41]	0.9975	0.9255	0.9745	0.9945	0.9965	0.9885	0.9850	0.9995	0.9827
CLaW (ours)	0.9995	**0.9970**	0.9995	0.9925	0.9630	0.9985	0.9970	0.9920	**0.9924**

Table 3: Robustness to variations in SD3.5-medium and FLUX.1-dev generation settings. We change one setting at a time from the default configuration. Default test settings are colored by green cells.

Factor	Setting	SD3.5-medium				FLUX.1-dev			
		BitAcc ↑	BitAcc (Adv.) ↑	CLIP ↑	PSNR ↑	BitAcc ↑	BitAcc (Adv.) ↑	CLIP ↑	PSNR ↑
Steps	15	0.9875	0.9497	0.8594	39.3099	0.9975	0.9700	**0.8597**	39.4380
	25	0.9913	0.9591	0.8606	39.3239	0.9958	0.9676	0.8568	40.2712
	40	**0.9929**	0.9601	**0.8626**	39.7695	0.9973	0.9707	0.8571	40.8469
	50	0.9910	0.9582	0.8625	40.0539	**0.9977**	**0.9708**	0.8572	**41.0504**
CFG	5.0	**0.9942**	**0.9676**	**0.8635**	**40.2391**	0.9975	**0.9718**	0.8569	40.2328
	7.5	0.9910	0.9580	0.8625	40.0539	**0.9977**	0.9708	**0.8572**	**41.0504**
	10.0	0.9904	0.9480	0.8602	39.6930	0.9960	0.9708	0.8559	40.0955
Resolution	512×512	**0.9910**	0.9588	0.8625	40.0539	**0.9977**	**0.9707**	0.8572	41.0504
	1024×1024	0.9875	**0.9704**	**0.8653**	**40.7770**	0.9865	0.9568	**0.8586**	**42.4153**

tions. This robustness aligns with its late-denoising injection and frozen codec decoder, which avoid fragile pixel-space patterns and costly inversion-based recovery.

We further evaluate robustness to generation-setting changes, including denoising steps, CFG scale, and output resolution. As shown in Table 3, CLaW remains stable on SD3.5-medium, maintaining high attacked BitAcc with only marginal CLIP variation. CFG scale has the most visible effect, likely because guidance strength directly alters the denoising trajectory. In contrast, denoising steps and resolution cause only minor fluctuations, indicating lower sensitivity to sampling budget and image size. Overall, CLaW remains effective across different sampling budgets, guidance strengths, and output resolutions without retraining the watermark codec.

4.3 Benchmarking Robustness beyond Standard Attacks

To assess robustness under more complex image processing, we evaluate ten conditions on SD3.5-medium, covering composite transformations and restoration. Composite transformations comprise three pipelines: degradation combines blur, noise, and JPEG compression; editing combines brightness adjustment, overlay, and JPEG compression; and geometric processing combines crop-and-resize with JPEG compression. Each pipeline is evaluated with one and three repetitions. Restoration

Table 4: Robustness on SD3.5-medium under composite transformations and restoration. Mean F1 is reported for each group, and Avg. denotes the mean over all ten conditions. Higher is better.

Method	Composite transformations			Restoration		Avg.
	Degradation	Editing	Geometric	Super-resolution	Denoising	
RivaGAN [27]	0.8877	0.5053	0.7406	0.3709	0.0392	0.5419
Gaussian Shading [22]	0.9847	**1.0000**	0.0000	**1.0000**	**1.0000**	0.7969
RingID [23]	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
PRC [38]	0.1304	0.4818	0.0000	0.7950	0.6207	0.4230
WIND [24]	0.9783	0.9804	0.0914	0.9934	0.9901	0.8071
GaussMarker [40]	0.9767	0.9975	0.0000	**1.0000**	**1.0000**	0.7948
SFW [39]	0.0575	0.4699	0.0000	0.8420	0.8118	0.4393
CLaW (ours)	**0.9873**	0.8293	**0.8030**	0.9473	0.9848	**0.9066**

Table 5: User-level traceability with increasing registry sizes. Trace accuracy is computed by matching each decoded 48-bit payload to a registry of candidate users. Higher is better.

Backbone	Method	10^4 users		10^5 users		10^6 users	
		Trace Acc.	Trace Acc. (Adv.)	Trace Acc.	Trace Acc. (Adv.)	Trace Acc.	Trace Acc. (Adv.)
SD3.5-medium	DWT-DCT-SVD [52]	0.9960	0.5090	0.9960	0.4768	0.9940	0.4464
	RivaGAN [27]	0.9950	0.9231	0.9920	0.9001	0.9870	0.8570
	CLaW (ours)	**1.0000**	**0.9861**	**1.0000**	**0.9786**	**0.9980**	**0.9649**
FLUX.1-dev	DWT-DCT-SVD [52]	**1.0000**	0.5610	**1.0000**	0.5251	**1.0000**	0.4866
	RivaGAN [27]	**1.0000**	0.9454	0.9990	0.9276	0.9970	0.8935
	CLaW (ours)	**1.0000**	**0.9936**	**1.0000**	**0.9896**	**1.0000**	**0.9803**

comprises four conditions: Real-ESRGAN $4\times$ super-resolution; super-resolution followed by downsampling, evaluated with one and three repetitions; and SwinIR denoising. For denoising, we use SwinIR pretrained for Gaussian noise with standard deviation $\sigma = 15$, without adding input noise.

As shown in Table 4, CLaW achieves the highest average F1 of 0.9066 and performs best under degradation and geometric processing. RivaGAN handles geometric processing relatively well but shows limited robustness to restoration, particularly denoising. Gaussian Shading and GaussMarker outperform CLaW under editing and restoration but perform poorly under geometric processing. These results demonstrate CLaW's robustness to diverse image attacks.

We further evaluate payload recovery from desktop screenshots and phone photographs of watermarked images displayed on a screen, using 100 images per condition. Watermark payloads are recovered directly from the captured images without manual cropping or geometric correction. Under these conditions, CLaW achieves mean BitAcc of 99.04% and 84.21%, respectively. These results demonstrate CLaW's robustness to screenshot and screen-shooting distortions.

4.4 Benchmarking User-Level Traceability

User-level traceability requires recovering a user-specific payload from the generated image and matching it to a registered user, rather than only detecting watermark presence. We therefore evaluate only methods that decode explicit bit strings. Several baselines in Table 1 are not directly applicable, as they target binary detection, often rely on slow inversion-based verification, and do not provide an efficient payload interface for large-scale attribution.

We assign each simulated user a distinct 48-bit payload and evaluate attribution over registries of 10^4, 10^5, and 10^6 users. As shown in Table 5, CLaW remains stable as the registry size increases, maintaining high attacked trace accuracy even at the largest scale. In contrast, RivaGAN degrades more noticeably with larger registries, while DWT-DCT-SVD [52] remains accurate on clean images but becomes unreliable after attacks. These results indicate that scalable attribution requires stable payload recovery under image transformations, not merely clean-image detectability.

We further assess open-set false attribution on SD3.5-medium with 10^6 registered users. Thresholds are set using $6{,}000$ unwatermarked images, then held fixed for testing on another $6{,}000$ unwatermarked images and $1{,}000$ watermarked images carrying distinct 48-bit payloads. Here, the

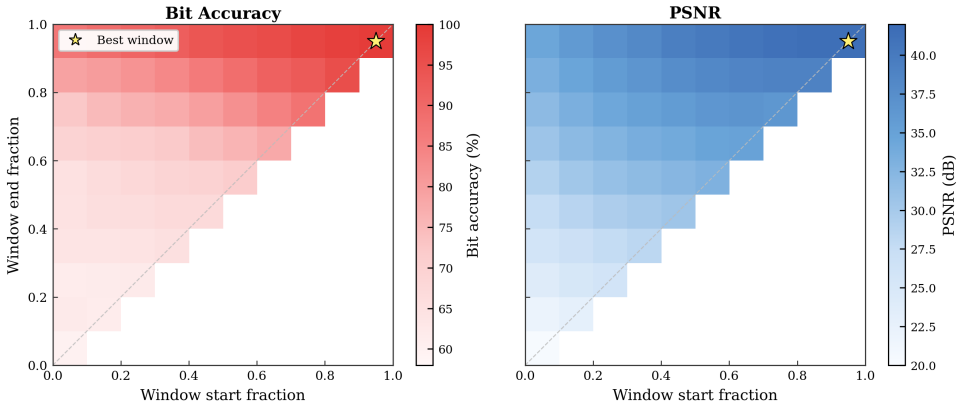


Figure 3: Effect of injection timing. Cells denote denoising-window start/end pairs, with BitAcc on the left and PSNR on the right. The starred late-window setting balances recovery and fidelity.

Table 6: Module ablation of CLaW on SD3.5-medium and FLUX.1-dev. Higher is better.

Variant	SD3.5-medium						FLUX.1-dev					
	BitAcc	BitAcc (Adv.)	AUC	AUC (Adv.)	CLIP	PSNR	BitAcc	BitAcc (Adv.)	AUC	AUC (Adv.)	CLIP	PSNR
w/o Late Denoising	0.5595	0.5413	0.5468	0.5557	**0.8702**	**57.54**	0.6116	0.5727	0.7095	0.6309	0.8667	**57.8904**
w/o Adaptive Feedback	0.9825	0.9283	0.9993	0.9979	**0.8702**	42.18	0.9890	0.9384	**1.0000**	0.9996	0.8674	43.1592
w/o Terminal Anchor	0.9911	0.9521	**1.0000**	0.9997	0.8700	40.62	0.9952	0.9604	**1.0000**	**1.0000**	0.8675	41.6392
Full	**0.9926**	**0.9595**	**1.0000**	**1.0000**	0.8700	40.04	**0.9971**	**0.9695**	**1.0000**	**1.0000**	**0.8677**	41.0571

unwatermarked images include natural images from COCO2017 [53] and images generated by SD3.5-medium. Across five runs, CLaW achieves a mean registry-level false-positive rate (FPR) of 0.0500% and 98.5% trace accuracy, with the attribution threshold set to 44 matching bits out of 48. Increasing the threshold to 45 matching bits reduces FPR to 0.0100%, while maintaining 97.3% trace accuracy. Overall, CLaW supports accurate attribution with low FPR at the million-user scale.

4.5 Discussion and Ablation

Figure 3 studies the effect of injection timing by varying the injection window along the denoising trajectory. High recovery is concentrated in late denoising windows, where the injected residual is less likely to be perturbed by subsequent updates. These windows also retain higher PSNR, showing that late-window injection balances effectiveness and fidelity. Additional analyses of watermark-strength and semantic consistency are provided in Appendix Figures 7(a) and 7(b).

Table 6 isolates the contribution of each CLaW component. Removing the late denoising window sharply reduces attacked BitAcc on both backbones, confirming that late injection is essential for payload persistence. Its higher PSNR mainly reflects a weaker watermark signal, since the payload is barely embedded. Adaptive feedback and the terminal anchor provide smaller but consistent gains: adaptive feedback improves sample-specific recoverability, while the terminal anchor stabilizes the final watermark signal before decoding.

5 Conclusion

In this paper, we introduce CLaW, a robust and efficient codec-guided latent watermarking framework for user-level traceability in diffusion image generation. CLaW enables low-cost sampling-time injection with a pretrained watermark codec, without retraining the diffusion backbone. We further design a late-window injection strategy to better balance watermark effectiveness and image fidelity, and propose a decoder-guided adaptive injection mechanism that improves robustness under image transformations. Extensive experiments show that CLaW achieves robust user-level traceability while maintaining high visual quality across diffusion backbones. A current limitation is that CLaW applies watermark residuals uniformly across latent channels. Future work could explore channel-aware injection rules that place watermark signals in more robust and less perceptually sensitive channels.

Acknowledgments and Disclosure of Funding

This work is supported by the Fundamental and Interdisciplinary Disciplines Breakthrough Plan of the Ministry of Education of China (JYB2025XDXM103), and the National Natural Science Foundation of China (62425114, 62121002, U23B2028, 62472395).

References

[1] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851, 2020.

[2] Yang Song, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations. In *International Conference on Learning Representations*, 2021.

[3] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10684–10695, 2022.

[4] Dustin Podell, Zion English, Kyle Lacey, Andreas Blattmann, Tim Dockhorn, Jonas Müller, Joe Penna, and Robin Rombach. Sdxl: Improving latent diffusion models for high-resolution image synthesis. In B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun, editors, *International Conference on Learning Representations*, volume 2024, pages 1862–1874, 2024.

[5] Patrick Esser, Sumith Kulal, Andreas Blattmann, Rahim Entezari, Jonas Müller, Harry Saini, Yam Levi, Dominik Lorenz, Axel Sauer, Frederic Boesel, et al. Scaling rectified flow transformers for high-resolution image synthesis. In *Forty-first international conference on machine learning*, 2024.

[6] Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. Hierarchical text-conditional image generation with clip latents. *arXiv preprint arXiv:2204.06125*, 1(2):3, 2022.

[7] James Betker, Gabriel Goh, Li Jing, Tim Brooks, Jianfeng Wang, Linjie Li, Long Ouyang, Juntang Zhuang, Joyce Lee, Yufei Guo, et al. Improving image generation with better captions. *Computer Science. https://cdn.openai.com/papers/dall-e-3.pdf*, 2(3):8, 2023.

[8] Chitwan Saharia, William Chan, Saurabh Saxena, Lala Li, Jay Whang, Emily L Denton, Kamyar Ghasemipour, Raphael Gontijo Lopes, Burcu Karagol Ayan, Tim Salimans, et al. Photorealistic text-to-image diffusion models with deep language understanding. *Advances in neural information processing systems*, 35:36479–36494, 2022.

[9] Google DeepMind. Nano banana 2. https://blog.google/innovation-and-ai/technology/ai/nano-banana-2/, 2026.

[10] Team Seedream, Yunpeng Chen, Yu Gao, Lixue Gong, Meng Guo, Qiushan Guo, Zhiyao Guo, Xiaoxia Hou, Weilin Huang, Yixuan Huang, et al. Seedream 4.0: Toward next-generation multimodal image generation. *arXiv preprint arXiv:2509.20427*, 2025.

[11] OpenAI. Gpt image 2. https://developers.openai.com/api/docs/models/gpt-image-2, 2026.

[12] Miles Brundage, Shahar Avin, Jack Clark, Helen Toner, Peter Eckersley, Ben Garfinkel, Allan Dafoe, Paul Scharre, Thomas Zeitzoff, Bobby Filar, et al. The malicious use of artificial intelligence: Forecasting, prevention, and mitigation. *arXiv preprint arXiv:1802.07228*, 2018.

[13] Rowan Zellers, Ari Holtzman, Hannah Rashkin, Yonatan Bisk, Ali Farhadi, Franziska Roesner, and Yejin Choi. Defending against neural fake news. *Advances in neural information processing systems*, 32, 2019.

[14] Gowthami Somepalli, Vasu Singla, Micah Goldblum, Jonas Geiping, and Tom Goldstein. Diffusion art or digital forgery? investigating data replication in diffusion models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 6048–6058, 2023.

[15] Gowthami Somepalli, Vasu Singla, Micah Goldblum, Jonas Geiping, and Tom Goldstein. Understanding and mitigating copying in diffusion models. *Advances in Neural Information Processing Systems*, 36:47783–47803, 2023.

[16] Yuxin Wen, Yuchen Liu, Chen Chen, and Lingjuan Lyu. Detecting, explaining, and mitigating memorization in diffusion models. In *The Twelfth International Conference on Learning Representations*, 2024.

[17] 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.

[18] Cheng Xiong, Chuan Qin, Guorui Feng, and Xinpeng Zhang. Flexible and secure watermarking for latent diffusion model. In *Proceedings of the 31st ACM International Conference on Multimedia*, pages 1668–1676, 2023.

[19] Rui Min, Sen Li, Hongyang Chen, and Minhao Cheng. A watermark-conditioned diffusion model for ip protection. In *European Conference on Computer Vision*, pages 104–120. Springer, 2024.

[20] Hai Ci, Yiren Song, Pei Yang, Jinheng Xie, and Mike Zheng Shou. Wmadapter: Adding watermark control to latent diffusion models. *arXiv preprint arXiv:2406.08337*, 2024.

[21] Yuxin Wen, John Kirchenbauer, Jonas Geiping, and Tom Goldstein. Tree-ring watermarks: Fingerprints for diffusion images that are invisible and robust. *arXiv preprint arXiv:2305.20030*, 2023.

[22] 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.

[23] Hai Ci, Pei Yang, Yiren Song, and Mike Zheng Shou. Ringid: Rethinking tree-ring watermarking for enhanced multi-key identification. In *European conference on computer vision*, pages 338–354. Springer, 2024.

[24] Kasra Arabi, Benjamin Feuer, R Teal Witter, Chinmay Hegde, and Niv Cohen. Hidden in the noise: Two-stage robust watermarking for images. *arXiv preprint arXiv:2412.04653*, 2024.

[25] Jiren Zhu, Russell Kaplan, Justin Johnson, and Li Fei-Fei. Hidden: Hiding data with deep networks. In *Proceedings of the European conference on computer vision (ECCV)*, pages 657–672, 2018.

[26] Matthew Tancik, Ben Mildenhall, and Ren Ng. Stegastamp: Invisible hyperlinks in physical photographs. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 2117–2126, 2020.

[27] Kevin Alex Zhang, Lei Xu, Alfredo Cuesta-Infante, and Kalyan Veeramachaneni. Robust invisible video watermarking with attention. *arXiv preprint arXiv:1909.01285*, 2019.

[28] Ingemar J Cox, Joe Kilian, Tom Leighton, and Talal Shamoon. Secure spread spectrum watermarking for images, audio and video. In *Proceedings of 3rd IEEE international conference on image processing*, volume 3, pages 243–246. IEEE, 1996.

[29] Joseph JK O'Ruanaidh and Thierry Pun. Rotation, scale and translation invariant digital image watermarking. In *Proceedings of International Conference on Image Processing*, volume 1, pages 536–539. IEEE, 1997.

[30] Martin Kutter and Fabien AP Petitcolas. Fair benchmark for image watermarking systems. In *Security and watermarking of multimedia contents*, volume 3657, pages 226–239. SPIE, 1999.

[31] Gerhard C Langelaar, Iwan Setyawan, and Reginald L Lagendijk. Watermarking digital image and video data. a state-of-the-art overview. *IEEE Signal processing magazine*, 17(5):20–46, 2000.

[32] Chin-Chen Chang, Piyu Tsai, and Chia-Chen Lin. Svd-based digital image watermarking scheme. *Pattern Recognition Letters*, 26(10):1577–1586, 2005.

[33] Ali Al-Haj. Combined dwt-dct digital image watermarking. *Journal of computer science*, 3(9):740–746, 2007.

[34] J Cox Ingemar, Matthew L Miller, A Bloom Jeffrey, Jessica Fridrich, and Ton Kalker. *Digital watermarking and steganography*. Elsevier Inc., 2008.

[35] Jamie Hayes and George Danezis. Generating steganographic images via adversarial training. *Advances in neural information processing systems*, 30, 2017.

[36] Arpit Bansal, Ping-yeh Chiang, Michael J Curry, Rajiv Jain, Curtis Wigington, Varun Manjunatha, John P Dickerson, and Tom Goldstein. Certified neural network watermarks with randomized smoothing. In *International Conference on Machine Learning*, pages 1450–1465. PMLR, 2022.

[37] Jianwei Fei, Zhihua Xia, Benedetta Tondi, and Mauro Barni. Supervised gan watermarking for intellectual property protection. In *2022 IEEE international workshop on information forensics and security (WIFS)*, pages 1–6. IEEE, 2022.

[38] Sam Gunn, Xuandong Zhao, and Dawn Song. An undetectable watermark for generative image models. *arXiv preprint arXiv:2410.07369*, 2024.

[39] Sung Ju Lee and Nam Ik Cho. Semantic watermarking reinvented: Enhancing robustness and generation quality with fourier integrity. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 18759–18769, 2025.

[40] Kecen Li, Zhicong Huang, Xinwen Hou, and Cheng Hong. Gaussmarker: Robust dual-domain watermark for diffusion models. *arXiv preprint arXiv:2506.11444*, 2025.

[41] Jan Kociszewski, Hubert Jastrzębski, Tymoteusz Stępkowski, Filip Manijak, Krzysztof Rojek, Franziska Boenisch, and Adam Dziedzic. Serum: Simple, efficient, robust, and unifying marking for diffusion-based image generation. In C. Vondrick, B. Hariharan, C. Raffel, L. Pinto, D. Yang, and A. Faust, editors, *International Conference on Learning Representations*, volume 2026, pages 71380–71405, 2026.

[42] Wenda Li, Huijie Zhang, and Qing Qu. Shallow diffuse: Robust and invisible watermarking through low-dimensional subspaces in diffusion models. *arXiv preprint arXiv:2410.21088*, 2024.

[43] Weitao Feng, Wenbo Zhou, Jiyan He, Jie Zhang, Tianyi Wei, Guanlin Li, Tianwei Zhang, Weiming Zhang, and Nenghai Yu. Aqualora: Toward white-box protection for customized stable diffusion models via watermark lora. *arXiv preprint arXiv:2405.11135*, 2024.

[44] Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 586–595, 2018.

[45] Stability AI. Stable diffusion 3.5 medium. https://huggingface.co/stabilityai/stable-diffusion-3.5-medium, 2024.

[46] Black Forest Labs. Flux.1-dev. https://huggingface.co/black-forest-labs/FLUX.1-dev, 2024.

[47] Leyi Pan, Sheng Guan, Zheyu Fu, Luyang Si, Huan Wang, Zian Wang, Hanqian Li, Xuming Hu, Irwin King, Philip S Yu, et al. Markdiffusion: An open-source toolkit for generative watermarking of latent diffusion models. *arXiv preprint arXiv:2509.10569*, 2025.

[48] Jack Hessel, Ari Holtzman, Maxwell Forbes, Ronan Le Bras, and Yejin Choi. Clipscore: A reference-free evaluation metric for image captioning. In *Proceedings of the 2021 conference on empirical methods in natural language processing*, pages 7514–7528, 2021.

[49] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. *Advances in neural information processing systems*, 30, 2017.

[50] Quan Huynh-Thu and Mohammed Ghanbari. Scope of validity of psnr in image/video quality assessment. *Electronics letters*, 44(13):800–801, 2008.

[51] Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*, 13(4):600–612, 2004.

[52] KA Navas, Mathews Cheriyan Ajay, M Lekshmi, Tampy S Archana, and M Sasikumar. Dwt-dct-svd based watermarking. In *2008 3rd international conference on communication systems software and middleware and workshops (COMSWARE'08)*, pages 271–274. IEEE, 2008.

[53] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.

[54] Gustavosta. Stable-diffusion-prompts. https://huggingface.co/datasets/Gustavosta/Stable-Diffusion-Prompts, 2022.

[55] Yuval Kirstain, Adam Polyak, Uriel Singer, Shahbuland Matiana, Joe Penna, and Omer Levy. Pick-a-pic: An open dataset of user preferences for text-to-image generation. *Advances in neural information processing systems*, 36:36652–36663, 2023.

[56] Anish Mittal, Rajiv Soundararajan, and Alan C Bovik. Making a "completely blind" image quality analyzer. *IEEE Signal processing letters*, 20(3):209–212, 2012.

[57] Vishal Asnani, John Collomosse, Tu Bui, Xiaoming Liu, and Shruti Agarwal. Promark: Proactive diffusion watermarking for causal attribution. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10802–10811, June 2024.

CLaW: Codec-Guided Adaptive Latent Watermarking for Traceable Diffusion Image Generation Appendix

Contents:

A Background **16**

A.1 Latent Diffusion Models 16

A.2 Flow-Matching Text-to-Image Backbones 16

A.3 Multi-bit Watermark Detection 16

B Additional Algorithmic Details and Analysis **16**

B.1 Late-Denoising Window Schedule and Terminal Anchor 17

B.2 Mechanistic Analysis of CLaW Injection 18

C Additional Experimental Details **19**

C.1 Codec Pretraining and Generation Setup 19

C.2 Evaluation Metrics and Image Attack Settings 20

D Additional Results **20**

D.1 Robustness under Image Attacks 20

D.2 Robustness under Regeneration Attack 21

D.3 Algorithmic Efficiency 21

D.4 Visual Quality and Semantic Fidelity 21

D.5 Qualitative Comparisons 21

E Additional Algorithm Design Analysis **26**

E.1 Injection Schedule Ablation 26

E.2 Watermark Strength and Semantic Consistency 26

F Conceptual Comparison with Related Methods **27**

A Background

A.1 Latent Diffusion Models

Diffusion models synthesize data by learning the reverse dynamics of a gradual noising process [1]. Latent diffusion models improve computational efficiency by transferring this denoising process from pixel space to the compact latent space of a pretrained autoencoder [3]. An image is encoded as $\boldsymbol{z} = E_{\text{vae}}(\boldsymbol{x})$ and decoded as $\boldsymbol{x} = D_{\text{vae}}(\boldsymbol{z})$. In the standard noise-prediction formulation, the forward process samples $\boldsymbol{z}_t \sim \mathcal{N}(\sqrt{\bar{\alpha}_t}\boldsymbol{z}_0, (1-\bar{\alpha}_t)\mathbf{I})$, and the denoiser is trained with:

$$\mathcal{L}_{\text{DM}} = \mathbb{E}_{t,\boldsymbol{z}_0,\epsilon} \left\| \epsilon - \epsilon_\theta(\boldsymbol{z}_t, t, c) \right\|_2^2 . \tag{9}$$

At inference time, sampling starts from Gaussian noise and produces a sequence of intermediate latents, with the final latent decoded by D_{vae}. CLaW is a plug-in inference-time mechanism: it keeps the text encoder, diffusion backbone, scheduler, and VAE frozen, and only injects a small payload-dependent residual into selected late denoising steps.

A.2 Flow-Matching Text-to-Image Backbones

Our experiments use modern latent text-to-image backbones such as SD3.5 and FLUX, which are better described as flow-matching or rectified-flow pipelines rather than classical DDPM samplers. These models learn a velocity field that transports a noise latent toward a data latent. A common straight-line path is $\boldsymbol{z}_t = (1-t)\epsilon + t\boldsymbol{z}_0$, with target velocity $u_t = \boldsymbol{z}_0 - \epsilon$ and generative ODE $d\boldsymbol{z}_t/dt = v_\theta(\boldsymbol{z}_t, t, c)$. The corresponding flow-matching loss is

$$\mathcal{L}_{\text{FM}} = \mathbb{E}_{t,\boldsymbol{z}_0,\epsilon} \left\| v_\theta(\boldsymbol{z}_t, t, c) - u_t \right\|_2^2 . \tag{10}$$

In high-resolution text-to-image systems, the velocity field v_θ is often parameterized by a transformer over latent tokens, as in rectified-flow transformer models [5]. However, CLaW does not depend on this internal parameterization. It only assumes access to intermediate latent states and the final VAE decoder, which are shared by both DDPM-style and flow-matching pipelines. Thus, CLaW operates purely at the sampler interface, injecting a codec-generated residual into the latent trajectory without modifying transformer blocks, attention layers, or model weights.

A.3 Multi-bit Watermark Detection

For user-level traceability, each registered user is assigned a payload $\boldsymbol{b} \in \{0,1\}^K$, and the detector recovers $\hat{\boldsymbol{b}} \in \{0,1\}^K$ from a suspicious image. Let $M(\boldsymbol{b}, \hat{\boldsymbol{b}}) = \sum_{k=1}^{K} \mathbb{I}[b_k = \hat{b}_k]$ be the payload match count. Under the null hypothesis, an unrelated image yields decoded bits that match a fixed payload independently with probability $1/2$ per bit, i.e., $M \sim \text{Binomial}(K, 1/2)$, so the false-positive probability for accepting a match at threshold m is:

$$\text{FPR}(m) = \Pr[M \geq m] = \sum_{j=m}^{K} \binom{K}{j} 2^{-K}. \tag{11}$$

This threshold defines the attribution criterion: CLaW accepts the best registry match only when it is unlikely under the random-bit null model.

B Additional Algorithmic Details and Analysis

Algorithm 1 summarizes the full CLaW pipeline, including watermark injection and image-level detection. Watermark Injection uses the frozen secret encoder to produce a payload-dependent latent residual, adds it in a late denoising window, and optionally adapts its gain using decoder-based preview confidence. Watermark Detection decodes the payload from a suspicious image and matches it against the registered user-payload database.

Algorithm 1 **C**odec-guided **La**tent **W**atermarking (CLaW)

Inject watermark:
Input: prompt c, seed s, payload $\boldsymbol{b}$
Given: sampler S_θ with timesteps $\{\tau_i\}_{i=0}^{N-1}$; frozen codec (E_s, D_s); VAE decoder D_{vae}
Hyperparameters: $\rho_s, \rho_e, \alpha, \tau, \lambda, g_{\min}, g_{\max}, s_{\text{pre}}$; optional terminal-anchor scale γ
Output: watermarked image $\boldsymbol{x}_w$
$\boldsymbol{z}^{(0)} \leftarrow \text{Init}(s)$
$\delta_s \leftarrow E_s(\boldsymbol{b})$;
$\bar{\delta}_s \leftarrow s_{\text{vae}}\delta_s$
$i_s \leftarrow \lfloor \rho_s N \rfloor$; $i_e \leftarrow \lceil \rho_e N \rceil$; $\mathcal{W} \leftarrow \{i_s, \ldots, i_e - 1\}$
$\{w_i\}_{i\in\mathcal{W}} \leftarrow \text{NormalizeSchedule}(\mathcal{W})$; $g_{i-} \leftarrow 1$
 # Late-Denoising Window Watermark Injection (Sec. 3.2).
for $i = 0, \ldots, N-1$ **do**
  if $i \in \mathcal{W}$ **then**
    # Decoder-Guided Adaptive Watermark Injection (Sec. 3.3).
    if $(i - i_s) \bmod s_{\text{pre}} = 0$ **then**
      $\boldsymbol{x}_{\text{pre}}^{(i)} \leftarrow D_{\text{vae}}(\boldsymbol{z}^{(i)})$
      $P_m(i) \leftarrow K^{-1} \sum_{k=1}^{K} \text{softmax}(D_s(\boldsymbol{x}_{\text{pre}}^{(i)})_k)_{b_k}$
      $g_i \leftarrow \text{clip}_{[g_{\min}, g_{\max}]}(1 + \lambda[\tau - P_m(i)]_+)$
    else
      $g_i \leftarrow g_{i-}$
    end if
    $\delta_i^{\text{ada}}(\boldsymbol{b}) \leftarrow \alpha w_i g_i \bar{\delta}_s$
    $\tilde{\boldsymbol{z}}^{(i)} \leftarrow \boldsymbol{z}^{(i)} + \delta_i^{\text{ada}}(\boldsymbol{b})$
  else
    $\tilde{\boldsymbol{z}}^{(i)} \leftarrow \boldsymbol{z}^{(i)}$
  end if
  $\boldsymbol{z}^{(i+1)} \leftarrow S_\theta(\tilde{\boldsymbol{z}}^{(i)}, \tau_i, c)$
end for
 # Optional terminal anchor: reinforce the watermark signal before VAE decoding.
if $\gamma > 0$ **then**
  $\boldsymbol{z}^{(N)} \leftarrow \boldsymbol{z}^{(N)} + \gamma\bar{\delta}_s$
end if
return $\boldsymbol{x}_w = D_{\text{vae}}(\boldsymbol{z}^{(N)})$

Detect watermark:
Input: suspicious image $\bar{\boldsymbol{x}}$
Given: frozen decoder D_s; payload database $\{\boldsymbol{b}^{(u)}\}_{u\in\mathcal{U}}$; threshold κ
Output: decoded payload $\hat{\boldsymbol{b}}$ and attributed user $\hat{u}$, or reject
$\hat{\boldsymbol{b}} \leftarrow \text{Decode}_{D_s}(\bar{\boldsymbol{x}})$
$\hat{u} \leftarrow \arg\max_{u\in\mathcal{U}} M(\boldsymbol{b}^{(u)}, \hat{\boldsymbol{b}})$
if $M(\boldsymbol{b}^{(\hat{u})}, \hat{\boldsymbol{b}}) \geq \kappa$ **then**
  return $\hat{\boldsymbol{b}}, \hat{u}$
else
  return $\hat{\boldsymbol{b}}$, reject
end if

B.1 Late-Denoising Window Schedule and Terminal Anchor

Let $L = |\mathcal{W}|$ and let $j(t)$ denote the local index of timestep $t \in \mathcal{W}$ in the sampling order from $\boldsymbol{z}_T$ to $\boldsymbol{z}_0$. We support flat, linear, and cosine base schedules:

$$\bar{w}_{j(t)} = \begin{cases} 1, & \text{flat}, \\ j(t) + 1, & \text{linear}, \\ 1 - \cos\left(\frac{j(t)+1}{L}\frac{\pi}{2}\right), & \text{cosine}, \end{cases} \qquad w_t = \frac{\bar{w}_{j(t)}}{\sum_{m=0}^{L-1} \bar{w}_m}. \tag{12}$$

The normalized schedule keeps the overall watermark budget controlled by the global scale α. After the final denoising step, CLaW optionally adds the terminal anchor $\boldsymbol{z}_0 \leftarrow \boldsymbol{z}_0 + \gamma\Delta_{\mathcal{M}}(\boldsymbol{b})$, where γ is the terminal-anchor scale. This update reinforces the terminal payload signal before VAE decoding.

B.2 Mechanistic Analysis of CLaW Injection

CLaW can be viewed as a controlled watermark perturbation of the latent generation trajectory. This subsection provides a local first-order mechanistic perspective on how this perturbation affects generation in modern DiT and flow-matching samplers. Let the clean conditional sampler dynamics be:

$$\boldsymbol{z}^{(i+1)} = \Phi_i(\boldsymbol{z}^{(i)}; c), \qquad i = 0, \dots, N-1, \tag{13}$$

where Φ_i denotes the one-step transition map induced by the transformer prediction and the scheduler update at step i. Starting from the same initial noise, CLaW adds a payload-dependent residual to the latent before selected sampler updates:

$$\tilde{\boldsymbol{z}}^{(i)} = \hat{\boldsymbol{z}}^{(i)} + \alpha_i \bar{\delta}_s, \qquad \hat{\boldsymbol{z}}^{(i+1)} = \Phi_i(\tilde{\boldsymbol{z}}^{(i)}; c), \tag{14}$$

with

$$\alpha_i = \begin{cases} \alpha\, w_i\, g_i, & i \in \mathcal{W}, \\ 0, & i \notin \mathcal{W}. \end{cases} \tag{15}$$

Here $\bar{\delta}_s$ is the scheduler-space watermark residual defined in Section 3.2, and α_i is determined by the global watermark scale, the late-window schedule, and the adaptive gain.

VAE-to-Sampler Residual Conversion. The injection rule above assumes a residual expressed in the sampler latent space. We now describe how the codec-produced VAE-space residual is converted into this form. During codec pretraining, the secret encoder maps each payload $\boldsymbol{b}$ to an additive residual $\delta_s = E_s(\boldsymbol{b})$ in the VAE latent space. Modern latent DiT and flow-matching generators, such as SD3.5-medium and FLUX.1-dev, evolve states in a VAE-derived latent representation, while their samplers use model-specific latent scaling or packed layouts. Accordingly, CLaW resizes the residual when necessary and applies the VAE scaling factor to obtain the scheduler-space residual $\bar{\delta}_s = s_{\text{vae}}\delta_s$ used in Eq. (14). This conversion maps the codec residual into the sampler space, allowing it to be added directly at selected denoising steps. Its persistence through the final image depends on the remaining sampler dynamics and the chosen injection window.

Late-Window Watermark Signal Propagation. We next use a first-order expansion around the clean sampling trajectory to analyze how this sampler-space residual propagates after injection. Let $\epsilon^{(i)} = \hat{\boldsymbol{z}}^{(i)} - \boldsymbol{z}^{(i)}$ denote the deviation between the watermarked and clean trajectories, which are initialized from the same noise. Since CLaW adds the scheduler-space residual before selected sampler updates, the next-step deviation is driven by both the accumulated trajectory deviation and the current watermark perturbation. A first-order expansion of Eq. (14) around the clean trajectory gives

$$\epsilon^{(i+1)} \approx J_i\big(\epsilon^{(i)} + \alpha_i \bar{\delta}_s\big), \qquad J_i := \frac{\partial \Phi_i(\boldsymbol{z}^{(i)}; c)}{\partial \boldsymbol{z}^{(i)}}. \tag{16}$$

where J_i denotes the local Jacobian of the one-step sampler transition evaluated at $\boldsymbol{z}^{(i)}$, and α_i is the effective watermark strength at step i. This recursion shows how the watermark residual is propagated by the local first-order sampler dynamics.

Unrolling Eq. (16) from the shared initialization $\epsilon^{(0)} = 0$ gives

$$\epsilon^{(N)} \approx \sum_{i=0}^{N-1} \left(\prod_{r=i}^{N-1} J_r \right) \alpha_i \bar{\delta}_s. \tag{17}$$

This expression decomposes the terminal latent deviation into the contributions of step-wise watermark perturbations. Each contribution is propagated through the remaining sampler transitions after its insertion. Therefore, an early residual is transformed by a longer chain of Jacobians, whereas a late-window residual follows a shorter path to the terminal latent.

The terminal latent deviation is finally projected to image space by the VAE decoder. Using a first-order approximation around the clean terminal latent, we obtain:

$$\boldsymbol{x}_w - \boldsymbol{x} \approx J_{\text{vae}}\, \epsilon^{(N)}, \tag{18}$$

Combining this approximation with Eq. (17) gives

$$\|\boldsymbol{x}_w - \boldsymbol{x}\| \le \|J_{\text{vae}}\| \sum_{i=0}^{N-1} \left\| \prod_{r=i}^{N-1} J_r \right\| \alpha_i \|\bar{\delta}_s\|. \tag{19}$$

The upper bound highlights the role of injection timing: early residuals traverse longer chains of sampler transitions and are more likely to perturb layout, object identity, or composition, whereas late-window residuals follow shorter paths after the main image content has largely formed.

To relate propagation to semantic preservation, consider an idealized local decomposition into semantic and watermark-sensitive detail components. Under this decomposition, the local Jacobian can be written in block form as:

$$J_i = \begin{bmatrix} J_i^{\mathrm{sem,sem}} & J_i^{\mathrm{sem,wm}} \\ J_i^{\mathrm{wm,sem}} & J_i^{\mathrm{wm,wm}} \end{bmatrix}, \tag{20}$$

where "sem" denotes semantic components such as layout, object identity, and composition, and "wm" denotes watermark-sensitive components relevant to payload decoding.

At late denoising steps, the main image content has largely formed, so semantic and watermark-sensitive components are expected to interact weakly:

$$\|J_i^{\mathrm{sem,wm}}\|,\ \|J_i^{\mathrm{wm,sem}}\| \le \varepsilon_{\mathrm{late}}, \qquad i \in \mathcal{W}. \tag{21}$$

This approximation provides an explanatory view: a watermark-sensitive residual can retain payload-relevant information while causing limited semantic drift, which helps explain why late-window injection maintains payload decodability with limited impact on the generated content.

Decoder-Guided Adaptive Watermark Injection. Since residual propagation depends on the local sampler dynamics, the effect of a fixed watermark strength varies across trajectories. CLaW therefore uses the frozen secret decoder as a lightweight feedback module to adapt the watermark strength during generation. At preview steps, CLaW decodes the current latent and computes the payload confidence $P_m(i)$ in Eq. (5). For a latent $\boldsymbol{z}$ and payload $\boldsymbol{b}$, we write

$$P_m(\boldsymbol{z}; \boldsymbol{b}) = \frac{1}{K} \sum_{k=1}^{K} \mathrm{softmax}(D_s(D_{\mathrm{vae}}(\boldsymbol{z}))_k)_{b_k}.$$

For a small perturbation $\eta \bar{\delta}_s$, a first-order expansion gives

$$P_m(\boldsymbol{z} + \eta \bar{\delta}_s; \boldsymbol{b}) \approx P_m(\boldsymbol{z}; \boldsymbol{b}) + \eta \left\langle \nabla_{\boldsymbol{z}} P_m(\boldsymbol{z}; \boldsymbol{b}), \bar{\delta}_s \right\rangle, \tag{22}$$

where η is a small scalar perturbation strength, $\nabla_{\boldsymbol{z}} P_m(\boldsymbol{z}; \boldsymbol{b})$ is the latent-space confidence gradient, and $\langle \cdot, \cdot \rangle$ denotes the inner product.

Eq. (22) suggests that the residual can increase payload confidence when its induced latent update has a positive projection onto the confidence gradient. CLaW does not compute this gradient explicitly. Instead, the pretrained codec provides a payload-specific residual direction, and the confidence gap $\tau - P_m(i)$ serves as a lightweight feedback signal. When the preview confidence is sufficient, the gain remains unchanged; otherwise, CLaW increases the gain within the clipping range to reinforce the current trajectory. This adaptive rule strengthens difficult trajectories while keeping the generator frozen, without requiring confidence to increase at every step.

C Additional Experimental Details

C.1 Codec Pretraining and Generation Setup

Watermark codec pretraining. CLaW uses the latent watermarking codec described in Section 3.1. The codec is trained on MS-COCO [53] training images resized to 512×512 and is reused for all CLaW generation experiments. Each batch samples random 48-bit payloads. A frozen VAE maps images to latent space, the secret encoder adds a payload residual, and the secret decoder recovers the payload from the decoded watermarked image. We train the secret encoder and decoder with AdamW using learning rate 1×10^{-3}, weight decay 1×10^{-4}, batch size 5, and 40 epochs. The learning rate follows a StepLR schedule with step size 2 and decay factor 0.8. The warm-up stage optimizes only the bit recovery loss, while the final stage adds LPIPS and PRVL regularization with weights 5.0 and 1.5, respectively. The augmentation layer samples identity, JPEG simulation, crop-and-resize, Gaussian blur, Gaussian noise, and color jitter. After pretraining, both E_s and D_s are frozen.

Table 7: Post-generation image attacks used for robustness evaluation. Each attack is applied before watermark detection and payload recovery.

Attack	Parameter	Robustness Aspect
JPEG Compression	quality $= 50$	compression-induced information loss
Crop + Resize	crop ratio $= 0.8$	geometric misalignment
Gaussian Blur	radius $= 3$	high-frequency attenuation
Gaussian Noise	noise std. $\sigma = 25$	additive pixel-level corruption
Brightness	factor $= 1.5$	global photometric shift
Mask	mask ratio $= 0.18$, number of masks $= 8$	partial spatial removal
Overlay	16 random strokes, stroke width $= 8$	localized occlusion
Adaptive Noise	intensity $= 0.9$, auto-select enabled	content-aware local corruption

Generation and injection settings. We evaluate CLaW on SD3.5-medium and FLUX.1-dev, covering both standard latent and packed-latent generation pipelines. For generation benchmarking, we sample prompts from the Stable-Diffusion-Prompts [54] dataset. Unless otherwise specified, we assign an independent 48-bit payload to each watermarked image. For both SD3.5-medium and FLUX.1-dev, we generate 512×512 images with 50 denoising steps, guidance scale 7.5, fp16 precision, and a FlowMatch-Euler scheduler. The backbones use their corresponding generation pipelines, i.e., StableDiffusion3Pipeline for SD3.5-medium and FluxPipeline for FLUX.1-dev. CLaW uses the same injection configuration on both backbones: $\alpha = 1.0$, $(\rho_s, \rho_e) = (0.9, 1.0)$, a linear schedule, terminal-anchor scale $\gamma = 0.1$, target probability $\tau = 0.98$, adaptive coefficient $\lambda = 2.0$, gain range $[0.75, 1.5]$, and preview period $s_{\text{pre}} = 1$.

C.2 Evaluation Metrics and Image Attack Settings

Evaluation metrics. For binary watermark detection, we report F1, accuracy, and AUC computed from detector scores over clean and watermarked images. For user-level traceability, BitAcc measures the fraction of correctly recovered payload bits. Trace accuracy is computed by matching the decoded payload to the registered user-payload database, and an attribution is accepted only when the nearest match exceeds the binomial threshold in Eq. (11).

For visual quality, we report both distribution-level and paired image-level metrics. FID [49] measures the distributional discrepancy between generated watermarked images and the MS-COCO [53] reference images. CLIPScore [48] evaluates semantic consistency between the generated image and its text prompt. PickScore [55] provides a preference-based image quality score, while NIQE [56] measures no-reference naturalness. For paired comparisons, LPIPS [44], PSNR, and SSIM [51] are computed between the watermarked image and the corresponding clean image generated with the same prompt and seed.

Image attack settings. To evaluate robustness, we apply a suite of post-generation image attacks before watermark detection and payload recovery. These attacks cover common frequency-domain, geometric, photometric, additive, and occlusion-based corruptions. Table 7 summarizes the parameter settings of each image attack and the robustness aspect it is intended to stress.

Figure 4 shows representative examples of the post-generation image attacks used in our robustness evaluation. The attack suite covers complementary robustness aspects, including compression-induced information loss, geometric misalignment, high-frequency attenuation, additive pixel-level corruption, global photometric shift, partial spatial removal, localized occlusion, and content-aware local corruption. Together, these attacks test watermark detectability and payload decodability under diverse image corruptions.

D Additional Results

D.1 Robustness under Image Attacks

Tables 8 and 9 report the per-attack robustness results omitted from the main text. Across both backbones, CLaW maintains stable detection and payload recovery across individual perturbations, complementing the averaged robustness results in the main tables.



Figure 4: Examples of post-generation image attacks on a FLUX.1-dev sample. The attacked results in the main text and supplementary tables are averaged over this perturbation suite.

D.2 Robustness under Regeneration Attack

Table 10 evaluates denoising-based regeneration, a stronger attack setting that partially re-synthesizes the image beyond isolated pixel-level perturbations. CLaW maintains stable detection and payload recovery across both backbones, whereas several baselines degrade substantially on FLUX.1-dev despite strong performance on SD3.5-medium. These results highlight the importance of backbone-agnostic robustness under regeneration attacks.

D.3 Algorithmic Efficiency

Table 11 compares the algorithmic efficiency of different watermarking methods. CLaW adds only minor generation overhead and enables millisecond-level detection with a single frozen-decoder forward pass, avoiding costly inversion.

D.4 Visual Quality and Semantic Fidelity

Tables 12 and 13 report full image-quality results on SD3.5-medium and FLUX.1-dev. CLaW preserves visual and semantic fidelity across both backbones, suggesting that its robustness is not driven by stronger visible artifacts.

D.5 Qualitative Comparisons

Figures 5 and 6 compare clean generations, baseline watermarked outputs, and CLaW outputs on representative prompts. The examples complement the quantitative quality metrics: several baselines introduce visible changes in color, composition, or local texture, whereas CLaW more consistently preserves the appearance of the clean generation across both backbones.

Table 8: Full attack-wise robustness on SD3.5-medium under common image transformations. Results are grouped by metric, and Avg. denotes the mean over the eight transformations.

METHODS	DISTORTIONS								
	JPEG	Crop+Resize	Blur	Noise	Brightness	Mask	Overlay	Adaptive Noise	Avg.
F1 ↑									
RivaGAN [27]	0.9189	0.9501	0.9213	0.9148	0.9496	0.9909	0.8318	0.9529	0.9288
Gaussian Shading [22]	**1.0000**	0.0000	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**	0.8750
RingID [23]	0.5380	0.0060	0.6591	0.4639	0.9712	0.9401	0.4962	0.8956	0.6213
PRC [38]	0.7390	0.0000	0.0198	0.3333	0.9224	0.9236	0.7063	0.8694	0.5642
WIND [24]	0.9901	0.0271	0.9886	0.9780	0.9877	0.9877	0.9804	0.9926	0.8665
GaussMarker [40]	**1.0000**	0.0000	0.9950	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**	0.8744
SFW [39]	0.6111	0.0000	0.7896	0.4791	0.9637	0.9610	0.5765	0.9276	0.6636
CLaW (ours)	0.9995	**0.9970**	0.9995	0.9924	0.9616	0.9985	0.9970	0.9919	**0.9922**
AUC ↑									
RivaGAN [27]	0.9250	0.9525	0.9270	0.9215	0.9520	0.9910	0.8560	0.9550	0.9350
Gaussian Shading [22]	**1.0000**	0.5000	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**	0.9375
RingID [23]	0.6840	0.5015	0.7140	0.6510	0.9720	0.9435	0.6650	0.9055	0.7546
PRC [38]	0.7930	0.5000	0.5050	0.6000	0.9280	0.9290	0.7730	0.8845	0.7391
WIND [24]	0.9900	0.4970	0.9885	0.9775	0.9875	0.9875	0.9800	0.9925	0.9251
GaussMarker [40]	**1.0000**	0.5000	0.9950	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**	0.9369
SFW [39]	0.7200	0.5000	0.8175	0.6575	0.9650	0.9625	0.7025	0.9325	0.7822
CLaW (ours)	0.9995	**0.9970**	0.9995	0.9925	0.9630	0.9985	0.9970	0.9920	**0.9924**
BitAcc ↑									
RivaGAN [27]	0.9383	0.9537	0.9422	0.9198	0.9565	0.9880	0.8945	0.9688	0.9452
Gaussian Shading [22]	0.9555	0.4962	0.8608	0.9024	0.9827	0.9820	0.9632	0.9623	0.8881
GaussMarker [40]	**0.9966**	0.5118	0.9762	**0.9857**	**0.9995**	**0.9995**	**0.9982**	**0.9953**	0.9329
CLaW (ours)	0.9729	**0.9625**	**0.9882**	0.9303	0.9255	0.9674	0.9501	0.9754	**0.9590**

Table 9: Full attack-wise robustness on FLUX.1-dev under common image transformations. Results are grouped by metric, and Avg. denotes the mean over the eight transformations.

METHODS	DISTORTIONS								
	JPEG	Crop+Resize	Blur	Noise	Brightness	Mask	Overlay	Adaptive Noise	Avg.
F1 ↑									
RivaGAN [27]	0.9610	0.9899	0.9770	0.9501	0.9822	**1.0000**	0.8270	0.9744	0.9577
Gaussian Shading [22]	0.9975	0.0000	**1.0000**	0.9529	**1.0000**	**1.0000**	**1.0000**	0.9744	0.8656
RingID [23]	0.0100	0.0000	0.0000	0.0000	0.9610	0.9583	0.7421	0.9071	0.4473
PRC [38]	0.2063	0.0000	0.0000	0.0296	0.9717	0.9899	0.9501	0.9247	0.5090
WIND [24]	0.9524	0.0645	**1.0000**	0.9852	0.9756	0.9804	0.9828	0.9852	0.8658
GaussMarker [40]	0.9950	0.0000	**1.0000**	**0.9950**	**1.0000**	**1.0000**	**1.0000**	**1.0000**	0.8737
SFW [39]	0.2379	0.0000	0.1900	0.0000	0.9873	0.9822	0.9418	0.9418	0.5351
CLaW (ours)	**1.0000**	**1.0000**	**1.0000**	0.9848	**1.0000**	**1.0000**	**1.0000**	0.9899	**0.9968**
AUC ↑									
RivaGAN [27]	0.9625	0.9900	0.9775	0.9525	0.9825	**1.0000**	0.8525	0.9750	0.9616
Gaussian Shading [22]	0.9975	0.5000	**1.0000**	0.9550	**1.0000**	**1.0000**	**1.0000**	0.9750	0.9284
RingID [23]	0.5025	0.5000	0.5000	0.5000	0.9625	0.9600	0.7950	0.9150	0.7044
PRC [38]	0.5575	0.5000	0.5000	0.5075	0.9725	0.9900	0.9525	0.9300	0.7388
WIND [24]	0.9500	0.4925	**1.0000**	**0.9850**	0.9750	0.9800	0.9825	0.9850	0.9188
GaussMarker [40]	0.9950	0.5000	**1.0000**	0.9950	**1.0000**	**1.0000**	**1.0000**	**1.0000**	0.9363
SFW [39]	0.5675	0.5000	0.5525	0.5000	0.9875	0.9825	0.9450	0.9450	0.7475
CLaW (ours)	**1.0000**	**1.0000**	**1.0000**	**0.9850**	**1.0000**	**1.0000**	**1.0000**	0.9900	**0.9969**
BitAcc ↑									
RivaGAN [27]	0.9581	**0.9783**	0.9733	0.9289	0.9831	0.9972	0.8934	0.9856	0.9622
Gaussian Shading [22]	0.8825	0.4880	0.8508	0.8014	0.9931	0.9972	0.9874	0.9577	0.8698
GaussMarker [40]	**0.9846**	0.4897	0.9772	**0.9437**	**0.9999**	**1.0000**	**0.9999**	**0.9862**	0.9227
CLaW (ours)	0.9725	0.9497	**0.9914**	0.9248	0.9832	0.9806	0.9697	0.9781	**0.9688**

Table 10: Robustness to denoising-based regeneration. The attack uses strength 0.15, 20 regeneration steps, guidance scale 4.5, and dataset prompts. Results are grouped by backbone.

Method	SD3.5-medium			FLUX.1-dev		
	F1 ↑	AUC ↑	BitAcc ↑	F1 ↑	AUC ↑	BitAcc ↑
RivaGAN [27]	0.7878	0.8250	0.9094	0.7097	0.7750	0.8828
Gaussian Shading [22]	**1.0000**	**1.0000**	0.9426	0.9899	0.9900	0.8077
RingID [23]	0.7879	0.8250	–	0.0000	0.5000	–
PRC [38]	0.7450	0.8000	–	0.0000	0.5000	–
WIND [24]	**1.0000**	**1.0000**	–	0.9615	0.9600	–
GaussMarker [40]	**1.0000**	**1.0000**	**0.9973**	**1.0000**	**1.0000**	**0.9439**
SFW [39]	0.9744	0.9750	–	0.0400	0.5100	–
CLaW (ours)	0.9744	0.9750	0.9396	0.9744	0.9750	0.9177

Table 11: Generation and detection efficiency on an H100 80G GPU. We report mean runtime with standard deviation. The detection time is measured per image after generation. Lower is better.

Method	SD3.5-medium		FLUX.1-dev	
	Gen. time (s)	Det. time (s)	Gen. time (s)	Det. time (s)
RivaGAN [27]	2.5617 ± 0.3592	0.5177 ± 0.0294	5.1161 ± 0.1984	0.5303 ± 0.0369
Gaussian Shading [22]	11.4012 ± 0.9618	1.5594 ± 0.2518	14.4887 ± 0.7680	4.7032 ± 0.0428
RingID [23]	2.2270 ± 0.3949	1.5889 ± 0.2624	4.5675 ± 0.0191	4.6583 ± 0.0238
PRC [38]	2.7029 ± 0.2068	3.4440 ± 0.4512	4.6993 ± 0.0314	6.8361 ± 0.8122
WIND [24]	2.3406 ± 0.3686	2.2410 ± 0.4159	4.6119 ± 0.0427	4.8455 ± 0.2438
GaussMarker [40]	11.2716 ± 0.8649	2.1072 ± 0.5018	13.9295 ± 0.6462	4.7024 ± 0.0410
SFW [39]	$\mathbf{1.8679} \pm 0.2650$	1.5767 ± 0.2165	$\mathbf{4.5857} \pm 0.1466$	4.6961 ± 0.0340
Clean	1.8341 ± 0.1897	–	4.6072 ± 0.1803	–
CLaW (ours)	1.9339 ± 0.1142	$\mathbf{0.0115} \pm 0.0047$	4.7331 ± 0.0251	$\mathbf{0.0104} \pm 0.0056$

Table 12: Full image-quality results on SD3.5-medium. Lower values indicate better performance for FID and NIQE, while higher values are better for CLIP, PickScore, PSNR, and SSIM.

Method	CLIP	FID (vs. COCO)	PickScore	NIQE	PSNR	SSIM
RivaGAN [27]	0.8758	**35.1276**	22.2388	10.0031	40.5322	**0.9820**
Gaussian Shading [22]	**0.8779**	35.5688	22.2567	10.5437	9.9909	0.1838
RingID [23]	0.8755	35.3834	22.3202	10.0119	8.5606	0.2031
PRC [38]	0.8755	35.9001	22.3141	10.1161	8.4404	0.1960
WIND [24]	0.8760	36.2584	22.2901	10.8505	8.3103	0.1901
GaussMarker [40]	0.8760	36.4749	**22.3337**	10.5729	8.4976	0.2000
SFW [39]	0.8731	37.1225	22.1375	**8.1922**	12.0641	0.4232
Clean	0.8748	35.4536	22.2853	10.2275	–	–
CLaW (ours)	0.8744	35.6994	22.1668	10.2541	**41.2460**	0.9427

Table 13: Full image-quality results on FLUX.1-dev. Lower values indicate better performance for FID and NIQE, while higher values are better for CLIP, PickScore, PSNR, and SSIM.

Method	CLIP	FID (vs. COCO)	PickScore	NIQE	PSNR	SSIM
RivaGAN [27]	**0.8693**	**38.3027**	22.5825	9.3157	40.3885	0.9817
Gaussian Shading [22]	0.8650	38.3992	22.5498	9.3590	10.8475	0.3057
RingID [23]	0.8634	38.7425	22.5805	9.5687	11.0946	0.3212
PRC [38]	0.8629	38.5748	**22.7049**	9.5986	11.0556	0.3060
WIND [24]	0.8627	39.3105	22.4199	9.7733	10.7848	0.3106
GaussMarker [40]	0.8650	38.7848	22.6271	9.4042	11.2247	0.2999
SFW [39]	0.8521	41.9550	22.1176	**8.6357**	13.8034	0.4437
Clean	0.8667	38.7804	22.6134	9.3310	–	–
CLaW (ours)	0.8677	38.3650	22.4992	9.3159	**41.0571**	**0.9948**

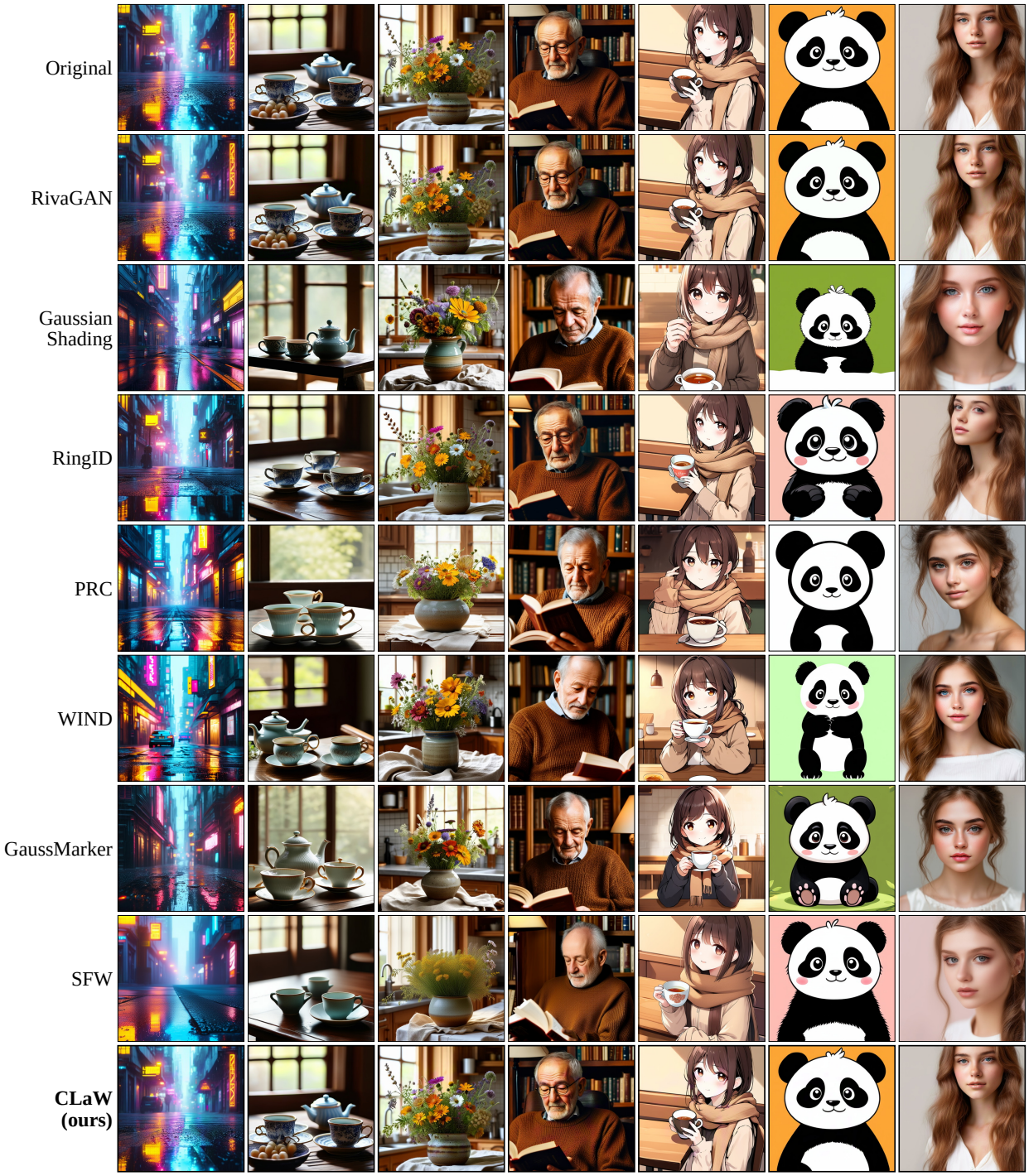


Figure 5: Qualitative comparison on SD3.5-medium. Rows show the clean generation, baseline watermarked outputs, and CLaW outputs. Columns cover representative prompt categories, including urban scenes, still-life objects, portraits, and stylized images.

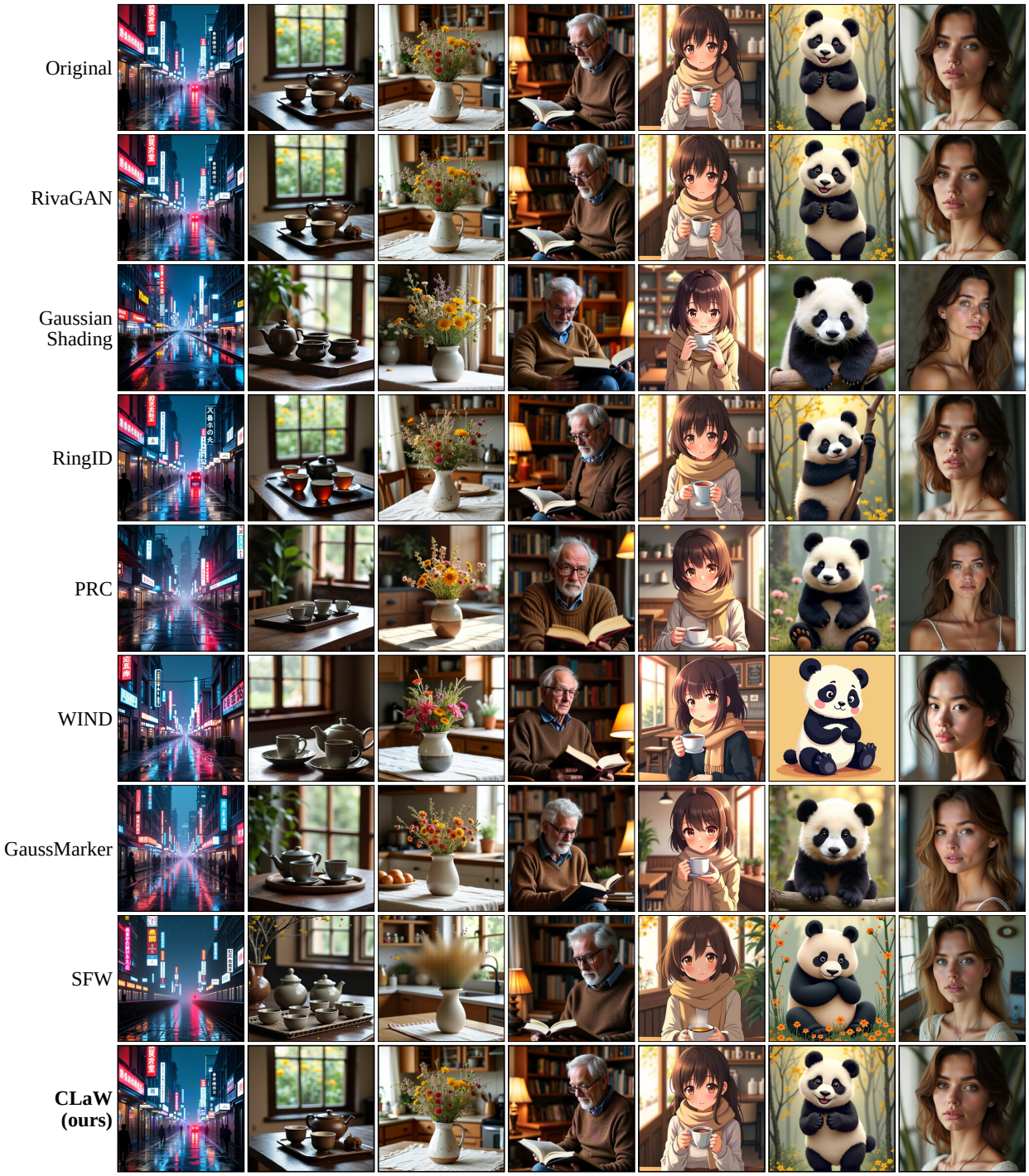
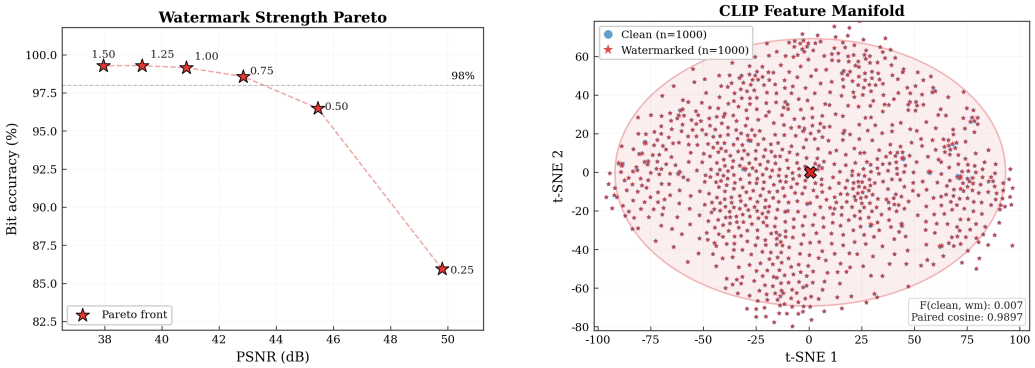


Figure 6: Qualitative comparison on FLUX.1-dev. Rows show the clean generation, baseline watermarked outputs, and CLaW outputs. Columns cover representative prompt categories, including urban scenes, still-life objects, portraits, and stylized images.

Table 14: Ablation of late-window injection schedules across backbones. Flat, linear, and cosine schedules are compared under the same injection window and adaptive controller.

Backbone	Schedule	F1 ↑	$F1_{adv}$ ↑	BitAcc ↑	$BitAcc_{adv}$ ↑	CLIP ↑	PSNR ↑	SSIM ↑
SD3.5-medium	cosine	1.0000	**0.9921**	**0.9933**	**0.9629**	0.8699	**40.0530**	0.9926
	flat	1.0000	0.9863	0.9913	0.9522	**0.8700**	39.7911	**0.9928**
	linear	1.0000	0.9914	0.9926	0.9598	0.8700	40.0424	0.9928
FLUX.1-dev	cosine	1.0000	**0.9972**	**0.9979**	**0.9723**	**0.8677**	41.0518	0.9947
	flat	1.0000	0.9959	0.9955	0.9635	0.8676	40.8491	0.9946
	linear	1.0000	0.9965	0.9971	0.9681	0.8677	**41.0571**	**0.9948**



(a) Watermark strength Pareto curve.

(b) CLIP-space visualization (t-SNE).

Figure 7: Additional CLaW design analysis. (a) PSNR–BitAcc Pareto curve under different watermark residual scales, with labels indicating tested scales. (b) t-SNE visualization of CLIP image features for clean generated images and watermarked generated images.

E Additional Algorithm Design Analysis

This section provides additional analysis of CLaW's algorithmic design choices. We first ablate the temporal schedule used for late-window injection, and then examine the watermark-strength operating point and a qualitative CLIP-space check for semantic consistency.

E.1 Injection Schedule Ablation

Table 14 compares flat, linear, and cosine late-window schedules across both backbones under the same injection window and adaptive controller. Since clean F1 is saturated for all schedules, the comparison mainly reflects attacked BitAcc and image fidelity. Cosine gives the strongest attacked BitAcc, while linear remains close in robustness and better preserves image fidelity. We therefore use linear by default, as it balances robustness and fidelity consistently across backbones.

E.2 Watermark Strength and Semantic Consistency

Figure 7 examines two design aspects of CLaW: the choice of watermark strength and a qualitative CLIP-space check for semantic consistency. The strength sweep shows the expected trade-off: smaller residual scales better preserve PSNR but weaken payload recovery, whereas larger scales improve BitAcc with diminishing returns. We choose the main setting near the knee of this curve, balancing recovery reliability and visual fidelity without using an unnecessarily large injection strength. We visualize CLIP image features using t-SNE as a qualitative check. Clean generated images and watermarked generated images occupy nearby regions in the CLIP feature space. This pattern is consistent with the CLIPScore and PSNR results, suggesting that CLaW preserves semantic content and image fidelity while enabling robust and efficient user-level traceability.

F Conceptual Comparison with Related Methods

This section provides a conceptual comparison between CLaW and related watermarking methods. We first summarize their attribution levels, generator modifications, and watermarking mechanisms in Table 15, and then discuss the key differences in watermark embedding and detection.

Table 15: Conceptual comparison of CLaW and related watermarking methods.

Method	Attribution Level	Generator Modification	Watermarking Mechanism
ProMark [57]	Concept-level	Diffusion model training	Concept-specific watermarks learned from watermarked training images
Stable Signature [17]	Model-level	VAE decoder fine-tuning	Fixed binary payload embedding through a fine-tuned VAE decoder
AquaLoRA [43]	Model-level	U-Net LoRA fine-tuning	Binary payload embedding through a diagonal scaling matrix in the watermark LoRA module
RivaGAN [27]	User-level	No diffusion/VAE update	Post-generation binary payload embedding with an attention-based encoder
SERUM [41]	User-level	No diffusion/VAE update	Initial-noise watermark embedding with separately trained pattern-specific detectors
CLaW (ours)	User-level	No diffusion/VAE update	Binary payload embedding via decoder-guided adaptive injection within a late denoising window

ProMark embeds concept-specific watermarks into training images and trains the diffusion model to retain them during generation. This enables concept-level attribution of generated images. CLaW instead embeds user-specific binary payloads during generation for attribution to registered users.

Stable Signature fine-tunes the VAE decoder to embed a fixed binary payload. Distinct user signatures therefore require separately fine-tuned decoders. **AquaLoRA** fine-tunes a watermark LoRA module in the U-Net and represents binary payloads through a diagonal scaling matrix, allowing payload changes without retraining. In comparison, CLaW supports user-specific payloads through a shared pretrained watermark codec without updating the diffusion backbone or native VAE.

RivaGAN embeds binary payloads into the final RGB image after generation using a learned attention-based encoder. Both RivaGAN and CLaW recover explicit binary payloads that can be matched to registered users. CLaW instead injects a payload-dependent latent residual within a late denoising window, using intermediate decoding confidence as feedback to adapt the watermark strength.

SERUM embeds watermark patterns into the initial diffusion noise and trains a separate detector for each pattern. Its multi-user extension assigns users distinct combinations of patterns and identifies them by combining the corresponding detector scores. Both methods keep the generator fixed, train auxiliary watermarking networks, and support inversion-free detection. In contrast to this pattern-based approach, CLaW embeds an explicit 48-bit binary payload through a shared pretrained watermark codec and directly recovers the payload bits for matching against a registered user database.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The paper contributes a codec-guided latent watermarking framework for traceable diffusion image generation, with the method introduced in Section 3, mechanistic analysis in Appendix B.2, and experimental verification in Section 4.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: We discuss the limitation of CLaW and possible directions for future work in Section 5, including its uniform latent-channel injection rule and channel-aware extensions.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [N/A]

Justification: The paper includes mathematical descriptions of the method, but does not state any theorems or provide formal proofs.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: Detailed experiment settings are provided in Section 4 and Appendix C.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [No]

Justification: The code is not publicly available at submission. We will make it open source upon acceptance.

Guidelines:

- The answer [N/A] means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (https://neurips.cc/public/guides/CodeSubmissionPolicy) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (https://neurips.cc/public/guides/CodeSubmissionPolicy) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

Answer: [Yes]

Justification: Detailed experiment settings are provided in Section 4 and Appendix C.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
- The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

Answer: [No]

Justification: Due to computational constraints, we report results under fixed experimental settings without multiple runs or statistical significance tests.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).

- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: All experiments are conducted on a single NVIDIA H100 GPU.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics https://neurips.cc/public/EthicsGuidelines?

   Answer: [Yes]

   Justification: We adhere to the NeurIPS Code of Ethics.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: The paper discusses how user-level traceability can help mitigate AI-generated content misuse and copyright infringement, while noting the risk of incorrect attribution.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.

- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: The paper does not release high-risk data or models that require safeguards.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: We use MS-COCO, Stable-Diffusion-Prompts, SD3.5-medium, FLUX.1-dev, and MarkDiffusion under their respective licenses and terms of use.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.

- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [N/A]

    Justification: The paper does not release new assets at submission. Code and documentation will be provided upon acceptance.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A].

    Justification: The paper does not involve crowdsourcing or research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: The paper does not involve human subjects research and therefore does not require IRB approval.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.

- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [N/A]

    Justification: LLMs were used only for language editing, such as grammar, spelling, and wording. The core method development does not involve LLMs as important, original, or non-standard components.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.