TARSteg: Generative Image Steganography Based on Latent TARFLOW

Erxiang Wang
Information Engineering University
Zhengzhou, China
wangerxiangxd@163.com

Chunfang Yang
Information Engineering University
Zhengzhou, China
chunfangyang@126.com

Long Yu
Information Engineering University
Zhengzhou, China
publicdragon@126.com

Chengyu Mo
Information Engineering University
Zhengzhou, China
MoChyu@163.com

Xueyuan Fu
Information Engineering University
Zhengzhou, China
redwards11@sfsu.edu

Yu Shao
Information Engineering University
Zhengzhou, China
20012019@outlook.com

***Abstract*—Existing normalizing-flow-based generative image steganography methods face two key challenges: rapidly increasing training cost as the stego-image resolution grows, and limited robustness in payload extraction. To address these issues, we propose TARSteg, a generative image steganography framework built upon latent TARFLOW. TARSteg first encodes the payload with spherical codes to improve the noise robustness of the input vectors to the flow model. It then leverages TARFLOW to map the payload into the image latent space, which reduces training cost while increasing the effective payload capacity. Finally, through adversarial training, TARSteg learns a robust image decoder and payload extractor for high-quality stego-image generation and accurate payload recovery. Extensive experiments on *LSUN-Churches*, *LSUN-Bedrooms*, and *FFHQ* show that TARSteg consistently outperforms representative generative image steganography methods, including *IDEAS*, *S2IRT*, and *LDStega*, in terms of stego-image quality, robustness, and hiding capacity. For example, under JPEG compression with a quality factor of 50, TARSteg improves the extraction accuracy from 60.20%, 70.46%, and 94.46% achieved by these methods to 96.44%.**

***Index Terms*—Generative image steganography, latent normalizing flow, TARFLOW, spherical-code, adversarial example**

I. Introduction

Digital image steganography exploits the perceptual redundancy of the human visual system to conceal secret information in an apparently normal cover image in an imperceptible manner, and then transmits it through a public channel to enable covert communication [1]. Traditional handcrafted schemes and early deep-learning-based steganographic methods often leave noticeable steganographic artifacts in the cover image, making the resulting stego images vulnerable to steganalysis, which poses a serious threat to secure information transmission. In contrast, *generative image steganography* (GIS) [2] avoids directly modifying a given cover image, thereby reducing modification traces and improving security. With the rapid development of deep learning, GIS methods have evolved into three major categories: GAN-based approaches [3], normalizing-flow-based approaches [4], [5], and diffusion-model-based approaches [6].

GANs exhibit strong image synthesis capability and can generate realistic images beyond the training distribution, which has been leveraged to increase the diversity of stego images. For example, the IDEAS [7] method proposed by Liu *et al.* establishes a unique and reversible mapping from payloads to stego images by coordinating multiple autoencoder components; in principle, different payloads correspond to different stego images. However, although GIS does not directly alter a cover image, GANs are prone to mode collapse [8], leading to unstable visual quality. As a result, low-quality stego images are more likely to arouse suspicion and be intercepted during Internet transmission. Moreover, the dimensionality of the noise input used to carry payloads is typically low, which limits the achievable capacity.

Diffusion-model-based GIS methods, represented by LDStega [2], address the image quality issues encountered in the above two categories. LDStega employs a pretrained autoencoder (e.g., a VAE [9]) for image generation and uses a latent diffusion model to learn the distribution in the autoencoder latent space. Due to the non-invertibility of diffusion models, the hiding strategy cannot be implemented by constructing an explicit mapping through the latent diffusion model. Therefore, for message embedding and extraction, LDStega first converts the message payload into a noise-like perturbation, adds it to the latent code, and then uses the decoder to transform the perturbed latent code into the stego image. By delegating high-resolution image synthesis to a pretrained autoencoder, this paradigm mitigates the low-quality high-resolution synthesis problem in flow-based GIS. Nevertheless, The surjective mapping from the VAE latent space to images limits the accurate recovery of latent-space codes (this will be elaborated in the Problem Description section). Moreover, because perturbation-based message payload typically occupies a limited numeric range, the available embedding space can be insufficient, making capacity control less flexible. Furthermore, perturbation-style payload is more susceptible to channel noise, resulting in limited robustness.

Normalizing-flow-based GIS alleviates the capacity limita-

tion of GAN-based schemes. For instance, the S2IRT [10] method proposed by Zhou *et al.* constructs an invertible mapping between message payloads and stego images by exploiting the strict bijection property of Glow. Normalizing flow requires the noise input and the image output to have the same dimensionality; therefore, generating higher-resolution images naturally increases the input dimensionality and, in principle, provides a larger space for information embedding. However, normalizing flow models exhibit poor scalability to high resolutions, since the training cost required to achieve pixel-level invertibility increases dramatically as the image resolution grows. Moreover, the robustness of payload recovery is insufficient.

In summary, compared with GAN-based and diffusion-model-based approaches, normalizing-flow-based generative image steganography methods offer two notable advantages: larger payload capacity and higher payload recovery accuracy at the same payload level. Accordingly, this paper focuses on the study of normalizing-flow-based generative image steganography. However, when the amount of hidden information increases, existing normalizing-flow-based generative image steganography methods must generate higher-resolution stego images, which leads to a dramatic increase in training cost and also suffers from insufficient robustness in payload extraction. To address these issues, this paper proposes **TARSteg**, a generative image steganography method based on latent **TARFLOW** [11], with innovations in the following three aspects:

1) We employ **TARFLOW**, a normalizing flow that can efficiently model the posterior, to establish a bijection between the autoencoder latent space and payload. It addresses the stego image distortion issue of normalizing-flow models under high-capacity embedding.
2) Inspired by adversarial examples [12], we add Gaussian perturbations with controlled magnitude to the latent code of a reference image to construct adversarial samples. Decoding these samples enforces a unique mapping from latent codes to synthesized images, providing a recoverable foundation for subsequent reversible steganography.
3) We design a reversible encoding scheme that maps bits to Gaussian noise tensors, enabling flexible message payload control while maintaining strong extraction performance and robustness under large-capacity settings.

II. Problem Description

The overall pipeline of the normalizing-flow-based generative image steganography method, termed **S2IRT**, is illustrated in Fig. 1 (a). Given a secret message to be concealed, S2IRT first applies a **Secret-to-Image (S2I)** transform to map the discrete payload into a Gaussian continuous-valued tensor that can be fed into the normalizing flow model *Glow*. The inverse pass of Glow then generates the corresponding stego image. At the receiver side, the stego image is processed by the forward pass of Glow to recover the Gaussian tensor, after which the inverse S2I transform is applied to reconstruct the hidden message.

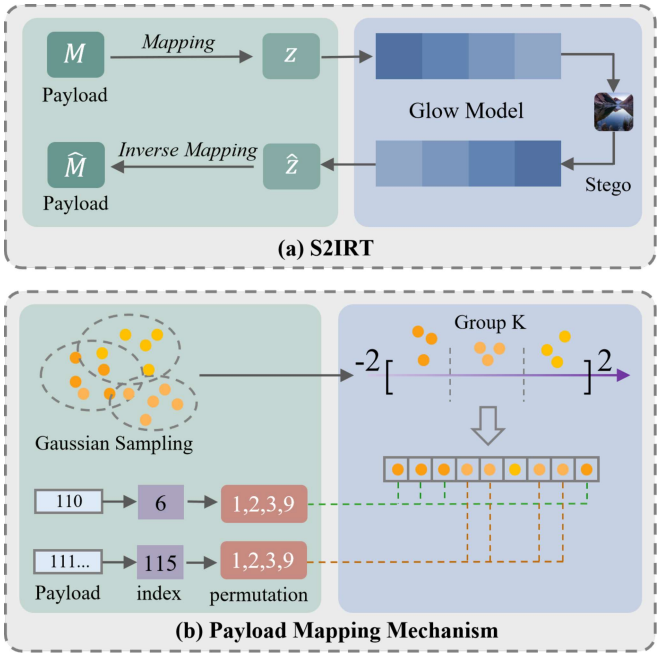


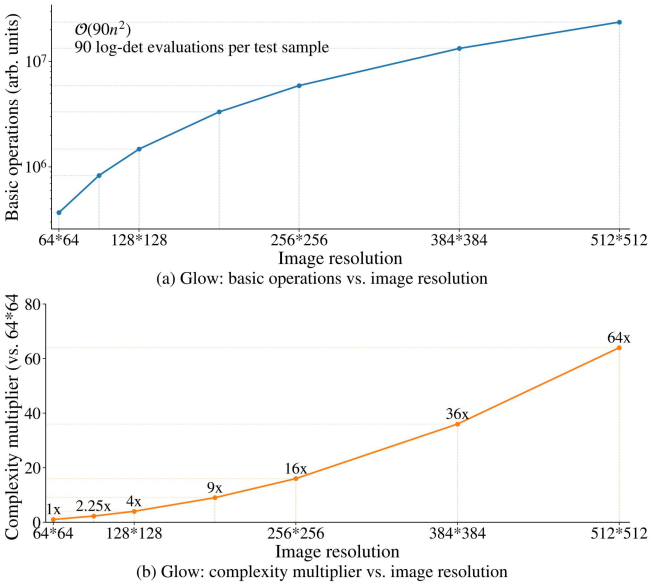
Fig. 1. (a) Overall architecture of S2IRT; (b) the mapping mechanism from the payload to the Gaussian noise tensor fed into Glow in S2IRT.

In what follows, we briefly analyze two key limitations of S2IRT: (*i*) the weakness of its payload mapping mechanism, and (*ii*) the degradation in payload recovery accuracy and the rapid increase in training cost as the image resolution grows.

A. Analysis of the Payload Mapping Mechanism

The payload mapping mechanism adopted in S2IRT, namely the **S2I** (*Secret-to-Image*) transform, is shown in Fig. 1 (b). This mechanism first samples a set of latent elements from a standard Gaussian distribution, and partitions the one-dimensional closed interval $[-2, 2]$ into K equiprobable sub-intervals to *group* these elements. It then keeps the sampled Gaussian values unchanged, while using bit slices of the payload to determine their spatial placement and permutation order in the continuous-valued tensor, thereby encoding the payload into the *positional permutation* of the sampled elements. At the receiver side, each element is assigned to a group according to its recovered numerical value, from which the permutation pattern of elements within the same group is inferred and subsequently decoded back into the payload.

A key limitation of this design lies in its reliance on *one-dimensional interval quantization* for grouping continuous Gaussian values. In practice, the extraction process inevitably introduces errors, which can be viewed as perturbations added to the recovered Gaussian values. For one-dimensional scalar elements, such perturbations act only along a single axis on the real line. Even a small perturbation may therefore move a scalar across the boundary of its original interval and into



(a) Glow: basic operations vs. image resolution

(b) Glow: complexity multiplier vs. image resolution

Fig. 2. Relationship between image resolution and computational complexity growth.

an adjacent one, causing an incorrect group assignment. As a result, the robustness of payload recovery is inherently limited.

B. Analysis of the Training Cost of the Normalizing Flow

Training a normalizing flow model requires repeated computation of Jacobian log-determinants in likelihood optimization, making its cost highly sensitive to image resolution. For a three-channel image of size $n \times n$, the data dimensionality is $d = 3n^2$, and the complexity of each Jacobian-related computation scales as $\mathcal{O}(d)$. Since the Glow model used in S2IRT stacks 96 invertible layers, increasing the resolution from 64×64 to 256×256 enlarges the per-layer cost by 16 times, leading to a substantial increase in overall training time and memory overhead. As illustrated in Fig. 2, the dominant training cost of Glow scales approximately as $\mathcal{O}(n^2)$. Under the setting of 90 accumulated Jacobian determinant evaluations per sample, the computational cost can be written as $\mathcal{O}(90n^2)$. In particular, at 256×256, the number of basic operations reaches about 6×10^6, which is 16 times larger than that under the 64×64 setting. These results show that the training cost of S2IRT grows sharply with image resolution, making high-resolution training increasingly expensive.

From the above analysis, it can be observed that existing normalizing-flow-based generative image steganography methods suffer from insufficient robustness in their payload mapping mechanisms, while their training cost increases dramatically as the resolution of the stego image grows.

III. Method

To address the degradation in payload recovery accuracy and the substantial increase in training cost suffered by normalizing-flow-based generative image steganography methods at high image resolutions, we propose TARSteg, a latent-TARFLOW-based generative steganography framework. Instead of directly modeling the high-dimensional image space with a normalizing flow, TARSteg first compresses images into a low-dimensional latent space via an autoencoder, and then performs steganographic modeling in the latent domain, thereby improving both training efficiency and payload extraction accuracy.

A. Hiding and Recovery Process

Fig. 3 (a) delineates the schematic workflow of TARSteg. During the hiding stage, the discrete payload M is transformed into a continuous variable z_1 using a Bit-to-Gaussian (B2G) mapping strategy. This transformation enforces probabilistic alignment with the standard Gaussian prior inherently required by the flow model. A latent normalizing flow, instantiated as TARFLOW [11], then establishes an invertible mapping between z_1 and the autoencoder's latent representation z_0. The stego image is subsequently generated by decoding z_0. On the receiver side, despite channel-induced corruptions (e.g., noise or compression), a pretrained extractor recovers the latent code z_0. By traversing the forward flow and the corresponding G2B mapping, the original payload M is accurately restored. Mathematically, the forward and backward processes are formulated as:

$$\begin{cases} z_1 = \mathrm{B2Gmap}(M), \\ z_0 = f_\theta^{-1}(z_1), \\ X_M = \mathrm{Dc}(z_0), \\ \widehat{z}_0 = \mathrm{Ex}(X'_M), \\ \widehat{z}_1 = f_\theta(\widehat{z}_0), \\ \widehat{M} = \mathrm{G2Bmap}(\widehat{z}_1). \end{cases} \tag{1}$$

Here, X_M is the generated stego image, and X'_M denotes the received (possibly attacked) image.

B. Pretraining Module

As shown in Fig. 4 (a), in a VAE decoder, the mapping f from the latent space code to the decoded image x is *surjective* rather than *injective*. That is, a single decoded image may correspond to multiple latent codes. Consequently, given only the decoded image (and its prior), one cannot identify a strictly defined inverse mapping that uniquely recovers the corresponding latent code. In contrast, the *injective* mapping in Fig. 4 (b) admits an inverse path in an objective sense, enabling the decoded image to be mapped back to its *unique* code. Therefore, the autoencoder paradigm required by steganography must follow the formulation in Fig. 4 (b).

Motivated by these observations, we train the autoencoder using the idea of adversarial examples. Specifically, we add perturbations with controlled magnitude to the latent code of a reference image X to generate multiple adversarial samples. Unlike a VAE, we do *not* constrain the generated images via a VAE-style reconstruction loss. Instead, we adopt adversarial training, which improves the realism of generated images while preserving their distinguishability from the reference image. This design enforces an injective latent-to-image mapping.

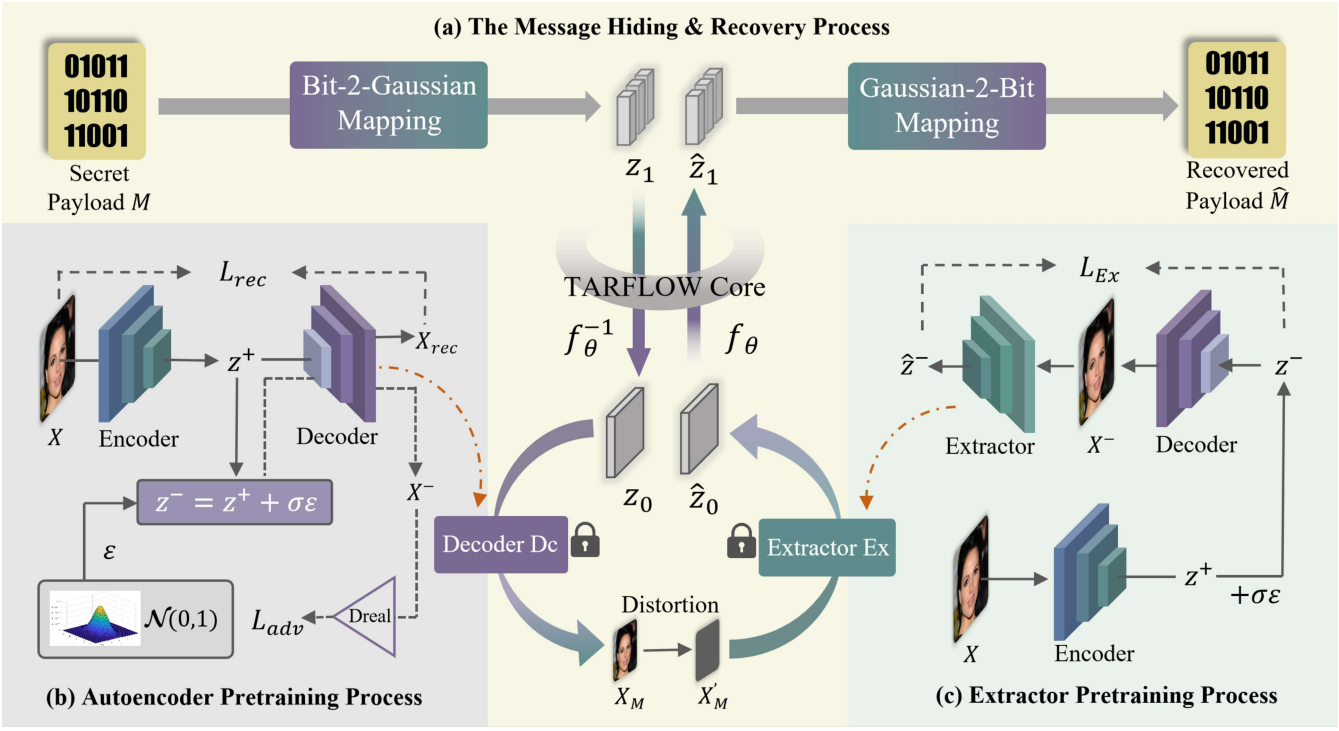


Fig. 3. Overall architecture of TARSteg: (a) The Message Hiding and Recovery Process; (b) Autoencoder pretraining Process, where Ec = Encoder, Dc = Decoder, and Dreal = Discriminator; (c) Extractor pretraining Process, where Ex = Extractor.

1) Pretrain an autoencoder.: We first pretrain an autoencoder to construct a latent code space and to synthesize high-quality images. The reference image X is encoded into a latent code z^+ and then reconstructed as X_{rec}:

$$z^+ = \mathrm{Ec}(X), \qquad X_{rec} = \mathrm{Dc}(z^+). \tag{2}$$

A reconstruction loss is used to provide supervision and prevent the decoder from producing meaningless samples:

$$\mathcal{L}_{rec} = \|X - X_{rec}\|_1. \tag{3}$$

2) Generate adversarial latent codes by Gaussian perturbation.: Given the latent code z^+, we inject Gaussian noise ϵ to obtain an adversarial code z^-:

$$\epsilon \sim \mathcal{N}(0, I), \qquad z^- = z^+ + \sigma\epsilon. \tag{4}$$

The standard deviation σ, which controls the perturbation strength, should be chosen to balance dispersion and concentration in the latent space. If σ is too large, adversarial codes become excessively scattered, which hinders TARFLOW from learning a coherent transport map. Conversely, if σ is too small, the adversarial cluster around z^+ becomes overly concentrated; sampling from such a narrowly supported region may induce overfitting in TARFLOW. Unless otherwise specified, we set $\sigma = 0.6$ in all experiments.

3) Adversarial training in the image space.: We decode z^- to obtain a generated image X^- and feed it into a discriminator Dreal:

$$X^- = \mathrm{Dc}(z^-). \tag{5}$$

We define the adversarial loss for the decoder (generator) as

$$\mathcal{L}_{adv}^{\mathrm{Dc}} = \mathrm{softplus}\big(-\mathrm{Dreal}(X^-)\big)\,, \tag{6}$$

and the discriminator loss as

$$\begin{aligned}\mathcal{L}_{adv}^{\mathrm{Dreal}} = {} & \mathrm{softplus}(-\mathrm{Dreal}(X)) \\ & + \mathrm{softplus}\big(\mathrm{Dreal}(X^-)\big)\,.\end{aligned} \tag{7}$$

where

$$\mathrm{softplus}(x) = \log(1 + e^x)\,. \tag{8}$$

Softplus function originates from [7]. Compared with a VAE that applies reconstruction constraints to *all* reparameterized samples, adversarial learning only encourages the realism of X^- through $\mathcal{L}_{adv}$. Importantly, it does not force multiple perturbed latent codes to be reconstructed into the *same* reference image, thereby avoiding a surjective latent-to-image mapping.

4) Pretrain an extractor for image-to-latent inversion.: As shown in Fig. 3 (c), we additionally pretrain an extractor Ex to recover the latent code from a decoded image with very low error. Since adversarial training promotes a stable injective mapping from the latent space to the image space, it becomes feasible (in principle) to learn the corresponding inverse mapping from images back to latent codes. During this stage, we freeze the encoder and decoder and train only the extractor. Concretely, we generate X^- from a perturbed code z^- and enforce the extractor output $\hat{z}^-$ to align with z^-:

$$\hat{z}^- = \mathrm{Ex}(X^-), \qquad \mathcal{L}_{\mathrm{Ex}} = \|\hat{z}^- - z^-\|_1. \tag{9}$$

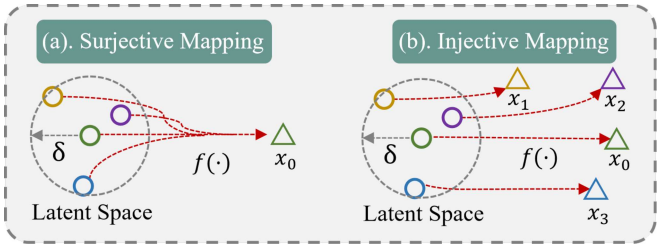


Fig. 4. Two mapping paradigms

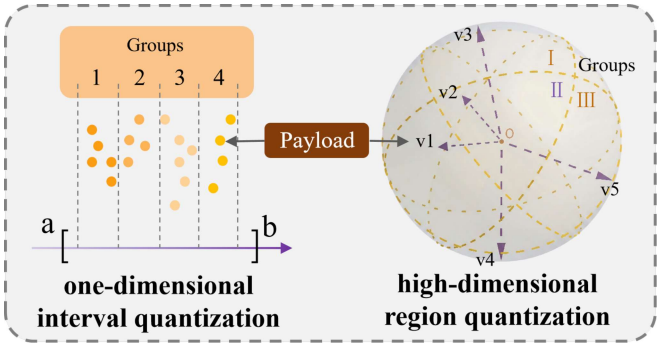


Fig. 5. Illustration of one-dimensional interval quantization versus high-dimensional interval quantization.

In this way, Ex establishes a stable correspondence between the image distribution p_X and the latent distribution p_Z, which lays the foundation for subsequent normalizing-flow modeling and inversion in the latent space.

C. Spherical-code-based Reversible Mapping Method for Message Payload

Similar to Glow, TARFLOW cannot directly process discrete payloads. Therefore, before being fed into TARFLOW, the payload must first be mapped into Gaussian continuous values. As mentioned in Section II, the payload mapping method used in S2IRT leads to insufficient robustness in the final payload recovery. To address this issue, this paper designs a spherical-code-based reversible payload mapping mechanism. As shown in Fig. 5, this mechanism uniformly partitions a high-dimensional hypersphere into multiple spherical-cone subregions, denoted as Region I, Region II, Region III, etc., and assigns a codeword vector v to the center of each region. The payload M is then mapped to a set of codeword vectors v from different regions, and finally transformed into a Gaussian noise matrix $Y \in \mathbb{R}^{H\times H}$, which is used as the Gaussian tensor z_1 input to TARFLOW. After channel transmission, the payload is recovered by determining the subregion to which each perturbed codeword vector belongs. The main procedure of the proposed reversible payload mapping mechanism is illustrated in Fig. 6 and is described as follows:

1) Basic setup: Our goal is to build a bijection between the payload M and a Gaussian noise tensor $Y \in \mathbb{R}^{H\times H}$. After flattening Y into a vector y, its dimensionality is H^2. We partition Y into non-overlapping blocks of size $B \times B$, which corresponds to splitting y into segments of length $d = B^2$. The number of segments is

$$y = \{y_1, y_2, \ldots, y_L\}, \quad L = \frac{H^2}{d}. \tag{10}$$

Each segment carries k bits, corresponding to a codebook size $Q = 2^k$.

2) Spherical codebook construction: For each segment index $b \in \{1, 2, \ldots, L\}$, we generate a spherical codebook

$$S = \{s_0, s_1, \ldots, s_{Q-1}\}, \quad \|s_q\|_2 = 1. \tag{11}$$

The codewords in the codebook S are d-dimensional vectors $\mathbf{s} \in \mathbb{R}^d$ pointing from the hypersphere center to the hypersphere $\mathbb{R}^{d-1}$, and the endpoints of $\mathbf{s}$ can uniformly partition the hypersphere. Here, q denotes the index of the codeword.

The extension of the Fibonacci lattice method to the case of $d = 4$ is to generate points that uniformly partition S^3 on the surface of a four-dimensional hypersphere S^4. The coordinates of a vector pointing to the hypersphere surface S^3 can be parameterized as follows:

$$\begin{cases} s_q^{(1)} = \cos\eta\, \cos\alpha, \\ s_q^{(2)} = \cos\eta\, \sin\alpha, \\ s_q^{(3)} = \sin\eta\, \cos\beta, \\ s_q^{(4)} = \sin\eta\, \sin\beta, \end{cases} \tag{12}$$

where α, η, and β denote the angular coordinates in the four-dimensional space, respectively, $\eta \in [0, \pi/2]$ and $\alpha, \beta \in [0, 2\pi]$.

To discretize the continuous parameter coordinates into N uniformly spaced points for generating N codeword vectors, this step adopts the spherical point generation method in the Fibonacci lattice. Specifically, starting from an initial vector, the vector is progressively rotated around the hypersphere center with equal incremental steps to produce new vectors. Therefore, for the q-th point on the hypersphere, i.e., when the codeword index is q, an incremental rotation step is defined as $x_q = \frac{q-0.5}{N}$. Then, η is rotated progressively with step x_q, while α and β are rotated with manually specified step sizes τ_1 and τ_2, respectively:

$$\begin{cases} \eta_q = \arcsin(\sqrt{x_q}), \\ \alpha_q = 2\pi\, \mathrm{frac}(q\tau_1), \\ \beta_q = 2\pi\, \mathrm{frac}(q\tau_2). \end{cases} \tag{13}$$

where τ_1 and τ_2 are irrational numbers (e.g., $\sqrt{2}$), and $\mathrm{frac}(\cdot)$ extracts the fractional part. The final codeword coordinate is

$$X_q = \begin{bmatrix} \cos\eta_q \cos\alpha_q \\ \cos\eta_q \sin\alpha_q \\ \sin\eta_q \cos\beta_q \\ \sin\eta_q \sin\beta_q \end{bmatrix}. \tag{14}$$

3) Forward mapping (B2G): Since TARFLOW takes Gaussian samples as its input, we must map the payload into samples that are identically distributed as Gaussian noise, so that the latent-flow model can generate a normal latent-space

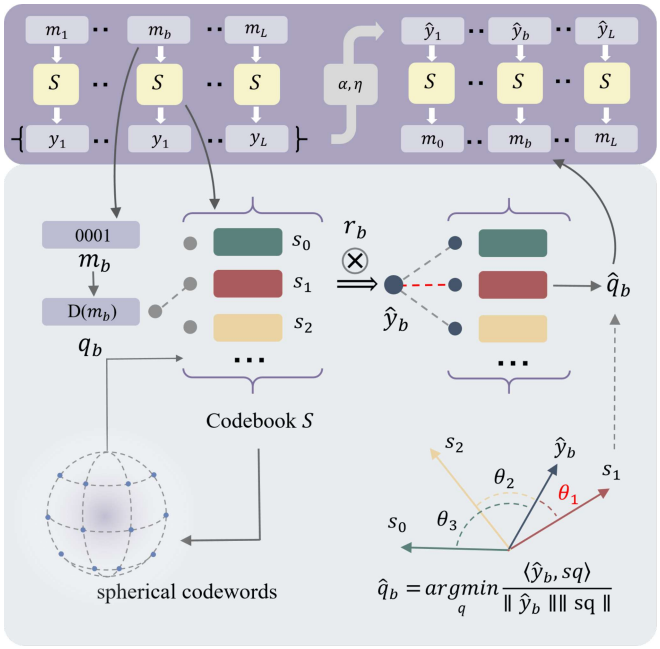


Fig. 6. Spherical-code-based reversible mapping method for message payload

representation for the image. To make each segment y_b closer to a Gaussian noise sample, we introduce a segment-specific radius r_b to "Gaussianize" the radius of the spherical code:

$$r_b = F^{-1}(u_b), \quad u_b = \mathrm{PRNG}(K, n, b), \tag{15}$$

where $F^{-1}(\cdot)$ denotes the inverse CDF of the desired radius distribution (matching the norm distribution of a d-dimensional Gaussian), and $\mathrm{PRNG}(\cdot)$ outputs a uniform random variable shared by the sender and the receiver via the parameters (K, n, b). With this design, the energy distribution of each segment matches the d-dimensional Gaussian norm, while the direction is provided by the spherical codebook, making the overall segment distribution closer to Gaussian.

We slice the payload into blocks of k bits, and interpret each block as an index

$$q_b \in \{0, 1, \ldots, Q-1\}. \tag{16}$$

Then the b-th noise segment is generated by

$$y_b = r_b \, s_{q_b}. \tag{17}$$

All segments are concatenated to recover y. Finally, we apply a global random permutation to y according to a predefined rule and reshape it back into Y, so that it more closely resembles i.i.d. Gaussian sampling.

4) Inverse mapping (G2B): On the receiver side, the global permutation is reversed to obtain the recovered sequence y. For each segment, we assume the channel effect can be approximated by

$$\widehat{y}_b \approx \omega \, y_b + \xi, \tag{18}$$

meaning that the segment may be affected by a multiplicative disturbance due to the channel scaling by a factor ω, as well as an additive disturbance ξ. During decoding, the receiver reconstructs the same codebook S and performs a nearest-neighbor decision based on similarity. To reduce sensitivity to the unknown scaling factor ω, we adopt cosine similarity:

$$\widehat{q}_b = \arg\max_q \frac{\langle \widehat{y}_b, s_q \rangle}{\|\widehat{y}_b\|_2}. \tag{19}$$

Note that $\|s_q\|_2 = 1$ in the denominator, and therefore this term is omitted. The selected index $\widehat{q}_b$ is then converted back to its corresponding k-bit block, thereby reconstructing the payload.

Compared with the payload mapping mechanism used in S2IRT, the proposed spherical-code-based reversible mapping provides two key advantages: improved robustness and higher capacity. By replacing one-dimensional interval quantization with high-dimensional hyperspherical region quantization, it makes payload recovery less sensitive to perturbations, since a codeword must be disturbed jointly across multiple dimensions before crossing into an adjacent region. Moreover, as high-dimensional vectors are used as codewords, the inter-codeword spacing remains relatively large, allowing the payload capacity to be increased by enlarging the codebook without introducing severe codeword interference.

IV. Experiments

In this section, we introduce the datasets and experimental settings used in TARSteg, followed by experimental results.

A. Experimental Setup

The experiments are implemented using the PyTorch deep learning framework and conducted on NVIDIA 6000 Ada GPU, and the latent variable size is fixed to $32 \times 32 \times 4$. At the input of TARFLOW, we use a 64×64 Gaussian tensor; with $B = 2$, we have $d = 4$, i.e., a codebook is constructed on a four-dimensional hypersphere.

To evaluate the applicability of the proposed method across different image domains, we conduct experiments on the LSUN [13] subsets *Churches* and *Bedrooms*, as well as the FFHQ [14] face dataset, The generated synthetic images are set to a resolution of 256×256. For evaluating steganographic capacity and extraction accuracy, we compare against representative generative image steganography baselines, including IDEAS [7], S2IRT, and LDStega. These baselines cover mainstream GIS paradigms based on GANs, diffusion models, and normalizing flows, and are representative in terms of model architectures and generation mechanisms.

B. Image Quality Evaluation

As shown in Table I, TARSteg achieves FID scores of 3.14, 3.12, and 2.85 on Churches, Bedrooms, and FFHQ, respectively. On all three datasets, TARSteg outperforms IDEAS, S2IRT, and LDStega. In particular, compared with LDStega, TARSteg further reduces the FID from 3.23 to 3.14 on Churches, from 4.14 to 3.12 on Bedrooms, and from 3.35 to 2.85 on FFHQ. These results indicate that TARSteg achieves the best image quality among the compared methods.

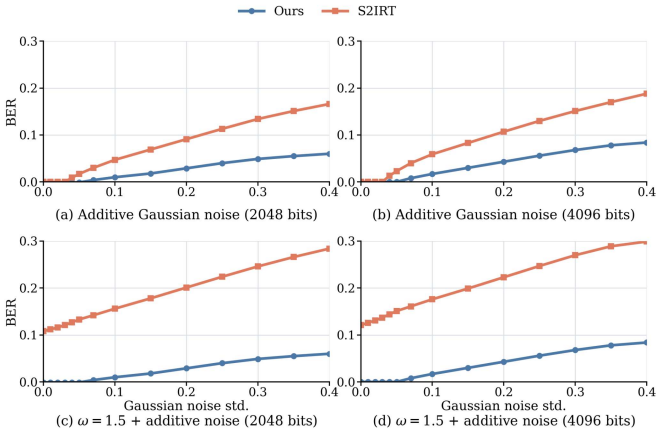


Fig. 7. Payload Robustness Comparison

TABLE I
FID OF STEGO IMAGES ON DIFFERENT DATASETS (LOWER IS BETTER).

Method	Churches	Bedrooms	FFHQ
IDEAS	16.34	13.51	15.36
S2IRT	20.31	15.57	13.38
LDStega	3.23	4.14	3.35
TARSteg	3.14	3.12	2.85

TABLE II
EXTRACTION ACCURACY UNDER NOISE AND JPEG COMPRESSION.

Attack	IDEAS	S2IRT	LDStega	TARSteg
Identity	85.42%	74.88%	98.50%	98.74%
Salt & Pepper ($\rho = 0.01\%$)	80.82%	72.45%	97.76%	96.79%
Salt & Pepper ($\rho = 0.04\%$)	75.21%	72.10%	94.69%	95.10%
JPEG compression (Q=90)	83.24%	74.02%	98.18%	98.35%
JPEG compression (Q=70)	79.94%	73.35%	96.95%	97.68%
JPEG compression (Q=50)	60.20%	70.46%	94.46%	96.44%

TABLE III
CAPACITY AND EXTRACTION ACCURACY UNDER PNG AND JPEG STORAGE.

Type	Method	Capacity	Churches	Bedrooms	FFHQ
PNG	IDEAS	2048	70.62%	69.35%	70.16%
	S2IRT	4096	70.89%	70.87%	70.69%
	LDStega	4096	98.69%	98.50%	97.75%
	TARSteg ($k = 4$)	4096	98.75%	98.50%	98.32%
	TARSteg ($k = 6$)	6144	95.37%	95.25%	95.03%
JPEG	IDEAS	2048	54.17%	60.30%	60.22%
	S2IRT	4096	70.14%	70.46%	70.09%
	LDStega	4096	95.73%	95.42%	96.15%
	TARSteg ($k = 4$)	4096	96.20%	95.85%	96.03%
	TARSteg ($k = 6$)	6144	91.30%	92.74%	91.89%

C. Robustness Evaluation

In practice, stego images transmitted through real channels may undergo distortions such as compression and additive noise. Therefore, we evaluate extraction accuracy under JPEG compression and noise attacks.

As shown in Table II, the extraction accuracy of all methods decreases as the noise level or JPEG compression strength increases. Under our attack settings, TARSteg consistently outperforms S2IRT and IDEAS, and achieves results close to LDStega, with slight advantages under certain attack strengths. These results demonstrate that the proposed method exhibits strong anti-interference capability. Compared with S2IRT, which builds a fragile bijection directly in image space, TARSteg constructs the bijection in latent space, whose representations are typically smoother under small perturbations, thereby improving robustness.

D. Capacity Evaluation

We compare extraction accuracy at fixed payloads and highlight the usability of TARSteg in large-capacity settings. Following the capacity setting described above, TARSteg is evaluated at two payload levels: $k = 4$ (4096 bits) and $k = 6$ (6144 bits). With a latent tensor size of $32 \times 32 \times 4$ and a spherical code dimension $d = 4$, there are $4096/4 = 1024$ blocks. Each block carries k bits, resulting in a total payload of $1024 \cdot k$ (i.e., 4096 bits for $k = 4$ and 6144 bits for $k = 6$). Table III reports extraction accuracy when stego images are saved in PNG (lossless) and JPEG (lossy) formats.

From Table III, we have the following findings. First, at 4096 bits, TARSteg ($k = 4$) achieves higher extraction accuracy than IDEAS, S2IRT, and StegaDDPM under both PNG and JPEG and reaches a level comparable to LDStega. In this setting, TARSteg may sometimes slightly underperform LDStega on certain datasets, potentially because the inverse mapping of the model can accumulate errors, whereas LDStega extracts perturbation-style secrets directly from the image latent space. Nevertheless, TARSteg offers greater flexibility in hiding strategies; with a proper mapping from messages to the base distribution, it can support higher payloads.

Second, when the payload increases to 6144 bits ($k = 6$), the extraction accuracy of TARSteg decreases but remains above 90%. This is because a larger k means more information per block, leading to a larger codebook and making it harder to maintain sufficient codeword separation under a fixed latent dimensionality, which increases nearest-neighbor confusion under JPEG compression and noise.

Third, for the same payload, PNG generally outperforms JPEG, because PNG is lossless, whereas JPEG introduces irreversible quantization errors that degrade the inversion accuracy of latent variables.

E. Robustness Evaluation of the Payload Mapping Scheme

In Eq. (18), ω denotes the multiplicative noise factor, which multiplies all values by ω. In the experiment, we compare the proposed spherical-code-based mapping mechanism with the one-dimensional mapping mechanism used in S2IRT under additive Gaussian noise and multiplicative noise.

As shown in Fig. 8, under additive Gaussian noise, the proposed method achieves lower BER than S2IRT in both the 2048-bit and 4096-bit settings. When the noise standard

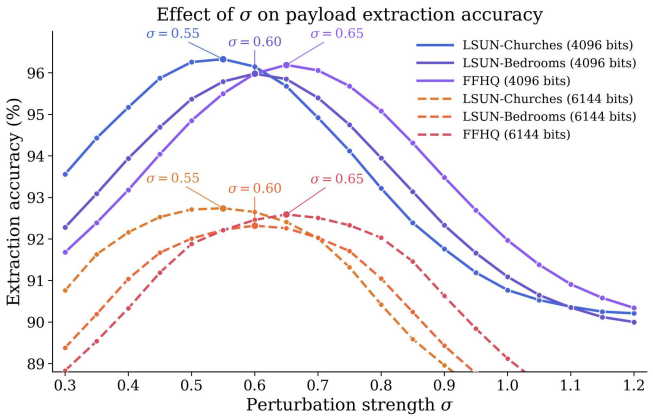


Fig. 8. Effect of σ on payload extraction accuracy under the 4096-bit and 6144-bit settings on LSUN-Churches, LSUN-Bedrooms, and FFHQ.

deviation is 0.40, the BER of the proposed method is 0.060 and 0.084, while the BER of S2IRT is 0.166 and 0.188, respectively. Under multiplicative noise with $\omega = 1.5$, the BER curves of the proposed method remain unchanged, whereas the BER of S2IRT increases by about 0.12 on average. When the noise standard deviation is 0.40, the BER of S2IRT further increases to 0.284 and 0.299.

These results show that the proposed mapping mechanism is immune to multiplicative noise and is also more robust to additive Gaussian noise.

F. Hyperparameter Analysis of σ

To study the effect of σ, we vary σ from 0.3 to 1.2 on LSUN-Churches, LSUN-Bedrooms, and FFHQ, and evaluate the payload extraction accuracy under the 4096-bit and 6144-bit settings.

As shown in Fig. 8, the best results under the 4096-bit setting are achieved at $\sigma = 0.55$, $\sigma = 0.60$, and $\sigma = 0.65$, with extraction accuracies of 96.33%, 95.98%, and 96.19% on LSUN-Churches, LSUN-Bedrooms, and FFHQ, respectively. Under the 6144-bit setting, the best results are also obtained at $\sigma = 0.55$, $\sigma = 0.60$, and $\sigma = 0.65$, with extraction accuracies of 92.74%, 92.32%, and 92.59%, respectively.

These results show that the optimal value of σ is consistently around 0.6. This indicates that such a setting can better balance the compactness and separability of the latent-space distribution, and thus leads to the highest extraction accuracy.

V. Conclusion

In this study, we propose TARSteg, a generative image steganography method based on latent TARFLOW. Our key idea is to shift the direct bijection between payload and images, originally realized by normalizing flows, into the latent space. Moreover, we train an auto-encoder tailored for steganography via an adversarial-sample strategy, establishing a bijective mechanism at the image–latent level to support robust payload extraction. We further introduce a key-driven Gaussian codebook index modulation scheme, constructing a robust bijection between payload and the base distribution of the latent normalizing flow. Experimental results demonstrate that our method achieves strong performance in terms of both capacity and security.

VI. Acknowledgment

This work was supported by the National Natural Science Foundation of China (Grant 62472440), and the Henan Provincial Distinguished Young Scholars Science Foundation (Grant 252300421064). The authors would like to acknowledge the support from Situation Awareness Laboratory, Information Engineering University for providing computing resources and scientific support.

References

[1] I. J. Kadhim, P. Premaratne, P. J. Vial, and B. Halloran, "Comprehensive survey of image steganography: Techniques, evaluations, and trends in future research," *Neurocomputing*, vol. 335, pp. 299–326, 2019.

[2] Y. Peng, Y. Wang, D. Hu, K. Chen, X. Rong, and W. Zhang, "Ldstega: Practical and robust generative image steganography based on latent diffusion models," in *Proceedings of the 32nd ACM International Conference on Multimedia*, pp. 3001–3009, 2024.

[3] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial networks," *Communications of the ACM*, vol. 63, no. 11, pp. 139–144, 2020.

[4] D. P. Kingma and P. Dhariwal, "Glow: Generative flow with invertible 1x1 convolutions," *Advances in neural information processing systems*, vol. 31, 2018.

[5] L. Dinh, J. Sohl-Dickstein, and S. Bengio, "Density estimation using real nvp," *arXiv preprint arXiv:1605.08803*, 2016.

[6] J. Ho, A. Jain, and P. Abbeel, "Denoising diffusion probabilistic models," *Advances in neural information processing systems*, vol. 33, pp. 6840–6851, 2020.

[7] X. Liu, Z. Ma, J. Ma, J. Zhang, G. Schaefer, and H. Fang, "Image disentanglement autoencoder for steganography without embedding," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 2303–2312, 2022.

[8] L. Metz, B. Poole, D. Pfau, and J. Sohl-Dickstein, "Unrolled generative adversarial networks," *arXiv preprint arXiv:1611.02163*, 2016.

[9] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013.

[10] Z. Zhou, Y. Su, J. Li, K. Yu, Q. J. Wu, Z. Fu, and Y. Shi, "Secret-to-image reversible transformation for generative steganography," *IEEE Transactions on Dependable and Secure Computing*, vol. 20, no. 5, pp. 4118–4134, 2022.

[11] S. Zhai, R. Zhang, P. Nakkiran, D. Berthelot, J. Gu, H. Zheng, T. Chen, M. A. Bautista, N. Jaitly, and J. Susskind, "Normalizing flows are capable generative models," *arXiv preprint arXiv:2412.06329*, 2024.

[12] A. Nguyen, J. Yosinski, and J. Clune, "Deep neural networks are easily fooled: High confidence predictions for unrecognizable images," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 427–436, 2015.

[13] F. Yu, A. Seff, Y. Zhang, S. Song, T. Funkhouser, and J. Xiao, "Lsun: Construction of a large-scale image dataset using deep learning with humans in the loop," *arXiv preprint arXiv:1506.03365*, 2015.

[14] T. Karras, S. Laine, and T. Aila, "A style-based generator architecture for generative adversarial networks," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 4401–4410, 2019.