STEGOCODER: LEARNING CHANNEL FOR STEGANOGRAPHIC CODING

Anonymous authors
Paper under double-blind review

ABSTRACT

Image steganography hides a message in a cover image with minimal perceptible distortion, producing a stego image from which an authorized receiver can recover the message. Robust steganography additionally requires recovery after the stego image is perturbed. Classical steganographic codes operate close to the rate-distortion bound, but on a fixed binary projection of the cover, typically its least-significant bit planes, in which the payload is destroyed by even mild perturbations. Neural steganography can be trained end-to-end to survive specific perturbations, but lacks the explicit rate–distortion control that coding theory provides. *StegoCoder* combines the two. A steganographic code first embeds the message into a fixed binary sequence and produces a binary stego sequence. A neural renderer then synthesizes the stego image from the cover image and the resulting binary stego. At the receiver, a learned bit predictor outputs soft estimates of the binary stego from the perturbed image, and the code decodes the message. The networks thus learn a reliable binary channel over which a classical steganographic code operates, splitting the problem into learning the channel and embedding the message efficiently over a binary cover. With a nested polar code, the same code acts as a lossy source code at the sender and as a channel code at the receiver, giving explicit control over the trade–off between embedding distortion and robustness. On CelebA-HQ, StegoCoder recovers most of the 500 test images without a single bit error under additive Gaussian noise of standard deviation 10^{-2} at 1 and 2 bits per pixel (bpp) and $5 \cdot 10^{-3}$ at 3 and 4 bpp, where least-significant-bit embedding, per-image optimization and end-to-end neural approaches recover none and quantization index modulation substantially fewer. A per-image refinement of the stego image further extends robustness across different payloads and co-training the scheme with the neural polar decoder decreases the distortion.

1 INTRODUCTION

Steganography is a branch of information hiding (Moulin & O'Sullivan, 2003) that studies how to embed a message (the payload) in a *cover* signal, such as an image or an audio, so that the resulting *stego* signal is perceptually indistinguishable from the original while an authorized receiver can recover the message (Fridrich, 2010). Closely related problems arise in watermarking (Cox et al., 2008), copyright protection (Lu, 2004), audio data hiding (Bilal et al., 2014) and privacy-preserving image sharing (Kishore et al., 2022).

Modifications to the cover are not equally perceivable at every position. For example, a change in a textured region of an image is harder to detect than one in a smooth region. *Adaptive steganography* exploits this by assigning a modification cost to every position and concentrating the embedding where the cost is low (Filler et al., 2011; Filler & Fridrich, 2010). Under the standard assumption that the embedding distortion is *additive*, i.e., the cost of modification does not depend on the other modifications, embedding becomes a quantization problem: among all signals that carry the intended message, the sender selects the one closest to the cover (Filler et al., 2011). The smallest achievable expected distortion for a given payload follows from a rate–distortion argument dual to lossy source coding (Barron et al., 2003). This is the basis of *steganographic coding* and, in particular, syndrome coding, where the message is the syndrome of the stego with respect to a linear code, and good codes approach the rate-distortion bound (Moulin & Koetter, 2005). One of the most widely used schemes asymptotically approaching the information-theoretic bound is based on syndrome-trellis

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

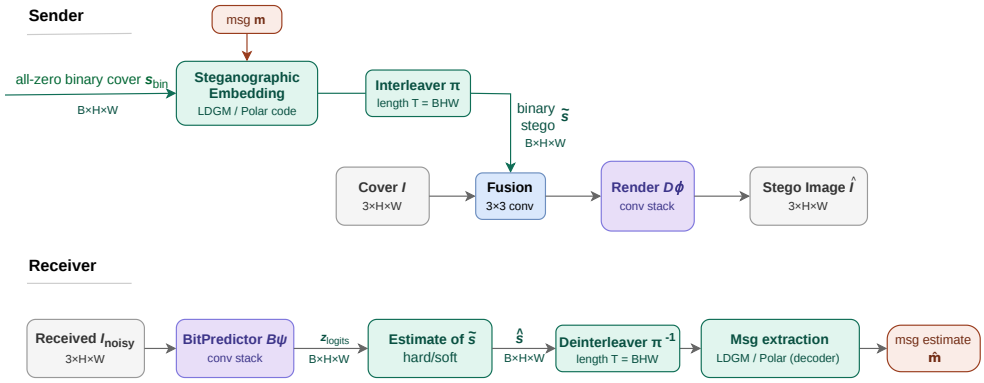

Figure 1: Overview of StegoCoder: the proposed steganographic scheme.

codes (STC) (Filler et al., 2011). Constructions based on low-density generator matrix (LDGM) codes (Yao et al., 2024) and polar codes (Ji et al., 2025) also approach the bound, and nested polar constructions add robustness to perturbations of the stego (Yao et al., 2025).

Steganographic codes operate on a fixed, hand-chosen discrete representation of the cover, such as its least-significant-bits (LSBs) or its quantized discrete cosine transform (DCT) coefficients, use hand-designed costs, and, with few exceptions (Yao et al., 2025; Ji et al., 2025; Kin-Cleaves & Ker, 2018; Feng et al., 2020), assume that the stego reaches the receiver unmodified, in contrast to watermarking codes such as quantization index modulation (QIM) (Chen & Wornell, 2001; Moulin & Koetter, 2005), which are robust but modify every position irrespective of its cost. Neural steganography removes these restrictions by learning the representation, embedding and the extraction from data. The first successful approaches used encoder–decoder architectures (Baluja, 2017; Zhu et al., 2018). Generative adversarial networks then improved both the embedding and the recovery (Zhang et al., 2019). More recent methods treat information hiding as a per-image optimization (Kishore et al., 2022): LISO learns an iterative optimizer and keeps the best iterate per image (Chen et al., 2023) with L-BFGS pixel optimization at the second stage, and neural cover selection searches the latent space of a diffusion model for a cover that suits the message (Chahine & Kim, 2024). Both sharply reduce the recovery error at high payloads. These methods reach low extraction error on an unperturbed stego signal and can be trained to keep it low under mild perturbations, but give up the rate-distortion control of steganographic coding. Moreover, per-image optimization is tied to the exact stego it produces: unless the perturbation that the image will undergo (*the transmission*) is placed inside the optimization loop, as done for JPEG in (Chen et al., 2023), nothing protects the message once the image is perturbed.

Steganographic codes and neural steganography thus make complementary trade-offs, while we combine them in this work. The proposed scheme, **StegoCoder**, consists of two neural modules built around a steganographic code (Fig. 1). First, the code embeds the message into a fixed binary cover s_{bin}, and outputs a binary stego $\tilde{s}$ that differs from s_{bin} in few positions. Then the binary stego sequence is embedded into the image by a pair of neural networks: the Render module synthesizes the stego image from the cover and the binary stego, while the BitPredictor estimates $\tilde{s}$ from the received image, from which the code recovers the message. Unlike LSB embedding, the steganographic embedding is performed over a fixed binary plane that does not depend on the cover. The Render and BitPredictor modules are trained jointly so that every change introduced into binary cover by the code remains recoverable from the stego image; in effect, they learn a reliable binary channel for the code. Moreover, each flip in the binary stego sequence contributes to the distortion of the rendered stego image largely independently of the others (App. B), so the additive Hamming distortion used by the codes is a reasonable proxy, and the standard steganographic codes can be applied without modification.

Our contributions can be summarized as follows:

A learned binary channel for steganographic codes. We introduce StegoCoder, a neural framework that renders a stego image carrying the binary stego sequence from the cover image and recovers the sequence from the received image after transmission perturbations, learning a reliable channel over which steganographic codes can operate, instead of over the pixel bit-planes.

Coding over the learned channel. We show that LDGM and nested polar steganographic codes can be utilized over the learned channel unmodified, thus preserving the coding-based control while benefiting from the reliability of the learned channel. The nested polar code's reserved set (Yao et al., 2025) turns it into an error-correcting code, providing additional robustness. A key-seeded interleaver makes the spatially correlated errors nearly independent within a block.

Co-training with a neural polar decoder. Since the induced binary channel has no explicit model, we co-train the networks with the neural polar decoder of Aharoni et al. (2024), thus the channel adapts to the decoding procedure, showing improvement in image quality across all tested payloads.

Pixel refinement. We introduce a post-hoc gradient-descent based (GD) refinement procedure that modifies the stego image before transmission to improve the recovery of the binary stego (Sec. 3.1).

Evaluation. We compare the proposed framework on CelebA-HQ to the main baselines (Sec. 5). While neural approaches recover none and QIM fewer, StegoCoder recovers most of the 500 test images without a single bit error under additive Gaussian noise of standard deviation 10^{-2} at 1 and 2 bpp and $5 \cdot 10^{-3}$ at 3 and 4 bpp. We evaluate JPEG compression (Sec. 5.2), 8-bit quantization (App. I) and a sender-side defence against steganalysis (App. L).

Throughout, we denote random variables by uppercase letters and their realizations by lowercase letters: X_i is the i-th element of the random vector $X_1^n = (X_1, \ldots, X_n)$, $X_1^i = (X_1, \ldots, X_i)$ denotes its first i elements, and $\boldsymbol{x} = x_1^n = (x_1, \ldots, x_n)$ is a realization of length n.

2 Preliminaries

This section introduces adaptive steganographic coding, and the LDGM- and polar-based coding schemes that we use.

2.1 Adaptive Steganographic Coding

Steganography hides a message into a cover signal, producing a stego signal that differs from the cover as little as possible. For binary embedding: the cover is a binary vector $\boldsymbol{x} \in \{0,1\}^n$, the sender embeds a message $\boldsymbol{m} \in \{0,1\}^q$, where $\alpha = q/n$ is the embedding rate, and produces the stego $\boldsymbol{y} \in \{0,1\}^n$. Assuming that the distortion has an additive form, the distortion between cover and stego is measured as:

$$D(\boldsymbol{x}, \boldsymbol{y}) = \sum_{i=1}^{n} \rho_i(x_i \oplus y_i), \tag{1}$$

where $\rho_i \geq 0$ is the cost of changing the i-th cover element x_i and $\oplus$ is addition modulo two. A payload-limited sender minimizes the expected distortion $\mathbb{E}[D]$ subject to a payload of q bits. For an additive distortion the optimal modifications are independent, $x_i \oplus y_i \sim \text{Bernoulli}(p_i)$ with the modification probability $p_i = P(y_i \neq x_i \mid x_i)$, and the entropy of the embedding is additive, so the sender minimizes $\sum_i \rho_i p_i$ subject to $\sum_i h_2(p_i) = q$, with h_2 denoting the binary entropy function. The solution follows the Gibbs distribution (Filler & Fridrich, 2010). Steganographic coding approaches this optimum with an efficient message extraction mechanism.

2.2 LDGM-based Steganographic Coding

LDGM is a linear code $\mathcal{C}(n,k)$ with a sparse generator matrix $G \in \{0,1\}^{k \times n}$ and parity-check matrix H satisfying $GH^T = \mathbf{0}$. To embed a message $\boldsymbol{m}$, the sender searches over all elements of the coset $c(\boldsymbol{m}) = \{\boldsymbol{u} \mid \boldsymbol{u}H^T = \boldsymbol{m}\}$ for the vector closest to the cover under Eq. 1. By construction $q \leq n - k$ and the search is performed approximately by belief propagation on the Tanner graph of the sparse generator matrix (Yao et al., 2024). The receiver extracts $\boldsymbol{m} = \boldsymbol{y}H^T$ (App. C).

The sparse generator matrix enables efficient approximate solutions to the constrained minimum-distortion problem, but the parity-check matrix is generally dense, so the extraction is sensitive to errors in the stego vector: a few flipped bits of $\boldsymbol{y}$ can result in substantial message-recovery errors.

2.3 Polar Codes for Steganographic Coding

Another steganographic coding scheme, near-optimal and additionally robust to channel noise, is based on polar codes (Arıkan, 2009; Yao et al., 2025). The polar transform splits a vector channel formed by n copies of a binary memoryless channel W into n synthetic subchannels whose qualities polarize as $n \to \infty$ (Arıkan, 2009), and the result extends to independent channels that are not identical (Chen et al., 2013; Ji et al., 2025). The polarization matrix G_n maps an input $\boldsymbol{u} \in \{0,1\}^n$ to the codeword $\boldsymbol{y} = \boldsymbol{u}G_n$, and the reliability of position i is measured by its Bhattacharyya parameter. A polar code is specified by $(n, k, \mathcal{F}, \boldsymbol{u}_{\mathcal{F}})$: the frozen set $\mathcal{F}$ holds the $n-k$ least reliable positions with values known to the receiver, and its complement carries the k information bits. The successive cancellation (SC) decoder estimates $\hat{\boldsymbol{u}}$ sequentially from the channel log-likelihood ratios (LLRs) and the known frozen values, and its list extension (SCL) with list size L improves accuracy at complexity $\mathcal{O}(L\, n \log n)$ (Tal & Vardy, 2015). Specifically, polar codes' optimality for lossy source coding (Korada & Urbanke, 2010) makes them a natural choice for steganography. Further construction and decoding details are provided in App. D.

Embedding. Polar steganographic embedding runs the SC decoder as a quantizer that maps the cover $\boldsymbol{x}$ to a nearby codeword. The modifications of the cover in Eq. 1 are modelled as n binary symmetric channels W_i with crossover probabilities p_i, $i = 1, \ldots, n$, given by the Gibbs distribution (Yao et al., 2025). The channels W_i are symmetric, since the costs in Eq. 1 do not depend on the direction of a change, and independent under the Gibbs distribution. The cover is the stego observed through these channels, and the decoder is initialized with the cover LLRs $l_i = (1-2x_i)\log\frac{1-p_i}{p_i}$ (Ji et al., 2025). In contrast to channel coding, the frozen positions carry the payload, $\boldsymbol{u}_{\mathcal{F}} = \boldsymbol{m}$ with $q = |\mathcal{F}|$, and the k information positions are quantized from the cover: for $i \in \mathcal{F}^c$ the decoder decides given the cover $\boldsymbol{x}$

$$\hat{u}_i = \arg\max_{u\in\{0,1\}} P_{U_i|U_1^{i-1},X_1^n}\big(u \mid \hat{u}_1^{i-1}, \boldsymbol{x}\big), \tag{2}$$

so that $\boldsymbol{y} = \hat{\boldsymbol{u}}G_n$ is close to $\boldsymbol{x}$ under Eq. 1 (Yao et al., 2025). The receiver applies the transform to the stego, $\boldsymbol{u} = \boldsymbol{y}G_n$ since $G_n^{-1} = G_n$ over $\mathbb{F}_2$, and gets $\boldsymbol{m} = \boldsymbol{u}_{\mathcal{F}}$.

Robustness through a reserved frozen budget. The extraction above requires the stego to be observed without errors. When it is perturbed, the receiver has to correct errors (Ji et al., 2025). Since the stego is already a polar codeword, a polar decoder can recover the message, but decoding requires positions whose values are known in advance. In the nested construction of Yao et al. (2025), a subset $\mathcal{F}' \subset \mathcal{F}$ of n_f positions is filled with values shared by the sender and the receiver, and the remaining $q = |\mathcal{F}| - n_f$ frozen positions carry the message. The construction then is based on two channels: the embedding channels $\{W_i\}$, which determine the cost of quantization, and the memoryless communication channel Q through which the stego is transmitted, which determines how reliably each position is recovered. The k positions most reliable for embedding are quantized from the cover, the n_f positions of $\mathcal{F}$ least reliable under Q form $\mathcal{F}'$, and the remaining q carry the message. The budget n_f controls the trade-off between distortion and robustness: a larger n_f for a fixed q increases the cover distortion but reduces the message error, and reliable extraction is asymptotically guaranteed for $n_f/n \geq h_2(\theta)$, where θ is the bit error probability for Q. This is shown for a binary symmetric channel with crossover probability θ (Yao et al., 2025), and by Fano's inequality it holds for any memoryless binary channel.

3 Proposed Method

The proposed scheme (Fig. 1) learns the channel for the binary stego sequence produced by the steganographic code. At the sender, the steganographic code embeds a message into a fixed binary sequence, producing a binary stego sequence, the Render generates the stego image from the cover image and the binary stego. At the receiver, the BitPredictor recovers the binary stego sequence from the received image, and the code extracts the message from the recovered sequence. The scheme tolerates transmission perturbations of the stego image that would destroy an embedding in the pixel

bit-planes, and it learns to recover the binary stego carrying the message while relying on its low distortion relative to the fixed binary sequence.

3.1 StegoCoder: Learned Binary Channel

The scheme consists of a steganographic code and a pair of jointly trained convolutional networks: the Render at the sender, and the BitPredictor at the receiver. Each is a four-layer stack of 3×3 convolutions with ReLU activations that keeps the size of the image at every layer, so that each position of the binary stego corresponds to a fixed image region (App. F).

Embedding and its simulation in training. The steganographic code $\mathcal{C}$ takes as a cover a binary array $\boldsymbol{s}_{\text{bin}} \in \{0,1\}^{B\times H\times W}$, with all zeros, which has B binary planes of the same size $H \times W$ as the cover image, and produces the binary stego sequence $\tilde{\boldsymbol{s}} = \mathcal{C}(\boldsymbol{m}, \boldsymbol{s}_{\text{bin}}) \in \{0,1\}^{B\times H\times W}$. The coding is a discrete operation that is not differentiable, so in training we model it by flipping every position of $\boldsymbol{s}_{\text{bin}}$ independently,

$$\Pr[\tilde{\mathcal{C}}(s_i) = s_i] = 1 - p_{\text{flip}}, \qquad \Pr[\tilde{\mathcal{C}}(s_i) = 1 - s_i] = p_{\text{flip}}. \tag{3}$$

Since the flips are random and independent of the image, the Render learns to embed the stego into the stego image, as otherwise the BitPredictor could not recover it. The range of p_{flip} sets how many changes of the given binary sequence the scheme is trained to embed and recover (App. A).

Render. The Render network $\mathcal{D}_\phi$ produces the stego image $\hat{\boldsymbol{I}} = \mathcal{D}_\phi\big(\text{Fusion}([\tilde{\boldsymbol{s}} \,\|\, \boldsymbol{I}])\big) \in [0,1]^{3\times H\times W}$ from the binary stego sequence $\tilde{\boldsymbol{s}}$ and the cover image $\boldsymbol{I} \in [0,1]^{3\times H\times W}$, where a learned fusion convolution combines the two groups before the rendering block, so that each bit modification produced by embedding is spread over a local image region rather than one pixel.

BitPredictor. The BitPredictor $\mathcal{B}_\psi$ estimates the binary stego positions $\hat{\boldsymbol{z}}_{\text{logits}} = \mathcal{B}_\psi(\hat{\boldsymbol{I}}_{\text{noisy}}) \in \mathbb{R}^{B\times H\times W}$ from the received image $\hat{\boldsymbol{I}}_{\text{noisy}}$, i.e., the stego image after transmission perturbations, producing soft estimates $\hat{\boldsymbol{s}} = S(\hat{\boldsymbol{z}}_{\text{logits}})$ through the sigmoid function $S(\cdot)$, which are thresholded at $1/2$ into hard decisions $\hat{\boldsymbol{s}}_{\text{hard}}$.

Training. During training, additive Gaussian noise of standard deviation σ_{train} is applied to the rendered image, producing $\hat{\boldsymbol{I}}_{\text{noisy}}$, and the binary stego recovery is performed on the perturbed stego image. The two neural modules are trained jointly with

$$\mathcal{L} = \lambda_{\text{rec}}\mathcal{L}_{\text{rec}}(\boldsymbol{I}, \hat{\boldsymbol{I}}) + \lambda_{\text{bit}}\mathcal{L}_{\text{bit}}(\hat{\boldsymbol{s}}, \tilde{\boldsymbol{s}}), \tag{4}$$

where $\mathcal{L}_{\text{rec}}$ is a reconstruction loss between the cover and the rendered stego image, and $\mathcal{L}_{\text{bit}}$ is the binary cross-entropy between the recovered binary sequence $\hat{\boldsymbol{s}}$ and the binary stego $\tilde{\boldsymbol{s}}$. The weights, the range of p_{flip} and the noise level are given in Sec. 4.

Pixel-Space Refinement. The stego image $\hat{\boldsymbol{I}}$ produced by the Render can be refined further before transmission: given the rendered image as an initial value $\hat{\boldsymbol{I}}_0 = \hat{\boldsymbol{I}}$, the cover $\boldsymbol{I}$ and the binary stego $\tilde{\boldsymbol{s}}$, we apply J gradient steps

$$\begin{aligned} \hat{\boldsymbol{I}}_{j+1} &= \text{clip}\Big(\hat{\boldsymbol{I}}_j - \eta\nabla_{\hat{\boldsymbol{I}}_j}\mathcal{L}_{\text{GD}},\ 0,\ 1\Big), \\ \mathcal{L}_{\text{GD}} &= \text{BCE}\Big(\mathcal{B}_\psi(\hat{\boldsymbol{I}}_j), \tilde{\boldsymbol{s}}\Big) + \lambda_{\text{LPIPS}}\,\text{LPIPS}(\hat{\boldsymbol{I}}_j, \boldsymbol{I}) + \lambda_{\text{MSE}}\,\text{MSE}(\hat{\boldsymbol{I}}_j, \boldsymbol{I}), \end{aligned} \tag{5}$$

which move the pixels toward values from which the BitPredictor recovers the binary stego more decisively, as in fixed-network steganography (Kishore et al., 2022). With this objective the refinement targets clean channel (App. H), while evaluating the BCE on the perturbed image, $\mathcal{B}_\psi(\hat{\boldsymbol{I}}_{\text{noisy},j})$ with noise of standard deviation σ_{GD}, makes it noise-aware, which is the refinement reported in Sec. 5.

Induced channel for binary stego. The image generation by the Render, the perturbation in transmission and the estimation by the BitPredictor together form a binary channel with the binary stego $\tilde{\boldsymbol{s}}$ as input and its estimate $\hat{\boldsymbol{s}}$ as output, with error probabilities

$$\theta_i = P(\hat{s}_{\text{hard},i} \neq \tilde{s}_i), \qquad i = 1, \ldots, B \cdot H \cdot W. \tag{6}$$

The estimate at each position depends on the whole $\tilde{\boldsymbol{s}}$, the cover $\boldsymbol{I}$ and the perturbation. The cover thus acts as a channel state known to the sender but not to the receiver, as in coding with side

information at the encoder (Gelfand & Pinsker, 1980; Moulin & O'Sullivan, 2003). The Render uses this state, and the GD refinement (Eq. 5) exploits it further, similar to dirty-paper precoding (Costa, 1983). The LDGM extraction has no error correction and needs $\theta_i = 0$ for exact recovery, while the polar scheme corrects errors with its reserved budget and is reliable once $n_f/n \geq h_2(\theta)$, with θ the average of θ_i (Sec. 2.3). This assumes a memoryless channel, which the induced channel only approximates: neighbouring positions share the receptive fields, so their errors are correlated. We address this by interleaving (Sec. 3.2) and by co-training with the neural decoder (Sec. 3.3).

3.2 Steganographic Coding on the Binary Planes

The fixed binary cover $\boldsymbol{s}_{\text{bin}}$ has $T = B \cdot H \cdot W$ bits. A code of block length n splits it into $\lfloor T/n \rfloor$ blocks and embeds q message bits into each, where $q = n-k$ for the LDGM code and $q = n-k-n_f$ for the polar code with n_f reserved positions.

Since the scheme is trained with a flip probability that is drawn per image but is the same at every position, there is no basis for position-dependent costs, and both codes are run with uniform costs $\rho_i = 1$ in Eq. 1.

Interleaving. In their natural order, consecutive binary stego positions are neighbouring pixels of one binary plane, so a block of n positions covers n/W neighbouring rows, and the errors, which are correlated between neighbouring positions, concentrate in a few blocks. We interleave all T positions of the flattened binary stego $\tilde{\boldsymbol{s}}$ with a key-seeded permutation π before reshaping into the $B \times H \times W$ planes and passing it to the Render. Interleaving spreads each code block over the whole image and makes the errors within a block close to the independent errors the code assumes. At the receiver, the estimates $\hat{\boldsymbol{s}}$ are de-interleaved with π^{-1} before being split into code blocks.

LDGM Code. The receiver extracts $\hat{\boldsymbol{m}} = \hat{\boldsymbol{s}}_{\text{hard}} H^T$ block by block; there is no error correction, so every error in a block propagates into the decoded message of that block.

Polar Code. The induced channel has no analytical model, and since the scheme is trained with the same flip probability at every position, the channel is approximately the same at every position, and we use one fixed reliability sequence from the 5G NR standard (Bae et al., 2019) for both orderings of Sec. 2.3. The n_f least reliable positions are the reserved set $\mathcal{F}'$ and hold a random key shared by the sender and the receiver, the next q least reliable positions carry the message, and the $k = n - q - n_f$ most reliable positions are quantized from the cover. Co-training (Sec. 3.3) further adapts the scheme to the code.

3.3 Adapting the Induced Channel to Successive-Cancellation Decoding

The polar construction treats the channel of Eq. 6 as a binary memoryless channel: the standard decoder at the receiver takes the soft estimates of the BitPredictor as LLRs, $l_i = \log \frac{1-\hat{s}_i}{\hat{s}_i}$ for the standard check-node and bit-node operations. The induced channel is not memoryless, and the reliability of the soft estimates depends on the image content. Rather than model this channel, we co-train the scheme with the data-driven neural polar decoder of Aharoni et al. (2024), which keeps the computational graph of the SC decoder with an embedding vector instead of a scalar LLR (App. D).

Training inside the scheme. The Render and the BitPredictor are pre-trained with independent flips at the same rate at every position (Sec. 3.1), so the induced channel is approximately the same at every position, and the polar code can use the ordering of Sec. 3.2 for the reserved and the message positions. The neural polar decoder is placed after BitPredictor, takes its soft estimates as input, and is trained with the binary cross-entropy between the sender's $\hat{u}_i$ of Eq. 2 and its predictions, summed over the stages of the polar factor graph with the message positions weighted by w_{msg}, while the Render, the Fusion layer and the BitPredictor are fine-tuned with the loss of Eq. 4. The flips in the binary stego are produced by the polar embedding, and the stego images are perturbed with Gaussian noise. The code at the sender is unchanged, and the co-trained networks are decoded with the standard SCL in all experiments.

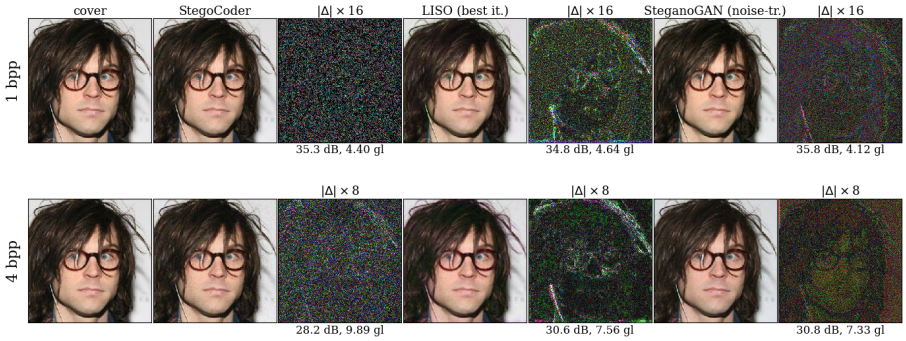

Figure 2: Cover, stego image and residual, comparison at 1 and 4 bpp.

4 Experimental Setup

Data. We train and test on CelebA-HQ (Karras et al., 2017) at 256×256 pixels, with $29{,}000$ training and 500 test images; an additional evaluation on AFHQ-Dog (Choi et al., 2020) is given in App. J.

StegoCoder. The separate model is trained for each number of steganographic planes $B = 2, 4, 6, 8, 9, 10$, and the configuration of each code and payload is given in Table 3 (App. A). At every training step the flip probability p_{flip} is drawn uniformly from the range provided there. Each model is trained with Gaussian noise of uniformly sampled standard deviation $\sigma \in [0, 0.13]$ per image injected in the stego, for $1{,}000$ epochs with AdamW at learning rate 10^{-3}, cosine annealing and gradient clipping at norm 0.5, with $\mathcal{L}_{\text{rec}}$ the LPIPS distance (Zhang et al., 2018) (AlexNet backbone) plus 0.1 times the mean squared error and $(\lambda_{\text{rec}}, \lambda_{\text{bit}}) = (0.008, 0.9)$.

Codes. As LDGM code, we use an $(n, k) = (384, 192)$ code from (Helmling et al., 2019) with $q = 192$ and 200 BP iterations with an annealing schedule (App. C). The polar code has length $n = 1024$ and reserved budget $n_f = 128$, a list size $L = 8$ for the SCL at both ends, and q chosen for the target rate (Table 3). The learned decoder is trained with $\sigma_{\max} = 0.013$, $w_{\text{msg}} = 5$.

Channel. During evaluation, the stego image is perturbed by additive Gaussian noise with standard deviation $\sigma \in \{0,\ 5 \cdot 10^{-3},\ 10^{-2}\}$ on the $[0, 1]$ scale, clipped to $[0, 1]$.

Baselines. LSB writes the message into the LSB plane (one plane at 1–3 bpp, two planes at 4 bpp). LISO (Chen et al., 2023) is run with the public checkpoints and 150 iterations of its learned optimizer. SteganoGAN (Zhang et al., 2019) is the dense encoder–decoder–critic, retrained on our training split and in matching our training noisy conditions with the published loss and schedule. We also report dither modulation (QIM) with step $\Delta = 24$ (Chen & Wornell, 2001).

Metrics. Image quality is measured between the cover $\boldsymbol{I}$ and the stego image $\hat{\boldsymbol{I}}$, both in $[0, 1]$: the peak signal-to-noise ratio $\text{PSNR} = 10 \log_{10} \left(1/\text{MSE}(\boldsymbol{I}, \hat{\boldsymbol{I}})\right)$ in dB, and the learned perceptual image patch similarity LPIPS (Zhang et al., 2018), the distance between deep features of the two images, computed with the VGG backbone (training uses AlexNet), lower for images that look more alike. Message recovery is measured by the bit-error rate $\text{BER} = \frac{1}{M} \sum_{i=1}^{M} \mathbb{1}[\hat{m}_i \neq m_i]$, for message length $M = \text{bpp} \times W \times H$, computed per image and averaged over the test set, and by the number of test images whose message is recovered without a single bit error, which is the criterion under which per-image optimization methods are usually judged (Chen et al., 2023).

5 Results

5.1 Comparison on CelebA-HQ

Table 1 compares StegoCoder with the baselines at matched payloads. We report StegoCoder (shaded rows) with the LDGM, polar, and the polar code co-trained with the neural polar decoder and decoded with the standard SCL, $L = 8$. Moreover, for each setting we report a per-image noise-aware GD refinement ($J = 200$ steps, $\sigma_{\text{GD}} = 10^{-2}$), as an analogue to the LISO (best iterate)

Table 1: The results on CelebA-HQ, 256×256, 500 test images. Bold marks the best value in each column within a payload block; shaded rows are StegoCoder.

method	PSNR ↑	$\text{LPIPS}_{\text{vgg}}$ ↓	$\sigma = 0$		$\sigma = 5{\cdot}10^{-3}$		$\sigma = 10^{-2}$	
			BER ↓	0-err ↑	BER ↓	0-err ↑	BER ↓	0-err ↑
1 bit per pixel								
LISO	32.01	0.176	$5.9{\cdot}10^{-4}$	0	0.068	0	0.205	0
LISO (best iterate)	36.36	0.111	$3.9{\cdot}10^{-6}$	425	0.063	0	0.195	0
SteganoGAN, noisy	35.99	0.055	$2.9{\cdot}10^{-3}$	20	$6.7{\cdot}10^{-3}$	0	0.034	0
QIM, $\Delta = 24$	36.00	0.159	**0**	**500**	$2.2{\cdot}10^{-6}$	435	0.018	0
LDGM	36.05	**0.053**	$2.0{\cdot}10^{-5}$	477	$5.3{\cdot}10^{-5}$	451	0.099	0
+ GD (noise-aware)	35.74	0.056	**0**	**500**	$3.1{\cdot}10^{-8}$	499	0.074	0
Polar, std. SCL	35.33	0.076	**0**	**500**	**0**	**500**	**0**	**500**
+ GD (noise-aware)	36.52	0.063	**0**	**500**	**0**	**500**	$2.4{\cdot}10^{-4}$	321
Polar, co-trained, std. SCL	36.84	0.059	**0**	**500**	**0**	**500**	$6.6{\cdot}10^{-4}$	181
+ GD (noise-aware)	**37.33**	0.054	**0**	**500**	**0**	**500**	$2.2{\cdot}10^{-3}$	15
2 bits per pixel								
LISO	26.74	0.318	$1.4{\cdot}10^{-3}$	0	0.040	0	0.163	0
LISO (best iterate)	31.77	0.201	$2.6{\cdot}10^{-5}$	142	0.040	0	0.165	0
SteganoGAN, noisy	33.94	0.140	$7.3{\cdot}10^{-3}$	0	0.015	0	0.073	0
QIM, $\Delta = 24$	32.97	0.196	**0**	**500**	$2.3{\cdot}10^{-6}$	374	0.018	0
LDGM	33.20	**0.111**	$3.5{\cdot}10^{-3}$	2	$9.6{\cdot}10^{-3}$	0	0.251	0
+ GD (noise-aware)	32.66	0.119	$1.8{\cdot}10^{-5}$	452	$2.0{\cdot}10^{-4}$	240	0.154	0
Polar, std. SCL	32.10	0.176	**0**	**500**	**0**	**500**	$4.8{\cdot}10^{-4}$	287
+ GD (noise-aware)	31.84	0.184	**0**	**500**	**0**	**500**	$\mathbf{2.8{\cdot}10^{-5}}$	**456**
Polar, co-trained, std. SCL	**34.41**	0.130	**0**	**500**	$2.3{\cdot}10^{-6}$	499	0.222	0
+ GD (noise-aware)	33.19	0.160	**0**	**500**	**0**	**500**	0.021	0
3 bits per pixel								
LISO	27.04	0.272	$3.4{\cdot}10^{-3}$	0	0.051	0	0.165	0
LISO (best iterate)	**32.33**	0.215	$7.9{\cdot}10^{-5}$	56	0.039	0	0.155	0
SteganoGAN, noisy	30.14	**0.164**	0.053	0	0.072	0	0.124	0
QIM, $\Delta = 24$	31.22	0.209	**0**	**500**	$2.3{\cdot}10^{-6}$	328	**0.018**	0
LDGM	30.97	0.205	0.024	0	0.058	0	0.365	0
+ GD (noise-aware)	30.69	0.214	$1.5{\cdot}10^{-3}$	3	$7.8{\cdot}10^{-3}$	0	0.302	0
Polar, std. SCL	30.21	0.242	$8.6{\cdot}10^{-6}$	493	$1.2{\cdot}10^{-4}$	447	0.145	0
+ GD (noise-aware)	30.04	0.246	$4.8{\cdot}10^{-7}$	499	$\mathbf{1.3{\cdot}10^{-6}}$	**497**	0.061	0
Polar, co-trained, std. SCL	31.50	0.209	$7.7{\cdot}10^{-5}$	483	$1.4{\cdot}10^{-3}$	299	0.432	0
+ GD (noise-aware)	31.07	0.220	$4.8{\cdot}10^{-7}$	499	$1.5{\cdot}10^{-5}$	480	0.316	0
4 bits per pixel								
LISO	25.34	0.331	$4.7{\cdot}10^{-3}$	0	0.088	0	0.194	0
LISO (best iterate)	31.40	0.240	$3.8{\cdot}10^{-4}$	0	0.072	0	0.185	0
SteganoGAN, noisy	30.44	**0.160**	0.091	0	0.118	0	0.180	0
QIM, $\Delta = 24$	**32.73**	0.198	**0**	**500**	0.014	0	**0.177**	0
LDGM	28.75	0.281	0.055	0	0.147	0	0.423	0
+ GD (noise-aware)	28.82	0.280	0.016	0	0.083	0	0.406	0
Polar, std. SCL	28.36	0.306	$9.5{\cdot}10^{-5}$	461	$1.6{\cdot}10^{-3}$	202	0.377	0
+ GD (noise-aware)	28.47	0.302	$5.7{\cdot}10^{-6}$	491	$\mathbf{2.0{\cdot}10^{-4}}$	**380**	0.314	0
Polar, co-trained, std. SCL	28.87	0.295	$6.2{\cdot}10^{-5}$	476	$1.9{\cdot}10^{-3}$	183	0.465	0
+ GD (noise-aware)	29.30	0.281	$2.1{\cdot}10^{-6}$	496	$2.1{\cdot}10^{-4}$	373	0.469	0

row, where the best iterate is chosen for a specific cover and message. Here we focus on recovery under perturbations; LSB and the methods optimized for the unperturbed stego image are compared in App. H.

Under noise, only StegoCoder keeps the message. With noise added, LISO and SteganoGAN recover no image without error at any payload, dither modulation (QIM) still recovers some, while StegoCoder with the polar code recovers all or almost all images at this noise level up to 3 bpp. At $\sigma = 10^{-2}$ every baseline loses all images, while StegoCoder with the polar code still recovers every image at 1 bpp, and at 2 bpp more than half with standard polar and most with GD refinement.

The reserved budget makes the polar code robust. The LDGM and polar codes introduce distortion of the same order, but the LDGM construction does not provide error correction, so it loses the message as the noise grows and fails even without noise at 3 and 4 bpp.

Table 2: JPEG channel, CelebA-HQ, 1 bpp, 500 test images; every model is trained and evaluated at one quality. *With Gaussian noise of $\sigma = 5{\cdot}10^{-3}$ after compression. Shaded rows are StegoCoder.

	quality 80					quality 90				
method	PSNR	BER	0-err	BER*	0-err*	PSNR	BER	0-err	BER*	0-err*
LISO-JPEG (Chen et al., 2023)	19.72	$6.0{\cdot}10^{-4}$	–	–	–	–	–	–	–	–
LISO + NCS (Chahine & Kim, 2024)	21.53	$6.0{\cdot}10^{-4}$	–	–	–	–	–	–	–	–
Polar, std. SCL	18.73	$4.8{\cdot}10^{-4}$	278	$6.7{\cdot}10^{-4}$	235	21.88	$2.7{\cdot}10^{-6}$	496	$7.5{\cdot}10^{-6}$	494
+ GD refinement	19.16	$3.9{\cdot}10^{-4}$	329	$6.1{\cdot}10^{-4}$	271	22.44	$3.9{\cdot}10^{-6}$	495	$1.6{\cdot}10^{-5}$	482
Polar, co-trained, std. SCL	19.89	$1.2{\cdot}10^{-2}$	1	$1.5{\cdot}10^{-2}$	0	23.64	$5.9{\cdot}10^{-4}$	244	$1.4{\cdot}10^{-3}$	155
+ GD refinement	20.36	$8.6{\cdot}10^{-3}$	9	$1.2{\cdot}10^{-2}$	1	23.99	$9.0{\cdot}10^{-4}$	223	$2.2{\cdot}10^{-3}$	119

The noise-aware refinement trades little distortion for robustness. Except for the polar codes at 1 bpp, GD refinement lowers the BER under noise at a cost of at most about a decibel of PSNR. At 1 bpp, it instead raises PSNR and gives up part of the recovery at $\sigma = 10^{-2}$.

Co-training reduces the image distortion. At every payload co-training raises PSNR and lowers LPIPS, at the price of part of the noise margin. At 1 and 2 bpp the co-trained scheme exceeds LISO (best iterate) in both PSNR and LPIPS; at 3 and 4 bpp LISO has a higher PSNR but recovers few or no images exactly even without noise.

Figure 2 shows the stego images and the amplified residuals at 1 and 4 bpp. Since the stego image is rendered, our modification is spread uniformly over the frame, while LISO concentrates its change on the edges and SteganoGAN on the face.

5.2 Robustness to JPEG Compression

JPEG compression is a harder channel than additive noise for a per-pixel embedding, since the quantization removes the details that the rendering of the binary stego relies on. Table 2 reports StegoCoder trained with differentiable JPEG at quality 80 and 90 and evaluated with the actual encoder. The published results report only the BER, and the PSNR of neural cover selection (Chahine & Kim, 2024) is measured against the selected cover rather than the original image.

Under JPEG, StegoCoder still recovers the message exactly. At quality 80, StegoCoder with the standard polar construction recovers 278 of the 500 images without a bit error, with a BER below the published results, though at a lower PSNR. At quality 90, it recovers perfectly almost all images even under additional Gaussian noise.

Co-training and the refinement reduce the image distortion. Co-training raises PSNR by more than 1 dB at both qualities, at the price of exact recovery. The per-image refinement adds about half a decibel in every setting, and it raises the number of images recovered without error at quality 80 and lowers it at quality 90.

6 Conclusion

StegoCoder combines classical steganographic coding with neural steganography: standard LDGM and nested polar codes produce the binary stego, while the scheme learns a binary channel to communicate it to the receiver through the stego image. This keeps the rate–distortion control of coding for binary embedding, and the nested structure of the polar code provides error correction through the reserved budget. As a result, StegoCoder keeps the message decoded without a single bit error for most images under moderate Gaussian noise, where the neural and per-image optimization baselines recover none, and under JPEG compression. Co-training with the neural polar decoder reduces the distortion, and the per-image refinement extends robustness.

Limitations. The induced channel has no analytical model, so the polar code uses a fixed reliability ordering that assumes independent errors. A model is trained per payload and perturbation. Since the Render spreads its modification over the whole image, at 3 and 4 bpp StegoCoder has a lower PSNR than the best neural baseline.

USE OF GENERATIVE AI

In this work, we used generative AI tools for for running the experiment workers and collecting their results, and for writing the plotting scripts of Figures 2 and 3. We have not used generative AI tools for generating research ideas or for implementing the method: the StegoCoder model, the LDGM and polar coding and Figure 1 were implemented by the authors, and data collection and annotation are not applicable to this work. Additionally, we used generative AI tools for polishing and rephrasing the text and for searching the literature for related and conflicting works. We have reviewed all AI-assisted work: every reported number was checked by the authors against the raw evaluation logs, and every reference found with AI assistance was read and verified by the authors. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

Steganography is dual-use: it supports private communication, watermarking and copyright protection, but can also hide communication from legitimate monitoring, and robustness to perturbations strengthens both uses. Pretrained steganalysers flag almost every undefended stego image; the sender-side defence of App. L is evaluated only against these frozen detectors. We use public datasets (CelebA-HQ, AFHQ-Dog) under their research licenses, embed random bit strings and involve no human subjects.

REPRODUCIBILITY STATEMENT

The networks, their training, the baselines and the code configurations are given in Sec. 3.1, Sec. 4 and Apps. A and F; the LDGM and polar constructions and the neural polar decoder in Sec. 2.3, Sec. 3.2 and Apps. C–D; the refinement settings in Sec. 5.1. All results use the test images with fixed seeds for the messages, the noise and the interleaver key. An anonymized implementation with the evaluation scripts is provided in the supplementary material.

REFERENCES

Ziv Aharoni, Bashar Huleihel, Henry D. Pfister, and Haim H. Permuter. Data-driven neural polar decoders for unknown channels with and without memory. *IEEE Transactions on Information Theory*, 70(12):8495–8510, 2024. doi: 10.1109/TIT.2024.3476681.

Erdal Arıkan. Channel polarization: A method for constructing capacity-achieving codes for symmetric binary-input memoryless channels. *IEEE Transactions on Information Theory*, 55(7): 3051–3073, 2009.

Jung Hyun Bae, Ahmed Abotabl, Hsien-Ping Lin, Kee-Bong Song, and Jungwon Lee. An overview of channel coding for 5G NR cellular communications. *APSIPA Transactions on Signal and Information Processing*, 8(1), 2019. ISSN 2048-7703. doi: 10.1017/ATSIP.2019.10. URL `http://dx.doi.org/10.1017/ATSIP.2019.10`.

Shumeet Baluja. Hiding images in plain sight: deep steganography. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, pp. 2066–2076, 2017. ISBN 9781510860964.

R.J. Barron, Brian Chen, and G.W. Wornell. The duality between information embedding and source coding with side information and some applications. *IEEE Transactions on Information Theory*, 49(5):1159–1180, 2003. doi: 10.1109/TIT.2003.810639.

Ifra Bilal, Rajiv Kumar, Mahendra Singh Roj, and P K Mishra. Recent advancement in audio steganography. In *2014 International Conference on Parallel, Distributed and Grid Computing*, pp. 402–405, 2014. doi: 10.1109/PDGC.2014.7030779.

Karl Chahine and Hyeji Kim. Neural cover selection for image steganography. In *Proceedings of the 38th International Conference on Neural Information Processing Systems*, NIPS '24, 2024. ISBN 9798331314385.

Brian Chen and Gregory W. Wornell. Quantization index modulation: A class of provably good methods for digital watermarking and information embedding. *IEEE Transactions on Information Theory*, 47(4):1423–1443, 2001. doi: 10.1109/18.923725.

Kai Chen, Kai Niu, and Jia-Ru Lin. Practical polar code construction over parallel channels. *IET Communications*, 7:620–627, 2013. doi: 10.1049/iet-com.2012.0428.

Xiangyu Chen, Varsha Kishore, and Kilian Q. Weinberger. Learning iterative neural optimizers for image steganography. In *International Conference on Learning Representations (ICLR)*, 2023.

Yunjey Choi, Youngjung Uh, Jaejun Yoo, and Jung-Woo Ha. StarGAN v2: Diverse image synthesis for multiple domains. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 8185–8194, 2020. doi: 10.1109/CVPR42600.2020.00821.

Max H. M. Costa. Writing on dirty paper. *IEEE Transactions on Information Theory*, 29(3):439–441, 1983.

Ingemar J. Cox, Matthew L. Miller, Jeffrey A. Bloom, Jessica Fridrich, and Ton Kalker. *Digital Watermarking and Steganography (Second Edition)*. The Morgan Kaufmann Series in Multimedia Information and Systems. Morgan Kaufmann, Burlington, second edition edition, 2008. ISBN 978-0-12-372585-1. doi: https://doi.org/10.1016/B978-012372585-1.50007-3. URL https://www.sciencedirect.com/science/article/pii/B9780123725851500073.

Bingwen Feng, Zhiquan Liu, Xiaotian Wu, and Yuchun Lin. Robust syndrome-trellis codes for fault-tolerant steganography. In Lakhmi C. Jain, Sheng-Lung Peng, and Shiuh-Jeng Wang (eds.), *Security with Intelligent Computing and Big-Data Services 2019*, pp. 115–127, Cham, 2020. Springer International Publishing. ISBN 978-3-030-46828-6.

Tomáš Filler and Jessica Fridrich. Gibbs construction in steganography. *IEEE Transactions on Information Forensics and Security*, 5(4):705–720, 2010. doi: 10.1109/TIFS.2010.2077629.

Tomáš Filler, Jan Judas, and Jessica Fridrich. Minimizing additive distortion in steganography using syndrome-trellis codes. *IEEE Transactions on Information Forensics and Security*, 6(3):920–935, 2011. doi: 10.1109/TIFS.2011.2134094.

Jessica Fridrich. Steganography in digital media: Principles, algorithms, and applications. *Steganography in Digital Media*, 01 2010. doi: 10.1017/CBO9781139192903.

S. I. Gelfand and M. S. Pinsker. Coding for channel with random parameters. *Problems of Control and Information Theory*, 9(1):19–31, 1980.

M. Helmling, S. Scholl, F. Gensheimer, T. Dietz, D. Kraft, S. Ruzika, and N. Wehn. Database of channel codes and ML simulation results. https://rptu.de/en/channel-codes, 2019.

Yujun Ji, Jinsheng Li, Ling Liu, Qi Cao, and Tao Dai. Pixel-sensitive and robust steganography based on polar codes, 06 2025.

Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of GANs for improved quality, stability, and variation. *ICLR 2018*, 2017. URL https://api.semanticscholar.org/CorpusID:3568073.

Christy Kin-Cleaves and Andrew D. Ker. Adaptive steganography in the noisy channel with dual-syndrome trellis codes. In *2018 IEEE International Workshop on Information Forensics and Security (WIFS)*, pp. 1–7, 2018. doi: 10.1109/WIFS.2018.8630779.

Varsha Kishore, Xiangyu Chen, Yan Wang, Boyi Li, and Kilian Q Weinberger. Fixed neural network steganography: Train the images, not the network. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=hcMvApxGSzZ.

Satish Babu Korada and Rüdiger L Urbanke. Polar codes are optimal for lossy source coding. *IEEE Transactions on Information Theory*, 56(4):1751–1768, 2010.

Chun-Shien Lu. *Multimedia Security: Steganography and Digital Watermarking Techniques for Protection of Intellectual Property*. IGI Global, USA, 2004. ISBN 1591401925.

P. Moulin and R. Koetter. Data-hiding codes. *Proceedings of the IEEE*, 93(12):2083–2126, 2005. doi: 10.1109/JPROC.2005.859599.

P. Moulin and J.A. O'Sullivan. Information-theoretic analysis of information hiding. *IEEE Transactions on Information Theory*, 49(3):563–593, 2003. doi: 10.1109/TIT.2002.808134.

Ido Tal and Alexander Vardy. How to construct polar codes. *IEEE Transactions on Information Theory*, 59(10):6562–6582, 2013. doi: 10.1109/TIT.2013.2272694.

Ido Tal and Alexander Vardy. List decoding of polar codes. *IEEE Transactions on Information Theory*, 61(5):2213–2226, 2015. doi: 10.1109/TIT.2015.2410251.

Guanshuo Xu, Han-Zhou Wu, and Yun-Qing Shi. Structural design of convolutional neural networks for steganalysis. *IEEE Signal Processing Letters*, 23(5):708–712, 2016. doi: 10.1109/LSP.2016.2548421.

Qiyi Yao, Weiming Zhang, Kejiang Chen, and Nenghai Yu. LDGM codes-based near-optimal coding for adaptive steganography. *IEEE Transactions on Communications*, 72(4):2138–2151, 2024. doi: 10.1109/TCOMM.2023.3342247.

Qiyi Yao, Kai Zeng, Weiming Zhang, and Kejiang Chen. Reliable robust adaptive steganographic coding based on nested polar codes. *IEEE Transactions on Signal Processing*, 73:12–25, 2025. doi: 10.1109/TSP.2024.3510755.

Weike You, Hong Zhang, and Xianfeng Zhao. A siamese CNN for image steganalysis. *IEEE Transactions on Information Forensics and Security*, 16:291–306, 07 2020. doi: 10.1109/TIFS.2020.3013204.

Kevin Alex Zhang, Alfredo Cuesta-Infante, Lei Xu, and Kalyan Veeramachaneni. SteganoGAN: High capacity image steganography with GANs. *ArXiv*, abs/1901.03892, 2019. URL `https://api.semanticscholar.org/CorpusID:59413797`.

Richard Zhang, Phillip Isola, Alexei A. Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *Proc. IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

Jiren Zhu, Russell Kaplan, Justin Johnson, and Li Fei-Fei. HiDDeN: Hiding data with deep networks. In *European Conference on Computer Vision (ECCV)*, pp. 657–672, 2018.

A CODE CONFIGURATIONS

For each payload we chose the number of planes B such that the steganographic code providing this payload on B planes flips 5 to 15 % of the positions of the binary stego. This operating point performed best in our experiments, and in this range the distortion introduced by the flips is approximately additive (App. B). Table 3 reports for each configuration the number of message bits per block q, the reserved budget n_f of the polar code and the number of binary planes B. We also report the fraction of flips of an ideal quantizer at the rate $(q + n_f)/n$ (the embedding rate for $n_f = 0$), $h_2^{-1}\big((q+n_f)/n\big)$, together with the fraction of flips measured for the code. Each model was trained on the flip range given in Table 3, and one model can serve several codes and payloads: the model with $B = 6$ is used both for the polar code at 2 bpp and for the LDGM code at 3 bpp, where the LDGM code flips 13.6 % of the positions, slightly above the training range.

Table 3 also gives the number of blocks and the payload of each configuration. The fixed binary sequence $\boldsymbol{s}_{\text{bin}}$ consists of B planes of 256×256 bits, i.e., $T = B \cdot 2^{16}$ positions. A code of block length n, with $n = 1024$ for the polar code and $n = 384$ for the LDGM code, splits it into $\lfloor T/n \rfloor$ blocks and embeds q message bits into each block, which gives $q\lfloor T/n \rfloor$ bits per image, about Bq/n bits per pixel.

Table 3: StegoCoder configurations.

Code	bpp	B	q	n_f	blocks	payload (bits)	$(q+n_f)/n$	h_2^{-1}	code p_{flip}	train p_{flip}
Polar	1	4	257	128	256	65 792	0.376	0.073	0.083	0.06–0.10
Polar	2	6	341	128	384	130 944	0.458	0.097	0.108	0.06–0.12
Polar	3	9	341	128	576	196 416	0.458	0.097	0.108	0.05–0.15
Polar	4	10	410	128	640	262 400	0.525	0.119	0.129	0.10–0.15
LDGM	1	2	192	–	341	65 472	0.5	0.110	0.136	0.08–0.15
LDGM	2	4	192	–	682	130 944	0.5	0.110	0.136	0.06–0.10
LDGM	3	6	192	–	1024	196 608	0.5	0.110	0.136	0.06–0.12
LDGM	4	8	192	–	1365	262 080	0.5	0.110	0.136	0.08–0.15

B ADDITIVITY OF THE FLIP DISTORTION

The steganographic codes of Sec. 2.1 assume that the distortion is a sum of per-position costs, Eq. 1. In the proposed scheme Render introduces distortion into the image from the flips in the binary stego. If the assumption on the per-cost independence holds, the resulting MSE between the stego image and the image produced by Render from all-zero binary plane $\boldsymbol{s}_{\text{bin}}$ should be a linear function of the number of introduced flips. Figure 3 directly tests this assumption. Here we plot the dependence between resulting MSE and a percent of the bit flips chosen uniformly at random, from 2 to 30 % of the $B \cdot 2^{16}$ positions, for considered $B = 4, 6, 9, 10$ planes corresponding to 1–4 bpp for polar code. Dashed lines were obtained by fitting the points inside the training flip range (shaded; Table 3).

If the flips contributed independently, the MSE would be proportional to the number of flips. The range of flip rates seen in training (5–15 %, Table 3) contains the rates at which the codes operate (7–13 %), and in this area function is approximately linear for all 4 considered B, with further growth of the cost per flip as the flip rate increases beyond operating point.

C LDGM CODE CONSTRUCTION

The LDGM code is built from the parity-check matrix H of a $(384, 192)$ LDPC code from (Helmling et al., 2019): H is brought to the systematic form $H_{\text{sys}} = [P \mid I_{n-k}]$ by Gaussian elimination over $\mathbb{F}_2$ and a column permutation, and $G = [I_k \mid P^T]$. Embedding runs 200 BP iterations with an annealing schedule of the decimation parameter (Yao et al., 2024).

D POLAR CODE CONSTRUCTION

The polar transform is the matrix $G_n = B_n F^{\otimes \log_2 n}$, with the block length n that is a power of two, the kernel $F = \left[\begin{smallmatrix} 1 & 0 \\ 1 & 1 \end{smallmatrix}\right]$, the Kronecker power $\otimes$ and the bit-reversal permutation B_n. Polar matrix

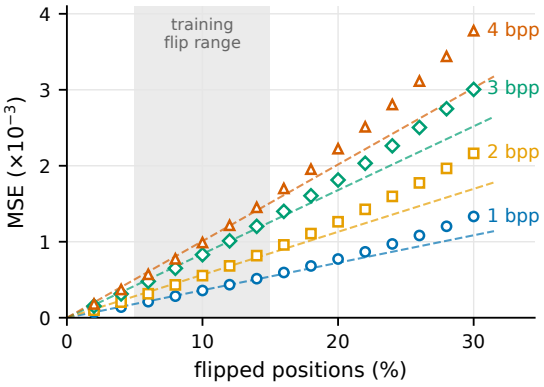

Figure 3: Distortion of the rendered image against the fraction of flipped positions in the binary stego. Each model with $B = 4, 6, 9, 10$ corresponds to 1–4 bpp for polar code.

G_n maps $\boldsymbol{u} \in \{0,1\}^n$ to $\boldsymbol{y} = \boldsymbol{u}G_n$, so that u_i is conveyed over the synthetic subchannel $W_n^{(i)}$. The reliability of position i is measured by the Bhattacharyya parameter $Z(W_n^{(i)}) \in [0,1]$, the smaller the more reliable, which can be estimated by a degrading-merging algorithm (Tal & Vardy, 2013) or, for channels without a model, by Monte-Carlo simulation.

Decoding. The successive cancellation (SC) decoder estimates $\hat{\boldsymbol{u}}$ sequentially from the log-likelihood ratios (LLRs) $l_i = \log \frac{\Pr(y_i=0|\boldsymbol{r})}{\Pr(y_i=1|\boldsymbol{r})}$, $i = 1, \ldots, n$, computed from the channel observations $\boldsymbol{r}$, and the decoder uses the known values $\boldsymbol{u}_{\mathcal{F}}$ of the frozen set. Its accuracy can be improved by successive cancellation list (SCL) decoder (Tal & Vardy, 2015), whose complexity is controlled by the list size L.

Neural polar decoder. The SC decoder is kept as a computation graph over the polar factor graph, and its elementary operations are replaced by small networks acting on an embedding of dimension d_e instead of a scalar LLR (Aharoni et al., 2024). The decoder keeps the $\mathcal{O}(n \log n)$ structure of SC decoding with a constant number of parameters ($d_e = 8$), and it runs inside the list decoder of Tal & Vardy (2015) with the same list size as the standard receiver. The reserved set and the message positions are the same as for standard polar in Sec. 3.2. The neural decoder observes the soft estimates of the BitPredictor and learns from data how to weigh the confidence of each estimate, whereas the standard receiver uses them directly as LLRs, $l_i = \log \frac{1-\hat{s}_i}{\hat{s}_i}$.

E Codes on a Synthetic Bit-Flip Channel

Table 4 reports steganographic coding performance for chosen codes on a synthetic noisy bit-flipping channel: the $(1024, 512)$ polar code and the $(384, 192)$ LDGM code are applied to a random binary cover, and the stego bits are flipped independently with a given probability before extraction. Without a reserved budget the two codes behave alike, changing 12–14% of the cover bits and losing the message under 1% of flips. The reserved budget closes this gap: at $n_f = 300$ the polar code tolerates 2% of flips at a message BER of 0.3%, at the price of changing 25% of the cover bits instead of 12% and of lowering the quantized fraction q/n from 0.50 to 0.21. Every reserved position removes one quantized position and adds one known value for decoding: the cover BER follows $h_2^{-1}\big((q + n_f)/n\big)$, while the message BER falls sharply once n_f/n exceeds $h_2(\theta)$ of the flip probability θ, as discussed in Sec. 2.3. The main results use $n_f = 128$, though the budget is a scheme's parameter and it can be raised for a noisier channel.

F Architecture Details

Both the Render and BitPredictor are four-layer convolutional stacks with 3×3 kernels, padding 1 and ReLU activation, and for each of the modules, the spatial size $W \times H$ is preserved. The

Table 4: Cover BER and message BER (%) of the LDGM $(384, 192)$ and polar $(1024, 512)$ codes on a random binary cover whose stego bits are flipped independently with the given probability. Values below 0.01% are shown as ≈ 0.

		Message BER (%) at flip probability							
Code	Cover BER (%)	0%	0.5%	1%	2%	3%	4%	5%	10%
LDGM $(384, 192)$	13.5	0	21.0	33.9	44.8	47.8	49.3	49.5	49.9
Polar, $n_f = 0$	12.1	0	26.5	36.3	44.0	47.1	48.7	49.1	50.1
Polar, $n_f = 50$	13.8	0	16.6	36.1	45.6	47.9	48.9	49.4	49.9
Polar, $n_f = 100$	15.6	0	2.4	15.9	41.3	47.1	48.8	49.3	50.0
Polar, $n_f = 150$	17.5	0	0.2	3.1	25.9	42.8	47.9	49.1	49.9
Polar, $n_f = 200$	19.7	0	≈0	0.4	8.0	31.4	43.7	47.6	49.7
Polar, $n_f = 250$	22.2	0	≈0	≈0	1.6	12.7	32.5	43.5	49.6
Polar, $n_f = 300$	25.0	0	≈0	≈0	0.3	2.7	14.6	32.1	49.7
Polar, $n_f = 350$	28.3	0	≈0	≈0	≈0	0.6	4.2	15.8	49.1
Polar, $n_f = 400$	32.1	0	≈0	≈0	≈0	0.1	0.9	4.4	48.2

steganographic code takes as a cover B binary planes, which are set to all zeros. Then the channel widths for each convolutional network are: Render $(B + 3) \to 128 \to 64 \to 32 \to 3$ with a final sigmoid; BitPredictor $3 \to 32 \to 64 \to 128 \to B$.

G Sensitivity to the Message Realization

Every row of Table 1 is computed with one random message per image. Since the stego image depends on the message and the Render produces the modified image, both the image quality and the BER are, in principle, functions of the drawn message. This section measures how strong that dependence is, and whether it differs between the proposed scheme and the LISO and SteganoGAN baselines.

We fix the first 10 test images and draw 100 independent messages per image. The noise realization is seeded per (image, σ) so that it is *identical* across the 100 draws, and only the message varies. For every $(\text{image}, \text{message})$ pair we embed, evaluate PSNR of the stego against the cover, and decode after the noise channel. We report the standard deviation averaged over the images for the polar scheme with standard and neural receiver at 3 bpp, for LISO and its best-iterate selection, and for SteganoGAN.

Results. The results of the proposed method are nearly message-invariant. At 3 bpp the per-image standard deviation of the stego PSNR across messages is 0.012 dB for the polar scheme, both standard and co-trained, and the BER varies by $8.9 \cdot 10^{-5}$ around a mean of $6.5 \cdot 10^{-3}$. SteganoGAN, also a feed-forward embedder, shows similar behaviour, while LISO is different: because its stego is the endpoint of a per-image optimization whose trajectory depends on the payload, its PSNR varies much more across messages.

Table 5: Message variability at matched payload 3 bpp: standard deviation across 100 message draws. $\overline{\text{BER}}$ is counted at $\sigma = 5 \cdot 10^{-3}$.

bpp	method	std PSNR (dB)	std BER	$\overline{\text{BER}}$
3	LISO	0.66	3.0e-3	5.1e-2
3	LISO (best iterate)	0.40	1.1e-2	4.0e-2
3	SteganoGAN (clean-trained)	0.008	9.6e-4	3.36e-1
3	SteganoGAN (noise-trained)	0.004	6.1e-4	7.4e-2
3	Polar, std. SCL	0.012	8.9e-5	6.8e-5
3	+ co-trained	0.013	3.1e-4	7.0e-4

Interpretation. In StegoCoder the message determines which positions of the binary stego flip, but their number is set by the code rate (Table 3), and the distortion of the rendered image depends

on the number of flips rather than on their positions (App. B). The quality and robustness of a stego image are therefore properties of the image and the configuration, and the single-message evaluation of Table 1 is representative. For the optimization-based baselines the message enters the objective, and their PSNR varies considerably across messages (Table 5).

H Refinement Objectives and the Clean Channel

Table 6 reports results specifically optimised for clean channel, including the StegoCoder with the pixel refinement for the non-perturbed image given in Eq. 5 and the SteganoGAN trained without stego image perturbations, as it was originally proposed. For StegoCoder, we provide the numbers before and after this refinement for reference. We also report LSB, the natural choice when the stego image is never perturbed; it writes b message bits into the last b bits of every pixel channel giving a capacity of $3b$ bpp. LSB has the highest PSNR at every payload (45.9–55.9 dB) and recovers every message without noise, but loses it completely at $\sigma = 5 \cdot 10^{-3}$.

Optimising for the clean channel improves image quality and the noiseless BER performance. SteganoGAN trained without noise has the highest PSNR of the learned methods at every payload, and the clean refinement adds between about half a decibel and 2.2 dB to every StegoCoder setting. The recovery of SteganoGAN, however, stays far from exact: on the noiseless channel it recovers only 12 of the 500 images without a bit error at 1 bpp and none at the higher payloads, and under noise it recovers no image at all. Clean GD refinement for StegoCoder also significantly improves message recovery for the noiseless channel, though for the channel with noise the recovery often gets worse.

Table 6: Optimising for the clean channel, CelebA-HQ. Shaded rows are StegoCoder framework.

method	PSNR ↑	$\text{LPIPS}_{\text{vgg}}$ ↓	$\sigma = 0$		$\sigma = 5{\cdot}10^{-3}$		$\sigma = 10^{-2}$	
			BER ↓	0-err ↑	BER ↓	0-err ↑	BER ↓	0-err ↑
1 bit per pixel								
LSB	**55.93**	**0.002**	**0**	**500**	0.497	0	0.498	0
SteganoGAN, clean-trained	39.34	0.020	$4.1{\cdot}10^{-4}$	12	0.229	0	0.351	0
SteganoGAN, noise-trained	35.99	0.055	$2.9{\cdot}10^{-3}$	20	$6.7{\cdot}10^{-3}$	0	0.034	0
LDGM	36.05	0.053	$2.0{\cdot}10^{-5}$	477	$5.3{\cdot}10^{-5}$	451	0.099	0
+ GD (clean)	38.28	0.037	**0**	**500**	0.054	0	0.432	0
Polar, std. SCL	35.33	0.076	**0**	**500**	**0**	**500**	**0**	**500**
+ GD (clean)	37.24	0.056	**0**	**500**	**0**	**500**	0.010	0
Polar, co-trained	36.84	0.059	**0**	**500**	**0**	**500**	$6.6{\cdot}10^{-4}$	181
+ GD (clean)	38.24	0.046	**0**	**500**	**0**	**500**	0.053	0
2 bits per pixel								
LSB	**52.90**	**0.007**	**0**	**500**	0.497	0	0.498	0
SteganoGAN, clean-trained	41.72	0.016	$4.6{\cdot}10^{-3}$	0	0.290	0	0.389	0
SteganoGAN, noise-trained	33.94	0.140	$7.3{\cdot}10^{-3}$	0	0.015	0	0.073	0
LDGM	33.20	0.111	$3.5{\cdot}10^{-3}$	2	$9.6{\cdot}10^{-3}$	0	0.251	0
+ GD (clean)	34.92	0.085	$9.5{\cdot}10^{-6}$	467	0.126	0	0.466	0
Polar, std. SCL	32.10	0.176	**0**	**500**	**0**	**500**	$\mathbf{4.8{\cdot}10^{-4}}$	**287**
+ GD (clean)	33.61	0.143	**0**	**500**	**0**	**500**	0.117	0
Polar, co-trained	34.41	0.130	**0**	**500**	$2.3{\cdot}10^{-6}$	499	0.222	0
+ GD (clean)	35.29	0.111	**0**	**500**	**0**	**500**	0.422	0
3 bits per pixel								
LSB	**51.14**	**0.008**	**0**	**500**	0.497	0	0.498	0
SteganoGAN, clean-trained	39.80	0.025	0.014	0	0.333	0	0.408	0
SteganoGAN, noise-trained	30.14	0.164	0.053	0	0.072	0	**0.124**	0
LDGM	30.97	0.205	0.024	0	0.058	0	0.365	0
+ GD (clean)	32.36	0.171	$7.8{\cdot}10^{-4}$	28	0.176	0	0.474	0
Polar, std. SCL	30.21	0.242	$8.6{\cdot}10^{-6}$	493	$1.2{\cdot}10^{-4}$	**447**	0.145	0
+ GD (clean)	31.04	0.217	$3.6{\cdot}10^{-7}$	499	$\mathbf{3.5{\cdot}10^{-5}}$	444	0.366	0
Polar, co-trained	31.50	0.209	$7.7{\cdot}10^{-5}$	483	$1.4{\cdot}10^{-3}$	299	0.432	0
4 bits per pixel								
LSB	**45.89**	**0.031**	**0**	**500**	0.466	0	0.497	0
SteganoGAN, clean-trained	39.50	0.040	0.011	0	0.341	0	0.409	0
SteganoGAN, noise-trained	30.44	0.160	0.091	0	0.118	0	**0.180**	0
LDGM	28.75	0.281	0.055	0	0.147	0	0.423	0
+ GD (clean)	29.88	0.246	$7.3{\cdot}10^{-3}$	0	0.243	0	0.472	0
Polar, std. SCL	28.36	0.306	$9.5{\cdot}10^{-5}$	461	$1.6{\cdot}10^{-3}$	**202**	0.377	0
+ GD (clean)	29.17	0.280	$3.3{\cdot}10^{-6}$	495	$1.2{\cdot}10^{-3}$	142	0.474	0
Polar, co-trained	28.87	0.295	$6.2{\cdot}10^{-5}$	476	$1.9{\cdot}10^{-3}$	183	0.465	0
+ GD (clean)	29.61	0.272	$2.0{\cdot}10^{-6}$	496	$\mathbf{1.0{\cdot}10^{-3}}$	187	0.485	0

Table 7: Quantizing the stego image to 8 bits, CelebA-HQ. The rendered stego image rounded to the 256-level integer grid before transmission.

refinement	stego	PSNR ↑	$\sigma = 0$		$\sigma = 5{\cdot}10^{-3}$		$\sigma = 10^{-2}$	
			BER ↓	0-err ↑	BER ↓	0-err ↑	BER ↓	0-err ↑
2 bits per pixel								
Polar, std. SCL	real-valued	32.13	0	500	0	500	$5.4{\cdot}10^{-4}$	250
	8-bit	32.10	0	500	0	500	$5.4{\cdot}10^{-4}$	264
3 bits per pixel								
Polar, std. SCL	real-valued	30.21	$8.6{\cdot}10^{-6}$	493	$1.2{\cdot}10^{-4}$	447	0.145	0
	8-bit	30.21	$9.6{\cdot}10^{-6}$	492	$1.4{\cdot}10^{-4}$	447	0.154	0
4 bits per pixel								
Polar, std. SCL	real-valued	28.36	$9.5{\cdot}10^{-5}$	461	$1.6{\cdot}10^{-3}$	202	0.377	0
	8-bit	28.40	$7.9{\cdot}10^{-5}$	463	$1.5{\cdot}10^{-3}$	196	0.382	0

I Quantization to 8 Bits

Table 7 provides results for StegoCoder with the rendered stego image rounded to the 256-level integer grid, so the transmitted stego image is an ordinary 8-bit image rather than a real-valued tensor. Rounding is a perturbation uniform in ± 0.5 grey levels, with standard deviation $1/(255\sqrt{12}) \approx 1.1 \cdot 10^{-3}$ on the $[0, 1]$ scale. This is about a fifth of the smallest nonzero noise level of the evaluation and well inside the noise range the scheme is trained with. Rounding therefore changes PSNR by at most 0.04 dB and leaves the recovery essentially unchanged.

J Transfer to AFHQ-Dog

Table 8 applies the CelebA-HQ models without retraining to 500 test images of AFHQ-Dog. The polar scheme with the standard receiver transfers without loss of robustness: as on CelebA-HQ, it recovers every image without a bit error at $\sigma = 5 \cdot 10^{-3}$ at both payloads, and almost every image at 1 bpp even at $\sigma = 10^{-2}$. The Render and the BitPredictor see only a small image region around each position of the binary stego, so the learned channel depends on local image statistics rather than on the dataset, and the code does not depend on the images at all. The image quality transfers as well: StegoCoder has the lowest LPIPS of the learned methods at 1 bpp and a higher PSNR than the transferred LISO at 2 bpp. LISO recovers no image under noise, even when trained on AFHQ-Dog itself.

Table 8: CelebA-HQ-trained models evaluated on AFHQ-Dog, 500 test images. LISO (AFHQ) is trained on AFHQ-Dog and is shown as an in-domain reference. Bold marks the best value in each column within a payload block; shaded rows are ours.

method	PSNR ↑	$\text{LPIPS}_{\text{vgg}}$ ↓	$\sigma = 0$		$\sigma = 5{\cdot}10^{-3}$		$\sigma = 10^{-2}$	
			BER ↓	0-err ↑	BER ↓	0-err ↑	BER ↓	0-err ↑
1 bit per pixel								
LISO (best iter)	35.90	0.098	$3.2{\cdot}10^{-6}$	461	0.053	0	0.185	0
LISO (best iter), AFHQ	**37.04**	0.086	$1.5{\cdot}10^{-6}$	477	0.071	0	0.206	0
Polar, std. SCL	35.28	0.053	**0**	**500**	**0**	**500**	$\mathbf{1.3{\cdot}10^{-6}}$	**498**
+ GD refinement	**37.04**	**0.038**	**0**	**500**	**0**	**500**	$1.1{\cdot}10^{-2}$	0
+ GD (noise-aware)	36.31	0.043	**0**	**500**	**0**	**500**	$2.3{\cdot}10^{-4}$	325
2 bits per pixel								
LISO (best iter)	30.68	0.205	$2.9{\cdot}10^{-5}$	152	0.037	0	0.160	0
LISO (best iter), AFHQ	**35.25**	**0.097**	$9.4{\cdot}10^{-6}$	396	0.056	0	0.183	0
Polar, std. SCL	31.79	0.146	**0**	**500**	**0**	**500**	$\mathbf{1.2{\cdot}10^{-3}}$	**193**
+ GD refinement	33.19	0.115	**0**	**500**	**0**	**500**	$1.3{\cdot}10^{-1}$	0
+ GD (noise-aware)	32.62	0.126	**0**	**500**	**0**	**500**	$9.5{\cdot}10^{-3}$	0

K Computational Cost

Both the sender and the receiver of StegoCoder are single feed-forward passes, linear in the number of pixels and nearly independent of the payload. The number of planes B enters only the Fusion layer, the first convolution of the Render and the last convolution of the BitPredictor, so going from 1 to 4 bpp raises the cost by $7\,\%$ (Table 9). SteganoGAN has the same feed-forward structure at about half the cost. LISO is different in kind: its sender is an unrolled optimizer that evaluates and backpropagates through its decoder at every iteration. One LISO iteration costs about as much as our sender and receiver together, but the 150 iterations of LISO put it two orders of magnitude above both feed-forward methods. All three networks have at most $0.2\,$M parameters.

On top of the networks, StegoCoder pays for the code. SCL decoding costs $\mathcal{O}(L\,n\log n)$ per block, i.e., $\mathcal{O}(L\,BHW\log n)$ per image, and the sender pays the same once for quantization. This is linear in the payload and logarithmic in the block length, and at $L = 8$, $n = 1024$ it stays more than two orders of magnitude below the convolutional work at every payload. We report operation counts rather than wall-clock times, since the runtime of our unoptimized SCL implementation would not represent the complexity.

Table 9: Inference cost per 256×256 image at batch size one: parameters at 1 bpp and multiply–accumulate operations (GMac).

			GMac			
Method	Module	kparams	1 bpp	2 bpp	3 bpp	4 bpp
SteganoGAN	dense encoder	60.9	3.98	4.04	4.10	4.16
	dense decoder	29.7	1.94	1.99	2.05	2.11
	total	*90.6*	*5.92*	*6.04*	*6.15*	*6.27*
StegoCoder	Fusion	0.4	0.03	0.05	0.09	0.10
	Render	101.3	6.63	6.78	7.00	7.08
	BitPredictor	97.9	6.40	6.55	6.78	6.85
	total	*199.6*	*13.05*	*13.37*	*13.86*	*14.03*
LISO	context net (once)	29.1	1.92	1.94	1.96	1.98
	decoder (per it.)	19.9	1.32	1.33	1.35	1.37
	update block (per it.)	108.0	7.06	7.06	7.06	7.06
	total, 1 it.	*176.9*	*14.2*	*14.3*	*14.4*	*14.5*
	total, 150 it.	*176.9*	*1653.8*	*1662.3*	*1670.9*	*1679.4*

L Detectability and Sender-Side Defence

The stego image is rendered from the cover, so a convolutional steganalyser can learn to detect it. We measure this with two pretrained detectors, XuNet (Xu et al., 2016) and KeNet from SiaStegNet (You et al., 2020), loaded with the public weights and preprocessing of Chen et al. (2023) and kept frozen throughout; an image is flagged when the detector's two-class margin is positive. The defence refines the stego image at the sender after embedding, while the receiver and the detector are frozen. Starting from the stego image of Table 1, 200 Adam steps minimise a detector term, a message term and an image term (MSE and LPIPS to the cover). Then the detector term switches off once the image is no longer flagged. During the optimisation, every ten steps the current image is decoded by the actual receiver under the Gaussian noise at $\sigma = 5 \cdot 10^{-3}$ and only images that decode without error are kept as candidates. After the optimisation, the five candidates with the lowest detector margin, together with the final iterate and the undefended stego, are tried again. The image that is sent is the least detectable of those that are successfully decoded.

The defence takes both detectors from 99–$100\,\%$ of images flagged to none, at a cost of 0.8 dB against XuNet and 1.3 dB against KeNet, while message recovery is unchanged: all 500 images are still recovered without any error (Table 10).

Authors in Chen et al. (2023) report that adding XuNet to the per-image optimisation of LISO at 1 bpp lowers detection from $100\,\%$ to $2\,\%$, at a cost of 2.2 dB and a roughly 40-fold increase in message error rate: LISO trades the detector loss against its decoding loss inside the optimisation,

Table 10: Sender-side defence at 1 bpp, CelebA-HQ, 500 test images, against frozen pretrained steganalysers. PSNR is measured on the defended images, against the cover; BER and 0-err at $\sigma = 5 \cdot 10^{-3}$ with the standard receiver. Undefended PSNR is 35.4 dB.

detector	PSNR ↑	flagged before	flagged after ↓	BER ↓	0-err ↑
XuNet	34.61	100 %	**0 %**	0	500/500
KeNet	34.09	99.2 %	**0 %**	0	500/500

whereas StegoCoder decodes each candidate with the deployed receiver and sends only those that decode without error.