Pulsar: Secure Steganography for Diffusion Models

Tushar M. Jois
City College of New York
tjois@ccny.cuny.edu

Gabrielle Beck
Johns Hopkins University
becgabri@cs.jhu.edu

Gabriel Kaptchuk
University of Maryland, College Park
kaptchuk@umd.edu

Abstract

Widespread efforts to subvert access to strong cryptography has renewed interest in steganography, the practice of embedding sensitive messages in mundane cover messages. Recent efforts at provably secure steganography have focused on text-based generative models and cannot support other types of models, such as diffusion models, which are used for high-quality image synthesis. In this work, we study securely embedding steganographic messages into the output of image diffusion models. We identify that the use of variance noise during image generation provides a suitable steganographic channel. We develop our construction, Pulsar, by building optimizations to make this channel practical for communication. Our implementation of Pulsar is capable of embedding $\approx$ 320–613 bytes (on average) into a single image without altering the distribution of the generated image, all in < 3 seconds of online time on a laptop. In addition, we discuss how the results of Pulsar can inform future research into diffusion models. Pulsar shows that diffusion models are a promising medium for steganography and censorship resistance.

1 Introduction

Years of sustained effort by the security and cryptography communities has led to widespread deployment of strong, secure communication technologies, including end-to-end encrypted messaging [PM; Wha17] and TLS [Res18]. While these technologies prevent high-resource attackers from viewing or manipulating the contents of communications, they do nothing to hide that the communication itself is occurring. In areas where encrypted communication technologies are blocked or cause for suspicion, leaking this metadata can have deadly consequences. As governments' suspicion of encrypted communication continues to grow around the world (e.g., KOSA in the USA [Unib], Online Safety Act in the UK [Unia], CSA regulation in the EU [Eur], TOLA in Australia [Aus], etc.), it is important to *proactively* develop new, generalizable, flexible techniques that can complement existing secure communication technologies and provide security and privacy to individuals at increased risk; if this development is not done proactively, the technology will be too immature for deployment when it is needed.

Steganography [Sim83] allows a sender to embed a sensitive message into a mundane context such that only the intended receiver is able to detect and extract the message, making it the ideal tool for communicating in areas where encryption cannot be safely used. Steganography can transform a conversation that might be seen as "subversive" into one that would not arouse suspicion—even when the content of the conversation itself is directly monitored by authorities. Importantly, this prevents censors from *selectively* blocking content (or selectively blocking encrypted communications). Steganography can also enable digital "dead drop" deployments, in which encoded messages are left on the public internet and the intended recipient can recover the message without leaving evidence of direct communication.

Although the value of efficient, secure steganography tools has been evident for decades, concrete constructions have long been lacking. Formal techniques, stemming from the cryptographic literature, provide

provable hiding guarantees but cannot be concretely deployed, either because they are inefficient or make unrealistic assumptions. Heuristic steganographic techniques, on the other hand, have been deployed in practice, but are often vulnerable to simple statistical attacks. For example, LSB steganography [Aur96] (in which bits of the sensitive message are embedded into the least significant bits of pixels' values) is often considered a viable choice, but the manipulation of low order bits is detectable [FPK07; FGD01; SCC07].

Steganography for generative models. A new line of research attempts to marry the formal techniques with the heuristic approaches by steganographically encoding messages into the output of generative, machine learning models. This provides a systematic divide between the provable guarantees and the heuristic assumptions such that they can be analyzed separately. These steganographic schemes can ensure that an adversary cannot distinguish between typical model output and model output carrying a sensitive message, but make no formal claims about an adversary attempting to detect if a message is generated with the help of a model (as opposed to created manually by the sender). While this approach falls short of providing end-to-end security, *this is the best one can hope for* when embedding into contexts where explicit descriptions of the statistical nature of the communication channel cannot be described (e.g., natural human language, art, or photographs). In these cases, encoded messages can only hope to achieve indistinguishability with the best known approximation we have of the communication channel; machine learning models are the best approximation of natural human language, art, or photographs currently available.

Existing steganographic proposals [KJG+21; DCW+23] are designed to work with popular model structures, like the transformer networks that form large language models [VSP+17]. These models iteratively generate output by producing an explicit probability distribution of different *tokens* (e.g., words, pixels) that could follow the given prompt. During typical operation, a single token is sampled from this distribution as output and appended to the prompt. The updated prompt is then fed back into the model, continuing until the output is the desired length. The randomness used to sample from the distribution is pulled from an arbitrary high entropy source.

When steganographically embedding a sensitive message into the output, the sampling is done as a function of the sensitive message such that the receiver can infer the bits of the sensitive message. For example, the sender might encode the leading bits of the message into a token using an arithmetic encoding scheme [KJG+21] or distribution copies [DCW+23]. To ensure that this steganographic encoding process does not change the statistical profile of the model output, the sensitive message is generally encrypted using a pseudorandom cipher,[1] making it, in effect, a high entropy source.

Steganography for diffusion models. Although text is a natural medium to consider for steganographic communication, embedding into text produces stegotext (the steganography equivalent of ciphertext) that is significantly longer than the initial message. This is because natural language text is actually quite low entropy and steganographic encoding rates are bound by the entropy in the communication channel. As such, sending even relatively short sensitive messages steganographically might require sending paragraphs or pages of model-generated text. While transmitting this much text might occasionally be appropriate, in many cases this will break steganography's illusion. This limitation makes existing, text-based approaches insufficient, motivating the need for encoding techniques that work with other media.

After text, images are the next media into which we might want to embed steganographic messages. Importantly, images do not share the same limitations as text: images have significantly higher entropy and it is commonplace to exchange or post large image files. These properties mean that it should (in principle) be possible to steganographically send large amounts of information using images[2] without arousing suspicion. Building on this intuition, Ding, Chen, Wang, Zhao, Zhang, and Yu [DCW+23] generate steganographic images using Image GPT, a neural network designed to produce images.

While transformer networks remain the most effective generative models in the natural language processing domain, a new model architecture, *diffusion models* [SWM+15; SME20; HJA20; DN21; RBL+22] have

[1] Some proposals [GGA+05; SS07; YHC+09; CC10; CC14; FJA17; VNB+17; YJH+18; Xia18; YGC+19; HH19; DC19; ZDR19] fail to properly encrypt the message before encoding, leading to an insecure construction.

[2] Note that we make a distinction between *existing images* that are used to hide content (e.g., by using least significant bits [Aur96; FGD01; FPK07; SCC07]) and *machine learning-generated images* that hide content during generation. We focus on the latter in this work, as the former cannot provide provable guarantees.

proven to produce higher quality output for images. Diffusion models have quickly captured the public's imagination and have become the *de facto* option for machine learning image generation. Unlike neural networks, diffusion models generate all the pixels in the output image at the same time. This significant departure from the typical transformer architecture means that existing steganographic approaches cannot be adapted for diffusion models. Some efforts have attempted to build diffusion-specific techniques [YZX+24; KSC+23; LCG23; PHW+23], but these approaches are non-black box in the underlying models. While these approaches yield impressive results, they risk making assumptions about low-level model details that may not hold as the rapidly improving field of diffusion models continues to evolve. As such, there is a need for more robust, generalizable techniques.

In this work, we study steganographic embedding mechanisms that can work for generic conceptions of diffusion models. Our techniques allow for the production of steganographic communication tools that can send large amounts of data without arousing suspicion. We find that encoding with diffusion models can be *faster* than encoding with other networks, because (1) encoding using existing text-like techniques requires querying a transformer model many times while diffusion models are *one shot* and (2) the flexibility of our solution facilitates interoperability with multiple concrete diffusion models, allowing us to take advantage of rapid improvements in diffusion model technologies. We are drawn to diffusion models both because posting the output of diffusion models has become common practice on the Internet [Red; Civ] and because they offer the best synthetic approximation of "natural" images shared in more traditional communication channels.

Our contributions. We study steganographic encoding schemes for images generated by diffusion models. Our goals are that our approach should be both principled and easily generalizable, such that it can be applied to new diffusion models as they are developed. In doing so, we have several concrete contributions:

- **A principled study of steganography for diffusion models.** As we are studying steganography for diffusion models, we begin our work by providing a thorough review of the various opportunities for hiding data that diffusion models afford, framed around the motivating deployment of steganographic dead drops. We begin by noting that cryptographically secure steganography is fundamentally about *randomness recovery*. As such, we can systematically iterate through the entropy sources consumed by a machine learning model and identify those that are most promising for hiding data.

- Pulsar**, a novel steganographic encoding scheme for diffusion models.** We design a novel, symmetric key steganographic encoding scheme that encodes data into the output of diffusion models that operate in the pixel space.[3] Pulsar embeds data into images by mapping pixels in the image to bits in the message; when sampling Gaussian noise for each pixel, the encoder uses one of two pseudorandom functions, one for pixels mapped to a zero bit and one for pixels mapped to a one bit. This results in a generic, but noisy, randomness recovery scheme, which can be improved using error correcting codes. We implement Pulsar and integrate it with existing diffusion models, showing that it can encode hundreds of bytes of plaintext information in a 256×256 pixel generated image. Our implementation encodes faster than state-of-the-art neural network steganographic systems.

- **Recommendations for steganography-friendly machine learning models.** We note that the empirical nature of machine learning research can lead to some model architectures incorporating semi-arbitrary design choices, while also significantly impairing the ability to steganographically embed messages into model output. Reflecting on our study of steganography for diffusion models, we highlight several ways in which model architecture can be improved to better support steganographic encoding. We hope that this can inspire designers of future model architectures to test if models can be made steganography friendly without reducing output quality.

Deployment scenario and threat model. In this work we assume a similar deployment scenario and threat model as recent work on symmetric key, model-based steganography [KJG+21; DCW+23]. Namely, a

[3] Pulsar is designed for models that operate in the pixel space, as opposed to a compressed latent space. We discuss this difference in Section 3.1.

sender and receiver generate shared key material out-of-band and select a (diffusion) model to use as a covert channel. We assume that their communications are monitored by a computationally powerful adversary (e.g., a state actor) who also has access to the selected diffusion model. The sender and receiver wish to disguise the contents of their communication such that the adversary cannot determine if their exchanged messages contain typical model output or steganographically encoded messages. As this work focuses on laying out a feasibility result, we make the simplifying assumption that the adversary does not launch active attacks and must distinguish based on observing encoded messages.

We do not attempt to formalize the ability of the adversary to distinguish between model output and "normal" human communication. For full-scale deployments of steganography, deployment designers must carefully consider how well steganographically generated messages fit in with existing communication channels (in addition to incorporating other best practices from cryptographic messaging like forward security). While this is certainly a limitation of our work, we note that humans exchanging the outputs of generative models has become increasingly common. For example, professional communications may be generated with the help of text models like ChatGPT [Kor23] and social media was flooded with "AI art" [Red; Vin22; Civ] after the release of Stable Diffusion [RBL+22] and similar models. These developments mitigate the risks associated with encoding information in model output. We discuss our motivating deployment more in Section 3.2.

2 Related Work

Steganography maps a message into a stegotext, a set of elements from a chosen target distribution. Typically, this distribution is mundane such that a censor would not find it suspicious. This strengthens encryption, in which a ciphertext can have an arbitrary distribution and does not aim to hide the fact that it is a ciphertext.

Provable steganography. Steganography was first formalized by Simmons [Sim83] and has since been the subject of a tremendous amount of theoretical research. The theoretical literature has focused on establishing the universal feasibility of steganography for arbitrary stegotext distributions, provided the distribution has some amount of entropy. For example, prior work has shown that universal steganography can be realized with information-theoretic security [AP98; ZFK+98; Mit99; Cac00], computational security, [HLv02; vH04; BC05] and statistical security [SSM+06a; SSM07; SSM+06b]. There are also symmetric-key [Cac00; HLv02; RR03] and public-key constructions [vH04; BC05; Le03; LK03] discussed in the literature. More recently, techniques deeply related to steganography have been studied as a tool to achieve security when an adversary is able to compel a receiver to decrypt ciphertexts [HPR+19; PPY22].

Practical steganography. A complementary line of research has studied the feasibility of concretely efficient, deployable steganography. Generally, these steganographic constructions either rely on heuristic security analyses, e.g., obs4/ScrambleSuit-style protocol obfuscators [WPF13] and domain fronting [FLH+15], or can only embed messages into very specific covertext distributions, e.g., pseudorandom bit streams. For example, SkypeMorph [MLD+12], CensorProofer [WGN+12], and FreeWave [HRB+13] tunnel Tor [RSG98; DMS04; Tor] traffic through Voice over IP traffic, which is usually encrypted with a pseudorandom cipher. Other examples include Format Transforming Encryption [LDJ+14; DCR+13; DCS15; OYZ+20], which requires implementers to explicitly describe the statistical properties of the target distribution.

Generative model steganography. Recently, there has also been work attempting to leverage generative neural networks to instantiate concretely efficient universal steganography by embedding messages into the output of the model. By using machine learning models, these works cleanly separate their heuristic guarantees from their formal ones. Specifically, these protocols offer no formal guarantees about how easy it is to detect that content has been produced by a machine learning model, but can make formal arguments about the statistical shifts induced by the steganographic embedding. This line of work started in the machine learning community with constructions that modified the output distribution of the model, thus falling short of any notion of provable security [Bal17; HWJ+18; Har18; SAZ+18; Cha19; WYL18]. Building on these works, Kaptchuk et al. [KJG+21] and Ding et al. [DCW+23] showed how to encode steganography messages into neural network output without modifying the output distribution. Kaptchuk et al. [KJG+21]

accomplish this by repeatedly re-encrypting the message using a stream cipher, using the resulting pseudo-random ciphertext bits to sample tokens from the neural network's probability distribution, and leveraging an arithmetic encoding scheme to enable a receiver to recover the message bits. Ding et al. [DCW+23] are able to achieve a better encoding rate by using the message bits to "rotate" the neural network's probability distribution before sampling. As we discuss in the next section, the techniques presented in these works cannot be adapted to work with diffusion models, as they make implicit assumptions about model architecture that do not hold for diffusion models.

Image steganography. Most often, image steganography is associated with techniques that hide information in the least significant bits (LSB) of an image [Aur96]. This approach is easy to deploy [L] and has a high encoding rate. Unfortunately, image steganography schemes can be broken with relatively simple statistical analyses (e.g., [FGD01; FPK07; SCC07]), rendering it insecure in practice.

Machine learning has also been studied for image steganography. Earlier works [Bal19; LWZ+21; JDX+21; XMH+22] create steganography-specific models take an existing cover image, and use it to hide a secret image. This approach intrinsically leaves detectable artifacts in the output (i.e., as compared to the cover image), meaning that they are only a marginal improvement over an LSB-based approach. Ding, Chen, Wang, Zhao, Zhang, and Yu adapt their work to run on Image GPT, an image generation model similar in structure to large language models for text [DCW+23]. Image GPT does not have the same image quality as diffusion models, however, and their implementation is too slow to be practical.

More recent works [YZX+24; KSC+23; LCG23; PHW+23] have attempted to build image steganography from diffusion models, an approach taken by this work. These efforts use a non-black box view of the underlying diffusion model, using specific properties of the model under consideration to provide steganography. Yu, Zhang, Xu, and Zhang [YZX+24], Kim, Shin, Choi, Jung, and Yoon [KSC+23], and Liu, Chen, and Gu [LCG23] do not provide cryptographic guarantees, opting to instead provide heuristic analyses based on purpose-built steganalysis models; Peng, Hu, Wang, Chen, Pei, and Zhang [PHW+23] provide an information theoretic proof of security. We discuss the details and shortcomings of these approaches below.

3 Diffusion Models & Steganography

We start by systematically exploring the opportunities that diffusion models provide for steganographic encoding, contextualizing our study around a motivating deployment.

3.1 Diffusion Models

Diffusion models [SWM+15; SME20; HJA20; DN21; RBL+22] are a novel generative model architecture tailored to produce multimedia. Rather than generate output one token at a time, as is common in large language models, diffusion models take in a random seed (and possibly a prompt) and produce an entire image (or audio file, etc.) at once. The model predicts the changes that would need to be applied to this seed to make the image closer to the desired distribution, e.g., photo-realistic images or fantasy art. We provide a visual representation of the diffusion model generation process in Figure 1.

Diffusion models are trained on a large corpus of training examples. Each example is a pair of images $(\mathsf{img}, \mathsf{img} + \mathcal{N}(0, I))$, where img is an image from the desired output distribution and $\mathsf{img} + \mathcal{N}(0, I)$ is a modified version of img with Gaussian noise added. Many examples are created from each img, with multiple values of I; the most extreme examples appear to be pure noise while others are close to the output image. Given these examples (and the value of I), the model is trained to predict a value from $\mathcal{N}(0, I)$ that can be subtracted from the example to recover img.

During image generation (Algorithm 1), the pipeline is reversed: a seed s_0 is sampled from a Gaussian distribution, and the model predicts a *noise residual* pred such that $s_0 - \mathsf{pred}$ is in the target distribution. Rather than remove pred all at once, the model takes pred as an indication of the *direction* in which it must modify the s_0 to get it to the desired distribution. As such, the model subtracts a function of pred rather than pred itself. For example, the models that we work with compute $s_1 = s_0 - \epsilon \cdot \mathsf{pred}$, for some $0 < \epsilon < 1$. This process is repeated a fixed number of times to produce the values $s_2, \ldots, s_{\mathsf{final}}$; recall that the model

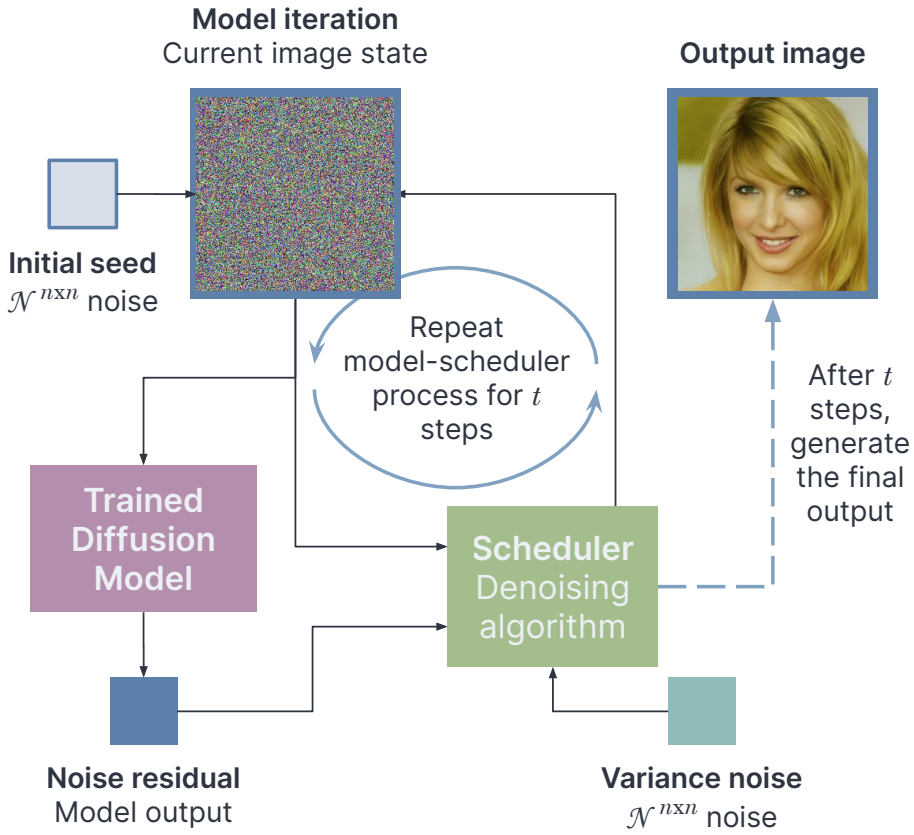


Figure 1: An overview of inference in a diffusion model. The model-scheduler feedback loop continues for all t steps of the scheduler, after which the final image is output.

was trained on many levels of noised images and can predict the noise that should be removed from $s_1, s_2, \ldots$ even though they are "less noisy" than s_0.

This iterative process is called *denoising* [SME20; HJA20] and is organized according to a *schedule*, which we denote with a set of functions $\{\mathsf{Scheduler}_i\}_{i\in[t]}$. Each function determines *how* the model prediction should be applied to the state s_{i-1} to produce an updated state s_i (e.g., applying the ϵ scaling factor). Different concrete instantiations of diffusion models may incorporate different modifications into each step of the schedule. One common modification is to apply additional Gaussian noise, called *variance noise*, to s_i in each step $i < t$, ensuring that image generation is *non-deterministic*. The final image is then generated deterministically by $\mathsf{Scheduler}_t$.

In the above, we have described the diffusion process as though the seed s_0 and intermediary states $s_1, s_2, \ldots$ are all elements in the image space (i.e., if the model is trained to generate 256×256 color images, then s_i is also a 256×256 color image). In practice, the space of s_i can be different than the image space; some diffusion models (e.g. Stable Diffusion [RBL+22]) operate in a latent space, a compressed space that is more succinctly able to represent and capture the complexity of images. When operating in the latent space, the model adds a final mapping step that maps s_{final} into a final image using a variational autoencoder (VAE) [KW13].

Algorithm 1: Typical Diffusion Model Operation

Output: Generated Image img

Set $s_0 \xleftarrow{\$} \mathcal{N}^{n \times n}$

for $1 \leq i < t$ **do**

  $r_i \xleftarrow{\$} \mathcal{N}^{n \times n}$

  $\mathsf{pred}_i \leftarrow \mathsf{Model}(s_{i-1})$

  $s_i \leftarrow \mathsf{Scheduler}_i(s_{i-1}, \mathsf{pred}_i; r_i)$

`// Deterministic Final Schedule`

$\mathsf{pred}_t \leftarrow \mathsf{Model}(s_{t-1})$

$\mathsf{img} \leftarrow \mathsf{Scheduler}_{\mathsf{final}}(s_{t-1}, \mathsf{pred}_t)$

Output img

3.2 A Motivating Deployment: Dead Drops

A motivating application of steganographic images is the *digital dead drop,* a digital re-imagining of a physical practice. Physical dead drops are secret locations where a sender and a receiver can pass documents without having to interact face-to-face. This process is asynchronous by design: a whistle-blower passing information to a journalist, for example, by hiding a document cache in a public park. Analogously, a digital dead drop allows these parties to pass information over the Internet instead. Implementations of digital dead drops exist [Swa+13], but are typically based on a heuristically secure system like Tor [Tor]. If we build a *steganographic* dead drop instead, a sender and receiver can provably guarantee that a censor will not detect their communication based on content (although subtle choice impacting the metadata associated with the dead drop deployment still remain). Much like in the physical case, a steganographic dead drop can re-use a public platform for hidden communication to essentially "hide in plain sight".

In addition to two-way communication, the steganographic dead drop can also be used by a single individual for the import and export of sensitive data. Consider an authoritarian regime that decrypts devices at border crossings to identify threats to its power [ZJG22]. A journalist can encode sensitive source data to themselves and upload it to a public, innocuous part of the Internet, and decode it after crossing the border. After completing their assignment, they can encode and upload their article while still in the authoritarian country, and recover it once they return home. The journalist would have no trace of the content on their devices at the border; all they would to remember the drop location and the secret key, which can be memorized by using a seed phrase.

Design. A steganographic dead drop scheme is relatively simple. Prior to the protocol, the sender and receiver share any necessary key material. The sender and receiver also agree upon a drop location (e.g., a website or forum), and a machine learning model that generates outputs suitable to the location. When the sender wishes to send some data to the receiver, the sender steganographically encodes it, and uploads the result to the drop location. The receiver periodically checks the drop location, and when they detect new content, they download it from the location. The receiver then decodes the content to recover the original data. The censor sees only some machine learning content on a public forum, and cannot distinguish a stegotext image from any other posts on the forum. The process remains asynchronous – the receiver waits for a "drop".

Implications. While the design is simple, a steganographic dead drop dictates interesting requirements of the underlying scheme:

1. *The drop location impacts the choice of model.* We must encode information into the same distribution expected by our drop location. Schemes like Discop [DCW+23] or Meteor [KJG+21] generate text, so a dead drop would be possible anywhere text is expected on the Internet. But, these schemes may need to generate hundreds of words to encode even a 100 B message, and large blocks of text are not very common on popular social media platforms – drawing censor scrutiny. On the other hand, public image communities like /r/aiArt/ [Red] or Civitai [Civ] have hundreds of thousands of users, allowing steganographic drops to hide among regular images. But we must generate *images* for these communities.

2. *We cannot assume any control over the drop location.* Ideal drop locations are social media sites and public forums, which means that we cannot expect changes to the overall platform to better support steganography—the censor may deem any platform changes suspicious. So, the images generated by the scheme should be robust to the format conversion or compression to meet the platform's requirements: typically, JPEG or PNG.

3. *We must minimize any requirements on the user.* The target audience for such a deployment are non-experts, and we cannot assume they have specialized hardware. So, any solution must be efficient on consumer hardware, and not require re-training of a machine learning model. We should also prefer the latest model innovations to speed up image generation.

3.3 Integrating Steganography

Given the structure of diffusion models, we can now turn to the task of studying the steganographic opportunities that it affords.

Why existing techniques fall short. A natural approach to steganography for diffusion models would be to extract the intuition behind steganographic approaches designed for other machine learning model architectures and adapt it for this new architecture. For example, Meteor [KJG+21] and Discop [DCW+23] are steganographic approaches for transformer-like architectures, which are the leading models for text generation. Both follow the same template: given some initial prompt (representing the context in which the encoded message will be sent) the sender uses the machine learning model to produce an explicit probability distribution over the token (e.g., a word) to be appended to the prompt. During typical generation, a model would select a token from this probability distribution at random. When encoding a message steganographically, both constructions instead sample the token as a function of the message.[4] Importantly, the receiver can use knowledge of the probability distribution and the selected token to efficiently recover a few bits of the message (in both cases, the receiver can recover a variable number of bits). The sender repeats this process, appending samples to the prompt until the entire message is encoded.

Critical to the approaches of Meteor and Discop is the assumption that the receiver gets access to the results of many (conditionally linked) sampling events. Although it is theoretically possible for a token to encode a large number of bits, the chances of this happening become vanishingly small as the number of bits increases. The throughput of the steganographic encoding scheme is tightly linked to the entropy in the channel: the expected number of bits to encode cannot exceed the instantaneous entropy in the model output distribution. For example, if a message is 128 bits long, it could only be encoded into a single token if the chances of choosing that token were 2^{-128} (assuming the sender and receiver share no prior information about the message distribution).

The structure required by Meteor and Discop is not present in diffusion models. Specifically, diffusion models do not produce explicit probability distributions from which the image is sampled. Moreover, the entire output that is accessible to the receiver is produced in a single shot, making subdivision (e.g., encoding into each pixel independently) impossible. Even adapting more classical steganographic schemes to diffusion models seems challenging. For example, foundational steganographic constructions (e.g. [Cac00; HLv02; vH04]) rely on the use of universal hash functions to subdivide the output space into segments that correspond to different bit sequences and then use rejection sampling to find a sample output that hashes to the desired message. While it is possible to use such a technique with diffusion models, the entire message would need to be encoded into the singular output image. Even though an image might be high entropy enough to encode a large number of bits, using rejection sampling to find an image that hashes to the desired bits is computationally infeasible.

Recently proposed techniques [YZX+24; KSC+23; PHW+23; LCG23] for diffusion model steganography also do not meet the requirements for a general steganographic deployment. The most promising of these techniques is StegaDDPM [PHW+23], a work developed concurrently to our own. The authors use specific

[4] Meteor encrypts the message first, to ensure the message is uniformly distributed. Discop instead uses an information theoretically secure approach to selecting the token, and thus does not require encryption.

properties of DDPM [HJA20] diffusion models— namely, their Gaussian noise residuals—to encode and decode information, achieving a very high encoding rate. While this acheives impressive results, exploiting low-level details of the model in this way prevents steganographic techniques from being lifted to new models as they are released; the proof of security is based on the model's attributes, and therefore does not immediately translate to other techniques. New techniques improve image quality and generation speed: for example, the DDIM scheduler [SME20] is faster than DDPM, which would improve encoding speed. Moreover, the fine details of generated images (i.e., in 32-bit model outputs) may be lost due to image file type or compression (i.e., 16-bit PNG pixels) on an image platform. This loss would impact decoding success rates, but adequate recovery mechanisms are not considered. So, while non-black box use of the model can yield strong encoding performance, we desire a flexible solution that meets deployment considerations and can easily adapt as models develop. As such, we must take a higher-level, general view to develop steganography for this rapdily developing field.

Opportunities for steganography. There are numerous techniques that can be used to construct steganography, but all of them rely on embedding the message into channel entropy. Because the space of messages that can be sent over a particular communication channel is fixed by the distribution within which the encoded message is supposed to hide, the *choice* of which message to send is the only subliminal channel available. Therefore, when evaluating the opportunities for steganographically embedding into the output of diffusion models, we begin by examining the sources of entropy.

When generating images, diffusion models use two sources of entropy: (1) the initialization seed, and (2) the noise added during each step of the schedule. To understand the viability of using these sources, we must understand how different randomness changes the image seen by the receiver. We study this question *empirically*, as randomness is largely a means to an end within these models.

1. *Initialization seed.* It is tempting to try to embed the message into the model's initialization seed. Because this seed is both large and high entropy, it is natural to assume that we could steganographically embed a large amount of information into the seed. Unfortunately the image generation process is not invertible. In our experiments with real-world diffusion models, we found that each step of the scheduler was lossy and not reversible. Even if only some information was lost in each step of the scheduler, most concrete instantiations of diffusion models have at least 50 steps, each one of which is lossy. As such, it is unclear how to steganographically embed information into the seed while allowing it to be recoverable.

2. *Variance noise added by scheduler.* Each step of the schedule adds a small amount of Gaussian noise to the current state s_i. In our experiments, we found that modifying the noise added to a single pixel in state s_i can result in a small, local change in the equivalent pixel in s_{i+1}. When operating in the latent space, the decompression results in changes not only in the equivalent point in the latent space, but also to surrounding points; this effect is amplified when the resulting latent space is mapped into the image space.

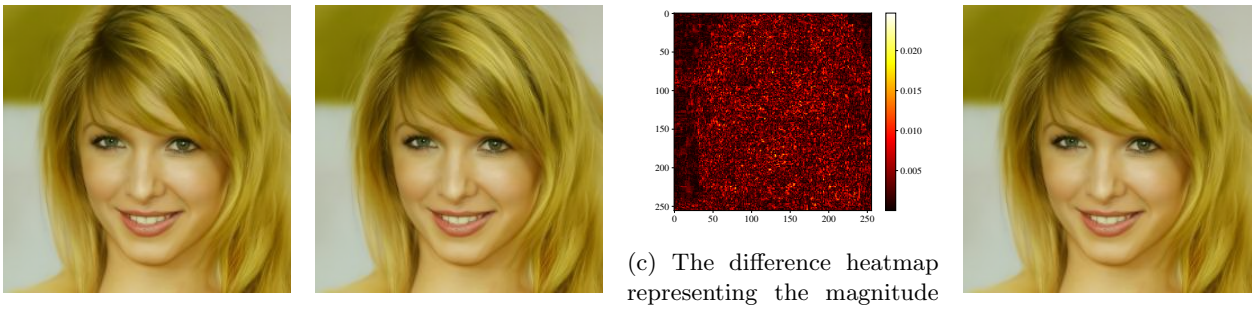
The localized nature of (2) provides an opportunity for an efficient steganographic channel, which we investigate below.

4 The Pulsar Steganography Scheme

We now describe Pulsar, a symmetric-key steganographic encoding scheme for pixel-space diffusion models.

4.1 Intuition

Pulsar leverages the steganographic channel enabled by the addition of Gaussian noise in the schedule. As discussed above, resampling the noise for a pixel in the final round of the schedule can result in a localized change to the output image. When this resampling is a function of the message, the receiver can observe these changes and recover the message.



(a) A Pulsar reference image img_0.

(b) A Pulsar reference image img_1.

(c) The difference heatmap representing the magnitude of the differences between Figures 2a and 2b.

(d) A Pulsar image which encodes 571 bytes.

Figure 2: Sample output from the `celebahq` model. Note that the differences shown in (c) are not changes introduced to encode a message, but a reflection of the entropy in the generative model that we exploit to embed steganographically.

The sender and receiver share key material that allows them to derandomize their model operation such that they can keep their models synchronized. Notably, this requires that the sender and receiver *start* their image generation with the same randomness. This allows the sender and receiver to have their models generate the same exact image. We divide this key material into three concrete PRG keys: a *seed* key k_s, and then a pair of *reference image* keys k_0 and k_1. The seed key is used by both the sender and receiver to synchronize their models until the final step of the scheduler, i.e. they use k_s to sample the model's initial seed and sample the variance noise in the first $t-1$ scheduler iterations. The sender and receiver can then generate two reference images $\mathsf{img}_0, \mathsf{img}_1$ by sampling the variance noise for the t^{th} iteration of the sampler with k_0 and k_1 respectively. Figures 2a and 2b are concrete reference images img_0 and img_1 generated by this process.

To build intuition, we make the following simplifying assumptions: (1) the model's state is simply the pixels of the image itself, and (2) the final iteration of the scheduler operates on each element of the state independently, i.e., each element of the final state is a function of exactly the corresponding element in the previous state – neighboring elements have no effect on one another. If these assumptions were true, a clear steganographic channel would emerge. The sender would divide the message m they want to send into bits $m_0, m_1, \ldots$ and then sample variance noise for each element of the state from k_0 or k_1 depending on the bit value of the message. For example, if message bit m_i were b, then the sender samples the variance noise for the i^{th} element using k_b. The receiver can then *guess* about the value of m_i by comparing the final output of the model to img_0 and img_1; whichever reference image the final output more closely resembles in the i^{th} pixel determine the receiver's guess about m_i. This would encode a single bit per pixel.

These simplifying assumptions serve a unified purpose: minimizing the error rate in the channel. Indeed, even without these assumptions, there may *still* be errors, as the reference images may be identical in particular pixels. For example, see Figure 2c, in which we give a heatmap of the difference between the two example reference images. Black pixels in this heatmap show that resampling the Gaussian noise has negligible effect on the final pixel value. Without assumption (2), sampling variance noise from different sources might have unpredictable effects, as the changes to one particular pixel might "contaminate" the signal in neighboring pixels. Finally, if the state does not have a clean correspondence to the image, it may not be clear where the receiver should look to recover information about how variance noise is sampled.

To recover from these errors, we introduce the use of a *binary error correcting code*, which introduces redundancy into a message. When the message is transmitted over a noisy channel (i.e., one that introduces bit flips), the receiver can run some recovery algorithm and output the initial message. Careful use of this error correcting code allows us to continue using the same intuition above, but ensures that the message can be recovered from the receiver, even without our simplifying assumptions.

We begin by removing assumption (2) and accept that changes in the way variance of noise is sampled for

a particular pixel may impact neighboring pixels—or, indeed, any pixel; we simply treat this as additional error to be corrected. The sender and receiver synchronize their models as before, reaching the final iteration of the scheduler. Before attempting to encode the message, the sender first encodes the message with a binary error correcting code, and then proceeds as before. Only one question remains: what *rate* should the sender use for their error correcting code, i.e., how *much* redundancy should the sender add into the message. If the error rate is high, the sender needs to increase the redundancy, as more information will be erased by the channel. On the other hand, adding unnecessary redundancy is wasteful, as fewer message bits will fit in the fixed capacity of the image.

To fix the rate of the error correcting code, the sender *estimates* the error rate. While it might be possible to use a fixed error rate, our experiments found that different images have very different error rates that are dependent on the structure of the image (discussed more in Section 5). As such, we instead have the sender estimate the error rate for each generated image by encoding random messages into the last iteration of the scheduler and measuring the number of errors by attempting to recover the encoded message. This process allows the sender to get a good estimate on the error rate, and then use conservative parameters on the error correcting code to ensure recovering the message is possible with high probability.

Relaxing assumption (1) appears to be more challenging. As discussed above, many concrete instantiations of diffusion models (e.g., Stable Diffusion) operate on a *latent space*, a compressed representation of the pixel space, in order to be more efficient. This design choice necessitates the use of an expanding mapping between the latent space and the pixel space. While we find that the approach outlined above can function for such models, the result is highly inefficient. As such, we restrict attention to diffusion models that operate directly in the pixel space. We included a more detailed discussion of diffusion models in the latent space in Section 7.

4.2 Formal Definitions

We lift our notation, formalization, and descriptions of a symmetric stenographic scheme largely from [KJG+21]. A scheme $\Sigma_{\mathcal{D}}$ is a triple of possibly probabilistic algorithms, $\Sigma_{\mathcal{D}} = (\mathsf{KeyGen}_{\mathcal{D}}, \mathsf{Encode}_{\mathcal{D}}, \mathsf{Decode}_{\mathcal{D}})$ parameterized by a covertext channel distribution $\mathcal{D}$.

- $\mathsf{KeyGen}_{\mathcal{D}}(1^\lambda)$ takes arbitrary input with length λ and generates k, the key material used for the other two functionalities.
- $\mathsf{Encode}_{\mathcal{D}}(\mathsf{k}, m, H)$ is a (possibly probabilistic) algorithm that takes a key k and a plaintext message m. Additionally, the algorithm can optionally take in a message history H, which is an ordered set of covertext messages $H = \{h_0, h_1, \ldots, h_{|H|-1}\}$, presumably that have been sent over the channel. Encode returns a stegotext message composed of $c_i \in \mathcal{D}$.
- $\mathsf{Decode}_{\mathcal{D}}(\mathsf{k}, c, H)$ is a (possibly probabilistic) algorithm that takes as input a key k and a stegotext message c and an optional ordered set of covertext messages H. Decode returns a plaintext message m on success or the empty string ε on failure.

Correctness. To be correct, we require that a scheme $\Sigma_{\mathcal{D}} = (\mathsf{KeyGen}_{\mathcal{D}}, \mathsf{Encode}_{\mathcal{D}}, \mathsf{Decode}_{\mathcal{D}})$ encoding and then decoding on a message m should recover the message m with overwhelming probability. Formally, for all messages m, histories H, and $\mathsf{k} \leftarrow \mathsf{KeyGen}_{\mathcal{D}}(1^\lambda)$,

$$\mathsf{Pr}[\mathsf{Decode}_{\mathcal{D}}(\mathsf{k}, \mathsf{Encode}_{\mathcal{D}}(\mathsf{k}, m, H), H) = m] \geq 1 - \mathsf{negl}(\lambda).$$

Security. As in [KJG+21], we adopt a symmetric-key analog of the security definitions for a steganographic system secure against a chosen hiddentext attacks in [vH04]. This is reminiscent of the real-or-random games used to formalize pseudorandom generators and functions. More formally, we say that a steganographic scheme $\Sigma_{\mathcal{D}}$ is secure against *chosen hiddentext attacks* if for all ppt. adversaries $\mathcal{A}_{\mathcal{D}}$, $k \leftarrow \mathsf{KeyGen}_{\mathcal{D}}(1^\lambda)$,

$$\left|\mathsf{Pr}[\mathcal{A}_{\mathcal{D}}^{\mathsf{Encode}_{\mathcal{D}}(\mathsf{k},\cdot,\cdot))} = 1] - \mathsf{Pr}[\mathcal{A}_{\mathcal{D}}^{O_{\mathcal{D}}(\cdot,\cdot)} = 1]\right| < \mathsf{negl}(\lambda)$$

where $O_{\mathcal{D}}(\cdot,\cdot)$ is an oracle that randomly samples from the $\mathcal{D}$.

Within the context of diffusion models, the history could be the prompt, when the model take a prompt as input, or can be left empty when no prompt is used. The models with which we experiment in Section 6 do not use prompts. We include history within our formal notation for generality, but omit it throughout the rest of the write-up of our algorithms below for readability.

4.3 Pulsar Description

Notation. We assume that the sender and receiver (and adversary) both have access to the same diffusion model that operates in the pixel space and generates $n \times n$ pixel images[5] with 3 color channels, resulting in a image space of $[0,255]^{n\times n\times 3}$.

Let $\mathcal{N}(0,I)$ be a Gaussian, and $\mathcal{N}^{n\times n}(0,I)$ be $n \times n$ copies of the underlying Gaussian. We denote a *keyed* Gaussian as $\mathcal{N}_k(0,I)$, in that samples drawn from the Gaussian are drawn using entropy generated by a pseudorandom generator [BM82; Ruh17] with key k. As the norm and standard deviation of the distribution are fixed model parameters, we omit them. We abuse notation and write $s+\mathcal{N}$ to denote sampling a value from $\mathcal{N}$ before adding it to s.

We index into images and messages using square bracket notation, i.e., $x[i]$ is the i^{th} element x. For images, we assume some arbitrary pixel ordering to facilitate single dimension indexing.

We assume that diffusion models have a set of schedulers for each time step $\{\mathsf{Scheduler}_i\}_{i\in[t]} : [0,255]^{n\times n\times 3}\times [0,255]^{n\times n\times 3} \times [0,255]^{n\times n\times 3} \to [0,255]^{n\times n\times 3}$. The scheduler at step i accepts the sample at step $i-1$, the diffusion model's prediction at step i, and some randomness, all in the size of the model.

Let ECC be an error correcting code made up of two algorithms: and encoding algorithm $\mathsf{ECC.Encode}$ and a recovery algorithm $\mathsf{ECC.Recover}$. Both take in a message and an error rate. We expand on error correcting codes and our use of them in Section 5.

Pulsar encoding and decoding. The encode and decode algorithms for Pulsar are described in Algorithm 2 and Algorithm 3 respectively. When encoding, the sender runs the typical image generation algorithm for the first $t-2$ iterations. The sender then estimates the error rate for encoding and adds redundancy to the message using an error correcting code ECC. Then, based on the bits of the expanded message m_{ECC}, the sender samples variance noise either using k_0 or k_1. Then, the final image is generated using $\mathsf{Scheduler}_t$. When decoding, the receiver estimates the error rate in the image and generates reference images img_0 and img_1. In order to recover m_{ECC}, the receiver compares each pixel to the reference images, and guesses the corresponding bit of m_{ECC} accordingly. m can then be recovered by applying $\mathsf{ECC.Recover}$.

Offline-online paradigm. We note that much of the logic of Pulsar can be run before the sender knows the message. Specifically, the first $t-2$ iterations of the schedule are completely independent of the message. As such, this computation can be run *offline*, and the results can be stored until the sender has a message they want to encode. At that time, the sender can run the *online* parts of Pulsar, which include ECC, sampling the variance noise r_{t-1}, and final image generation. We highlight the offline and online phases of encoding and decoding in Algorithm 2 and Algorithm 3.

Proof. We provide a proof that Pulsar is secure against chosen hiddentext attacks (see Section 4.2) in Appendix E.

Randomness synchronization. Pulsar assumes that the sender and receiver can *start* their image generation process with the same randomness. While the sender and receiver are able to do this because they share key information, it requires synchronization. For example, the sender and the receiver may run the offline phase of hundreds of images before transmission, each with different randomness (i.e, imagine the key k in Algorithm 2 and Algorithm 3 are each sampled from a single pseudorandom function queried on sequential inputs). The receiver may detect images out of order (as is likely to happen in the dead-drop deployment scenario discussed in Section 3.2), and thus will need to figure out which per-image randomness key material to use in the decoding process. Thankfully, synchronizing in this way is relatively simple, as

[5] We assume square images for simplicity, but other dimensions are trivially supported.

Algorithm 2: Pulsar Encode

Input: Plaintext Message m, Key k
Output: Stegotext Image img

Parse $(k_s, k_0, k_1) \leftarrow k$ // Offline Phase
Set $s_0 \xleftarrow{\$} \mathcal{N}_{k_s}^{n \times n}$
for $1 \leq i < t-1$ **do**
  $r_i \xleftarrow{\$} \mathcal{N}_{k_s}^{n \times n}$
  $\mathsf{pred}_i \leftarrow \mathsf{Model}(s_{i-1})$
  $s_i \leftarrow \mathsf{Scheduler}_i(s_{i-1}, \mathsf{pred}_i; r_i)$
Compute rate $\leftarrow \mathsf{EstimateRate}(s_{t-2})$
$m_{\mathsf{ECC}} \leftarrow \mathsf{ECC.Encode}(m, \mathsf{rate})$ // Online Phase
for $0 \leq j < |m_{\mathsf{ECC}}|$ **do**
  if $m_{\mathsf{ECC}}[j] = 0$ **then**
    $r_{t-1}[j] \xleftarrow{\$} \mathcal{N}_{k_0}$
  else
    $r_{t-1}[j] \xleftarrow{\$} \mathcal{N}_{k_1}$
$\mathsf{pred}_{t-1} \leftarrow \mathsf{Model}(s_{t-2})$
$s_{t-1} \leftarrow \mathsf{Scheduler}_{t-1}(s_{t-2}, \mathsf{pred}_{t-1}; r_{t-1})$
// Deterministic Final Schedule
$\mathsf{pred}_t \leftarrow \mathsf{Model}(s_{t-1})$
$\mathsf{img} \leftarrow \mathsf{Scheduler}_t(s_{t-1}, \mathsf{pred}_t)$
Output img

the images bearing encoded messages are perceptually very similar to the ones generated by the receiver in the offline phase. Thus, the receiver can maintain a local map of the images created in the offline phase to the key material used to generate those images. To do a lookup on this map, the receiver simply finds the image in the domain of the map that is most similar (e.g. using a perceptual hash function) to the newly encountered image.

Latent diffusion. Pulsar is designed for *pixel diffusion* models, where the internal states of the model s_i are the same size as the output image. As mentioned in Section 3.1, another type of models uses a smaller latent space to represent s_i. These *latent diffusion* models have seen increased popularity, with Stable Diffusion [RBL+22] being the most famous. But, while Stable Diffusion has made latent diffusion approaches popular [Vin22], we note that pixel diffusion models are still useful, including Imagen [SCS+22], one of the state-of-the-art works in image synthesis. While there is nothing stopping Pulsar from being used in a latent setting, the model has some differences that impact throughput. We discuss the relationship between Pulsar and latent diffusion models further in Section 7.

5 Error Correction for Pulsar

Error correction allows Pulsar to function when image details are lost during transmission, so selecting the right code is critical for deployment. We now investigate how to build EstimateRate and select an ECC for Pulsar to maximize throughput.

5.1 Channel Error Structure

A naive approach to computing EstimateRate would simply have the sender generate a random message, encode it into an image, locally attempt to recover it and measure the decoding error rate. They would then select an ECC that can successfully encode at that error rate. For instance, an estimate for the error rate of Figure 2d is high (29.1%), but we can build a code for this high error rate.

Algorithm 3: Pulsar Decode

Input: Stegotext Image img, Key k
Output: Plaintext Message m

Parse $(k_s, k_0, k_1) \leftarrow k$ `// Offline Phase`
Set $s_0 \xleftarrow{\$} \mathcal{N}_{k_s}^{n \times n}$
for $1 \leq i < t - 1$ **do**
  $r_i \xleftarrow{\$} \mathcal{N}_{k_s}^{n \times n}$
  $\mathsf{pred}_i \leftarrow \mathsf{Model}(s_{i-1})$
  $s_i \leftarrow \mathsf{Scheduler}_i(s_{i-1}, \mathsf{pred}_i; r_i)$
Compute $\mathsf{rate} \leftarrow \mathsf{EstimateRate}(s_{t-2})$
$\mathsf{pred}_{t-1} \leftarrow \mathsf{Model}(s_{t-2})$
`// Generate reference image 0`
Set $r_{t-1}^0 \xleftarrow{\$} \mathcal{N}_{k_0}^{n \times n}$
Set $s_{t-1}^0 \leftarrow \mathsf{Scheduler}_{t-1}(s_{t-2}, \mathsf{pred}_{t-1}; r_{t-1}^0)$
$\mathsf{pred}_t^0 \leftarrow \mathsf{Model}(s_{t-1}^0)$
$\mathsf{img}_0 \leftarrow \mathsf{Scheduler}_t(s_{t-1}^0, \mathsf{pred}_t^0)$
`// Generate reference image 1`
$r_{t-1}^1 \xleftarrow{\$} \mathcal{N}_{k_1}^{n \times n}$
Set $s_{t-1}^1 \leftarrow \mathsf{Scheduler}_{t-1}(s_{t-2}, \mathsf{pred}_{t-1}; r_{t-1}^1)$
$\mathsf{pred}_t^1 \leftarrow \mathsf{Model}(s_{t-1}^1)$
$\mathsf{img}_1 \leftarrow \mathsf{Scheduler}_t(s_{t-1}^1, \mathsf{pred}_t^1)$
for $0 \leq j < |m_{\mathsf{ECC}}|$ **do** `// Online Phase`
  if $|\mathsf{img}[j] - \mathsf{img}_0[j]| < |\mathsf{img}[j] - \mathsf{img}_1[j]|$ **then**
    $m_{\mathsf{ECC}}[j] \leftarrow 0$
  else
    $m_{\mathsf{ECC}}[j] \leftarrow 1$
Output $m \leftarrow \mathsf{ECC.Recover}(m_{\mathsf{ECC}}, \mathsf{rate})$

To improve on this baseline, we investigate the sources of error a little more closely. An error occurs in Pulsar when, for a bit b, the difference in the corresponding pixel between the encoded img and $\mathsf{img}_{b'}$ is closer than that of img and img_b, where $b \neq b'$. There is, therefore, a inverse relationship between the absolute *magnitude* of differences between the references images img_0 and img_1 and the error rate: roughly speaking, the greater the difference for a pixel, the less likely decoding that pixel will result in an error. This naive approach assumes that the distribution of errors is uniform.

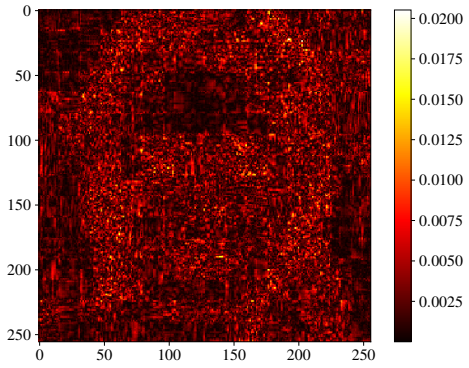
A closer look at the heatmap in Figure 2c reveals that the differences between img_0 and img_1 are not uniformly distributed in the image—there is structure. The heatmap is dark for the regions of the image that make up the background in Figure 2d, creating a semi-visible silhouette of the face. In this significant background region, the differences between img_0 and img_1 are consistently low. Because Pulsar leverages differences in order to steganographically embed, the background will contribute more to the error rate than the rest of the image. Put another way, any single bit encoded in this part of the image will have a much higher error rate.

This notion of "difference regions" with varying magnitudes is more obvious in Figure 3a and the associated heatmap Figure 3b. In addition to the background, parts of the generated face are also less discernible, and the estimated overall error rate is 34.1%: over one in three bits is incorrectly decoded. Not only are there distinct difference regions, but they also differ from image to image.

5.2 Variable Error Correction

The naive approach above fails to fully utilize the high entropy present in some regions of the image. The model is more detailed in some areas, leading to high entropy (and low error). In other areas, the model is less detailed, and so the available entropy is lower. These regions are essentially *guaranteed* to introduce

(a) A Pulsar image which encodes 257 bytes.



(b) The difference heatmap for the img_0 and img_1 used to create Figure 3a.

Figure 3: Another sample output for the `celebahq` model.

high amounts of error, diluting the overall error rate. Unlike many applications of error correcting codes, in Pulsar we actually *can* estimate where the errors will happen; we can divide the communication channel into smaller channels, each of which has a predictable error rate.

An obvious solution would be to leverage the sender's and receiver's shared knowledge of the image structure to *only* encode into those regions that contain low error rates. This would entail selecting a code for low error regimes and using that on these regions. This approach would work, but is also inefficient. The number of low error regions is image-dependent; an image like Figure 2d has many more than an image like Figure 3a. Additionally, this approach excludes "medium" error rate regions, i.e., where the error rate is between low and high. These regions will have relatively higher error rates, but could still encode *some* amount of data.

So, what we need is a *library* of error correcting codes, selecting the best code for a region based on its error rate. Intuitively, if we can define these difference regions, we can estimate the error rates within each region separately, and then select error correcting codes per-region instead of over the image overall. This approach helps us more efficiently use the low error rate regions for encoding.

Our region-based error rate estimation for EstimateRate proceeds as follows. First, we use the naive error estimate strategy on each difference region of the image individually. To do so, we compute l trial encodings at the current model state on random messages. These trail encodings are then subsequently decoded to generate difference heatmaps (like those seen in Figures 2c and 3b). To systematically define the difference regions from these heatmaps, we divide the pixels in the image into u buckets based on the magnitude of the difference shown in the heatmap. Each bucket represents one unique difference region of the image—all of the pixels in a bucket have a similar magnitude of their differences and therefore (roughly) similar error rates during Pulsar encoding. We chose to bucketize the differences rather than apply image analysis (e.g., convolution to find region boundaries) because the latter would involve more computation and may miss non-contiguous regions.

We calculate the error rate for each difference regions, as defined by its bucket, based on the messages from the trial encoding. rate is now a list of regions with associated error rates, we select the appropriate ECC for each region. To do so, we start with the lowest error regions, and use codes suitable for those regions. We then work our way up the difference regions, selecting a code from our library that works at each region's calculated error rate. Eventually, we reach a region that cannot be encoded using our codes due to its significant noise, and we consider encoding finished.

Using this approach, Figure 2d can encode 571 bytes of information, and Figure 3a can encode 257 bytes of information. We also note that EstimateRate is still a fully offline operation (Section 4.3), as the difference regions for an image can be estimated without any information about the message to encode. Both the

sender and receiver can perform multiple runs of EstimateRate in preparation for a communication. Also note that, while EstimateRate is randomized, we can use the key shared between the sender and receiver to generate any randomness used by EstimateRate. This ensures that the sender and receiver agree on the encoding and decoding algorithms without explicitly sharing this information.

A more formal description of EstimateRate can be found in Appendix A. Our EstimateRate function has two parameters: the number of buckets for each estimate u, and the number of estimates l. We experimentally determine these parameters in Appendix D.

5.3 Identifying Candidate Codes

Our variable error correction approach requires building a library of error correcting codes that support different error rates, so more efficient codes can be used in difference regions with low error. We approach the development of this library *heuristically*, as developing optimal codes for our setting is beyond the scope of this work. There may be significant room for Pulsar performance improvement by identifying other superior codes.

Error correcting codes. In this work we will consider linear $[N, k, d]_q$ codes, which are codes defined over a field $\mathbb{F}_q$ of size q. The size of the codeword N is called the *block length*, k is the *dimension* and d is the *distance* of the code. The *rate* of a code is given by $\frac{k}{N}$ and the unique decoding radius (the number of errors a code can tolerate while still uniquely recovering the original codeword) is $\lfloor \frac{d}{2} \rfloor$. The channel we have identified earlier can be characterized as a binary symmetric channel (BSC_p), which is parameterized by the probability p that an error is introduced in each bit. More formally, if Y is the output of the channel and X is the input, $\Pr[Y = 1 - b | X = b] = p$ for $b \in \{0, 1\}$. The *capacity* for BSC_p—which is the highest possible rate we could hope to achieve—is $1 - H_2(p)$ where $H_2(x) = -(x \cdot \log_2(x) + (1 - x) \cdot \log_2(1 - x))$.

Error correcting for binary channels. When we quantize the image into u different difference regions, the result is that our communication channel is actually the concatenation of u channels $Ch_1, \ldots Ch_u$. We assume that each channel Ch_i functions as a binary symmetric channel BSC_{p_i} for some probability $p_i \in [0, 1]$. Our goal is to come up with an algorithm so that, given block lengths $N_1, \ldots N_u$, we identify u codes $\mathsf{ECC}_1, \ldots \mathsf{ECC}_u$ such that $\forall j \in [u]$ (1) ECC_j can recover from errors induced by a BSC_{p_i} channel with all but low probability, (2) ECC_j has as high a rate as possible, and (3) ECC_j has a practically efficient encoder and decoder.

One approach to constructing binary codes is to make use of *concatenated codes* [For65]. A concatenated code $\mathsf{ECC}_{out} \circ \mathsf{ECC}_{in}$ encodes messages by first using the code ECC_{out} and then encoding each resulting symbol with ECC_{in}. To be more precise, let the *outer code* ECC_{out} be an $[N, k, d]_Q$ where $Q = q^{k'}$ for some $k' \in \mathbb{Z}^+$ and the *inner code* ECC_{in} an $[N', k', d']_q$ code. To encode a message $\vec{m} \in \mathbb{F}_q^k$, first produce $\bar{c} = (\bar{c}_1, \ldots \bar{c}_N) = \mathsf{ECC}_{out}.\mathsf{Encode}(\vec{m})$; the final codeword is $\mathsf{ECC}_{in}.\mathsf{Encode}(\bar{c}_1), \ldots \mathsf{ECC}_{in}.\mathsf{Encode}(\bar{c}_N)$. The resulting concatenated code is an $[NN', kk', z]$ linear code where $z \geq dd'$. For binary concatenated codes, $q = 2$.

It is well known that finding binary concatenated codes with rate approaching capacity over a BSC_p channel can be done by using a "small" inner code that has rate close to capacity and then applying an outer code to handle most of the error patterns that could arise from incorrect decoding of the inner code [GRS12]. Constructing the inner code is done by brute force and takes $O(2^{N'^2})$ time. The outer code may be either Reed-Solomon or another binary code leading to either super-polynomial or polynomial construction time, respectively. The resulting decoder for the concatenated code uses a maximum likelihood decoder for $\mathsf{ECC}_{in}.\mathsf{Recover}$ and then applies an efficient unique decoding algorithm for $\mathsf{ECC}_{out}.\mathsf{Recover}$. While this would produce high quality codes, it is not computationally feasible to find such an inner code, and finding "optimal" codes is beyond the scope of this work. After identifying simple codes via an ad-hoc approach, we designed a strategy for finding a reasonably good code for a given BSC_{p_i}. First, we manually search for a binary inner code with low decoding error probability and high capacity. Rather than fully testing the code, we empirically test its performance on the BSC_{p_i} for at least 40,000 inputs to get a loose decoding error probability. Once we manually identify the best such codes, we then choose the outer codes. Let the random variable X_ℓ be the number of decoding errors that occur when ℓ randomly chosen messages

Table 1: Average execution time in seconds for offline, encoding, and decoding steps on Desktop and Laptop for 100 runs.

Model	Desktop			Laptop		
	Offline	Encoding	Decoding	Offline	Encoding	Decoding
`church`	$\bar{x} = 2.96$, $s = 0.06$	$\bar{x} = 1.61$, $s = 0.10$	$\bar{x} = 4.11$, $s = 0.84$	$\bar{x} = 9.99$, $s = 0.23$	$\bar{x} = 2.97$, $s = 0.16$	$\bar{x} = 3.78$, $s = 0.49$
`celebahq`	$\bar{x} = 2.96$, $s = 0.01$	$\bar{x} = 1.57$, $s = 0.08$	$\bar{x} = 3.68$, $s = 0.58$	$\bar{x} = 9.98$, $s = 0.11$	$\bar{x} = 2.93$, $s = 0.13$	$\bar{x} = 3.53$, $s = 0.40$
`bedroom`	$\bar{x} = 2.96$, $s = 0.01$	$\bar{x} = 1.48$, $s = 0.11$	$\bar{x} = 2.93$, $s = 0.73$	$\bar{x} = 10.02$, $s = 0.11$	$\bar{x} = 2.88$, $s = 0.14$	$\bar{x} = 3.10$, $s = 0.44$
`cat`	$\bar{x} = 2.96$, $s = 0.01$	$\bar{x} = 1.52$, $s = 0.15$	$\bar{x} = 3.34$, $s = 1.21$	$\bar{x} = 10.00$, $s = 0.11$	$\bar{x} = 2.89$, $s = 0.17$	$\bar{x} = 3.33$, $s = 0.78$

Table 2: Bytes encoded per image for 100 runs.

Model	$\bar{x}$	s	Success	Throughput
`church`	613.74	200.02	93.0%	$E[X] = 570.78$
`celebahq`	437.73	98.37	89.0%	$E[X] = 389.58$
`bedroom`	320.07	133.33	93.0%	$E[X] = 297.66$
`cat`	439.94	301.88	97.0%	$E[X] = 426.74$

are encoded via $\mathsf{ECC}_{in}.\mathsf{Encode}$, sent over the channel BSC_{p_i}, and decoded. For a given outer code block length N, we find the minimum $z \in \{0, \ldots, N\}$ so that $\Pr(X_N \leq z) \geq 0.95$. We then set the outer code's distance and dimension to allow recovery of up to z errors. [6] See Appendix A for the codes we identified for Pulsar.

6 Implementation & Evaluation

We implemented Pulsar with PyTorch and `diffusers` [Hug] using the DDIM scheduler [SME20]. We use SageMath [The23] for error correcting codes. We implement HMAC-DRBG [BK15] to generate synchronized randomness., which we bootstrap into a keyed Gaussian $\mathcal{N}_k(0, I)$, for encoding using inverse transform sampling [BFS11]. We have released our code and benchmarks as artifacts[7] for the community.

Our implementation uses the following diffusion models (which generate the following images): `church` (churches and other places of worship), `celebahq` (celebrity faces), `bedroom` (bedrooms), `cat` (cats), all of which were published by Google on Hugging Face [Goo22d; Goo22c; Goo22a; Goo22b]. Example images from each model can be found in Appendix B.

We run our implementation on (1) a Jetstream2 `g3.medium` instance, with 8 vCPUs, 30 GiB of RAM, and access to 25% of a NVIDIA A100 GPU (i.e., 10 GiB of VRAM) running Ubuntu 22.04, and (2) a MacBook Pro with an M1 Pro system-on-chip with 16 GiB of RAM running macOS Ventura. We consider (1) representative of an entry-level AI workstation, so we call it "Desktop"; we believe (2) is representative of consumer hardware, and call it "Laptop".

We choose the DDIM scheduler [SME20] for our implementation over the DDPM scheduler [HJA20] used in prior work [PHW+23]. In our preliminary testing, standard (non-steganographic) image generation on Desktop using the DDPM scheduler took ≈ 50 seconds, while the newer DDIM scheduler took ≈ 3 seconds. We take advantage of DDIM's better efficiency for our solution, which we are able to do because of the flexibility of Pulsar. When better schedulers are created, it would be a matter of swapping out DDIM for the new option—the underlying scheme would remain the same.

There are two parameters in EstimateRate that we have yet to determine: the number of buckets to generate for our difference regions u, and the total number of estimates to generate l. We experimentally determine that $u = 100$ and $l = 1$ are appropriate parameters for our purposes. For details, see Appendix D.

We run 100 trials of our implemented Pulsar system from end to end for each of the models we consider on both Desktop and Laptop. In each trial, we run the offline step of EstimateRate, encode a random message

[6] This means setting $d \approx 2z$ and calculating k based on d and N.

[7] Our evaluated artifact is available on Zenodo: https://zenodo.org/doi/10.5281/zenodo.12785497. Active development and updates can be found on our GitHub: https://github.com/spacelab-ccny/pulsar.

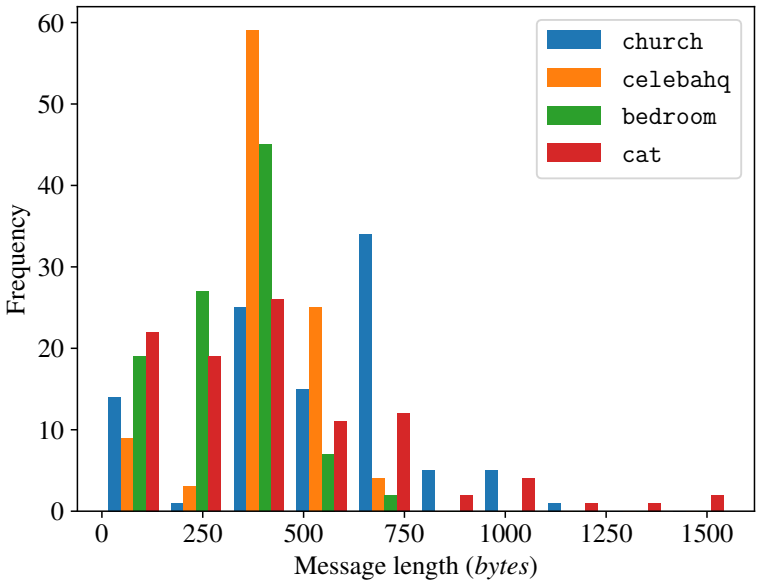


Figure 4: The distribution of message lengths in 100 images generated for each of the models used in Pulsar.

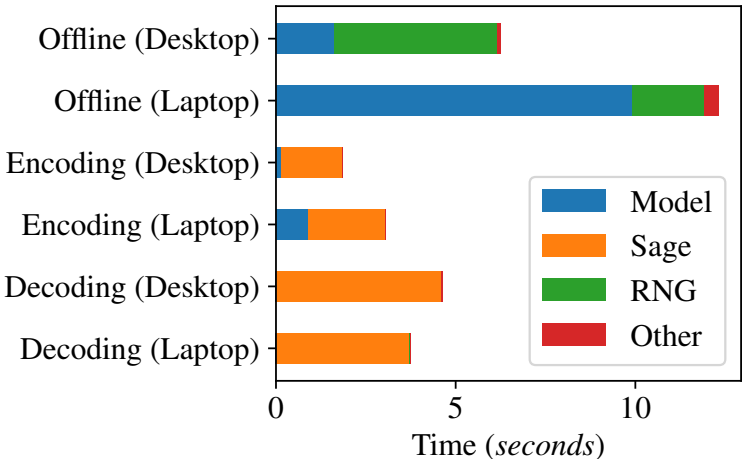


Figure 5: The mean time spent to perform Pulsar tasks for the `church` model on Desktop and Laptop.

at the sender into a 16-bit PNG image, and decode this image back into the original message. Note that we provide results for the offline step of Algorithm 3; since Algorithm 2's offline step is a subset of that of Algorithm 3, we can use the same offline state for both encoding and decoding.

Throughput. We first seek to understand throughput: the number of bytes per image that can be communicated steganographically with Pulsar. Table 2 summarizes the information, with the mean $\bar{x}$ and standard deviation s of the bytes encoded in each image, along with the percentage of encoded messages can be successfully decoded. As mentioned above, decoding can fail if the estimate is not representative of the actual message encoding. The expected throughput is therefore this decoding success probability times the mean bytes per image $\bar{x}$. While the success rate is not 100%, we do note that it is high enough to support throughputs in the hundreds of bytes. Moreover, because Pulsar is a symmetric key scheme, the sender will *know* when decoding will fail and can abort.

We also seek to understand how the number of bytes per image varies between the models. We chart this distribution for each model in Figure 4. Each model has a different distribution, which is reasonable considering that each model generates substantially different images. For example, note that `celebahq` is mostly tightly centered, likely because human faces have a core structure that does not deviate significantly. On the other hand, `cat` has high variance, which we attribute to the model's lower quality (see Appendix B).

Time. Table 1 contains runtime for each phase of Pulsar for each model on both Desktop and Laptop. The offline phase is about the same for every model, which is in line with how diffusion models work. Even though each model generates different images, all models have the same number of steps (and therefore PyTorch calls to their weights), and should have the same runtimes. We see that decoding is generally slower than encoding. We can also see that the runtime of Laptop is about three times that of Desktop, and that encoding and decoding times vary between the models. In particular, the encoding and decoding times are proportional to the number of bytes the model generates (see Table 2).

To investigate these results more, we separate each runtime by the constituent parts of Pulsar, and chart it in Figure 5. "Model" is time spent generating an image using diffusion in PyTorch, "Sage" is time spent in SageMath for error correction, "RNG" is time spent generating random numbers for encoding, and "Other" is time spent outside of the prior two tasks. The offline phase is dominated by the model and random number generation, as it is generating the estimate that will be used for the online phases. Laptop's slower runtime can be attributed to its weaker model processing ability.

Encoding and decoding are instead dominated by SageMath. So, models that require more calls to SageMath are slower in encoding and decoding; the models that call SageMath more are those with a higher capacity for encoding bytes. For example, the `church` model takes the most time to encode/decode because it needs to encode/decode more bytes. Thus, our results are consistent.

As an aside, our implementations of random number generation and ECCs are unoptimized and could be improved for a production deployment. Our HMAC-DRBG implementation is in pure Python, and a native implementation would reduce Pulsar runtime. Moreover, our ECC implementation performs a call to SageMath as a subprocess, which means our runtimes include a relatively lengthy interpreter start-up; a direct implementation of the underlying ECCs (without SageMath) can help further reduce Pulsar runtime.

Comparison to text model methods. We now compare our results to that of prior work on text-based steganography. We specifically compare Pulsar to [DCW+23], where the authors provide seconds per bit results for their construction, as well as those of other modern baselines such as Meteor [KJG+21]. Their hardware configuration is similar to our Desktop one. Based on the evaluation in Table II of [DCW+23], Pulsar represents a performance improvement over prior work. Using the numbers in Tables 1 and 2, the `church` model takes 1.01×10^{-3} seconds per bit to perform both the offline and encode steps, which is about twice as fast as the best performance metrics reported for text-based models. Even our *Laptop* configuration is relatively performant, with 2.84×10^{-3} seconds per bit, which is similar to Discop and Meteor on *Desktop*.

We do not claim that our solution is strictly better. Text-based solutions have high utilization of the available entropy in the channel, which means that it can efficiently send smaller messages in a few hundred words. Pulsar has much lower utilization due to the noisiness of the channel. Once the messages are in the hundreds of bytes, though, Discop and related work require several paragraphs of text to encode. Pulsar

instead sends an image, which, while strictly larger in file size, may be more acceptable in a channel than pages of text. So, we recommend text-based steganography for short messages, and image-based steganography for long messages.

Comparison to image model methods. We now compare Pulsar to two prior image model works, Liu, Chen, and Gu [LCG23] and StegaDDPM [PHW+23], based on the results presented in the respective works.

Liu, Chen, and Gu present a heuristic scheme, and are able to embed 50 bits per image into outputs of the `church` and `celebahq` models; Pulsar on the other hand is both secure and able to embed several hundred bytes of information per message (i.e., several thousand bits). Liu, Chen, and Gu do not provide runtime information in their work.

StegaDDPM is a secure scheme, and is actually able to encode several bits *per pixel*, resulting in images that have thousands of bytes of information. This impressive rate outperforms that of Pulsar by an order of magnitude. We note that this throughput is likely due to StegaDDPM's non-black box nature; Pulsar actively trades off a higher throughput for greater flexibility. The StegaDDPM work also does not provide execution time benchmarks. But, based on our testing mentioned above, our use of the DDIM scheduler means that Pulsar likely runs faster than StegaDDPM and its DDPM scheduler.

Steganalysis. Although Pulsar is a provably secure scheme (see Appendix E), we validate our results against two ML-based steganalysis techniques, YeNet [YNY17] and SRNet [BCF18]. We generate both Pulsar steganographic images and non-steganaographic images using the same models except without embedding. We convert both sets of images to grayscale, which these steganalysis tools require as input. Each model was then trained, using 5000 pairs for training and 1000 for validation. The models were then tested for accuracy in distinguishing between 5000 pairs of steganographic and non-steganaographic images. On this task, SRNet was 49% accurate, while YeNet was 50% accurate. Note that the ideal performance is near 50%: the ability of the tool to distinguish images is no better than a random guess. For details, see Appendix C.

Deployment. Pulsar's flexibility in model types, concrete efficiency on consumer hardware, and resistance to channel errors make it suitable for the dead drop deployment discussed in Section 3.2. As a step towards deployment, we validated that Pulsar could be used on common image platforms. We were successfully able to upload and decode a Pulsar image on Imgur, Reddit, and Twitter.

We believe these and other online communities are candidate drop locations; to truly establish this, though, we require user feedback. As future work, we propose a user study to identify locations and social impacts to these communities. This study would determine where dead drops are feasible, effective, and welcome, guiding both the deployment and development of future steganography.

7 Better Steganography for Diffusion Models

While Pulsar is able to encode significant amounts of information into the images generated by diffusion models, we are limited by the structure of the current generation of diffusion models. Thus, there remains capacity within these images on which we are unable to capitalize. To argue cryptographic security, we must adhere to the structure of existing models – changing the structure would mean that an adversary might distinguish between an image generated by our refined model and the one used more generally. We note, however, that the empirical nature of modern machine learning model development means that the structural choices within existing models are not inherent; *future* iterations of the diffusion model paradigm may both have better generative performance (based on metrics used within the machine learning literature) while also increasing the resulting image's steganographic capacity.

In this section, we highlight several ways models could be changed in order to admit higher capacity steganography. In essence, each of these changes is a way in which randomness recovery could be made easier. These changes should not be integrated simply because they make models steganography friendly—if they result in worse model performance, the *quality* of the steganography will also suffer. Instead, we hope they can be explored as "win-win" opportunities, in which steganographic performance can be improved and model performance either improves or stays consistent.

Randomized final scheduler step. Pulsar encodes information in the second-to-last step of the scheduler since the final step is deterministic. Adding the randomness, however, is followed by another iteration of the model being applied, which perturbs the encoded information and introduces errors into the encoding. In Pulsar, we solve this through the use of error correcting codes. If the final scheduler step were randomized, we could use the same techniques developed in Pulsar to encode at the final step instead. Because there would be no additional model iteration to apply after encoding, the decoding process would be less error prone. This lack of errors would in turn allow for smaller (or even no) error correcting codes, resulting in higher utilization of the image.

Another consequence of having a deterministic final scheduler step is that we are only able to use one color channel in the generated image. Applications of the model propagates changes from one color channel to the others. In our experiments, we found that any information embedded into the variance noise of a second channel is effectively wiped out when the model is applied. As such, two thirds of the image is unusable; if the final step of the scheduler were randomized, we could theoretically triple our throughput.

Reversible model iterations. Instead of introducing randomness to the final scheduler iteration, it would also be sufficient to design model iterations that are reversible. That is, given access to the final image, it would be possible to *recover intermediary states* of the model or even the initial Gaussian noise that form the seed for the diffusion model. As discussed in Section 3.3, this seed is an excellent source of entropy that could be leveraged for steganographic embedding, but current model structures prevent the receiver from recovering the seed efficiently. Each model iteration applied to the seed modifies the image and each step is not efficiently reversible due to the nature of the neural network model.

Recovering the initialization seed would require non-trivial changes to the diffusion model structure, i.e., to become reversible. We note that non-linearity within the model will always make inversion more difficult, and it may be that highly non-linear models are inherently superior. Alternatively, model designers could train *pairs* of models that are able to both generate images (forwards) and recover seeds (backwards). If models with this property were possible, we could apply Pulsar's embedding strategy to the seed directly. In this case we would have (ideally) $n \times n \times 3$ bits of information for encoding per image, exceeding our current results (Table 2).

Detailed and contextually appropriate models. Pulsar is best able to encode information images that are highly detailed. For example, both Figures 2d and 3a are *realistic* images of human faces. As the respective difference heatmaps in Figures 2c and 3b show, however, Figure 2d is more *detailed* (e.g., at the top of the head), which means it can encode more information in Pulsar, and is therefore a better encoding target than Figure 3a. As such, having access to models that produce highly detailed images is desirable. We note that there is a distinction between "detailed" and "realistic", and, of course, the quality of the generated images is critical. For instance, `cat` is a detailed model, but its relatively low quality images (see Appendix B) may make censors more suspicious. Moreover, its encoding rate has high variance, as seen in Figure 4. Thus, future models need to prioritize detail, but not at the expense of quality.

Having models that produce images that are contextually appropriate is also critical. Our running examples in this work come from `celebahq` as they have the highest level of realism, but the model's outputs are heavily biased towards generating white, conventionally attractive faces, reflecting the biases of its training set. Thus, the images have limited contexts in which they would be appropriate. Future diffusion models should be trained on more diverse data sets. The problem of bias in model outputs is not unique to our use case, but only having a few types of outputs would be detrimental to deploying Pulsar. A censor monitoring a channel may expect only certain types of images; perhaps images like `celebahq` or `church` would be considered strange at best, or subversive at worst. More diverse models would allow users to steganographically generate images without arousing a censor's suspicion.

Invertible latent diffusion architectures. The models for our Pulsar evaluation are all *pixel* diffusion models, in which each model iteration performs operations on all pixels of the output image at once. Our models operate over tensors of shape $256 \times 256 \times 3$ to generate an image of dimension 256×256 with 3 color channels. As mentioned in Section 3.1, an alternative to this approach are *latent* diffusion models like Stable Diffusion [RBL+22]. Operations are performed on latents, which are a smaller representation of the image,

and upscaled at the end using a VAE. Latent diffusion architectures have good results, even on relatively limited hardware.

While the core embedding strategy of Pulsar can be applied to latent diffusion models, there are barriers to making this approach productive. One is encoding space. The latent space used by these models typically have shape $64 \times 64 \times 3$, much smaller than that of pixel diffusion models and potentially reducing throughput.

The more concerning issue is the VAE itself. Our experiments found that the VAE made it extraordinarily difficult for a receiver to determine anything about how noise was injected during model generation. In particular, because the VAE "decompresses" the latent space, applying noise to even a single element of the latent would result in noticeable changes in *many* pixels. As such, a receiver has a hard time differentiating between the effects of two neighboring latents (i.e., the effects contaminate each other). We found that encoding was possible if we spread out the locations in the latents into which we steganographically encoded, but is has *extraordinarily low* performance, e.g., 16 or 32 bits per image.

Existing approaches to VAE inversion involve estimation, and the result is not guaranteed to be the original input. Our preliminary experiments showed that attempts to reverse the VAE step resulted in too many errors to accurately reconstruct that original latents. If a VAE could be deterministically invertible, it would be possible to encode with very high rate into the latent space. Future work could identify ways to recover latents from an image.

8 Conclusion

We present Pulsar, the first provably secure steganography scheme with support for generic diffusion models. Pulsar is practical, encoding hundreds of bytes per image at speeds exceeding that of prior solutions. Future work in both steganography and diffusion models can use Pulsar's lessons to create even more efficient systems.

Acknowledgments

The authors would like to thank Katharina Schaar for helping to correct issues in a previous version of this paper.

This work was funded, in part, by the National Science Foundation's Convergence Accelerator Program, Track G under contract number 49100422C0024. The first author would like to acknowledge support from the National Science Foundation under Grant #1955172. The second author was supported by DARPA under Contract No. HR001120C0084. The third author is supported by the NSF under Grant #2030859 to the Computing Research Association for the CIFellows Project and is supported by DARPA under Agreement No. HR00112020021. A significant amount of this work was completed while the third authors was at Boston University. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of their employers or the sponsors.

References

[AP98] Ross J Anderson and Fabien AP Petitcolas. On the limits of steganography. *IEEE Journal on selected areas in communications*, 16(4):474–481, 1998.

[Aur96] Tuomas Aura. Practical invisibility in digital communication. In *Information Hiding: First International Workshop Cambridge, UK, May 30–June 1, 1996 Proceedings 1*, pages 265–278. Springer, 1996.

[Aus] Australian Government. Telecommunications and other legislation amendment (assistance and access) act 2018. https://www.legislation.gov.au/Details/C2018A00148.

[Bal17] Shumeet Baluja. Hiding images in plain sight: deep steganography. In *Neural Information Processing Systems*, 2017. URL: http://www.esprockets.com/papers/nips2017.pdf.

[Bal19] Shumeet Baluja. Hiding images within images. *IEEE transactions on pattern analysis and machine intelligence*, 42(7):1685–1697, 2019.

[BC05] Michael Backes and Christian Cachin. Public-key steganography with active attacks. In Joe Kilian, editor, *TCC 2005*, volume 3378 of *LNCS*, pages 210–226. Springer, Heidelberg, February 2005. DOI: 10.1007/978-3-540-30576-7_12.

[BCF18] Mehdi Boroumand, Mo Chen, and Jessica Fridrich. Deep residual network for steganalysis of digital images. *IEEE Transactions on Information Forensics and Security*, 14(5):1181–1193, 2018.

[BFS11] Paul Bratley, Bennet L Fox, and Linus E Schrage. *A guide to simulation*. Springer Science & Business Media, 2011.

[BK15] Elaine Barker and John Kelsey. Nist special publication 800-90a revision 1 recommendation for random number generation using deterministic random bit generators, 2015.

[BM82] Manuel Blum and Silvio Micali. How to generate cryptographically strong sequences of pseudo random bits. In *23rd FOCS*, pages 112–117. IEEE Computer Society Press, November 1982. DOI: 10.1109/SFCS.1982.72.

[Cac00] Christian Cachin. An information-theoretic model for steganography. Cryptology ePrint Archive, Report 2000/028, 2000. https://eprint.iacr.org/2000/028.

[CC10] Ching-Yun Chang and Stephen Clark. Linguistic steganography using automatically generated paraphrases. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, HLT '10, pages 591–599, Los Angeles, California. Association for Computational Linguistics, 2010. ISBN: 1-932432-65-5. URL: http://dl.acm.org/citation.cfm?id=1857999.1858083.

[CC14] Ching-Yun Chang and Stephen Clark. Practical linguistic steganography using contextual synonym substitution and a novel vertex coding method. *Computational Linguistics*, 40(2):403–448, June 2014. ISSN: 1530-9312. DOI: 10.1162/coli_a_00176. URL: http://dx.doi.org/10.1162/COLI_a_00176.

[Cha19] Marc Chaumont. Deep learning in steganography and steganalysis from 2015 to 2018, 2019. arXiv: 1904.01444 [cs.CR].

[Civ] Civitai. The home of open-source generative ai. https://civitai.com/.

[DC19] Falcon Dai and Zheng Cai. Towards near-imperceptible steganographic text. *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 2019. DOI: 10.18653/v1/p19-1422. URL: http://dx.doi.org/10.18653/v1/p19-1422.

[DCR+13] Kevin P. Dyer, Scott E. Coull, Thomas Ristenpart, and Thomas Shrimpton. Protocol misidentification made easy with format-transforming encryption. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 2013*, pages 61–72. ACM Press, November 2013. DOI: 10.1145/2508859.2516657.

[DCS15] Kevin P. Dyer, Scott E. Coull, and Thomas Shrimpton. Marionette: a programmable network traffic obfuscation system. In *24th USENIX Security Symposium (USENIX Security 15)*, pages 367–382, Washington, D.C. USENIX Association, 2015. ISBN: 978-1-931971-232. URL: https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/dyer.

[DCW+23] J. Ding, K. Chen, Y. Wang, N. Zhao, W. Zhang, and N. Yu. Discop: provably secure steganography in practice based on "distribution copies". In *2023 IEEE Symposium on Security and Privacy (SP)*, pages 2238–2255, Los Alamitos, CA, USA. IEEE Computer Society, May 2023. DOI: 10.1109/SP46215.2023.00155. URL: https://doi.ieeecomputersociety.org/10.1109/SP46215.2023.00155.

[DMS04] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: the second-generation onion router. In *Proceedings of the 13th Conference on USENIX Security Symposium - Volume 13*, SSYM'04, pages 21–21, San Diego, CA. USENIX Association, 2004. URL: http://dl.acm.org/citation.cfm?id=1251375.1251396.

[DN21] Prafulla Dhariwal and Alexander Nichol. Diffusion models beat gans on image synthesis. *Advances in neural information processing systems*, 34:8780–8794, 2021.

[Eur] European Commission. Proposal for a regulation of the european parliament and of the council laying down rules to prevent and combat child sexual abuse. https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=COM:2022:209:FIN.

[FGD01] Jessica Fridrich, Miroslav Goljan, and Rui Du. Detecting lsb steganography in color, and gray-scale images. *IEEE multimedia*, 8(4):22–28, 2001.

[FJA17] Tina Fang, Martin Jaggi, and Katerina Argyraki. Generating steganographic text with lstms. *Proceedings of ACL 2017, Student Research Workshop*, 2017. DOI: 10.18653/v1/p17-3017. URL: http://dx.doi.org/10.18653/v1/p17-3017.

[FLH+15] David Fifield, Chang Lan, Rod Hynes, Percy Wegmann, and Vern Paxson. Blocking-resistant communication through domain fronting. *Proceedings on Privacy Enhancing Technologies*, 2015(2):46–64, 2015.

[For65] G David Forney. Concatenated codes. 1965.

[FPK07] Jessica Fridrich, Tomáš Pevný, and Jan Kodovský. Statistically undetectable jpeg steganography: dead ends challenges, and opportunities. In *Proceedings of the 9th Workshop on Multimedia & Security*, MM&Sec '07, pages 3–14, Dallas, Texas, USA. Association for Computing Machinery, 2007. ISBN: 9781595938572. DOI: 10.1145/1288869.1288872. URL: https://doi.org/10.1145/1288869.1288872.

[GGA+05] Christian Grothoff, Krista Grothoff, Ludmila Alkhutova, Ryan Stutsman, and Mikhail Atallah. Translation-based steganography. In *International Workshop on Information Hiding*, pages 219–233. Springer, 2005.

[Goo22a] Google. Ddpm-bedroom-256. https://huggingface.co/google/ddpm-bedroom-256, 2022.

[Goo22b] Google. Ddpm-cat-256. https://huggingface.co/google/ddpm-cat-256, 2022.

[Goo22c] Google. Ddpm-celebahq-256. https://huggingface.co/google/ddpm-celebahq-256, 2022.

[Goo22d] Google. Ddpm-church-256. https://huggingface.co/google/ddpm-church-256, 2022.

[GRS12] Venkatesan Guruswami, Atri Rudra, and Madhu Sudan. Essential coding theory. *Draft available at http://www. cse. buffalo. edu/atri/courses/coding-theory/book*, 2(1), 2012.

[Har18] Harveyslash. Harveyslash/deep-steganography. https://github.com/harveyslash/Deep-Steganography, April 2018.

[HH19] SHIH-YU HUANG and Ping-Sheng Huang. A homophone-based chinese text steganography scheme for chatting applications. *Journal of Information Science & Engineering*, 35(4), 2019.

[HJA20] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851, 2020.

[HLv02] Nicholas J. Hopper, John Langford, and Luis von Ahn. Provably secure steganography. In Moti Yung, editor, *CRYPTO 2002*, volume 2442 of *LNCS*, pages 77–92. Springer, Heidelberg, August 2002. DOI: 10.1007/3-540-45708-9_6.

[HPR+19] Thibaut Horel, Sunoo Park, Silas Richelson, and Vinod Vaikuntanathan. How to subvert backdoored encryption: security against adversaries that decrypt all ciphertexts. In Avrim Blum, editor, *ITCS 2019*, volume 124, 42:1–42:20. LIPIcs, January 2019. DOI: 10.4230/LIPIcs.ITCS.2019.42.

[HRB+13] Amir Houmansadr, Thomas J. Riedl, Nikita Borisov, and Andrew C. Singer. I want my voice to be heard: IP over voice-over-IP for unobservable censorship circumvention. In *NDSS 2013*. The Internet Society, February 2013.

[Hug] Hugging Face. Diffusers. https://github.com/huggingface/diffusers.

[HWJ+18] D. Hu, L. Wang, W. Jiang, S. Zheng, and B. Li. A novel image steganography method via deep convolutional generative adversarial networks. *IEEE Access*, 6:38303–38314, 2018. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2018.2852771.

[JDX+21] Junpeng Jing, Xin Deng, Mai Xu, Jianyi Wang, and Zhenyu Guan. Hinet: deep image hiding by invertible network. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 4733–4742, 2021.

[KJG+21] Gabriel Kaptchuk, Tushar M. Jois, Matthew Green, and Aviel D. Rubin. Meteor: cryptographically secure steganography for realistic distributions. In Giovanni Vigna and Elaine Shi, editors, *ACM CCS 2021*, pages 1529–1548. ACM Press, November 2021. DOI: 10.1145/3460120.3484550.

[Kor23] Jennifer Korn. Microsoft outlook will soon write emails for you. CNN, October 2023.

[KSC+23] Daegyu Kim, Chaehun Shin, Jooyoung Choi, Dahuin Jung, and Sungroh Yoon. Diffusion-stego: training-free diffusion generative steganography via message projection. *arXiv preprint arXiv:2305.18726*, 2023.

[KW13] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[L] Chris L. Steganography online. https://stylesuxx.github.io/steganography/.

[LCG23] Tengjun Liu, Ying Chen, and Wanxuan Gu. Deniable diffusion generative steganography. In *2023 IEEE International Conference on Multimedia and Expo (ICME)*, pages 67–71. IEEE, 2023.

[LDJ+14] Daniel Luchaup, Kevin P. Dyer, Somesh Jha, Thomas Ristenpart, and Thomas Shrimpton. Libfte: a toolkit for constructing practical, format-abiding encryption schemes. In *23rd USENIX Security Symposium (USENIX Security 14)*, pages 877–891, San Diego, CA. USENIX Association, 2014. ISBN: 978-1-931971-15-7. URL: https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/luchaup.

[Le03] Tri Van Le. Efficient provably secure public key steganography. Cryptology ePrint Archive, Report 2003/156, 2003. https://eprint.iacr.org/2003/156.

[LK03] Tri Van Le and Kaoru Kurosawa. Efficient public key steganography secure against adaptively chosen stegotext attacks. Cryptology ePrint Archive, Report 2003/244, 2003. https://eprint.iacr.org/2003/244.

[LWZ+21] Shao-Ping Lu, Rong Wang, Tao Zhong, and Paul L Rosin. Large-capacity image steganography based on invertible neural networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10816–10825, 2021.

[Mit99] Thomas Mittelholzer. An information-theoretic approach to steganography and watermarking. In *International Workshop on Information Hiding*, pages 1–16. Springer, 1999.

[MLD+12] Hooman Mohajeri Moghaddam, Baiyu Li, Mohammad Derakhshani, and Ian Goldberg. SkypeMorph: protocol obfuscation for Tor bridges. In Ting Yu, George Danezis, and Virgil D. Gligor, editors, *ACM CCS 2012*, pages 97–108. ACM Press, October 2012. DOI: 10.1145/2382196.2382210.

[OYZ+20] Jonathan Oakley, Lu Yu, Xingsi Zhong, Ganesh Kumar Venayagamoorthy, and Richard Brooks. Protocol proxy: an fte-based covert channel. *Computers & Security*, 92:101777, May 2020. ISSN: 0167-4048. DOI: 10.1016/j.cose.2020.101777. URL: http://dx.doi.org/10.1016/j.cose.2020.101777.

[PHW+23] Yinyin Peng, Donghui Hu, Yaofei Wang, Kejiang Chen, Gang Pei, and Weiming Zhang. Stegaddpm: generative image steganography based on denoising diffusion probabilistic model. In *Proceedings of the 31st ACM International Conference on Multimedia*, pages 7143–7151, 2023.

[PM] Trevor Perrin and Moxie Marlinspike. He double ratchet algorithm. Available at https://whispersystems.org/docs/specifications/doubleratchet/.

[PPY22] Giuseppe Persiano, Duong Hieu Phan, and Moti Yung. Anamorphic encryption: private communication against a dictator. In Orr Dunkelman and Stefan Dziembowski, editors, *EUROCRYPT 2022, Part II*, volume 13276 of *LNCS*, pages 34–63. Springer, Heidelberg, May 2022. DOI: 10.1007/978-3-031-07085-3_2.

[RBL+22] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10684–10695, 2022.

[Red] Reddit. /r/aiArt. https://old.reddit.com/r/aiArt.

[Res18] Eric Rescorla. Rfc 8446 - the transport layer security (tls) protocol version 1.3. https://datatracker.ietf.org/doc/html/rfc8446#page-160, 2018.

[RR03] Leonid Reyzin and Scott Russell. Simple stateless steganography. Cryptology ePrint Archive, Report 2003/093, 2003. https://eprint.iacr.org/2003/093.

[RSG98] M. G. Reed, P. F. Syverson, and D. M. Goldschlag. Anonymous connections and onion routing. *IEEE Journal on Selected Areas in Communications*, 16(4):482–494, May 1998. ISSN: 0733-8716. DOI: 10.1109/49.668972.

[Ruh17] Sylvain Ruhault. SoK: security models for pseudo-random number generators. *IACR Trans. Symm. Cryptol.*, 2017(1):506–544, 2017. ISSN: 2519-173X. DOI: 10.13154/tosc.v2017.i1.506-544.

[SAZ+18] Ayon Sen, Scott Alfeld, Xuezhou Zhang, Ara Vartanian, Yuzhe Ma, and Xiaojin Zhu. Training set camouflage. *Decision and Game Theory for Security*:59–79, 2018. ISSN: 1611-3349. DOI: 10.1007/978-3-030-01554-1_4. URL: http://dx.doi.org/10.1007/978-3-030-01554-1_4.

[SCC07] Yun Q Shi, Chunhua Chen, and Wen Chen. A markov process based approach to effective attacking jpeg steganography. In *Information Hiding: 8th International Workshop, IH 2006, Alexandria, VA, USA, July 10-12, 2006. Revised Selcted Papers 8*, pages 249–264. Springer, 2007.

[SCS+22] Chitwan Saharia, William Chan, Saurabh Saxena, Lala Li, Jay Whang, Emily Denton, Seyed Kamyar Seyed Ghasemipour, Burcu Karagol Ayan, S. Sara Mahdavi, Rapha Gontijo Lopes, Tim Salimans, Jonathan Ho, David J Fleet, and Mohammad Norouzi. Photorealistic text-to-image diffusion models with deep language understanding, 2022. arXiv: 2205.11487 [cs.CV].

[Sim83] Gustavus J. Simmons. The prisoners' problem and the subliminal channel. In David Chaum, editor, *CRYPTO'83*, pages 51–67. Plenum Press, New York, USA, 1983.

[SME20] Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. *arXiv preprint arXiv:2010.02502*, 2020.

[SS07] Mohammad Shirali-Shahreza and M. H. Shirali-Shahreza. Text steganography in sms. *2007 International Conference on Convergence Information Technology (ICCIT 2007)*:2260–2265, 2007.

[SSM+06a] K. Solanki, K. Sullivan, U. Madhow, B. S. Manjunath, and S. Chandrasekaran. Provably secure steganography: achieving zero k-l divergence using statistical restoration. In *2006 International Conference on Image Processing*, pages 125–128, October 2006. DOI: 10.1109/ICIP.2006.312388.

[SSM+06b] Kenneth Sullivan, Kaushal Solanki, B. S. Manjunath, Upamanyu Madhow, and Shivkumar Chandrasekaran. Determining achievable rates for secure, zero divergence, steganography. In *ICIP*, pages 121–124. IEEE, 2006.

[SSM07] A. Sarkar, K. Solanki, and B. S. Manjunath. Secure steganography: statistical restoration in the transform domain with best integer perturbations to pixel values. In *IEEE International Conference on Image Processing (ICIP)*, San Antonio, Texas, September 2007. URL: https://vision.ece.ucsb.edu/sites/default/files/publications/sarkar_icip07_fractional_binwidth.pdf.

[Swa+13] Aaron Swartz et al. SecureDrop. https://github.com/freedomofpress/securedrop, 2013.

[SWM+15] Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *International conference on machine learning*, pages 2256–2265. PMLR, 2015.

[The23] The Sage Developers. *SageMath, the Sage Mathematics Software System (Version x.y.z)*. https://www.sagemath.org. 2023.

[Tor] Tor Project. The tor project: privacy and freedom online. https://www.torproject.org/.

[Unia] United Kingdom Parliament. Online safety act 2023. https://bills.parliament.uk/bills/3137.

[Unib] United States Congress. S.1409 - kids online safety act. https://www.congress.gov/bill/118th-congress/senate-bill/1409.

[vH04] Luis von Ahn and Nicholas J. Hopper. Public-key steganography. In Christian Cachin and Jan Camenisch, editors, *EUROCRYPT 2004*, volume 3027 of *LNCS*, pages 323–341. Springer, Heidelberg, May 2004. DOI: 10.1007/978-3-540-24676-3_20.

[Vin22] James Vincent. Ai-generated selfies could be the next snapchat filters. The Verge, October 2022.

[VNB+17] Denis Volkhonskiy, Ivan Nazarov, Boris Borisenko, and Evgeny Burnaev. Steganographic generative adversarial networks, 2017. arXiv: 1703.05502 [cs.MM].

[VSP+17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, undefinedukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, pages 6000–6010, Long Beach, California, USA. Curran Associates Inc., 2017. ISBN: 9781510860964.

[WGN+12] Qiyan Wang, Xun Gong, Giang T. K. Nguyen, Amir Houmansadr, and Nikita Borisov. CensorSpoofer: asymmetric communication using IP spoofing for censorship-resistant web browsing. In Ting Yu, George Danezis, and Virgil D. Gligor, editors, *ACM CCS 2012*, pages 121–132. ACM Press, October 2012. DOI: 10.1145/2382196.2382212.

[Wha17] WhatsApp. WhatsApp Encryption Overview. Available at https://scontent.whatsapp.net/v/t61/68135620_760356657751682_6212997528851833559_n.pdf/WhatsApp-Security-Whitepaper.pdf, December 2017.

[WPF13] Philipp Winter, Tobias Pulls, and Jürgen Fuß. Scramblesuit: A polymorph network protocol to circumvent censorship. *CoRR*, abs/1305.3199, 2013. arXiv: 1305.3199. URL: http://arxiv.org/abs/1305.3199.

[WYL18] Pin Wu, Yang Yang, and Xiaoqiang Li. Stegnet: mega image steganography capacity with deep convolutional network. *Future Internet*, 10(6):54, June 2018. ISSN: 1999-5903. DOI: 10.3390/fi10060054.

[Xia18] Lingyun Xiang. Reversible natural language watermarking using synonym substitution and arithmetic coding, 2018.

[XMH+22] Youmin Xu, Chong Mou, Yujie Hu, Jingfen Xie, and Jian Zhang. Robust invertible image steganography. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 7875–7884, 2022.

[YGC+19] Z. Yang, X. Guo, Z. Chen, Y. Huang, and Y. Zhang. Rnn-stega: linguistic steganography based on recurrent neural networks. *IEEE Transactions on Information Forensics and Security*, 14(5):1280–1295, May 2019. DOI: 10.1109/TIFS.2018.2871746.

[YHC+09] Zhenshan Yu, Liusheng Huang, Zhili Chen, Lingjun Li, Xinxin Zhao, and Youwen Zhu. Steganalysis of synonym-substitution based natural language watermarking, 2009.

[YJH+18] Zhongliang Yang, Shuyu Jin, Yongfeng Huang, Yujin Zhang, and Hui Li. Automatically generate steganographic text based on markov model and huffman coding, 2018. arXiv: 1811.04720 [cs.CR].

[YNY17] Jian Ye, Jiangqun Ni, and Yang Yi. Deep learning hierarchical representations for image steganalysis. *IEEE Transactions on Information Forensics and Security*, 12(11):2545–2557, 2017.

[YZX+24] Jiwen Yu, Xuanyu Zhang, Youmin Xu, and Jian Zhang. Cross: diffusion model makes controllable, robust and secure image steganography. *Advances in Neural Information Processing Systems*, 36, 2024.

[ZDR19] Zachary M. Ziegler, Yuntian Deng, and Alexander M. Rush. Neural linguistic steganography, 2019. arXiv: 1909.01496 [cs.CL].

[ZFK+98] Jan Zöllner, Hannes Federrath, Herbert Klimant, Andreas Pfitzmann, Rudi Piotraschke, Andreas Westfeld, Guntram Wicke, and Gritta Wolf. Modeling the security of steganographic systems. In *International Workshop on Information Hiding*, pages 344–354. Springer, 1998.

[ZJG22] Maximilian Zinkus, Tushar M. Jois, and Matthew Green. SoK: cryptographic confidentiality of data on mobile devices. *PoPETs*, 2022(1):586–607, January 2022. DOI: 10.2478/popets-2022-0029.

Table 3: The library of error correcting codes used for Pulsar.

Error Rate	Outer Code	Inner Code	Input Size (*bytes*)	Output Size (*bits*)	Code Rate
0.05	`GeneralizedReedSolomonCode(255, 200)`	`HammingCode(GF(2), 3)`	200	3570	0.44
0.10	`GeneralizedReedSolomonCode(255, 100)`	`HammingCode(GF(2), 3)`	100	3570	0.22
0.15	`GeneralizedReedSolomonCode(193, 177)`	`BCHCode(GF(2), 51, 17)`	243	9843	0.20
0.20	`GeneralizedReedSolomonCode(289, 187)`	`BCHCode(GF(2), 51, 17)`	257	14739	0.14
0.25	`GeneralizedReedSolomonCode(271, 183)`	`BCHCode(GF(2), 73, 18)`	228	19783	0.09
0.30	`GeneralizedReedSolomonCode(255, 200)`	`BinaryReedMullerCode(1, 7)`	200	32640	0.05
0.35	`GeneralizedReedSolomonCode(255, 100)`	`BinaryReedMullerCode(1, 7)`	100	32640	0.02

Algorithm 4: Optimized EstimateRate

Input: Model state s_{t-2}
Output: Difference region-based ECC rates rate
`// Get error rates at each pixel`
err $\leftarrow$ CalcErrors(l)
`// Bucket regions based on error rates`
regions $\leftarrow$ Bucketize(err, u)
for region $\in$ regions **do**
    regionErr $\leftarrow$ mean(err[region])
    `// Select appropriate ECC parameters`
    params $\leftarrow$ $\text{Library}_{\text{ECC}}$[regionErr])
    rate[region] $\leftarrow$ params
Output rate

A Additional Details on Error Correction in Pulsar

Pulsar uses error correcting codes to improve the performance of the steganographic channel inside of diffusion models. A more detailed treatment of the EstimateRate used in Pulsar can be found in Algorithm 4. Note that our optimized EstimateRate requires the estimation of the errors in the current model state by performing trial encoding and decoding of random messages. We define a subroutine in Algorithm 5, CalcErrors, that performs this estimation for EstimateRate. Note that EstimateRate is parameterized by the number of buckets in each estimate u and l, which we discuss in Appendix D.

Table 3 contains information on the concrete codes library that we built for Pulsar. This library forms the $\text{Library}_{\text{ECC}}$ in Algorithm 4.

As we discovered these codes experimentally, we make no claims about optimality. Each concatenated code was tested on 500 inputs. The worst code achieved a decoding error probability of about 6 percent. The inner code is written in a way that the code can be instantiated using SageMath. The outer code is written as `GeneralizedReedSolomonCode(n,k)` where `n` is the block length and `k` is the dimension. The field the code is defined over is of size 2^z where z is the smallest integer such that $2^z > \mathtt{n}$. When $n \neq 2^z - 1$ the evaluation points are chosen at random. We leave achieving concatenated codes with higher rates and concrete correctness guarantees to future work.

Algorithm 5: CalcErrors

Input: Model state s_{t-2}, Number of estimates l
Output: Mean error rates at each pixel location err
for $1 \leq g < l$ **do**
  `// Similar to the online phase of encoding`
  $m \xleftarrow{\$} [0,1]^{n \times n}$
  for $0 \leq j < |m|$ **do**
    if $m[j] = 0$ **then**
      $r_{t-1}[j] \xleftarrow{\$} \mathcal{N}_{k_0}$
    else
      $r_{t-1}[j] \xleftarrow{\$} \mathcal{N}_{k_1}$
  $\mathsf{pred}_{t-1} \leftarrow \mathsf{Model}(s_{t-2})$
  $s_{t-1} \leftarrow \mathsf{Scheduler}_{t-1}(s_{t-2}, \mathsf{pred}_{t-1}; r_{t-1})$
  `// Deterministic Final Schedule`
  $\mathsf{pred}_t \leftarrow \mathsf{Model}(s_{t-1})$
  $\mathsf{img} \leftarrow \mathsf{Scheduler}_t(s_{t-1}, \mathsf{pred}_t)$
  `// Similar to the offline phase of decoding`
  `// Generate reference image 0`
  Set $r^0_{t-1} \xleftarrow{\$} \mathcal{N}^{n \times n}_{k_0}$
  Set $s^0_{t-1} \leftarrow \mathsf{Scheduler}_{t-1}(s_{t-2}, \mathsf{pred}_{t-1}; r^0_{t-1})$
  $\mathsf{pred}^0_t \leftarrow \mathsf{Model}(s^0_{t-1})$
  $\mathsf{img}_0 \leftarrow \mathsf{Scheduler}_t(s^0_{t-1}, \mathsf{pred}^0_t)$
  `// Generate reference image 1`
  $r^1_{t-1} \xleftarrow{\$} \mathcal{N}^{n \times n}_{k_1}$
  Set $s^1_{t-1} \leftarrow \mathsf{Scheduler}_{t-1}(s_{t-2}, \mathsf{pred}_{t-1}; r^1_{t-1})$
  $\mathsf{pred}^1_t \leftarrow \mathsf{Model}(s^1_{t-1})$
  $\mathsf{img}_1 \leftarrow \mathsf{Scheduler}_t(s^1_{t-1}, \mathsf{pred}^1_t)$
  `// Similar to the online phase of decoding`
  for $0 \leq j < |m|$ **do**
    if $|\mathsf{img}[j] - \mathsf{img}_0[j]| < |\mathsf{img}[j] - \mathsf{img}_1[j]|$ **then**
      $m'[j] \leftarrow 0$
    else
      $m'[j] \leftarrow 1$
  `// Save the n × n matrix of magnitudes to a list`
  $\mathsf{errs}[g] \leftarrow \mathsf{abs}(m - m')$
`// Mean of all estimated magnitudes`
Output $\mathsf{err} \leftarrow \mathsf{mean}(\mathsf{errs})$

B Sample Pulsar Images

Figures 6 to 9 contain additional sample images from our Pulsar evaluation. In each figure, we show generated images achieving the (a) best and (b) worst (b) encoding lengths when instantiating Pulsar with each of the pretrained models we used in our evaluation.

(a) A Pulsar image which encodes 1143 bytes.

(b) A Pulsar image which encodes 100 bytes.

Figure 6: Sample Pulsar outputs for the `church` model.

(a) A Pulsar image which encodes 743 bytes.

(b) A Pulsar image which encodes 228 bytes.

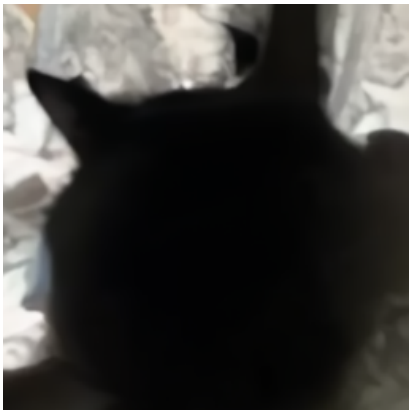
Figure 7: Sample Pulsar outputs for the `celebahq` model.

(a) A Pulsar image which encodes 771 bytes.

(b) A Pulsar image which encodes 100 bytes.

Figure 8: Sample Pulsar outputs for the `bedroom` model.

(a) A Pulsar image which encodes 1557 bytes.



(b) A Pulsar image which encodes 100 bytes.

Figure 9: Sample Pulsar outputs for the `cat` model.

C Steganalysis of Pulsar

As discussed in Section 6, we ran Pulsar against two steganalysis systems, YeNet [YNY17] and SRNet [BCF18], to experimentally validate its provable security (based on the proof in Appendix E). We ran both YeNet and SRNet on 10000 images, where 5000 were generated by Pulsar, and 5000 were generated using the same models as Pulsar but without any steganographic embedding. YeNet correctly classified 4895 out of these 10000 images, for an accuracy of approximately 49%. SRNet correctly classified 5003 out of these 10000 images, for an accuracy of approximately 50%.

We note that, based on our definition of security in Section 4.2, this distinguisher should have negligible advantage, i.e., do no better than a random guess. To show this, we turn to a hypothesis test. We consider the null hypothesis that the steganalysis methods are choosing from random; in statistical terms, the steganalyzer is a Bernoulli random variable X with $p = 0.5$. We say $X = 1$ when the steganalyzer believes an image is steganographic, and $X = 0$ when it believes an image is not. Because of the large sample size $n = 10000$, we use a Normal distribution approximation.

First, we consider YeNet. The alternative hypothesis is YeNet is *not* a Bernoulli(0.5), i.e., is not selecting from random. We calculate our test statistic:

$$Z = \frac{X - \bar{X}}{\sqrt{n}\sqrt{Var(X)}} = \frac{X - np}{\sqrt{np(1-p)}} = \frac{4895 - (10000)(0.5)}{\sqrt{(10000)(0.5)(0.5)}} = -2.1$$

Then, we can calculate the two-tailed p-value:

$$P(|Z| \geq 2.1) = 0.036$$

At the $\alpha = 0.01$ level of significance, we fail to reject the null hypothesis that YeNet is a Bernoulli(0.5), and therefore cannot show that it is doing better than choosing randomly.

Next, we consider SRNet. The alternative hypothesis once again is SRNet is *not* a Bernoulli(0.5), i.e., is not selecting from random. We calculate our test statistic:

$$Z = \frac{X - \bar{X}}{\sqrt{n}\sqrt{Var(X)}} = \frac{X - np}{\sqrt{np(1-p)}} = \frac{5003 - (10000)(0.5)}{\sqrt{(10000)(0.5)(0.5)}} = 0.06$$

Then, we can calculate the p-value:

$$P(|Z| \geq 0.06) = 0.9522$$

At the $\alpha = 0.01$ level of significance, we fail to reject the null hypothesis that SRNet is a Bernoulli(0.5), and therefore cannot show that it is doing better than choosing randomly either.

We note that, for a distinguisher to reject the null hypothesis, out of 10000 images, it would have to correctly classify somewhere *outside* of the range of

$$\bar{X} \pm Z_{\frac{\alpha}{2}}\sqrt{n}\sqrt{Var(X)} = (10000)(0.50) \pm (2.576)\sqrt{(10000)(0.5)(0.5)} = 4871.2 \leq x \leq 5128.8$$

images at the $\alpha = 0.01$ level of significance.

Table 4: Averages from 100 trials of our u experiment.

Buckets	Estimation Time (sec)	Message Length (bytes)
25	$\bar{x} = 6.26$, $s = 0.25$	$\bar{x} = 488.44$, $s = 172.05$
50	$\bar{x} = 6.43$, $s = 0.25$	$\bar{x} = 566.70$, $s = 184.05$
100	$\bar{x} = 6.45$, $s = 0.25$	$\bar{x} = 610.12$, $s = 194.24$
125	$\bar{x} = 6.47$, $s = 0.25$	$\bar{x} = 605.43$, $s = 204.59$
150	$\bar{x} = 6.48$, $s = 0.25$	$\bar{x} = 623.10$, $s = 206.37$

Table 5: Averages from 100 trials of our l experiment.

Estimates	Estimation Time (sec)	Success Rate
1	$\bar{x} = 6.15$, $s = 0.06$	89.0%
3	$\bar{x} = 6.88$, $s = 0.07$	94.0%
5	$\bar{x} = 7.44$, $s = 0.07$	89.0%
10	$\bar{x} = 8.84$, $s = 0.07$	86.0%
30	$\bar{x} = 14.48$, $s = 0.09$	85.0%

D Parameter Selection

In Section 5.2, we discussed our optimized EstimateRate approach that variably encodes using different error correcting codes based on the available entropy in the image, and in Section 5.3, we discussed how we determined the library of codes to support this approach. There are two parameters in EstimateRate that we have yet to determine: the number of buckets to generate for our difference regions u, and the total number of estimates to generate l. We experimentally determine these parameters for our implementation.

Number of buckets in each estimate u. We first want to find the number of buckets to define difference regions in EstimateRate. We can increase the number of buckets to get more granular regions, but this increases execution time and may result in over-fitting. So, we fixed $l = 1$ and ran 100 iterations of EstimateRate on Desktop for multiple candidate u values on the `church` model. Our results can be found in Table 4. $u = 100$ appears to be the best balance between mean message length, variability of the message length, and estimation time, so we choose that as the parameter.

Number of estimates l. The other parameter is the number of estimates generated during EstimateRate. We want to know if increasing *offline* estimates improves the *online* success rate—the percentage of encoded images that can be successfully decoded. Each estimate involves an iteration of a model, so we want to ensure that these additional estimates are worth the longer computation. In each trial, we fix $u = 100$ (based on our evaluation for u above), and run EstimateRate , varying l. We then encode a message using the estimated regions and see if the generated image decodes back into the same message. This shows if more estimates lead to a better chance of a successful encoding. We run 100 of these trials on Desktop and the `church` model, and show our results in Table 5. Interestingly, our results demonstrate that additional estimates do not meaningfully improve the success probability of encoding. We use $l = 1$ for the remainder of our evaluation because of its faster speed, but note that $l = 3$ is also a candidate option due to its higher success rate.

E Proof of Security for Pulsar

Correctness. Observe that the correctness of our scheme is not guaranteed with overwhelming probability if the encode algorithm outputs an image without checking that the decode algorithm will succeed. However, it is trivial to modify the encoding algorithm to locally run the decode algorithm and restart as necessary. With a polynomial number of attempts, the probability that the encoder can find no randomness that permits encoding is vanishingly small in the security parameter. As such, correctness can be guaranteed with this trivial modification.

Security. To prove security, we show that we can reduce the security of Pulsar to the security of the PRG that underlies our construction. We do this in two steps for clarity. First, we reduce the security of Pulsar to the security of a non-standard version of the real-or-random PRG security game in which the adversary either gets access to two oracles that produce honest randomness or two instances of the PRG under different keys. Second, we show that this non-standard game reduces to the standard real-or-random PRG security game.

Step 1. Consider the following version of the real-or-random security experiment, which we call *Experiment 1*: a challenger flips a coin $b \in \{0,1\}$. If $b = 0$, then the polynomial time adversary $\mathcal{A}$ is given access to two oracles that, when prompted, produce honest, uniform randomness. Otherwise, if $b = 1$, then $\mathcal{A}$ is given access to the oracles that produce output from PRF_{k_0} and PRF_{k_1} when prompted, where k_0, k_1 are sampled uniformly from the key space. The adversary can interact with these oracles at will, and then must output a guess b'. If $b' = b$, then adversary wins, and loses otherwise. We say the advantage of the adversary in *Experiment 1* is the gap between probability that the adversary wins and $\frac{1}{2}$.

We now show that the security of Pulsar reduces to the security of *Experiment 1*. Specifically, for an adversary $\mathcal{A}_0$ against Pulsar, we can construct a adversary $\mathcal{A}_1$ with the same advantage in *Experiment 1* that runs $\mathcal{A}_0$ as a subroutine:

- $\mathcal{A}_1$ is initialized with access to either oracles that produce true randomness or produce output from PRGs.
- Whenever $\mathcal{A}_0$ makes a query to the encoding oracle, $\mathcal{A}_1$ produces a response using its two oracles. That is, $\mathcal{A}_1$ follows Pulsar encoding algorithm, but each time it would draw randomness from $\mathcal{N}_{k_0}$ or $\mathcal{N}_{k_1}$, it draws that randomness from either its first or second oracles respectively.
- When $\mathcal{A}_0$ outputs a guess b', $\mathcal{A}_1$ outputs b' as its guess.

Notice that in the case where $\mathcal{A}_1$ is initialized with access to oracles that produce true randomness, then all queries made by $\mathcal{A}_0$ are answered with samples distributed exactly according $O_{\mathcal{D}}(\cdot,\cdot)$. Similarly, if $\mathcal{A}_1$ is initialed with access to oracles that produces output from PRGs, then all queries made by $\mathcal{A}_0$ are answered with samples distributed exactly according to $\mathsf{Encode}_{\mathcal{D}}(\mathsf{k},\cdot,\cdot))$. As such, if $\mathcal{A}_0$ is a ppt. algorithm such that

$$\left|\mathsf{Pr}[\mathcal{A}_{0_{\mathcal{D}}}^{\mathsf{Encode}_{\mathcal{D}}(\mathsf{k},\cdot,\cdot))} = 1] - \mathsf{Pr}[\mathcal{A}_{0_{\mathcal{D}}}^{O_{\mathcal{D}}(\cdot,\cdot)} = 1]\right| > \mathsf{negl}(\lambda),$$

then $\mathcal{A}'$ has the same, non-negligible advantage in *Experiment 1*.

Step 2. Next, we show that *Experiment 1* outlined in *Step 1* reduces to the standard real-or-random security experiment. To see this, note that when $\mathcal{A}_1$ above is initialized with access to two oracles that produce output from PRGs, we can replace the second oracle with an oracle that produces uniform randomness. We call this *Experiment 2*. If $\mathcal{A}_1$ could distinguish *Experiment 1* from *Experiment 2*, we can trivially construct an adversary $\mathcal{A}_2$ for the standard real-or-random experiment:

- $\mathcal{A}_2$ is initialized with access to an oracle that either produces true randomness or produces output from a PRG.
- $\mathcal{A}_2$ samples a key for the first oracles k
- Whenever $\mathcal{A}_1$ makes a query to the first oracle, $\mathcal{A}_2$ answers this query with output from the PRG using key k.

– Whenever $\mathcal{A}_1$ makes a query to the second oracle, $\mathcal{A}_2$ answers this query by querying its own oracle and returning the result.
– When $\mathcal{A}_1$ outputs a guess b', $\mathcal{A}_2$ outputs b' as its guess.

Notice that if $\mathcal{A}_2$ is initialized with access to an oracle that produces pseudorandom output, then all queries made by $\mathcal{A}_1$ are answered according to the initial experiment unless the key k sampled by $\mathcal{A}_2$ is exactly the same as the key sampled by its oracle. This happens with inverse probability to the key space, which we assume is exponential in the security parameter. Similarly, if $\mathcal{A}_2$ is initialized with access to an oracle that produces true random output, then all queries made by $\mathcal{A}_1$ are answered according to this new experiment. Thus, $\mathcal{A}_1$ can not distinguish between these two experiments with non-negligible advantage.

Finally, we show that if $\mathcal{A}_1$ has non-negligible advantage in *Experiment 2*, then we can use it to construct an adversary in the standard real-or-random security experiment. Specifically, we construct this adversary $\mathcal{A}_3$ as follows:

– $\mathcal{A}_3$ is initialized with access to an oracle that either produces true randomness or produces output from a PRG.
– Recall that in *Experiment 2*, $\mathcal{A}_1$ is given access to a first oracle that is either true randomness or pseudorandomness and a second oracle which is always true randomness.
– Whenever $\mathcal{A}_1$ makes a query to its first oracle, $\mathcal{A}_3$ answers this query by querying its own oracle and returning the result.
– Whenever $\mathcal{A}_1$ makes a query to its second oracle $\mathcal{A}_3$ samples uniform randomness and returns that result to $\mathcal{A}_1$
– When $\mathcal{A}_1$ outputs a guess b', $\mathcal{A}_3$ outputs b' as its guess.

Notice that whenever $\mathcal{A}_3$ is initialized with access to an oracle that produces true randomness, then $\mathcal{A}_1$'s queries are answered in the same manner as when both its oracles are true randomness. Similarly, whenever $\mathcal{A}_3$ is initialized with access to an oracle that produces pseudorandomness, then $\mathcal{A}_1$'s queries are answered in the same manner as when one of its oracles is true randomness and the other is pseudorandom. Thus, $\mathcal{A}_1$'s advantage in *Experiment 2* must be bounded by $\mathcal{A}_3$'s advantage in the real-or-random game. Because we assume that there exists no adversary with non-negligible advantage in the real-or-random PRG game, this means that $\mathcal{A}_1$ cannot have non-negligible advantage in *Experiment 2*. This concludes our proof.