Assemblage: Chipping Away at Censorship with Generative Steganography

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

Cora Rowena Ruiz
City College of New York
New York, NY, USA
cruiz1@ccny.cuny.edu

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

Abstract

Increasing Internet censorship has led to developments in steganography that are promising pathways to build circumvention technologies. We present Assemblage, a novel scheme which applies these developments to encode messages within unassuming AI-generated images. Assemblage communicates messages using popular public platforms for these images as rendezvous points, effectively circumventing the censor and opening communication.

Keywords

censorship circumvention, generative steganography, rendezvous

1 Introduction

Modern Internet censorship is pernicious and spreading. To circumvent this censorship, tools often include *steganography*, which hides a sensitive message in a benign-looking cover channel. With steganography, the *presence* of a message itself is hidden, a powerful property for censored users and one that has made steganography a key primitive in censorship circumvention technologies. A classic example is Collage [10], which builds a framework for steganographically sending and receiving messages over online platforms for *user-generated content*, e.g., image-sharing websites.

At the same time, the Internet is undergoing a monumental upheaval with the advent of publicly available generative artificial intelligence (AI) models. A rapidly developing research area poised to take advantage of this AI boom is *generative steganography* [18, 43, 44], which hides messages (*plaintext*) inside of AI-generated outputs. At a high level, a censor cannot distinguish between an output that contains a message (*stegotext*) and a regular model output (*covertext*). Unlike prior efforts at steganography (e.g., [36, 48, 61]), generative steganographic techniques both (1) have provable, cryptographic security and (2) can run on realistic, human-like channels. Moreover, with easily available generative AI models for images and text, everyday people are generating and posting *AI-generated content* to the Internet [46, 67] – even those in authoritarian regimes [65]. These outputs are high quality and online subcultures have risen [52] to share and discuss this content, leading to plausible locations where steganographic messages can be dropped for others to receive without alerting a censor.

But, despite the explosion of AI content on the Internet and its potential to circumvent censors, generative steganography has not seen the transition from academia to practice. While works like Collage have studied embedding data into user-generated content, AI-generated content has different properties and is less understood by the community. This mismatch does not preclude deployment; rather, it means we must adapt circumvention frameworks to fit.

In this work, we design a system for censorship circumvention based on state-of-the-art results in generative image steganography. Just as Collage introduced a framework for circumventing censorship via user-generated content, we introduce *Assemblage*, which does so via AI-generated content. The goal of our work is to highlight how generative steganography can be deployed to circumvent censorship while using previous work as a blueprint; but more broadly, it is to spark a wider conversation about how best to apply generative steganography to censorship circumvention and what additional work is required for the future.

Contributions. With the above in mind, we contribute:

- *Assemblage*, a novel scheme for circumventing censorship using image-based generative steganography, constituting its first concrete application in censorship circumvention.
- The first analysis of the image platforms and communities that would be suitable for *rendezvousing* circumvention messages between senders and receivers in generative steganography.
- An evaluation of Assemblage on these image platforms, highlighting its end-to-end resilience and efficiency.
- Directions for future work in the field implied by Assemblage.

2 From Collage to Assemblage

We begin by describing Collage, and how it and recent developments in steganography lead us to our solution, Assemblage.

Collage. In the original Collage paper, Burnett et al. [10] discuss two key properties to circumvent censorship. The first, *availability*, is when clients can still communicate even if a censor starts targeting access to different types of content. The second, *deniability*, is when censors cannot detect if users are engaging in circumvention. To evade blocking while meeting these properties, these communication schemes typically force the censor into taking "collateral damage": to block potential circumvention traffic, the censor must also block a service with social or economic value [26]. In Collage, the collateral damage would be blocking an image-sharing platform.

Collage separates its solution design three parts. The *vector layer* provides the cover medium (e.g., images or text) for hiding a message as well as functionality for short data chunks to be encoded into and decoded from this cover medium. Then, the *message layer* takes a message of arbitrary length and breaks it into chunks which can be encoded into vectors. Finally, senders and receivers *rendezvous* on a content host to store and receive vectors containing hidden messages. Collage meets the above two properties by the

Free and Open Communications on the Internet 2026(1), 33–42

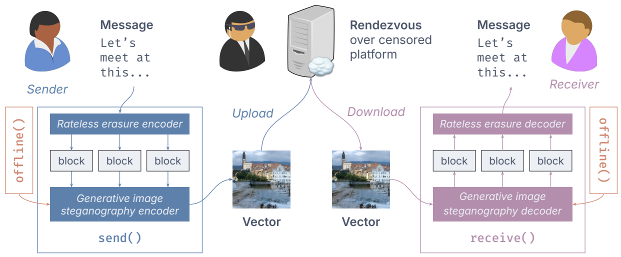


Figure 1: Assemblage overview. The censor only sees an innocuous post, but the receiver can decode the actual message when they detect that a stegotext has been posted.

steganographic properties of the vectors, the robustness of message chunking, and the deniability of the chosen rendezvous technique.

Steganography. The intuitive idea of steganography makes it attractive for censorship resistance applications such as Collage. Beginning with the classic work of Simmons [56], there has been a significant treatment of steganography in the cryptographic literature, including constructions of *universal* steganography for arbitrary channels [5, 12, 36, 60], but these mostly theoretical works have unrealistic assumptions on the cover channel and have not seen deployment. In parallel, various *heuristic* schemes for steganography have been developed and deployed, but either do not meet cryptographic notions of security (e.g., [4, 6, 14, 15, 66]), or are specific to individual cover channels (e.g., [23, 37, 48, 61, 62]). The original Collage construction [10] implements heuristic steganography as its vector layer due to its ease of deployment, even though its authors note that attacks against it existed at the time [27].

Recent developments in *generative* steganography attempt to reconcile this tradeoff between security and flexibility. Works in this area [18, 43, 44] embed sensitive messages into the outputs of generative AI models. These outputs – both text and images – have both runtime overhead (requiring relatively slow machine learning operations) and message overhead (only encoding data on the order of hundreds of bytes), but are increasingly realistic and human-like while maintaining cryptographic provable security.

Assemblage. In Assemblage, we adapt the Collage framework to meet the properties of availability and deniability using generative steganography. As we discuss above, generative steganography avoids the issues of prior attempts, but considers AI-generated rather than user-generated content. The aforementioned popularity of this content [46, 67] (even in censored regimes [65]) opens a new path for censorship circumvention. Thus, to take advantage of these developments, Assemblage takes the Collage framework – vectors, messages, and rendezvous – to fit the model and use of AI-generated content. An overview can be found in Figure 1.

3 Assemblage design

We now introduce Assemblage, going layer-by-layer as in Collage.

3.1 Vector layer

For Assemblage's core vector primitive, we choose Pulsar [43], which embeds content using diffusion models [35, 58], the state-of-the-art for image generation. The encoder and decoder locally

Algorithm 1 offline(identifier, n)

Create a list to hold local states.
for each i from 1 to n **do**
  Generate a local state using the underlying generative steganographic algorithm.
  Append the local state to the list.
return the list of n local states.

Algorithm 2 send(states, data)

Create a rateless erasure encoder for data.
for each state in states **do**
  Retrieve a block from the erasure coder to meet the vector's encoding capacity.
  encode the block into the state to create a vector using the underlying generative steganographic algorithm.
  Publish the vector on a AI-generated content host such that receivers can find it. See discussion on rendezvousing in Section 3.3.

Algorithm 3 receive(identifier)

Create a rateless erasure decoder.
while the decoder cannot decode the message **do**
  Find and fetch a vector from an AI-generated content host.
  Check if the vector contains encoded data for this identifier. See discussion on rendezvousing in Section 3.3.
  if the vector is encoded with message data **then**
    decode payload from the vector using the underlying generative steganographic algorithm.
    Provide each decoded block to the erasure decoder.
return decoded message from erasure decoder

Figure 2: Pseudocode presented as Algorithm 1 (offline), Algorithm 2 (send), and Algorithm 3 (receive).

generate state that allows for steganographic embedding. Then, the encoder uses a local state to embed data into an image. The decoder can then use its own local state to retrieve the data encoded into the image. Pulsar operates on a symmetric key assumption, as in the vector layer of Collage. But unlike the heuristic steganography vectors used in Collage, which were vulnerable to attack even then [27], a Pulsar-encoded image vector is provably secure [43].

3.2 Message layer

Our choice of an AI-generated vector impacts the design of our message layer. We divide message transmission into three algorithms – offline, send, and receive – with pseudocode definitions available in Figure 2. In the first, offline, the message layer creates several local Pulsar states corresponding to candidate AI-generated image vectors. This step allows Assemblage to "prepare for" future send and receive operations by having local states ready before they are needed. As there is no offline step in the original Collage work, this represents one of the ways Assemblage updates the Collage framework for modern steganography.[1]

When the sender calls send on a message, we choose an offline state, encode the message into this state, and return an image that can be posted on an AI-generated content host. As in Collage, we employ rateless erasure codes [11, 47, 55] to create short chunks of data from the message for encoding into the vector. This allows sender to handle messages of arbitrary length over multiple vectors, even though a single vector's encoding size is limited.

Correspondingly, in receive, the recipient's message layer watches posts on the AI image community and check to see if an image is intended for them (i.e., matches their local state); if so, then the recipient downloads the image and performs a Pulsar decoding to retrieve the encoded data.[2] The decoded data is then passed back to a rateless erasure decoder, and after enough vectors are processed, it can return the sender's message. On the recipient side, the rateless erasure code allows for vectors to be received out-of-order or not at all and still reassemble to the original message.

3.3 Rendezvous

In designing its rendezvous approach, Collage utilizes popular platforms for user-generated content for availability, and formalizes a database of *tasks* that the sender and receiver agree upon beforehand for deniability. For Assemblage, Internet communities for sharing AI-generated images are promising rendezvous locations. Also, in addition to the social value of its associated communities to its users, AI images also have economic value [69]: the kind of "collateral damage" censors want to avoid. Given the popularity of AI-generated content, we instead choose a single, global task for Assemblage: senders post to a AI-generated content community, and receivers view and download these posts. If both parties can post to the community, communication could also be bi-directional. So, to build rendezvousing, we must blend Assemblage posts within this community, and we can achieve the twin goals of availability and deniability.[3] Collage address three challenges in rendezvousing; we do the same for Assemblage below:

Choosing deniable tasks. The first challenge is in selecting tasks that resemble innocuous user behavior. In Assemblage, this is posting and viewing AI-generated content, but an important question to answer is *where* to do so. Pulsar [43] mentions /r/AIArt [52] as an AI-generated content location, but does not discuss where else images can be plausibly placed to enable deniable communication.

So, to understand the sharing of AI-generated content, we studied popular social media platforms to identify viable posting locations and observe their communities' cultures. Table 1 details the communities we observed, their style of social media, and their relative reach measured by community size and platform monthly active users (MAU). Our examination of these channels revealed vibrant communities showcasing AI-generated images, discussing generated content, and facilitating troubleshooting. Despite differences in platform structure, they are similar in their relaxed, participatory culture that emphasizes building community. To the best of our knowledge, no prior work has done such a study of potential rendezvous locations for generative AI steganography.

A key property is that rendezvous should be *deniable* against the normal behaviors of the users of a platform [10]. Prior work that has considered generative steganography for censorship circumvention [7] has looked not at images but at AI-generated *text*, in which ~100 B messages will generate hundreds of words of text [18, 44]. But, while posting images is common, long-form text is rare on social media, and long-form AI-generated text even less so. Thus, we see AI image posting behaviors (and therefore Assemblage) as better meeting the deniability criterion than those of prior work.

Identifying suitable vectors. To address the next challenge, we must understand the *norms* in AI image communities: what kinds of things do they post about? Community norms for rendezvous locations are relatively understudied but critically important for deniability. If a post does not match those of the rest of the community, it may compromise the presence of Assemblage. As an example, a satirical comic may be suspicious when posted a more serious community on AI-generated cinematic realism. Moreover, no prior works discuss post metadata; each community has different expectations regarding what information is posted alongside an image, which also must be followed to avoid arousing suspicion.

Considering the above, we collected requirements by analyzing community guidelines and existing posts as well as how those posts were received. We also study image style, as this varied across platforms with distinct tonal differences. Matching these norms supports the authenticity of a post, allowing it to blend into the surrounding community. We discuss further in Section 4.1.

Agreeing on tasks for a message. We mention above that the global task for Assemblage is simply posting to and downloading from AI image communities. But, monitoring the community and naïvely attempting to decode every single post on it would be prohibitively expensive and ineffective for the receiver. Instead, we note that Pulsar images generated by a given local state are visually similar regardless of the content encoded into them [43]. So, we can apply a *perceptual hash* – a type of fuzzy hash robust to slight variations in images – to each new post on the platform, and see if it matches that of a local state.[4] If it does, then we download the image and decode the vector. Locally checking a perceptual hash is fast (as we will see in Section 4) and induces no behavior visible to the censor, so deniability is maintained. Here again, Assemblage takes advantage of a steganography innovation to optimize the deployment of censorship circumvention.

4 Evaluation

We implement Assemblage based on the public Pulsar artifact [42]. We generate image vectors of places of worship [32], celebrity faces [31], bedrooms [29], and cats [30]. To implement perceptual hashing, we use the PDQ algorithm [24], and to implement rateless erasure coding, we use Raptor codes [55]. Our code and benchmarks are available at https://github.com/spacelab-ccny/assemblage as artifacts for the community.

[1]Note that, as Pulsar is probabilistic [43], not every offline state will successfully generate a decodable message. But, it is possible to detect this in the offline step (i.e., before message transmission) by performing a trial decoding.

[2]By the security of Pulsar, only the intended recipient can generate this state and therefore perform this detection; we refer to that paper [43] for the proof.

[3]We note that accounts used to rendezvous should establish a history of AI-generated posts. Otherwise, participating heavily in these communities may be seen as suspicious.

[4]We merely use perceptual hashing to identify images in our local offline set and not for the security-sensitive purposes for which perceptual hashing has failure modes [51].

Table 1: Potential locations for AI-generated content, and if Assemblage can be built on top of them.

Platform	Style	Viable Drop Locations	Reach	Metadata Required?	Compresses Images?	Successful?
DeviantArt	Media sharing	Platform-wide Public AI image communities	65 mil MAU [25] 800+ groups [3]	✓	✗	✓
Discord	Messaging	Platform-wide “AI Hub by Weights” “ai Art” “NightCafeLounge”	200 mil MAU [22] 521k members [21] 6k members [19] 78.2k members [20]	✗	✓	✓
Imgur	Media sharing	Platform-wide “aiart” hashtag	300 mil MAU [40] 1k posts [41]	✓	✓	✓
Rednote	Media sharing	Platform-wide	300 mil MAU [28]	✓	✓	✗
Reddit	Discussion board	Platform-wide “/r/AIArt” subreddit “/r/AIArtwork” subreddit	1.21 bil MAU [16] 608k members [52] 83k members [53]	✓	✗	✓
X (Twitter)	Micro-blogging	Platform-wide “Generative AI Community”	611 mil MAU [64] 191.7k members [63]	✓	✗	✓
Telegram	Messaging	Broadcast groups “Robots and Art” group “Atlas AI Art” group	1 bil MAU [17] 103.3k members [54] 1.2k members [2]	✗	✓	✓
WeChat	Messaging	Broadcast groups	1.3 bil MAU [57]	✗	✓	✗

(a) celebahq-256, “Angela”

(b) bedroom-256, “Snooze”

(c) cat-256, “Gary”

(d) church-256, “Chapel”

Figure 3: The four encoded test images used during testing of potential channels, and their respective models.

4.1 Candidate Platforms

To deploy Assemblage, we must first evaluate the interactions between Assemblage vectors and candidate rendezvous platforms.

Resilience. Platforms can compress or otherwise modify uploaded images [9], and these modifications interfere with Assemblage vectors. To test the end-to-end *resilience* of Assemblage, we conducted trials over our roster of AI-generated content platforms. Each trial used Assemblage image vectors (see Figure 3) from one of the aforementioned models. Vectors were uploaded to the platform with the required metadata and downloaded from a different account to emulate rendezvousing. We considered the trial a success if the downloaded vector was successfully decodable into the original message sent. All vectors were promptly deleted after verifying behavior to avoid disrupting communities.

We can see from Table 1 that the majority of platforms we tested successfully transmitted data over image vectors, with the biggest factor being *lossy compression.* For some of these platforms (e.g., Discord), an image downloaded via “Save image as...” from the browser context menu is often a scaled-down, compressed version (~30 KB in size), not the original upload (~300 KB in size). We could not receive from the scaled version. Instead, if we use the platform’s native “Download” option to obtain the original image, receive works. We speculate this is related to CDN-style optimizations; platforms prefer to serve compressed thumbnails when possible, and provide the original only when requested. But, this trick did not extend to all platforms; inhibiting us from building Assemblage on platforms that only provide lossy downloads (e.g., Rednote).

Platform metadata. A vital component of a successful rendezvous is ensuring that the post carrying the stegotext image is associated with metadata that fits in with the target community. Having the correct metadata is an important part of achieving deniability. We now discuss the required and optional metadata required to post by platform type outlined in Table 1.

Messaging: no metadata required. In Discord [19–21] members are typically allowed to upload an image without a caption or context. However, some communities [21] ask brief questions during the on-boarding process – such as “What kind of experience are you looking for?” or “What are your areas of interest?” – to help place users in relevant channels/groups or assign server nicknames accordingly. Telegram groups [2, 54] had no standing requirements at the time of our examination.

Micro-blogging: minimal metadata required. X (Twitter) had low requirements, with the Generative AI Community [63] requiring only a post title. We note that all X Communities only allow public accounts to contribute to making posts within the group.

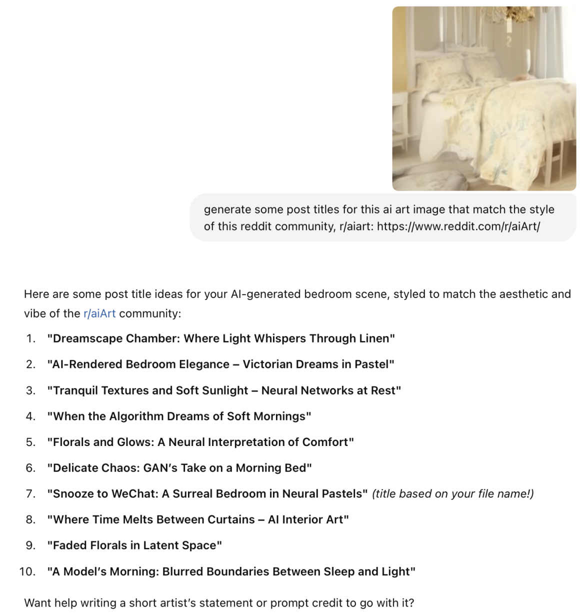
Media sharing: minimal metadata required. In our testing, media-sharing platforms such as Imgur [41], DeviantArt [3], and Rednote [25] also had minimal requirements. Note that Imgur [41] has an optional tag which discloses if the uploaded piece was generated using AI; this was enabled during all test uploads. These platforms only required a post title, which based on our observations are informal in nature and can be presented in a variety of styles – comedic, descriptive, but consistently brief.

Discussion board: More metadata required. Reddit's forum structure has more involved metadata requirements, with posts requiring a title, caption, and *flair*, which is a tag that categorizes a post. They are created by community moderators and can vary from place to place. For example, /r/AIArt [52] has 25 flair options and a write-in section to denote the type of generator used to make the attached image. During our testing, all images posted to /r/AIArt had the "Diffusion" flair. [52] Conversely, /r/AIArtwork [53] only asks for the type of post being made, with options such as discussion, artwork, news, guide, and series. Similar to media sharing metadata, the title and captions are informal and brief.

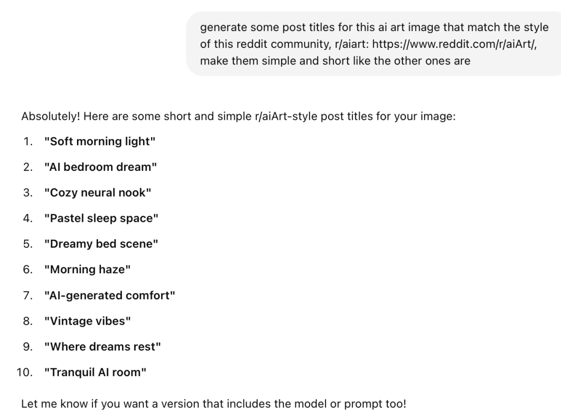
Generating metadata. Pulsar does not generate accompanying metadata with its encoded images. So, we decided to use a large language model to gauge its effectiveness at matching metadata.[5] We used ChatGPT-4o to generate captions and titles for our experimental posts. We note that ChatGPT and other large language models tend to output generation with unnatural or overly polished language, which can stand out. We found that the most effective way to generate metadata was to include an example of the target style – such as a screenshot or link of existing posts – and to use keywords such as "simple" and "short" in the prompt. Figure 4 illustrates this contrast, showing how better prompting can significantly improve the quality of metadata generation.

Figure 5 presents an encoded image deployed in a potential channel, showing metadata qualities we discuss. The encoded image does not attract undue attention and appears like a natural community output in a search for a "bedroom". We deleted this post once we verified this behavior.

Image styles. In addition to the metadata required for each post, communities have common stylistic tropes that should be matched by any Assemblage vector. We provide examples of image styles we observed in our analysis in Appendix A.



(a) Unnatural sounding metadata generated by ChatGPT-4o.



(b) Natural metadata output from ChatGPT-4o.

Figure 4: Examples of metadata generations from ChatGPT: (a) a verbose version and (b) a more natural sounding version by using keywords in the prompt.

4.2 Performance

We now turn to the performance of Assemblage. We first study the efficiency of all stages of the message layer when sending messages short enough to fit in one vector. We then study how messages spread out over several messages using rateless erasure codes fare. All evaluations were run on a MacBook Pro with an M4 Pro system-on-chip running macOS Sequoia 15.6.

Efficiency of vectors. For our first set of experiments, we ran each stage of the Assemblage message layers – offline, send, and receive – 30 times on each of the four models mentioned above. We discard trials that do not result in a successful offline state due to the probabilistic nature of Pulsar [43]. We wish to study runtime and disk storage required for each operation in detail, and as such we choose a message length small enough to fit within a single vector; we study longer messages (using rateless erasure codes) below. Also, given the number of images generated, we do not perform rendezvous on a live platform in these experiments, focusing instead on efficiency on local hardware.

[5]As an aside, by using another form of AI-generated content as a part of Assemblage rendezvous, one could transmit additional messages using post *metadata* as a vector, perhaps by using text-based steganography such as Meteor [44] or Discop [18].

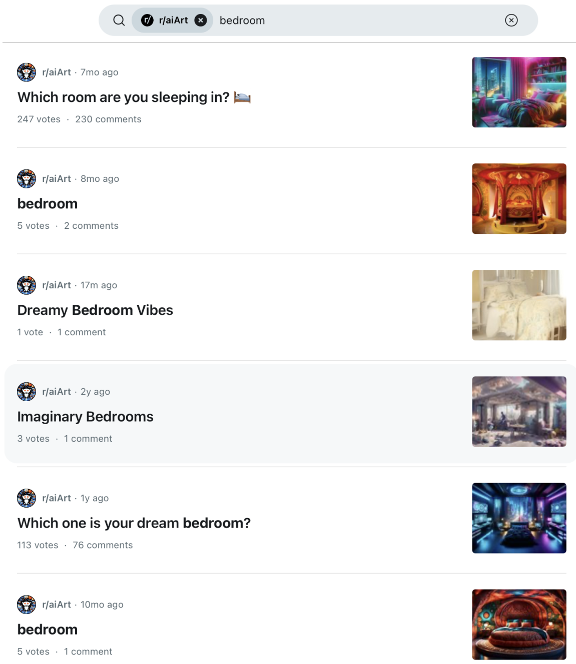


Figure 5: Reddit search results from /r/AiArt showing results for the term "bedroom". The image vector "Snooze" is the 3rd result from top. This post has since been deleted.

Our full results can be found in Table 2. One generation of a local offline state – loading the AI model, running the model to create the state, and serializing it for storage – takes ~9.5 sec, and each local state requires only ~5.6 MB on disk. The encoding phase is similarly fast, taking ~4.4 sec to go from a local state on disk to a saved image ready to be uploaded. Each generated image vector holds ~300-600 B on average. Finally, we measured that perceptual hash detection of image vectors takes ~0.0013 sec and steganographic decoding after that is ~4.2 sec; we did not observe any false positive hash matches in our tests. These experimental results show our design is efficient enough for deployment.

Efficiency with rateless erasure codes. In Section 3, we describe how our message layer takes advantage of rateless erasure codes [11, 47], as in Collage [10], to allow for arbitrary size messages to be sent over vectors in any order. Thus, achieve a truly end-to-end evaluation, we also wish to measure the impact of rateless erasure codes on the performance of Assemblage.

We instantiate rateless erasure coding using Raptor codes [55], as implemented by the Python `raptorq` library. The reconstruction probability for the decoder after receiving $K+h$ blocks is $1-\frac{1}{256}^{h+1}$, where K is the number of blocks of the original message, and h is the number of extra blocks received. We configure `raptorq`'s encoder to $h = \lceil K \cdot 0.5 \rceil$, such that 50% more blocks are generated (rounded up). We configure the decoder to reconstruct the message based on some size K subset of the $K + h$ generated blocks.[6]

[6]We choose $h = 0$ during recovery as the recovery probability is $1 - \frac{1}{256} \approx 99.6\%$. We did not see any failures to decode Raptor-encoded blocks during our experiments, though some Assemblage users may wish to use more messages to be more robust.

For our experiments, we combine Raptor codes with our existing Pulsar-based Assemblage implementation. To generate the data used as input, we randomly select 300 English words using the Python `wonderwords` library, and then apply `zlib` compression to the output. The idea here is to support a realistic long-form message that requires multiple vectors to transmit. Next, we create $K + h$ Raptor-encoded blocks from this data, and perform the offline() and send() operations in sequence to create $K + h$ Pulsar vectors. We then perform the receive() operation on K vectors to recover the Raptor code blocks, and finally recover the original message.

We performed 30 experiments per model, and Table 3 shows the results of our experiment. We provide information on the size of the input data, runtime of the Raptor encoder and decoder, as well as runtime of offline(), send(), and receive() on the Raptor-encoded message blocks. Note that K is dependent on the capacity of each image vector; as a safe estimate, we use a vector capacity of $\bar{x} - s$ (i.e., one standard deviation below the mean) from Table 2 for each model. But, this estimate is not exact, so there may be some scenarios where a larger or smaller K is necessary for a specific image. As such, in Table 3, the values of K and h are the mode.

Based on our results, rateless erasure coding itself has a marginal effect on the runtime of Assemblage, with Raptor code blocks being generated almost instantaneously. Also, with compression, sending 300 English words over Assemblage requires only $K + h = 4 + 2$ image vectors using our best model (church-256), which can be sent (Raptor encoding, offline(), and send()) in less than 90 seconds on the laptop used for our experiment, and received (receive() and Raptor decoding) in less than 30 seconds.

The church-256 model had the lowest times because it generates higher capacity vectors on average. As mentioned previously, K varies with vector capacity, and as K increases, the number of vectors generated increases as well; more vectors mean more offline states, which is where the bottleneck is, as we saw in Table 2. celebahq-256 and bedroom-256 have lower capacity but still do not require too many vectors to transmit 300 words, and therefore are suitable for deployment. The low capacity of cat-256, however, results in an impractically large amount of images generated (over double than that of the next model, bedroom-256), meaning it is far slower end-to-end than the other models.

Our current implementation performs offline() on-demand to generate offline states before a send(). We note that, in a production workflow, offline() to generate dozens of offline states in the background before any message was ready to be sent, as discussed in Section 3.2. Given the relatively low size of an offline state (Table 2), and the relative speed of send() and receive(), this approach would result in a better deployment. Additionally, our use of $\bar{x} - s$ from Table 2 as an estimate of the capacity is somewhat conservative and is likely a lower bound. By generating offline states before sending a message, we could get a precise value for K and optimize the number of vectors that we have to send() and receive().

Moreover, 4–6 image posts (K for most of our models) are relatively few in the context of the hundreds of images that are shared on some of these communities every day, especially if all generated image vectors are shared as an "album" of related images on the AI-generated content community. Sending very large messages or using lower-quality models (such as cat-256) may require dozens of image vectors, however; to meet deniability, image vectors may

Table 2: Performance of our scheme over n successful trials for on the bedroom-256 ($n = 26$), cat-256 ($n = 28$), celebahq-256 ($n = 29$), and church-256 ($n = 28$) models across message layer operations. Trials vary due to the probabilistic nature of Pulsar [43]. Runtimes (sec) and data sizes (bytes) are presented as $\bar{x} \pm s$, where $\bar{x}$ is the mean and s is the standard deviation. For some runtimes, we only provide the mean $\bar{x}$ as the standard deviations are very small ($\approx 10^{-6}$ sec).

Operation	**bedroom-256** [29]	**cat-256** [30]	**celebahq-256** [31]	**church-256** [32]
offline()				
Model Load Runtime	0.3914 ± 0.0534 sec	0.2836 ± 0.0669 sec	0.3900 ± 0.0269 sec	0.4239 ± 0.0935 sec
Model Runtime	9.1887 ± 0.0192 sec	9.2075 ± 0.0237 sec	9.1808 ± 0.0218 sec	9.2893 ± 0.3073 sec
State Serialization Runtime	0.0069 ± 0.0001 sec	0.0068 ± 0.0006 sec	0.0069 ± 0.0006 sec	0.0073 ± 0.0005 sec
Perceptual Hash Runtime	0.0041 ± 0.0001 sec	0.0043 ± 0.0002 sec	0.0042 ± 0.0002 sec	0.0046 ± 0.0023 sec
Vector Capacity	376.19 ± 142.30 bytes	368.64 ± 262.26 bytes	417.55 ± 89.30 bytes	618.21 ± 210.28 bytes
Serialized Local State Size	5.6816 ± 0.0648 MB	5.6649 ± 0.0908 MB	5.6926 ± 0.0416 MB	5.7597 ± 0.0849 MB
Local State Hash Size	3.9547 ± 0.0005 KB	3.9546 ± 0.0006 KB	3.9547 ± 0.0005 KB	3.9545 ± 0.0010 KB
send()				
Local State Deserialization Runtime	0.0027 ± 0.0001 sec	0.0027 ± 0.0001 sec	0.0027 ± 0.0001 sec	0.0028 ± 0.0003 sec
Model Load Runtime	0.2701 ± 0.0293 sec	0.2140 ± 0.0185 sec	0.2683 ± 0.0241 sec	0.0213 ± 0.0213 sec
Pulsar Encoding Runtime	4.1725 ± 0.0466 sec	4.1388 ± 0.0490 sec	4.1949 ± 0.0417 sec	4.2293 ± 0.2127 sec
Image Save Runtime	0.0558 ± 0.0007 sec	0.0563 ± 0.0007 sec	0.0559 ± 0.0007 sec	0.0567 ± 0.0007 sec
Image Size	383.9161 ± 3.8510 KB	372.4326 ± 9.8040 KB	386.7969 ± 2.9960 KB	378.6861 ± 7.4133 KB
receive()				
Perceptual Hash Runtime	3.33×10^{-4} sec	3.36×10^{-4} sec	4.04×10^{-3} sec	3.00×10^{-4} sec
Find Hash Match Runtime	1.54×10^{-5} sec	2.54×10^{-5} sec	4.48×10^{-5} sec	3.53×10^{-5} sec
Local State Deserialization Runtime	0.0038 ± 0.0005 sec	0.0037 ± 0.0003 sec	0.0038 ± 0.0038 sec	0.0036 ± 0.0002 sec
Image Load Runtime	0.0090 ± 0.0004 sec	0.0112 ± 0.0084 sec	0.0092 ± 0.0007 sec	0.0103 ± 0.0076 sec
Pulsar Decoding Runtime	4.1976 ± 0.1831 sec	4.1191 ± 0.2741 sec	4.2463 ± 0.1409 sec	4.3999±0.2671 sec

have to be posted on multiple communities or posted over the span of multiple hours. But, because even hundreds of words can be sent in just a few images, our results show that the rateless erasure coding approach in Assemblage improves its practical applicability.

5 Discussion and Future Work

Broadly, our work is a first step towards recent steganography research realizing its lofty censorship resistance aspirations. We conclude by discussing insights from our design and results.

Feasibility. Through our evaluation, we can see that image-based generative steganography is indeed a feasible technique for censorship circumvention. Our investigations show there are several active communities where posting AI images is commonplace, and our Assemblage design can be established over many of them.

Moreover, our efficiency results demonstrate the practicality of Assemblage. For instance, generating 30 local states – more than enough for sending thousands of message bytes – takes only ~5 minutes of runtime on a laptop, and ~150 MB of disk space. In terms of message capacity, note that each image vector can hold at least one standard X/Twitter post (280 B) on average, and potentially 2 depending on the model/image vector. Based on our experiments with rateless erasure codes, the cat-256 model may have too low capacity to be deployable for longer messages, but the rest of the models (and especially church-256) are performant enough for deployment. Also, our perceptual hashing approach is practical for detecting Assemblage stegotexts; perceptual hashing every image posted to /r/AIArt in one day (~150 [52]), for instance, would require less than a second. Thus, decoders do not have to expend excess compute trial decoding to participate in Assemblage.

We show good efficiency on our laptop hardware, but we expect that a significant number of users would run Assemblage over smartphone hardware as well. Given the rise of accelerated machine learning hardware on mobile devices [38, 39], paralleling developments in laptops, generative steganography could be reasonably deployed on smartphones. But, as these devices have resource and battery constraints, future work must develop new optimizations and perhaps even new algorithms for generative steganography.

Image compression. Table 1 shows that our candidate design can be deployed on major social media platforms, and is even resilient to some compression. But, our design fails under lossy compression, as a result of the underlying Pulsar scheme; while it does include error correction, significant modifications to the pixel data – even if the visual appearance is preserved – can destroy any embedded content. So, rather than just looking at increasing the capacity of generative image steganography [33, 50], more research into *modification resistance* is necessary, which will require new approaches in cryptography, machine learning, and/or coding theory. Assemblage failed on two platforms known for censorship – WeChat and Rednote – for this reason, making the need especially acute.

Bootstrapping. Our results in Section 4 show that generative steganography is already feasible for deployment, both in terms of rendezvous locations and efficiency. But, an inherent limitation

Table 3: Performance of rateless error coding using the average image sizes of vectors generated by the bedroom-256, cat-256, celebahq-256, and church-256 models. Data sizes (bytes) are presented as $\bar{x} \pm s$, where $\bar{x}$ is the mean and s is the standard deviation. The average number of vectors required to send a message K and number of extra vectors generated h are also provided, based on the capacity $\bar{x} - s$ in Table 2 for each model. Runtimes (sec) are computed for each message operation over these generated vectors. For some runtimes, we only provide the mean $\bar{x}$ as the standard deviations are very small ($\approx 10^{-6}$ sec).

Operation	**bedroom-256** [29]	**cat-256** [30]	**celebahq-256** [31]	**church-256** [32]
offline()				
Data Length	2408.60 ± 38.50 bytes	2424.33 ± 41.41 bytes	2403.06 ± 46.47 bytes	2421.73 ± 47.84 bytes
Compressed Data Length	1328.43 ± 20.88 bytes	1333.46 ± 22.62 bytes	1322.80 ± 22.33 bytes	1332.96 ± 24.31 bytes
Vectors Generated	$K = 6$, $h = 3$	$K = 13$, $h = 7$	$K = 5$, $h = 3$	$K = 4$, $h = 2$
offline() Runtime	88.22 ± 0.41 sec	194.05 ± 3.96 sec	72.79 ± 8.88 sec	60.86 ± 3.23 sec
send()				
Raptor Encoder Setup Runtime	1.66×10^{-4} sec	1.95×10^{-4} sec	1.68×10^{-4} sec	1.70×10^{-4} sec
Raptor Encode Runtime	1.30×10^{-5} sec	1.59×10^{-5} sec	9.97×10^{-6} sec	9.27×10^{-6} sec
send() Runtime	27.52 ± 0.34 sec	63.06 ± 2.03 sec	21.52 ± 2.25 sec	18.41 ± 0.36 sec
receive()				
Raptor Decoder Setup Runtime	5.64×10^{-5} sec	5.82×10^{-5} sec	6.12×10^{-5} sec	6.05×10^{-6} sec
Raptor Decode Runtime	1.51×10^{-4} sec	1.46×10^{-4} sec	1.33×10^{-4} sec	1.55×10^{-4} sec
receive() Runtime	27.27 ± 0.39 sec	55.68 ± 0.65 sec	21.34 ± 2.16 sec	19.07 ± 0.23 sec

of most generative steganographic schemes is that it assumes the sender and receiver share a symmetric key. Public-key steganography has been shown in the theoretical literature [60], and even through some generative models [44, 68], but currently does not efficiently support channels common on the Internet like images and text, so we do not use it in Assemblage. Indeed, bootstrapping shared state was a problem even in the original Collage work [10].

We argue that, despite these limitations, there are still situations where Assemblage would be viable. First, consider situations where users *transit* between censored and uncensored regions. For instance, a journalist may be traveling on assignment to a more repressive environment, and can take the shared secret with them (either on disk or as a seed phrase) so they can instantiate Assemblage when needed to communicate with the outside world. Next, many users are in environments where censorship is active in waves [1]; a user can establish the shared secret when tensions are lower, and use Assemblage when censorship is reactivated. Finally, we could also utilize a hybrid approach: a one-time-use signaling channel [59] could establish the shared secret, and Assemblage could then be used for any subsequent communication.

Communities. In Section 3.3, we discuss suitable tasks and vectors for Assemblage, and provide further analysis in Section 4.1. But, for Assemblage to blend in, we must better understand our cover distribution: AI image communities. The primary question: how do we create authentic-looking AI community accounts, posts, and metadata? We provide some initial results towards that question in this work. But, another question here is how online communities will be impacted if they were used as a drop location. While we would hope for solidarity, we cannot expect it, especially since this system may lead to additional scrutiny on the community's posts. Moreover, we found that some communities have restrictions on which accounts can post (e.g., based on account age, posting history, or relationship to existing community members).

The answers to the above questions require future research. We propose working with the members of AI image communities more explicitly, such as through a need-finding study. Any deployment would have to work within – not around – these human factors considerations if we wish to achieve the most capable deployment.

Concluding thoughts. We emphasize that we do not make value judgments about the *nature* of the AI-generated content communities. Opinions are mixed on the societal benefits of AI image generation [8, 45, 49] and if its outputs constitute art [13, 34]; we do not settle this debate here. In any case, it is clear that communities around sharing AI images are popular, and our results show Assemblage is resilient and efficient. So, as long as these communities exist, we can repurpose AI through generative steganography to chip away at censorship.

Acknowledgments

The authors would like to thank Tasnim Hanaif and Ana Stanisavljevic for their work implementing a prototype that applied perceptual hashing to Pulsar offline states.

This work is supported by PSC-CUNY Research Awards from the City University of New York and the U.S. Department of Education under the TREAD program at the City College of New York. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily those of the sponsors.

References

[1] Access Now. 2024. Shrinking democracy, growing violence: Internet shutdowns in 2023. https://www.accessnow.org/wp-content/uploads/2024/05/2023-KIO-

Report.pdf. Online; accessed 1 Sept 2024.

[2] Atlas Art. 2025. Telegram Atlas AI Art. @atlasart_collection.

[3] Deviant Art. 2000. Deviant Art - AI Art. https://www.deviantart.com/tag/ai.

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

[5] Michael Backes and Christian Cachin. 2005. Public-Key Steganography with Active Attacks. In *TCC 2005 (LNCS, Vol. 3378)*, Joe Kilian (Ed.). Springer, Berlin, Heidelberg, 210–226. https://doi.org/10.1007/978-3-540-30576-7_12

[6] Shumeet Baluja. 2017. Hiding Images in Plain Sight: Deep Steganography. In *Neural Information Processing Systems*. http://www.esprockets.com/papers/nips2017.pdf

[7] Luke A Bauer, James K Howes, Sam A Markelon, Vincent Bindschaedler, and Thomas Shrimpton. 2024. Leveraging generative models for covert messaging: Challenges and tradeoffs for" dead-drop" deployments. In *Proceedings of the Fourteenth ACM Conference on Data and Application Security and Privacy*. 67–78.

[8] Oliver Bendel. 2023. Image synthesis from an ethical perspective. *AI & SOCIETY* (2023), 1–10.

[9] Flavio Bertini, Rajesh Sharma, and Danilo Montesi. 2022. Are social networks watermarking us or are we (unawarely) watermarking ourself? *Journal of Imaging* 8, 5 (2022), 132.

[10] Sam Burnett, Nick Feamster, and Santosh Vempala. 2010. Chipping Away at Censorship Firewalls with User-Generated Content. In *USENIX Security 2010*. USENIX Association, 463–468.

[11] John W Byers, Michael Luby, Michael Mitzenmacher, and Ashutosh Rege. 1998. A digital fountain approach to reliable distribution of bulk data. *ACM SIGCOMM Computer Communication Review* 28, 4 (1998), 56–67.

[12] Christian Cachin. 2000. An Information-Theoretic Model for Steganography. Cryptology ePrint Archive, Report 2000/028. https://eprint.iacr.org/2000/028

[13] Eva Cetinic and James She. 2022. Understanding and creating art with AI: Review and outlook. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)* 18, 2 (2022), 1–22.

[14] Ching-Yun Chang and Stephen Clark. 2010. Practical linguistic steganography using contextual synonym substitution and vertex colour coding. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 1194–1203.

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

[16] DemandSage. 2025. Reddit User Statistics 2025. https://www.demandsage.com/reddit-statistics.

[17] DemandSage. 2025. Telegram User Statistics 2025. https://www.demandsage.com/telegram-statistics.

[18] Jinyang Ding, Kejiang Chen, Yaofei Wang, Na Zhao, Weiming Zhang, and Nenghai Yu. 2023. Discop: Provably Secure Steganography in Practice Based on "Distribution Copies". In *2023 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 2238–2255. https://doi.org/10.1109/SP46215.2023.10179287

[19] Discord. 2022. Discord Discover - ai Art. https://discord.com/servers/ai-art-1016879474928795718.

[20] Discord. 2022. Discord Discover - Night Cafe Lounge. https://discord.com/servers/nightcafe-lounge-943306099019370566.

[21] Discord. 2023. Discord Discover - AI Hub by Weights. https://discord.com/invite/aihub.

[22] Discord. 2025. About Discord - Our Mission. https://discord.com/company.

[23] Kevin P. Dyer, Scott E. Coull, Thomas Ristenpart, and Thomas Shrimpton. 2013. Protocol misidentification made easy with format-transforming encryption. In *ACM CCS 2013*, Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung (Eds.). ACM Press, 61–72. https://doi.org/10.1145/2508859.2516657

[24] Facebook. 2025. ThreatExchange GitHub repository. https://github.com/facebook/ThreatExchange/tree/master/pdq.

[25] FastCompany. 2024. With Its New App, DeviantArt Finally Has A Mobile Place For Those 65 Million Monthly Visitors. https://www.fastcompany.com/deviantart-app.

[26] David Fifield, Chang Lan, Rod Hynes, Percy Wegmann, and Vern Paxson. 2015. Blocking-resistant communication through domain fronting. *PoPETs* 2015, 2 (April 2015), 46–64. https://doi.org/10.1515/popets-2015-0009

[27] Jessica Fridrich, Miroslav Goljan, and Dorin Hogea. 2002. Attacking the outguess. In *Proceedings of the ACM Workshop on Multimedia and Security*, Vol. 2002.

[28] Claire Fu and Meaghan Tobin. 2025. Red Note, a Chinese app, is dominating downloads, thanks to TikTok users. https://www.nytimes.com/2025/01/17/us/politics/red-note-a-chinese-app-is-dominating-downloads-thanks-to-tiktok-users.

[29] Google. 2022. ddpm-bedroom-256. https://huggingface.co/google/ddpm-bedroom-256.

[30] Google. 2022. ddpm-cat-256. https://huggingface.co/google/ddpm-cat-256.

[31] Google. 2022. ddpm-celebahq-256. https://huggingface.co/google/ddpm-celebahq-256.

[32] Google. 2022. ddpm-church-256. https://huggingface.co/google/ddpm-church-256.

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

[34] Aaron Hertzmann. 2018. Can computers create art?. In *Arts*, Vol. 7. MDPI, 18.

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

[36] Nicholas J. Hopper, John Langford, and Luis von Ahn. 2002. Provably Secure Steganography. In *CRYPTO 2002 (LNCS, Vol. 2442)*, Moti Yung (Ed.). Springer, Berlin, Heidelberg, 77–92. https://doi.org/10.1007/3-540-45708-9_6

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

[38] Han Hu, Yujin Huang, Qiuyuan Chen, Terry Yue Zhuo, and Chunyang Chen. 2023. A first look at on-device models in iOS apps. *ACM Transactions on Software Engineering and Methodology* 33, 1 (2023), 1–30.

[39] Andrey Ignatov, Radu Timofte, William Chou, Ke Wang, Max Wu, Tim Hartley, and Luc Van Gool. 2018. Ai benchmark: Running deep neural networks on android smartphones. In *Proceedings of the European Conference on Computer Vision (ECCV) Workshops*. 0–0.

[40] Imgur. 2025. Imgur - Our Journey. https://imgurinc.com/.

[41] Imgur. 2025. Imgur #aiart. https://imgur.com/t/aiart.

[42] Tushar M. Jois, Gabrielle Beck, and Gabriel Kaptchuk. 2024. *Artifact for "Pulsar: Secure Steganography for Diffusion Models"*. https://doi.org/10.5281/zenodo.13324377

[43] Tushar M. Jois, Gabrielle Beck, and Gabriel Kaptchuk. 2024. Pulsar: Secure Steganography for Diffusion Models. In *ACM CCS 2024*, Bo Luo, Xiaojing Liao, Jun Xu, Engin Kirda, and David Lie (Eds.). ACM Press, 4703–4717. https://doi.org/10.1145/3658644.3690218

[44] Gabriel Kaptchuk, Tushar M. Jois, Matthew Green, and Aviel D. Rubin. 2021. Meteor: Cryptographically Secure Steganography for Realistic Distributions. In *ACM CCS 2021*, Giovanni Vigna and Elaine Shi (Eds.). ACM Press, 1529–1548. https://doi.org/10.1145/3460120.3484550

[45] Amelia Katirai, Noa Garcia, Kazuki Ide, Yuta Nakashima, and Atsuo Kishimoto. 2024. Situating the social issues of image generation models in the model life cycle: a sociotechnical approach. *AI and Ethics* (2024), 1–18.

[46] Weixin Liang, Yaohui Zhang, Mihai Codreanu, Jiayu Wang, Hancheng Cao, and James Zou. 2025. The Widespread Adoption of Large Language Model-Assisted Writing Across Society. *arXiv preprint arXiv:2502.09747* (2025).

[47] Petar Maymounkov. 2002. *Online codes*. Technical Report. Technical report, New York University.

[48] Hooman Mohajeri Moghaddam, Baiyu Li, Mohammad Derakhshani, and Ian Goldberg. 2012. SkypeMorph: protocol obfuscation for Tor bridges. In *ACM CCS 2012*, Ting Yu, George Danezis, and Virgil D. Gligor (Eds.). ACM Press, 97–108. https://doi.org/10.1145/2382196.2382210

[49] Jonas Oppenlaender, Johanna Silvennoinen, Ville Paananen, and Aku Visuri. 2023. Perceptions and realities of text-to-image generation. In *Proceedings of the 26th International Academic Mindtrek Conference*. 279–288.

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

[51] Jonathan Prokos, Neil Fendley, Matthew Green, Roei Schuster, Eran Tromer, Tushar M. Jois, and Yinzhi Cao. 2023. Squint Hard Enough: Attacking Perceptual Hashing with Adversarial Machine Learning. In *USENIX Security 2023*, Joseph A. Calandrino and Carmela Troncoso (Eds.). USENIX Association, 211–228.

[52] Reddit. 2017. /r/AiArt. https://www.reddit.com/r/aiArt.

[53] Reddit. 2017. /r/AIArtwork. https://www.reddit.com/r/AIArtwork.

[54] Telegram Robots, Art AI Art, and Pictures. 2025. Telegram Robots and Art. https://t.me/images_pictures.

[55] Amin Shokrollahi. 2006. Raptor codes. *IEEE transactions on information theory* 52, 6 (2006), 2551–2567.

[56] Gustavus J. Simmons. 1983. The Prisoners' Problem and the Subliminal Channel. In *CRYPTO'83*, David Chaum (Ed.). Plenum Press, New York, USA, 51–67. https://doi.org/10.1007/978-1-4684-4730-9_5

[57] Sinorbis. 2025. WeChat Statistics. https://blog.sinorbis.com/wechat-statistics.

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

[59] Paul Vines, Samuel McKay, Jesse Jenter, and Suresh Krishnaswamy. 2024. Communication Breakdown: Modularizing Application Tunneling for Signaling Around Censorship. *Proceedings on Privacy Enhancing Technologies* (2024).

[60] Luis von Ahn and Nicholas J. Hopper. 2004. Public-Key Steganography. In *EUROCRYPT 2004 (LNCS, Vol. 3027)*, Christian Cachin and Jan Camenisch (Eds.). Springer, Berlin, Heidelberg, 323–341. https://doi.org/10.1007/978-3-540-24676-3_20

[61] Qiyan Wang, Xun Gong, Giang T. K. Nguyen, Amir Houmansadr, and Nikita Borisov. 2012. CensorSpoofer: asymmetric communication using IP spoofing for censorship-resistant web browsing. In *ACM CCS 2012*, Ting Yu, George Danezis, and Virgil D. Gligor (Eds.). ACM Press, 121–132. https://doi.org/10.1145/2382196.2382212

[62] Zachary Weinberg, Jeffrey Wang, Vinod Yegneswaran, Linda Briesemeister, Steven Cheung, Frank Wang, and Dan Boneh. 2012. StegoTorus: a camouflage proxy for the Tor anonymity system. In *ACM CCS 2012*, Ting Yu, George Danezis, and Virgil D. Gligor (Eds.). ACM Press, 109–120. https://doi.org/10.1145/2382196.2382211
[63] X. 2006. Generative AI. https://x.com/i/communities/1601841656147345410.
[64] X. 2025. X User Statistics 2025. https://www.demandsage.com/telegram-statistics.
[65] Zeyi Yang. 2022. There's no Tiananmen Square in the new Chinese image-making AI. MIT Technology Review.
[66] Zhongliang Yang, Shuyu Jin, Yongfeng Huang, Yujin Zhang, and Hui Li. 2018. Automatically Generate Steganographic Text Based on Markov Model and Huffman Coding. arXiv:1811.04720 [cs.CR]
[67] Ece Yildirim. 2025. ChatGPT delays rollout of image generator for free users because it's too 'popular'. Quartz.
[68] Xin Zhang, Kejiang Chen, Jinyang Ding, Yuqi Yang, Weiming Zhang, and Nenghai Yu. 2024. Provably secure public-key steganography based on elliptic curve cryptography. *IEEE Transactions on Information Forensics and Security* 19 (2024), 3148–3163.
[69] Summer Zhen. 2024. Asia Hedge funds eye Chinese tech leading AI charge in 2025. Reuters.

A Styles of Generated Images

Generating images in styles that are popular within the receiving channel is similar to having correct metadata: shaping how the image is perceived by aligning it with community norms. In our preliminary analysis of these communities, the two most dominant observed styles were "cinematic fantasy" and "realistic parody". We provide representative examples in Figure 6. Cinematic fantasy pieces are heavily genre-driven, often featuring folk and storybook creatures and places in dramatic illustration. In contrast, realistic parody applies a photorealistic lens to fictitious scenarios, visualizing comically absurd situations grounded in reality. This style is notably reactive to culture and trends, leaning into ironic messaging or social and political commentary. More research is necessary to definitively identify trends and more closely match art styles.

(a) A man in a futuristic city facing a portal to a magical realm done in a cinematic illustrative style.

(b) An anachronistic birthday party with famous historical figures and a fictitious cat generated in a photorealistic style.

Figure 6: Examples of popular art styles observed across investigated channels: (a) cinematic fantasy and (b) realistic parody. These are sourced from the communities, and do not contain steganographic content.