Compressing steganographic payloads with LLM assistance

Jaisal Ahmadullah
Latymer Upper School
London, United Kingdom
jaisal.ahmadullah@gmail.com

Abstract

Steganography is the practice of concealing messages or information within other non-secret text or media to avoid detection. A central challenge in steganography is balancing payload size with detectability and media constraints—larger payloads increase the risk of detection and require proportionally larger or higher-capacity carriers. In this paper, we introduce a novel approach that combines Huffman coding, suitable dictionary identification, and large language models (LLMs) rephrasing techniques to significantly reduce payload size. This enables more efficient use of limited-capacity carriers, such as images, while minimizing the visual or statistical footprint. Our method allows for the embedding of larger payloads into fixed-size media, addressing a key bottleneck in traditional steganographic systems. By optimizing payload compression prior to encoding, we improve both the stealth and scalability of steganographic communication.

1 Introduction

Censorship arises both in authoritarian regimes, whereby a government restricts the free flow of information, as well as in the corporate world in cases where a corporation wishes to silence employees and potential whistleblowers. Escaping such censorship requires the creation of undetectable methods for transmitting and receiving information. This is the field of *steganography*.

Steganography is different from encryption in that messages are "hidden in plain sight". In contrast, encryption hides the content of the messages but not its existence. That said, the two techniques work in tandem: a message can first be encrypted and then sent using steganography.

Steganography has historically been used in a variety of contexts. Firstly, intelligence agencies and the military can use steganography during conflicts or in espionage operations. This is especially important when knowledge of a message being transmitted in the first instance is something which one wants to hide.

Secondly, civilians—especially journalists and civil right activists—may use steganography to communicate among one another in authoritarian regimes; to protect their messages from censorship.

Finally, companies also have uses for steganography; they may for instance embed watermarks in digital media to prove copyright ownership to detect piracy.

Despite its unique advantages, steganography faces several technical and practical challenges that hinder its widespread and effective use. One of the most fundamental difficulties is to achieve a balance between transmission rate and stealthiness. The more plaintext data that one adds to a file, the easier it is for an analysis tool to detect it [17].

Steganography also faces the issue of a limited payload size. In particular, the size of a payload that can be encoded into a media form (an image, for instance) is constrained by the data size of the media form. For example, if you have an image which is 1MB you may only be able to encode a payload which is 30 bytes.

Note that these constraints severely restrict the amount of data one can encode into a reasonably sized image or media form. Thus, any method which could allow larger payloads to be encoded for the same-sized image would be highly beneficial, as larger images with more data are both more suspicious and easier to detect with analysis tools.

There are several tools that utilize steganography. Examples include Stegano[8], HStego[3], Stego-retweet[9] (which hides messages using Twitter/X), and OpenStego[6].

The primary limitation of these tools is that they all have an upper limit in the size of a payload which can be hidden for a given image or media form size. This means that these tools fail to hide several larger payloads.

1.1 Our Proposal

Our proposal seeks to reduce the size of payloads by means of compression, in an effective manner such that larger payloads can be sent with the same size image.

Our approach is to leverage public online data sources as dictionaries to compress the payloads, and then use LLMs to further improve the dictionary fit and reduce the size of original message. Finally, we combine this with lossless compression (e.g., 7zip), achieving significant compression of the plaintext—often around 50% in our evaluation.

In more detail, our approach first identifies a suitable dictionary (e.g., Wikipedia page). Suitability is established locally with a copy of a web index (e.g., Common Crawl[1]), and with the use of distance metrics such as cosine similarity to the plaintext. Then, with the use of local (very small) LLMs we rephrase the plaintext to increase the cosine similarity and reduce the number of words and characters—thereby further improving compression. Finally, a shortened URL to the public dictionary in addition to the compressed payload is further compressed with lossless compression like zlib. This is the final payload that is then encoded into a media file using stegranography (we use Stegano [8]).

Cryptanalysis tools designed to detect steganography, such as Aletheia [17] can be very effective in detecting steganography. Our approach results in most of these tools being worse at detecting payloads in images, as less data is hidden in an image. However, it should be noted that "tamper checks" which work when a tool like Aletheia compares a stenographic image to its source, are still an effective way to detect the use of steganography. Fortunately, this is rare as it requires an attacker to have the raw, unedited image. However, several tools can also conduct Google Reverse Image searches and find the original image in some cases.

We note that as with all technologies, steganography can be used for malicious purposes, such as enabling criminals to securely

communicate without detection of their communications in the first place. This is not the intent of our work, but we nevertheless acknowledge the moral and ethical implications of our work.

2 Background

2.1 Background on steganography

Steganography is a technique to hide information in "plain sight". The earliest recorded instances of steganography date back to 440BC in ancient Greece, as described by Herodotus in Histories. He recounts how Histiaeus sent a secret message to his vassal, Aristagoras, by shaving the head of a trusted servant, writing the message onto his scalp, and then waiting for the hair to grow back before sending him on his journey.

Steganography is still used today, though documentation is very sparse; after all, successful use of this technology implies nobody learns about it. That said, there are documented cases of industrial espionage; several sources show that Chinese intelligence personnel have commonly used this method [4].

2.2 Our setting

There are two common cases for our work.

In Scenario A, two users Alice and Bob wish to engage in covert messaging. Whilst Alice and Bob know one another, they are in different geographic locations. Alice and Bob have a shared cryptographic key that they exchanged out of band. Alice wants to send a message to Bob, but does not want a censor to stop her message.

In Scenario B, Alice wishes to engage in broadcasting whereby she wants to get a message out with no particular recipient in mind. Note that in this case there is no prior cryptographic key or other material exchanged in advance.

2.3 Threat Model

We assume that there is a censor that can monitor all communications between Alice and the world. The censor can inspect each one of Alice's messages. We assume that Alice's device is not compromised and has been pre-configured with the necessary steganographic software that she wishes to use prior to being monitored by the censor. The censor may know that Alice has steganographic software, but doesn't know when she is using it. The censor may also run steganography detecting software such as Aletheia [17]. The censor can inspect TLS traffic (e.g., may force Alice to install a root certificate in her browser or machine).

2.4 Our Goals

The goal is for Alice to successfully, without detection, send a message to Bob in scenario A or to a public forum such as Reddit or X in scenario B. Alice can use any steganographic software of her choosing. We will make the size of the message Alice wants to send smaller so that she can either send bigger messages or leave a smaller trace of her message in the picture (which will in turn improve security).

2.5 Prior and related work

Existing stego software. Modern steganography employs a variety of approaches that leverage both classical techniques and advanced computational methods. Traditionally, steganographic systems have relied on embedding secret messages within innocuous cover media such as images, audio files, or text. These methods aim to ensure that the alterations to the cover medium are imperceptible to the human eye or ear, thereby preserving plausible deniability and preventing detection.

Several software tools have been developed to automate this embedding process. For instance, OpenPuff [5] allows users to hide data in multimedia files using layered security, including cryptographic and steganographic mechanisms. Steghide [7] is another widely used tool that embeds encrypted data within lossless image and audio formats such as BMP and WAV, preserving the statistical properties of the cover medium. DeepStego [16] and SteganoGAN [20] represent more recent, AI-driven approaches, using generative adversarial networks (GANs) to embed messages within images in a way that is resistant to both statistical detection and visual analysis. These tools typically rely on either the modification of least significant bits or deep neural network models trained to hide and extract information with minimal distortion.

Formal guarantees. The work of Berndt and Liskiewicz [11] aims to understand the formal guarantees provided by steganography. This paper derives a universal, unconditionally secure protocol that achieves optimal embedding rates across any channel. Again, their approach is complementary to ours because we are not modifying the actual steganographic mechanism, but rather increasing the compression of the underlying plaintext.

Reducing size. There is a lot of work on generic data compression, but below we focus on those applied to steganography in particular.

The most closely related work to ours is work that focuses on minimizing the size of the combined file or medium that contains both the cover and the embedded message. This is typically achieved through compression techniques, entropy reduction, and optimized encoding schemes that aim to reduce the amount of visible or statistical distortion caused by embedding. For instance Sallee et al. [18] find ways to preserve or reduce JPEG image size after encoding a message in a JPEG image.

Lossless data compression methods, such as Huffman coding[14] or arithmetic coding, are traditionally applied to the payload prior to embedding, reducing message size without any loss of information. In image steganography, techniques like pixel-value differencing and transform-domain embedding target perceptually insignificant regions to optimize capacity and reduce detectability. During this process, Sallee et al. [18] explore model-based JPEG steganography built upon similar principles. Our approach diverges from these insofar as we combine a variety of techniques: we use LLM rewording to shrink the message without losing content and to better match it to a chosen dictionary; we use Huffman encoding[14] with the dictionary to further reduce the size of the message; and we finish with URL shortening and lossless compression to reduce the size of the dictionary's URL and the remaining message. This holistic approach produces the best results.

2.6 What about LLM watermarks?

At a conceptual level, both steganography and LLM watermarking rely on modifying some aspect of an output medium, whether that

be an image, audio clip, or a block of generated text—to encode hidden information. In steganography, this is typically done by subtly altering low-level features of a file, such as the least significant bits of an image pixel. In contrast, LLM watermarks involve making controlled alterations to the statistical distribution of token outputs during text generation. These changes are imperceptible to the reader but can be detected using secret keys or known statistical methods to verify whether the output was machine-generated. For instance, the work by Kirchenbauer et al. achieves this [15].

The core distinction lies in their primary use cases and threat models. Steganography is usually employed to conceal the existence of a message, serving applications such as covert communication, data exfiltration, or censorship resistance. Meanwhile, watermarking in LLMs is focused on provenance; it aims to reveal the origin of text by embedding signals that indicate whether it was generated by a particular model. In this sense, watermarking is more akin to digital fingerprinting than to covert messaging.

2.7 How do steganography detectors work?

Steganalysis, the detection of hidden messages in media—relies on identifying subtle statistical anomalies or inconsistencies that arise when information is embedded. Classical stego detectors use handcrafted features derived from image statistics, such as histograms, co-occurrence matrices, and noise residuals, to identify patterns that deviate from those found in unaltered media. An example of such a tool is Aletheia [17].

Fridrich [13] introduces a feature based way to compare two JPEG images to detect the use of steganography.

More recent detectors like the Universal Steganalytic Covariance Network (US-CovNet) [12] leverage deep learning models, especially convolutional neural networks, which can learn hierarchical features directly from data. These models have shown strong performance in detecting stego images, even when traditional statistical features are insufficient. However, their success heavily depends on training data: detectors typically need large datasets of stego and cover images generated using known embedding algorithms to generalize effectively.

Despite their growing sophistication, stego detectors face several limitations. One of the most significant is their vulnerability to cover-source mismatch, a situation where the images used during training differ in distribution from those encountered in deployment. If a stego detector is trained on one set of image sources and tested on another, its accuracy often drops dramatically. This relationship was observed by Zhang et al. [21], who found that CNNs performed worse with unseen images (usually a 7-10% drop in accuracy) although this effect was not uniform. This makes real-world deployment challenging unless the distribution of cover media is well understood and consistent.

How to avoid stego detectors? Based on the above, an effective way of evading stego detection is by guaranteeing that a photo will not be in the training data set. This can be achieved, for example, by simply taking a photo of an everyday object with a smartphone camera or using generative AI to create images with careful perturbations. Since steganalysis methods depend on comparing features to what is "typical", any significant deviation in the image content—such as from synthetic or rare photographs—can confuse detectors [21].

Ultimately, while stego detection has made considerable strides, it remains an arms race against increasingly sophisticated embedding strategies. The interplay between new media synthesis, adversarial robustness, and statistical generalization continues to define the frontier of this cat-and-mouse game.

3 Design & Implementation

Our key idea is to implement compression prior to encoding the payload into an image or media file, allowing for longer or more complex messages to be embedded. This approach is complementary to other methods which compress after the stego message has been embedded in the medium.

We use a combination of techniques to achieve this. These include standard methods such as lossless compression (e.g., using 7zip or zlib), as well as the novel use of Huffman encoding[14] with an automatically selected public dictionary. We identify suitable dictionaries, typically web pages, with high lexical overlap with the payload, using TF-IDF[19] cosine similarity. In our preliminary testing, Wikipedia has proven particularly effective due to its large article base covering a wide range of topics. Our software either computes this similarity or loads it from a cache if present.

Our existing implementation works as follows: Given a payload, the user is asked to upload an uncommon image (photo of a mundane object, or generative AI). This ensures the image diverges from existing steganography detector training datasets. The program then computes the similarity between the payload and a large set of potential dictionaries—most often a cached corpus of Wikipedia articles, though any public webpage could be used. The best-matching dictionary URL is selected and shortened using a service such as TinyURL [10], minimizing the length of the reference. This dictionary is then used for Huffman encoding [14].

In parallel, the program calculates the size of the payload under a variety of compression strategies:

- Raw compression with 7zip or zlib
- Huffman encoding using the selected dictionary
- Huffman encoding followed by 7zip or zlib
- Raw payload (no compression)

To further optimize compression, a local LLM (we use Gemma-12b-it-qat [2]) is called to rephrase the payload without altering its meaning. The LLM is also instructed to preferentially use vocabulary found in the selected dictionary, improving Huffman encoding efficiency. The reworded payload is then independently compressed using all aforementioned techniques, and the encoding method with the smallest resulting payload size is selected.

More generally, one could use Common Crawl [1] data. One would download a WET (Web Extracted Text) file, containing cleaned plaintext from much of the web. After parsing, TF-IDF [19] vectors could be precomputed for this dataset. When a payload is provided, the program would identify top-matching pages using cosine similarity, retrieve them using curl, and feed them to a local LLM. The LLM would then determine the best-fitting page and reword the payload to align more closely with its vocabulary.

On the receiving end, the decoder extracts the payload using Stegano [8], then infers the compression method from flags embedded in the payload. If the payload is a raw UTF-8 string, it is directly output. If it contains w: (Wikipedia) or t: (TinyURL) flags,

Message	Size	Wiki	Similarity	C. Size	LLM Rewrite	LLM Size	LLM C.Size	Enc (s)	Dec (s)	Comp(%)
Weather conditions unfavourable for extraction. Surveillance drone spotted circling eastern approach. Will remain at Safehouse Echo and suspend all movement until further notice. Rations sufficient.	198	Extraction	0.2124	163	Bad weather. Drone near east approach. Staying Echo. Movement halted. Rations okay.	83	86	768.13	0.02	58.08%
Entry to archive achieved via custodial route at 0215 hours. No alarms triggered. Retrieved copy of ledger as instructed. Have obscured tampering. Returning via overland courier system.	185	Tampering	0.2615	129	Archive access via staff route, 0215. No alarms. Ledger copy secured, tampering masked. Returning overland.	107	85	748.90	0.38	54.05%
Spoke to three sources at the Ministry today. None would go on record, but each confirmed a shift in internal rhetoric. Security around press briefings has tightened. I'll send the draft through the backchannel once reviewed.	227	Rhetoric (Aristotle)	0.2025	154	Three Ministry contacts confirmed shift. No names. Briefing security increased. Draft via secure route.	103	88	833.39	0.35	61.23%
It's been quiet here since you left. The rain hasn't stopped all week, and the house feels colder than usual - though maybe that's just me. I still make two cups of tea in the morning out of habit. Your books are exactly where you left them, spine-up on the armchair. I haven't touched them. I saw someone today who looked like you from behind - same coat, same walk. I almost called out. Funny how memory plays tricks on us. Write when you can. Always, M.	464	Wikipedia Wikipediaholism test	0.3094	305	Quiet here, rain constant. House colder. Two teas daily. Books untouched, armchair. Saw someone similar - coat, walk. Almost called. Memory's strange. Write soon. M.	167	132	799.35	0.33	71.55%

Figure 1: Compression statistics and LLM rewrites

the corresponding dictionary page is fetched and used for Huffman decoding. Similarly, if zlib or 7zip compression is detected, the payload is extracted using standard decompression methods.

4 Evaluation

We use a MacBook Pro (M4 Pro, 24GB of RAM). In our threat model, we assume that the user's device already contains all the tools they need to perform steganography. In our case, we have a TF-IDF cache of 1 million Wikipedia articles (16.15 GB), though we could have used the Common Crawl database (6.24 TB compressed).

Our system finds the best compression method to use, and works with several existing stegotext systems. We have tested on hundreds of examples with an average compression rate of 53.35%. The messages were randomly generated by AI with a prompt of "generate a brief message sent by a journalist, a spy, or someone traveling abroad". Figure 1 shows 4 concrete examples.

For these 4 examples, our approach achieved 61.2% compression, indicating that the average payload decreases in size by 61.2% from its original size. The size, C.Size (compressed size of the original payload), LLM Size (the size of the raw, rewritten payload by the LLM), and the LLM C.Size (the size of the rewritten LLM payload compressed), are all measured in bytes. Every time compression was used we first used huffman encoding with our chosen dictionary and then zlib of the dictionary's URL and the huffman result. The similarity metric refers to TF-IDF cosine similarity between the payload and a giving Wikipedia article (listed under Wiki). Enc (s) and Dec (s) represent the total encode and decode time in seconds.

In terms of encoding time, most comes from cosine similarity between the target text and the cache dictionaries: 93.7% of the time is spent on cosine similarity, 6.2% is spent on encoding/compression, and 0.01% of the time is used for LLM rewording.

In all of our experiments we found that huffman encoding with our chosen dictionary was better than using 7zip or zlib alone. Further, we found that LLM rewording is particularly effective if a message is verbose, as an LLM is able to convey the same message using fewer words, and thus gives a smaller payload size. LLM rewording works better if a Local LLM with a greater number of parameters is used (e.g., gemma-27b-it-qat [2]). However these models take much longer to run than their smaller parameter counterparts.

In general, we find that if the TF-IDF cosine similarity is higher, then our compression metric also improves as huffman encoding is more effective on the payload. It should be noted that in some cases for very short payloads, compression can actually increase the final size of the payload to be encoded in an image. This is because adding a 't:' or 'w:' flag for TinyURL and Wikipedia pages respectively, adds a couple extra bytes which may not offset the compression gain from using huffman encoding. This flag must be in the message in order for the decoder to be able to fetch the pages from the internet.

The end result of these significant decreases in payload sizes is that users can encode larger payloads into the same image, or decrease the chance of detection as fewer bits are changed in the output image.

Acknowledgment

The author thanks Sebastian Angel (University of Pennsylvania) for his guidance and mentorship during this project.

References

[1] Common crawl. https://commoncrawl.org.

[2] Gemma3. https://blog.google/technology/developers/gemma-3/.

[3] Hstego. https://github.com/daniellerch/hstego.

[4] Industrial espionage: How china sneaks out america's technology secrets. https://www.bbc.co.uk/news/world-asia-china-64206950.

[5] Openpuff. https://embeddedsw.net/OpenPuff_Steganography_Home.html.

[6] Openstego. https://www.openstego.com.

[7] Stegahide. https://steghide.sourceforge.net/.

[8] Stegano. https://github.com/cedricbonhomme/Stegano.

[9] Stego-retweet. https://github.com/daniellerch/stego-retweet.

[10] Tinyurl. https://tinyurl.com.

[11] S. Berndt and M. Liśkiewicz. Provable secure universal steganography of optimal rate: Provably secure steganography does not necessarily imply one-way functions. In *Proceedings of the 4th ACM Workshop on Information Hiding and Multimedia Security*, pages 81–92, 2016.

[12] X.-Q. Deng, B.-L. Chen, W.-Q. Luo, and D. Luo. Universal image steganalysis based on convolutional neural network with global covariance pooling. *Journal of Computer Science and Technology*, 37(5):1134–1145, 2022.

[13] J. Fridrich. Feature-based steganalysis for jpeg images and its implications for future design of steganographic schemes. In *International Workshop on Information Hiding*, pages 67–81. Springer, 2004.

[14] D. A. Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 40(9):1098–1101, 1952.

[15] J. Kirchenbauer, J. Geiping, Y. Wen, J. Katz, I. Miers, and T. Goldstein. A watermark for large language models, 2024.

[16] O. Kuznetsov, K. Chernov, A. Shaikhanova, K. Iklassova, and D. Kozhakhmetova. Deepstego: Privacy-preserving natural language steganography using large language models and advanced neural architectures. *Computers*, 14(5):165, 2025.

[17] D. Lerch-Hostalot and D. Megías. Aletheia: an open-source toolbox for steganalysis. *Journal of Open Source Software, 2024, 9 (93)*, 2024.

[18] P. Sallee. Model-based steganography. In *International workshop on digital watermarking*, pages 154–167. Springer, 2003.

[19] G. Salton, A. Wong, and C.-S. Yang. A vector space model for automatic indexing. *Communications of the ACM*, 18(11):613–620, 1975.

[20] K. A. Zhang, A. Cuesta-Infante, L. Xu, and K. Veeramachaneni. Steganogan: High capacity image steganography with gans. *arXiv preprint arXiv:1901.03892*, 2019.

[21] X. Zhang, X. Kong, P. Wang, and B. Wang. Cover-source mismatch in deep spatial steganalysis. In *International Workshop on Digital Watermarking*, pages 71–83. Springer, 2019.