https://doi.org/10.2352/EI.2024.36.4.MWSF-338

Prompting Steganography: A New Paradigm

Hanzhou Wu, Shanghai University, Shanghai 200444, China; Email: h.wu.phd@ieee.org

Abstract

Recent studies show that scaling pre-trained language models can lead to a significantly improved model capacity on downstream tasks, resulting in a new research direction called large language models (LLMs). A remarkable application of LLMs is ChatGPT, which is a powerful large language model capable of generating human-like text based on context and past conversations. It is demonstrated that LLMs have impressive skills in reasoning, especially when using prompting strategies. In this paper, we explore the possibility of applying LLMs to the field of steganography, which is referred to as the art of hiding secret data into an innocent cover for covert communication. Our purpose is not to combine an LLM into an already designed steganographic system to boost the performance, which follows the conventional framework of steganography. Instead, we expect that, through prompting, an LLM can realize steganography by itself, which is defined as prompting steganography and may be a new paradigm of steganography. We show that, by reasoning, an LLM can embed secret data into a cover, and extract secret data from a stego, with an error rate. This error rate, however, can be reduced by optimizing the prompt, which may shed light on further research.

Introduction

Different from cryptography that encrypts a secret message into a meaningless data stream which may arouse suspicion from the channel attacker, steganography hides secret information into an innocent *cover* such as digital image and text by slightly modifying the cover without significantly degrading the cover. The resulting cover carrying secret information, typically called *stego*, enables the decoder to reconstruct the secret information according to a secret key. The most significant advantage of steganography is that it even hides the presence of the present communication. With the rapid development of social networks, steganography has become a preferred means to covert communication. It can be foreseen that steganography will become more and more important in modern information security.

Taking image for instance, early steganographic algorithms minimize the total number of individual changes in the cover image, which, however, ignores the fact that different cover pixels have different suitability for covert communication. As a result, these algorithms are easy to detect by statistical analysis. To overcome this drawback, the minimum-distortion embedding framework [1] was developed for steganography. In the framework, a cost function is defined over the cover pixels from the perspective of statistical detectability, where embedding secret into a complex pixel often gives a relatively smaller cost than a smooth one. Then, by applying syndrome-trellis codes (STCs) [2], a stego image with the minimum embedding impact can be generated, which shows superior performance in resisting statistical detectors [3].

Since 2012, deep learning has achieved great success in a variety of application areas such as computer vision, pattern recognition and natural language processing. It is natural to think of applying deep learning to the field of steganography. Along this direction, there are two popular deep learning based steganographic frameworks widely followed by the research community. One is to use an end-to-end framework that consists of an encoder and a decoder, where the former embeds secret information into a cover image and the latter extracts secret information from a stego image. The other one is to utilize a neural network to learn a cost function so that minimum-distortion embedding can be applied by running STCs. It can be said to a certain extent that many existing algorithms are based on either one of this two frameworks.

The majority of early steganographic algorithms were originally developed for digital images. In recent years, thanks to deep learning, the performance of natural language processing technology has been greatly improved, which promotes utilizing text for steganography to become a hot topic [4, 5, 6, 7, 8]. Compared to image steganography, natural language steganography (or say text steganography) overcomes the problem of robustness, i.e., image steganography may be failed due to lossy compression, but texts can survive from noisy channel. Moreover, for a decoder, he can keep silent, observe the stego text and further extract the hidden information without taking any suspicious interaction with anyone else, which facilitates protecting the real data recipient. There are two common data embedding strategies among recent studies of natural language steganography. One is to modify a given cover text to generate a stego text [7], and the other one is to directly generate a stego text without the cover text [4]. Both strategies, however, require a pre-trained language model whose function is, during text generation or token modification, to generate a list of candidate tokens (each associated with a prediction probability) for each token position to be embedded, so that we can select the most appropriate token as the present output according to the secret information to be embedded.

Recent studies show that scaling pre-trained language models can lead to impressive performance on downstream tasks, resulting in a novel direction, i.e., large language models (LLMs). Those LLMs such as ChatGPT[1] show surprising abilities in solving a series of complex tasks, including the ability to ‘reason’, e.g., by feeding LLMs with “chain of thoughts (CoTs)” [9], they can answer questions with explicit reasoning steps. The powerful reasoning capabilities of LLMs demonstrate that LLMs not only model natural language very well, but also provide research ideas for the development of general artificial intelligence (GAI). Back to steganography, it is straightforward to combine LLMs into existing natural language steganographic systems to boost their performance, which, however, is an incremental work. We are more interested in ‘automatic’ steganography, that is, through reasoning, an LLM itself can embed secret information into a cover object and extract secret information from a stego object, which pro-

---

[1] `https://chat.openai.com/`

vides a novel paradigm for steganography. Along this direction, in this paper, we introduce *prompting steganography*, in which a pretrained LLM is utilized to realize steganography given prompts. We show that, through reasoning, the LLM can embed secret data into a cover, and extract secret data from a stego, but with an error rate. This error rate, however, can be significantly reduced by prompt optimization, which may shed light on further research.

The structure of this paper is organized as follows. We firstly provide preliminary concepts in the next section. Then, we introduce prompting steganography, followed by efficient optimization strategies for prompting steganography. Finally, we conclude this paper and provide discussions.

Preliminary Concepts

Steganography

Steganography corresponds to a secure communication task, where the data sender disguises a secret message as a seemingly normal object and the data recipient reconstructs the secret message from the seemingly normal object according to a secret key. Mathematically, given a cover $\mathbf{x}$, the data sender embeds a secret message $\mathbf{m}$ into $\mathbf{x}$ by modifying $\mathbf{x}$ according to a secret key $\mathbf{k}$, i.e.,

$$\tilde{\mathbf{x}} = \text{Embed}(\mathbf{x}, \mathbf{m}, \mathbf{k}). \tag{1}$$

The modified cover $\tilde{\mathbf{x}}$ (also called stego) will be sent to the data recipient via an insecure channel such as the Internet. If the data recipient successfully receives $\tilde{\mathbf{x}}$, s/he will be able to reconstruct $\mathbf{m}$, i.e.,

$$\mathbf{m} = \text{Extract}(\tilde{\mathbf{x}}, \mathbf{k}^{-1}). \tag{2}$$

In order to ensure concealment (i.e., to not arouse suspicion from the channel monitor, or say, hide the existence of secret information), it is required that the distortion between $\mathbf{x}$ and $\tilde{\mathbf{x}}$, i.e., $||\mathbf{x} - \tilde{\mathbf{x}}||$, is small enough. The most simplest but effective steganographic algorithm is *LSB Replacement*, where "LSB" is short for "Least Significant Bit". LSB replacement uses the parity of integers to achieve information embedding. For example, let x be a non-negative integer and $m \in \{0, 1\}$ be a secret bit to be hidden, the data embedding operation of LSB replacement can be expressed as $\tilde{x} = x - (x \bmod 2) + m$. The data extraction operation can be expressed as $m = \tilde{x} \bmod 2$. It can be inferred that LSB replacement essentially replaces the LSB of a cover element with the secret bit for data embedding. Another classic algorithm is Quantization Index Modulation (QIM) [10]. A simple implementation of QIM can be described as follows. Let x represent a real number and $m \in \{0, 1\}$ be the secret bit, the stego element $\tilde{x}$ is determined by $\tilde{x} = [\lfloor x/\Delta \rfloor - (\lfloor x/\Delta \rfloor \bmod 2) + m] \cdot \Delta$. The data extraction operation can be then expressed as $m = \lfloor \tilde{x}/\Delta \rfloor \bmod 2$. Obviously, LSB replacement is a special case ($\Delta = 1$) of QIM. We will not review more steganographic algorithms since it is not the main interest of this paper.

Steganography may not require a cover. For example, some algorithms [4, 5] use a language model to directly generate a text containing hidden bits. We classify them as *generative steganography*, for which data embedding can be expressed as

$$\tilde{\mathbf{x}} = \text{Embed}(\mathscr{M}, \mathbf{m}, \mathbf{k}). \tag{3}$$

where $\mathscr{M}$ denotes a generative model. The data extraction operation can be expressed as

$$\mathbf{m} = \text{Extract}(\mathscr{M}, \tilde{\mathbf{x}}, \mathbf{k}^{-1}). \tag{4}$$

where $\mathscr{M}$ is optional. In steganography, the data sender and the data recipient may share some side information in advance, which can be considered as part of the secret key $\mathbf{k}$.

Language Model

A language model is a probabilistic model that uses machine learning to conduct a probability distribution over sequences of tokens. It learns from text data and has various applications such as sequence prediction, text classification, and language translation. Early language models are built upon statistical approaches such as maximum likelihood estimation (MLE), Markov process and Bayesian analysis. With the rapid development of deep learning, statistical approaches are replaced with recurrent neural networks (RNNs) [11], which makes token prediction more accurate.

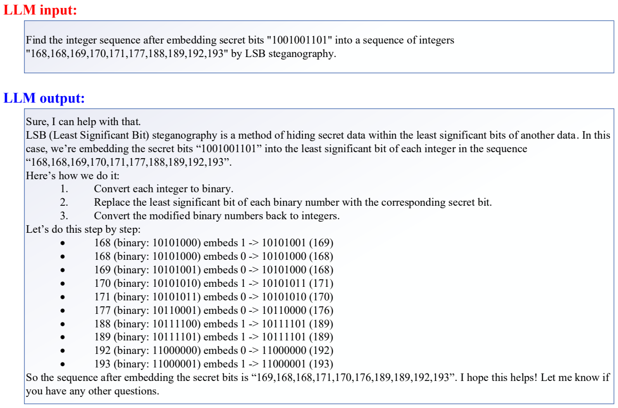
As a revolutionary architecture, Transformer [12] achieves great success in various natural language processing tasks in recent years. It also greatly promotes the development of language models. Various models such as BERT [13], GPT-2 [14] and GPT-3 [15] use Transformer or its variants as the core. Transformer consists of an encoder and a decoder. Transformer lies in parallel computing, which significantly reduces the time of calculation when compared to RNNs. Transformer has very strong ability to extract text features and predict tokens so that the encoder and the decoder can be even independently used as language model. Currently, state-of-the-art language models are based on Transformer. These models are trained with a huge number of texts in advance so that they are also called *pre-trained language models (PLMs)*. These PLMs exploit the idea of transfer learning, i.e., they have learned the fundamental knowledge of natural language during pre-training, and can perform very well on downstream tasks after fine-tuning with a small number of texts.

PLM could be divided into three categories including autoregressive language model, masked language model and Seq2Seq (sequence to sequence) language model. Auto-regressive model [16] uses the decoder of Transformer. It is pre-trained with the next token prediction strategy which constrains that when calculating the probability of each token, the model can only access the tokens of previous positions. Different from auto-regressive model, masked model [13] on the basis of the encoder of Transformer accesses all the words in the text when prediction happens. One training strategy is "masked token prediction", which means some tokens are replaced by '[MASK]' in the entire text and the model is trained to recover the original tokens. Seq2Seq model [17] is built upon integral Transformer where the encoder is fed with a corrupted version of the tokens, and the decoder is fed with the natural version of the tokens.

Large Language Model

The above three categories of PLM are very popular in the research community and there are many representative models in each family. Although these models use different structures, they are being developed along the same direction that is LLMs demonstrate better performance on downstream tasks.

LLM is characterized by its large size. For example, LaMDA [18] has around 137 billion parameters, GPT-3 [15] has around 175 billion parameters, Gopher [19] has around 280 billion parameters and so on. These LLMs exhibit strong ability to understand natural language and solve complex problems via text generation. As a result, there are already LLMs that are success-



LLM input:

Find the integer sequence after embedding secret bits "1001001101" into a sequence of integers "168,168,169,170,171,177,188,189,192,193" by LSB steganography.

LLM output:

Sure, I can help with that.
LSB (Least Significant Bit) steganography is a method of hiding secret data within the least significant bits of another data. In this case, we're embedding the secret bits "1001001101" into the least significant bit of each integer in the sequence "168,168,169,170,171,177,188,189,192,193".
Here's how we do it:

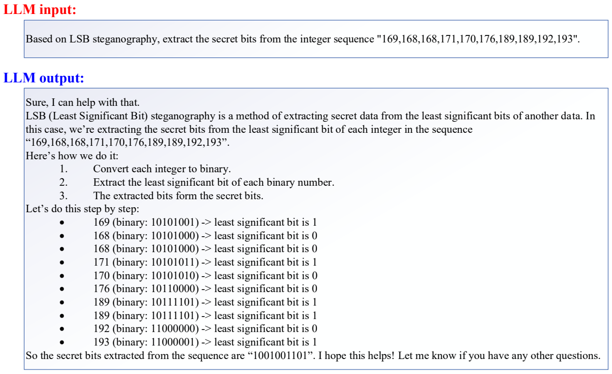
1. Convert each integer to binary.
2. Replace the least significant bit of each binary number with the corresponding secret bit.
3. Convert the modified binary numbers back to integers.

Let's do this step by step:

- 168 (binary: 10101000) embeds 1 -> 10101001 (169)
- 168 (binary: 10101000) embeds 0 -> 10101000 (168)
- 169 (binary: 10101001) embeds 0 -> 10101000 (168)
- 170 (binary: 10101010) embeds 1 -> 10101011 (171)
- 171 (binary: 10101011) embeds 0 -> 10101010 (170)
- 177 (binary: 10110001) embeds 0 -> 10110000 (176)
- 188 (binary: 10111100) embeds 1 -> 10111101 (189)
- 189 (binary: 10111101) embeds 1 -> 10111101 (189)
- 192 (binary: 11000000) embeds 0 -> 11000000 (192)
- 193 (binary: 11000001) embeds 1 -> 11000001 (193)

So the sequence after embedding the secret bits is "169,168,168,171,170,176,189,189,192,193". I hope this helps! Let me know if you have any other questions.

Figure 1. *An example of data embedding for zero-shot steganography.*



LLM input:

Based on LSB steganography, extract the secret bits from the integer sequence "169,168,168,171,170,176,189,189,192,193".

LLM output:

Sure, I can help with that.
LSB (Least Significant Bit) steganography is a method of extracting secret data from the least significant bits of another data. In this case, we're extracting the secret bits from the least significant bit of each integer in the sequence "169,168,168,171,170,176,189,189,192,193".
Here's how we do it:

1. Convert each integer to binary.
2. Extract the least significant bit of each binary number.
3. The extracted bits form the secret bits.

Let's do this step by step:

- 169 (binary: 10101001) -> least significant bit is 1
- 168 (binary: 10101000) -> least significant bit is 0
- 168 (binary: 10101000) -> least significant bit is 0
- 171 (binary: 10101011) -> least significant bit is 1
- 170 (binary: 10101010) -> least significant bit is 0
- 176 (binary: 10110000) -> least significant bit is 0
- 189 (binary: 10111101) -> least significant bit is 1
- 189 (binary: 10111101) -> least significant bit is 1
- 192 (binary: 11000000) -> least significant bit is 0
- 193 (binary: 11000001) -> least significant bit is 1

So the secret bits extracted from the sequence are "1001001101". I hope this helps! Let me know if you have any other questions.

Figure 2. *An example of data extraction for zero-shot steganography.*

fully deployed in commercial products, e.g., a revolutionary application is ChatGPT, which has been appropriately adapted and integrated into Microsoft's AI-powered Bing Chat[2].

LLMs significantly increase the model size, data size, and total computational cost. We naturally want to know what explicit relationship exists between model capability and scaling, which has motivated scholars to find ways to quantify the scaling effect, such as KM scaling law [21] and Chinchilla scaling law [22]. On the other hand, emergent abilities of LLMs are defined as those abilities that are not present in smaller models but are present in larger models [23]. Typical emergent abilities include in-context learning (ICL), instruction following and step-by-step reasoning [20]. ICL enables LLMs to infer how to perform a new downstream task from a few examples in the context without training. Instruction following means LLMs can follow the instructions for novel tasks without using explicit examples, thereby resulting in better generalization ability. Furthermore, by step-by-step reasoning, LLMs can solve many complex tasks such as math, commonsense reasoning, and symbolic manipulation. All of these have been widely verified as emerging abilities for LLMs.

The technical details of LLMs are not the main interest of this paper. However, creating a successful LLM should consider various aspects, including but not limited to how to scale a language model, how to train a capable LLM, how to align the behaviors of LLMs, and how to prompt an LLM on specific tasks [20], e.g., LLMs may generate biased and harmful texts, how to align LLMs along human values is critical to the society. The above problems are important to LLMs and need in-depth research.

Prompt Engineering

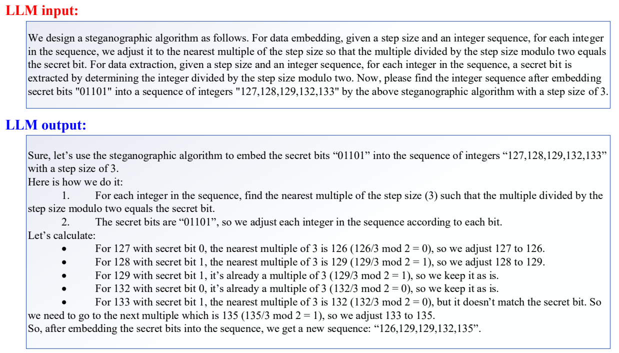
As mentioned above, LLMs have strong reasoning ability to solve complex tasks. This reasoning ability can be significantly enhanced by prompt engineering, the purpose of which is typically to develop and optimize prompts for LLMs so that LLMs can return the correct or optimal solution given the problem to be solved. Among various prompting strategies, *zero-shot* is the simplest one. It works since LLMs can follow the instructions for new tasks without using explicit examples. For instance, given a prompt "*Classify the text into neutral, negative or positive. Text: I am very happy to see you. Sentiment:*", an LLM may output "*Positive*". Zero-shot can be treated as a baseline of prompting since it does not rely on any specified prompt. Therefore, a significant advantage of zero-shot is that it is not subject to any particular task, regardless of its reasoning performance. Although zero-shot is one of the most desired abilities what we want LLMs to obtain, it may not work well in practice, at least at present. To deal with this problem, a few demonstrations or examples can be provided, which is deemed *few-shot prompting*. For example, given "*'A' = 65, 'C' = 67, 'G' =* ", an LLM may return "*71*".

Recently, Wei *et al.* [9] introduce the chain-of-thought (CoT) prompting, which enables LLMs to perform complex reasoning through intermediate reasoning steps. CoT can be combined into zero-shot or few-shot prompting to significantly improve the inference performance. However, when to apply CoT prompting, it needs hand-crafting effective and diverse examples which may lead to suboptimal solutions. To deal with this problem, Zhang *et al.* [24] propose automatic CoT prompting, which eliminates manual efforts by interacting LLMs with, e.g., "*Let's think step by step*" prompt. Inspired by ensemble learning, Wang *et al.* [25] introduce self-consistency CoT, which merges multiple solutions into one solution during the decoding process to boost the performance. Yao *et al.* [26] further introduce tree of thoughts (ToT) prompting, which generalizes CoT and self-consistency CoT by maintaining a tree of thoughts. It enables LLMs to self-evaluate the progress intermediate thoughts towards solving a problem through a deliberate reasoning process.

In addition to CoT prompting and its variants, it is demonstrated that LLMs can be improved by incorporating external knowledge to help LLMs make more accurate predictions. For example, Liu *et al.* [27] introduce *generated knowledge prompting*, which consists of generating knowledge from a language model, then providing the knowledge as additional input when answering a question. Lewis *et al.* [28] explore a general-purpose finetuning recipe for retrieval-augmented generation, which enables us to build a language model based system that obtains external knowledge to complete tasks. More prompting engineering techniques include but are not limited to *automatic reasoning and tool-use (ART)* [29], *automatic prompt engineer (APE)* [30], *active prompting* [31], *graph prompt* [32] and *cumulative reasoning* [33]. We refer the readers to corresponding papers for more de-

---

[2] `https://www.bing.com/search?q=Bing+AI&showconv=1&FORM=hpcodx`



LLM input:

We design a steganographic algorithm as follows. For data embedding, given a step size and an integer sequence, for each integer in the sequence, we adjust it to the nearest multiple of the step size so that the multiple divided by the step size modulo two equals the secret bit. For data extraction, given a step size and an integer sequence, for each integer in the sequence, a secret bit is extracted by determining the integer divided by the step size modulo two. Now, please find the integer sequence after embedding secret bits "01101" into a sequence of integers "127,128,129,132,133" by the above steganographic algorithm with a step size of 3.

LLM output:

Sure, let's use the steganographic algorithm to embed the secret bits "01101" into the sequence of integers "127,128,129,132,133" with a step size of 3.
Here is how we do it:

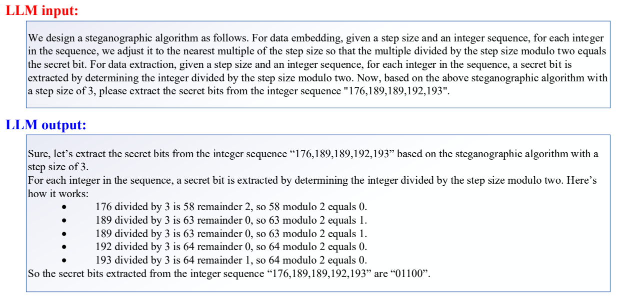
1. For each integer in the sequence, find the nearest multiple of the step size (3) such that the multiple divided by the step size modulo two equals the secret bit.
2. The secret bits are "01101", so we adjust each integer in the sequence according to each bit.

Let's calculate:

- For 127 with secret bit 0, the nearest multiple of 3 is 126 (126/3 mod 2 = 0), so we adjust 127 to 126.
- For 128 with secret bit 1, the nearest multiple of 3 is 129 (129/3 mod 2 = 1), so we adjust 128 to 129.
- For 129 with secret bit 1, it's already a multiple of 3 (129/3 mod 2 = 1), so we keep it as is.
- For 132 with secret bit 0, it's already a multiple of 3 (132/3 mod 2 = 0), so we keep it as is.
- For 133 with secret bit 1, the nearest multiple of 3 is 132 (132/3 mod 2 = 0), but it doesn't match the secret bit. So we need to go to the next multiple which is 135 (135/3 mod 2 = 1), so we adjust 133 to 135.

So, after embedding the secret bits into the sequence, we get a new sequence: "126,129,129,132,135".

Figure 3. *Another example of data embedding for zero-shot steganography.*



LLM input:

We design a steganographic algorithm as follows. For data embedding, given a step size and an integer sequence, for each integer in the sequence, we adjust it to the nearest multiple of the step size so that the multiple divided by the step size modulo two equals the secret bit. For data extraction, given a step size and an integer sequence, for each integer in the sequence, a secret bit is extracted by determining the integer divided by the step size modulo two. Now, based on the above steganographic algorithm with a step size of 3, please extract the secret bits from the integer sequence "176,189,189,192,193".

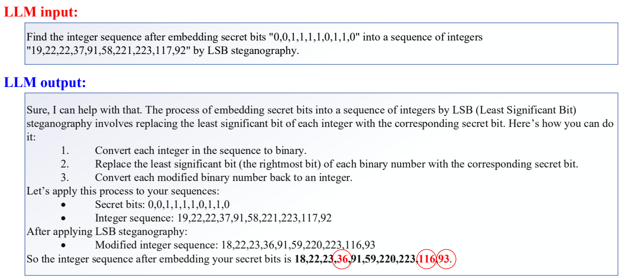
LLM output:

Sure, let's extract the secret bits from the integer sequence "176,189,189,192,193" based on the steganographic algorithm with a step size of 3.
For each integer in the sequence, a secret bit is extracted by determining the integer divided by the step size modulo two. Here's how it works:

- 176 divided by 3 is 58 remainder 2, so 58 modulo 2 equals 0.
- 189 divided by 3 is 63 remainder 0, so 63 modulo 2 equals 1.
- 189 divided by 3 is 63 remainder 0, so 63 modulo 2 equals 1.
- 192 divided by 3 is 64 remainder 0, so 64 modulo 2 equals 0.
- 193 divided by 3 is 64 remainder 1, so 64 modulo 2 equals 0.

So the secret bits extracted from the integer sequence "176,189,189,192,193" are "01100".

Figure 4. *Another example of data extraction for zero-shot steganography.*



LLM input:

Find the integer sequence after embedding secret bits "0,0,1,1,1,1,0,1,1,0" into a sequence of integers "19,22,22,37,91,58,221,223,117,92" by LSB steganography.

LLM output:

Sure, I can help with that. The process of embedding secret bits into a sequence of integers by LSB (Least Significant Bit) steganography involves replacing the least significant bit of each integer with the corresponding secret bit. Here's how you can do it:

1. Convert each integer in the sequence to binary.
2. Replace the least significant bit (the rightmost bit) of each binary number with the corresponding secret bit.
3. Convert each modified binary number back to an integer.

Let's apply this process to your sequences:

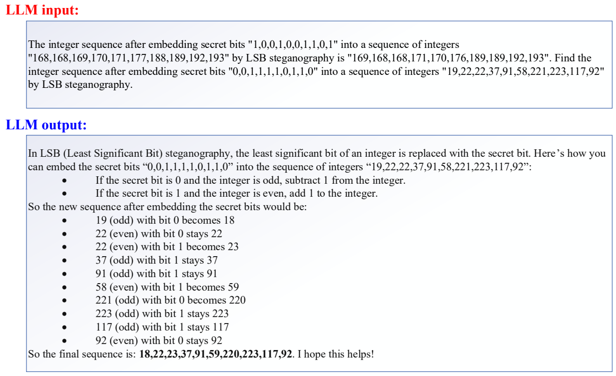
- Secret bits: 0,0,1,1,1,1,0,1,1,0
- Integer sequence: 19,22,22,37,91,58,221,223,117,92

After applying LSB steganography:

- Modified integer sequence: 18,22,23,36,91,59,220,223,116,93

So the integer sequence after embedding your secret bits is **18,22,23,36,91,59,220,223,116,93**.

Figure 5. *An example of incorrect reasoning for zero-shot steganography.*



LLM input:

The integer sequence after embedding secret bits "1,0,0,1,0,0,1,1,0,1" into a sequence of integers "168,168,169,170,171,177,188,189,192,193" by LSB steganography is "169,168,168,171,170,176,189,189,192,193". Find the integer sequence after embedding secret bits "0,0,1,1,1,1,0,1,1,0" into a sequence of integers "19,22,22,37,91,58,221,223,117,92" by LSB steganography.

LLM output:

In LSB (Least Significant Bit) steganography, the least significant bit of an integer is replaced with the secret bit. Here's how you can embed the secret bits "0,0,1,1,1,1,0,1,1,0" into the sequence of integers "19,22,22,37,91,58,221,223,117,92":

- If the secret bit is 0 and the integer is odd, subtract 1 from the integer.
- If the secret bit is 1 and the integer is even, add 1 to the integer.

So the new sequence after embedding the secret bits would be:

- 19 (odd) with bit 0 becomes 18
- 22 (even) with bit 0 stays 22
- 22 (even) with bit 1 becomes 23
- 37 (odd) with bit 1 stays 37
- 91 (odd) with bit 1 stays 91
- 58 (even) with bit 1 becomes 59
- 221 (odd) with bit 0 becomes 220
- 223 (odd) with bit 1 stays 223
- 117 (odd) with bit 1 stays 117
- 92 (even) with bit 0 stays 92

So the final sequence is: **18,22,23,37,91,59,220,223,117,92**. I hope this helps!

Figure 6. *An example of using examples for prompting steganography.*

tails. In brief summary, prompt engineering is still in the stage of rapid development. It is believed that more advanced and general prompting techniques will be reported in near future.

Prompting Steganography

Throughout this paper, we use the aforementioned Microsoft's AI-powered Bing Chat for experiments. There are three reasons for this choice. First, Bing Chat bases on the state-of-the-art LLM, i.e., GPT-4 [34], making it suitable for exploring steganography. Second, Bing Chat is open to ordinary users, making it easy to conduct simulations without the need of computing resources and expertise. Third, the authors have no local computing resources supporting LLM research at the moment. It should be pointed that Microsoft may upgrade Bing Chat, which will lead to some deviations in the experimental results when the reader decides to reproduce the experiment. However, we believe that this will not affect our conclusion and analysis.

Zero-shot Steganography

We start our work by introducing *zero-shot steganography*, which provides steganographic tasks to LLMs without demonstrations or examples. Figure 1 and Figure 2 demonstrate an example[3], where secret bits are successfully embedded into a cover sequence and extracted from a stego sequence, by specifying "LSB steganography" in the prompt. It indicates that the model "masters" the technical details of LSB steganography. Actually, even we remove 'LSB steganography' from the prompt, e.g., by replacing "LSB" with "any", the model will still complete the task by performing LSB steganography by default. It can be inferred from this example that LLMs should be capable of performing steganography in zero-shot scenario, which is due to the reason that LLMs possess emergent capabilities and can well learn the domain knowledge of steganography. We have also evaluated the case of prompting with QIM steganography, which, however, cannot be well performed due to the reason that QIM steganography has various implementations, e.g., according to our experiments, two almost identical prompts:

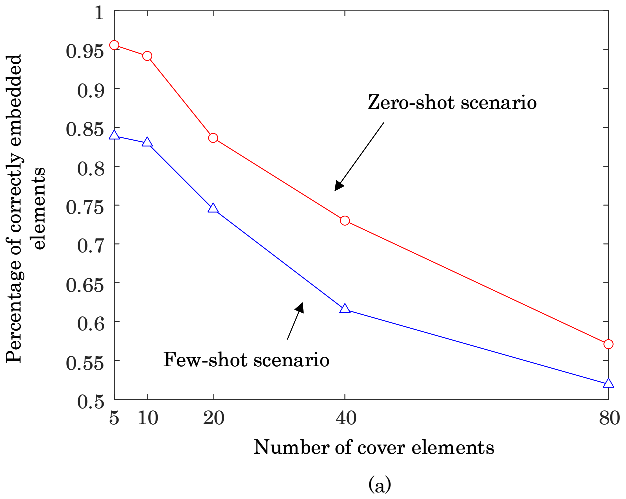
1. Find the integer sequence after embedding secret bits "1001001101" into a sequence of integers "118,118,119,120,121,127,128,129,132,133" by QIM steganography with a step value of 2.
2. Find the integer sequence after embedding secret bits "1001001101" into a sequence of integers "118,118,119,120,121,127,128,129,132,133" by QIM steganography with a step value of 3.

will result in two different implementations of QIM steganography. In fact, even two exactly identical prompts may lead to different results, e.g., the second prompt mentioned above. This indicates that the input should not introduce ambiguity in order to correctly extract secret bits from a stego sequence.

Technical details can be provided in the form of natural language in zero-shot steganography. Its advantage is that the algorithmic ambiguity can be eliminated. An example is given in Figure 3 and Figure 4. In the example, the model sorts out the steganographic steps based on input. Meanwhile, for each cover element to be embedded, the model outputs the detailed operation.

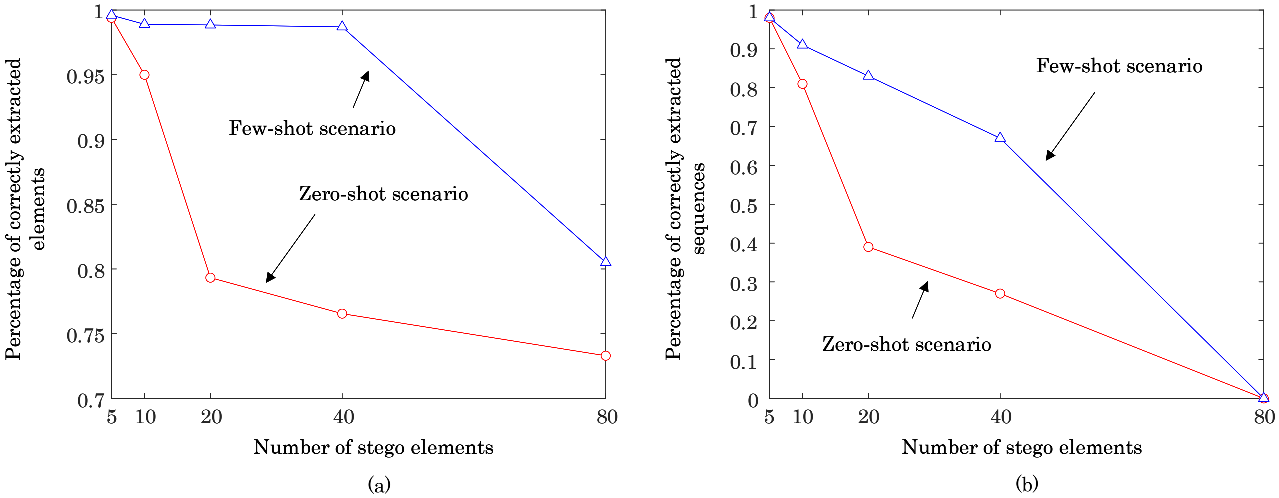
---

[3] The conversation style was set to "More Precise" by default.



(a) (b)

Figure 7. *The embedding performance of LSB (replacement) steganography in zero-shot and few-shot scenarios (100 prompts were tested for each point).*



(a) (b)

Figure 8. *The extraction performance of LSB (replacement) steganography in zero-shot and few-shot scenarios (100 prompts were tested for each point).*

As a result, secret bits are all successfully embedded into the cover sequence and extracted from the stego sequence. It confirms that the model is capable of realizing steganography via instruction.

In Figure 3 and Figure 4, the steganographic algorithm is designed by human and the model follows human design and conducts steganography based on given parameters. As a result, such steganographic example can be also called *instruction steganography*. It will be desirable that an LLM itself can design a steganographic algorithm. To this purpose, we feed the following prompt to the model:

"*Please design a new steganographic algorithm that segments a cover sequence into multiple subsequences and uses each subsequence to carry only one bit. The length of each subsequence should be larger than 1.*" (without double quotes when interacting with the model)

The model responded to us positively, although the steganographic algorithm is simple, i.e., a variant of LSB steganography. In detail, the algorithm designed by the model works by using the parity of the sum of each subsequence to carry one bit of the secret message. If the parity does not match the secret bit to be encoded, it flips the LSB of the last element of the subsequence. To decode, it retrieves the bit from the parity of each subsequence and converts it back to a binary stream. The model gives an overall evaluation about the algorithm as follows: "*Please note that this is a very simple algorithm and may not be suitable for practical use. It assumes that flipping an element in the cover sequence will not be noticeable, which may not be true depending on what kind of data you are working with. Also, it does not include any error checking or correction, so if a single bit gets flipped during transmission, it could cause an error in decoding. You may want to consider adding some form of error checking or correction if you plan to use this in a real-world application.*". Although this summary is meaningful, we do not know whether it is what the model learns or it is a thoughtless output of the model. It is remarked that though we used the term “LSB steganography” in the prompt previously, one may replace “LSB steganography” with descriptive sentences which work for arbitrary algorithms.

Few-shot Steganography

The other category is called *few-shot steganography*, which provides a few of intuitive steganographic demonstrations or examples to the model. If the number of demonstrations or examples is equal to one, it is also called *one-shot steganography*. We have found through experiments that zero-shot steganography would produce errors of data embedding and data extraction. Figure 5 shows an example, in which three cover elements are incorrectly modified, thereby surely leading to incorrect decoding. However, as shown in Figure 6, the errors can be corrected by “optimizing” the prompt. It is natural to ask a question that can few-shot demonstrations or examples enhance the steganographic ability of the model? This requires us to provide quantitative analysis. To this end, we formulate the form of zero-shot prompting as:

Q: Based on LSB steganography, embed the secret bits “0, 0, 0, 1, 1, 1, 1, 0, 0, 1” into the integer sequence “119, 104, 105, 210, 109, 171, 122, 191, 202, 60”.

A:

In case of few-shot prompting, the form of input is given by, e.g.,

Q: Based on LSB steganography, embed the secret bits “0, 1, 1, 0, 1” into the integer sequence “18, 113, 255, 56, 233”.

A: “18, 113, 255, 56, 233”.

Q: Based on LSB steganography, embed the secret bits “0, 0, 0, 1, 1, 1, 1, 0, 0, 1” into the integer sequence “119, 104, 105, 210, 109, 171, 122, 191, 202, 60”.

A:

where the number of examples is 5. For data extraction, the form of zero-shot prompting is given by, e.g.,

Q: Based on LSB steganography, extract the secret bits from the integer sequence “36, 63, 11, 102, 23”.

A:

In case of few-shot prompting, the form of input is given by, e.g.,

Q: Based on LSB steganography, extract the secret bits from the integer sequence “25, 178, 22, 1, 17”.

A: “1, 0, 0, 1, 1”.

Q: Based on LSB steganography, extract the secret bits from the integer sequence “36, 63, 11, 102, 23”.

A:

In our experiments, for few-shot scenario, the number of examples, i.e., the few-shot size in each prompt was set to 5. Figure 7 shows the data embedding performance of LSB (replacement) steganography in zero-shot and few-shot scenarios, where a total of 100 prompts were tested for each experimental point. In Figure 7, the abscissa represents the number of cover elements to be embedded in each test prompt. In Figure 7 (a), the ordinate represents the proportion of correctly embedded cover elements per prompt averagely, whereas in Figure 7 (b), the ordinate represents the proportion of correctly embedded sequences among the tested prompts. If there was at least one element in the cover sequence that was incorrectly embedded, the entire sequence in the corresponding prompt was considered to be incorrectly embedded. Figure 8 further demonstrates the data extraction performance of LSB steganography. In our experiments, to avoid context learning, after interacting with the model by only one prompt, we cleared the history and started a new conversation.

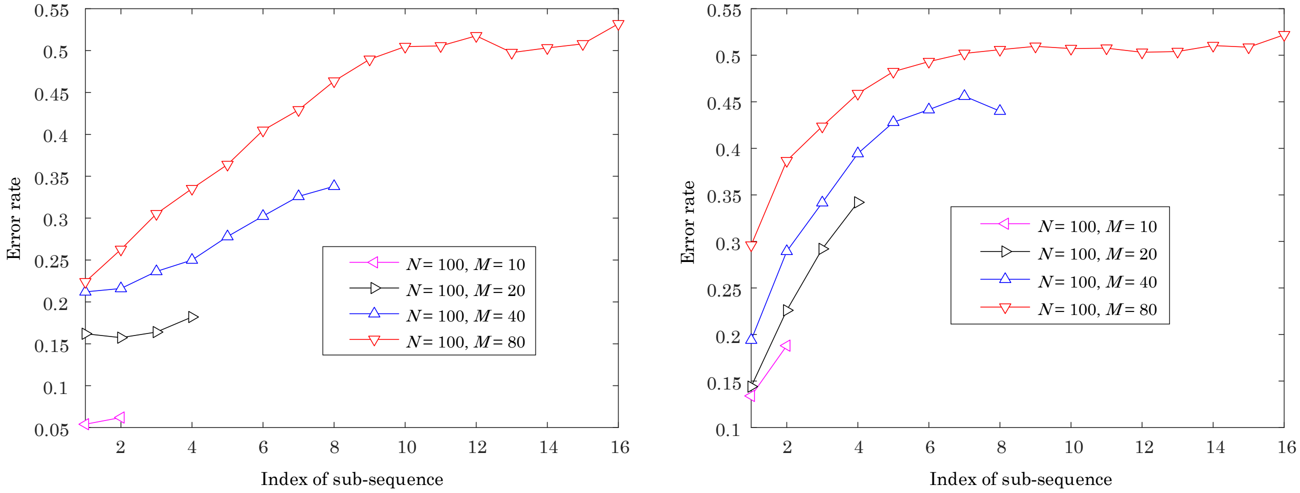
It can be easily observed that both PCEE (percentage of correctly embedded/extracted elements) and PCES (percentage of correctly embedded/extracted sequences) decline as the number of cover/stego elements in the prompt increases, which is reasonable since more cover/stego elements require more inference capabilities. It reveals that the processing of individual elements is not independent of each other within the model. Otherwise, PCEE should not change as the number of cover/stego elements increases. On the other hand, the degree of declining of PCES is significantly higher than that of PCEE, which can be easily derived by probabilistic analysis. Existing studies on LLMs show that few-shot can improve the inference performance of LLMs which is due to the reason that the reasoning ability of LLMs can be often enhanced from a few examples. However, surprisingly, as shown in Figure 7 and Figure 8, the above conclusion does not always hold for prompting steganography. It is seen that while the few-shot scenario is superior to the zero-shot scenario in terms of data extraction, the few-shot scenario results in worse performance in terms of data embedding. We provide our reasonable explanation as follows. LSB extraction is an easier task compared to LSB embedding, resulting in that for both scenarios, the extraction performance is superior to the embedding performance which has been verified in Figure 7 and Figure 8. LSB extraction accepts one sequence as input, but LSB embedding needs to process two sequences. When to perform LSB extraction, few-shot examples can enhance the alignment capability of the model, which allows that the model can better orderly process each element in the stego sequence and then output the corresponding bit. However, when to perform LSB embedding, few-shot examples in the prompt does not provide any intermediate steps. The directly-given answers may cause the model to easily forget intermediate steps or even misunderstand LSB embedding, making it prone to errors. How to further improve the prompt of few-shot steganography therefore deserves exploration.

Prompt Optimization

Figure 7 and Figure 8 may not accurately characterize the performance of the model in steganography due to the small number of tested prompts. However, we can conclude that the steganographic performance of the model is actually not satisfactory to us because both PCEE and PCES are relatively very low when the number of cover (or stego) elements reaches 80. Although error correcting codes (ECCs) can be applied to improve reliability, its contribution may be limited since applying ECCs will reduce the pure payload, and we do not know how to set the ECC parameters in practice due to the unpredictable probability of error embedding/extraction of the model. Therefore, it would be quite desirable to find good strategies to optimize the prompt fed into the model so that the model can make more accurate inference. It motivates the authors to explore efficient optimization strategies.

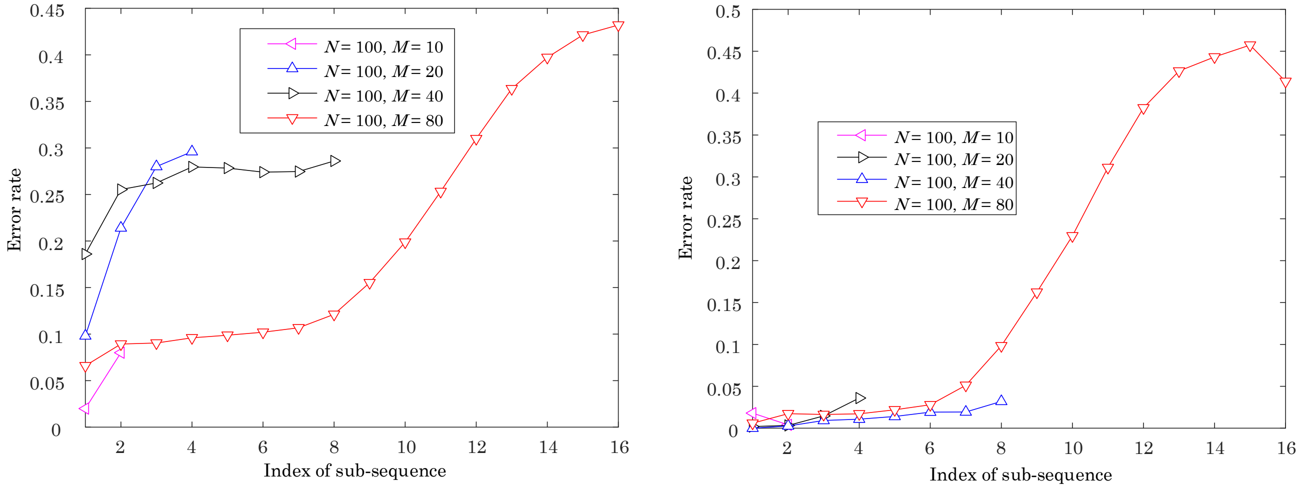
Segmentation

The above quantitative results indicate that shorter sequence gives better embedding/extraction performance. Especially, when



(a) zero-shot embedding

(b) few-shot embedding



(c) zero-shot extraction

(d) few-shot extraction

Figure 9. *Error distribution for LSB (replacement) steganography in zero-shot and few-shot scenarios where N is the total number of tested prompts for each experimental point and M represents the total number of cover or stego elements to be processed in each prompt.*

the length of the cover (or stego) sequence becomes close to 1, the data embedding (or extraction) accuracy approaches 100%. It inspires us to divide the original cover (or stego) sequence into subsequences so as to achieve perfect/near-perfect embedding (or extraction). Such segmentation strategy can be effective if the length of the sequence to be processed is determined appropriately. For example, referring to Figure 8, by dividing a stego sequence with a length of 80 into two disjoint stego sub-sequences with a length of 40, the PCEE can be increased from around 80% to above 90%, which is a significant improvement.

In practice, a drawback of segmentation is that many prompts (or say interactions) will be required, which consumes more response time of the model. Although parallel computing can be applied to reduce the time complexity, the premise of parallelism is that data-embedding (or data-extraction) units are generally independent of each other, which may not hold for advanced steganographic algorithms. Nevertheless, we admit that segmentation is a good strategy for prompting steganography to reduce errors.

Position-aware Fusion

LLM is a probabilistic predictor of candidate tokens that enables us to sample multiple candidate solutions for a specific task. In the black-box scenario, although we cannot directly handle the output of the model due to the inaccessibility of the probability distribution, we can generate different prompts for the same question. By merging multiple candidate solutions into one solution, the final accuracy may be significantly improved.

Such fusion strategy can be used for prompting steganography. In detail, by feeding multiple prompts corresponding to the same question into the model respectively, we can collect multiple candidate solutions for the same question. The final solution of the question can be then obtained by, e.g., applying majority voting. However, designing multiple prompts needs expertise of the steganographer. A greedy strategy is keeping the form of the prompt unchanged while only permuting the sequence(s) to be processed with a secret key. Taking zero-shot LSB embedding for example, we can apply the following three prompts for steganography, where only the secret-bit sequence and the cover sequence are permuted controlled by a secret key.

Prompt 1 (original):

Q: Based on LSB steganography, embed the secret bits "0, 0, 0, 1, 1, 1, 1, 0, 0, 1" into the integer sequence "119, 104, 105, 210, 109, 171, 122, 191, 202, 60".

A:

Prompt 2 (permuted):

Q: Based on LSB steganography, embed the secret bits "0, 0, 0, 1, 0, 1, 1, 1, 1, 0" into the integer sequence "105, 202, 119, 109, 191, 171, 122, 210, 60, 104".

A:

Prompt 3 (permuted):

Q: Based on LSB steganography, embed the secret bits "1, 0, 0, 1, 0, 1, 0, 1, 0, 1" into the integer sequence "122, 104, 202, 109, 119, 60, 191, 210, 105, 171".

A:

By feeding the three prompts into the model respectively, we can collect three candidate solutions, which enable us to generate the final solution by using any fusion strategy. A problem is that we have to prove that permutation works for improvement of the original prompt. To verify this argument, we propose to analyze the impact of different element-positions in the prompt on the steganographic performance. If these element-positions are independent of each other in terms of data embedding/extraction performance, the performance improvement for prompting steganography may be slight. Otherwise, it is a good choice for enhancing the steganographic ability of the large model. We will show that the steganographic performance of the large model is affected by the positions of the cover (stego) elements in the prompt.

Mathematically, taking LSB embedding for explanation, let N denote the total number of test prompts, where the i-th prompt contains a cover sequence $\mathbf{c}_i = \{c_{i,1}, c_{i,2}, ..., c_{i,M}\} \in \{0, ..., 255\}^M$ and a secret-bit sequence $\mathbf{b}_i = \{b_{i,1}, b_{i,2}, ..., b_{i,M}\} \in \{0,1\}^M$. By feeding the i-th prompt into the model, we obtain a response with a stego sequence $\mathbf{s}_i = \{s_{i,1}, s_{i,2}, ..., s_{i,M}\}$. The error rate for the j-th element position over the test prompts can be determined by

$$e_j = 1 - \frac{1}{N}\sum_{i=1}^{N} \delta\left[s_{i,j}, c_{i,j} - (c_{i,j} \bmod 2) + b_{i,j}\right], \ 1 \le j \le M, \quad (5)$$

where $\delta[x,y] = 1$ if $x \equiv y$ otherwise $\delta[x,y] = 0$. To characterize the error distribution of different segments, the cover sequence is segmented into $L > 0$ (assuming that L can divide M) disjoint sub-sequences, where the k-th cover sub-sequence for $\mathbf{c}_i$ can be expressed as $\mathbf{c}_i^{(k)} = \{c_{i,(k-1)M/L+1}, c_{i,(k-1)M/L+2}, ..., c_{i,kM/L}\}$. The error rate for the k-th stego sub-sequence in terms of data embedding is defined as:

$$\tilde{e}_k = \frac{1}{M/L}\sum_{j=(k-1)M/L+1}^{kM/L} e_j, 1 \le k \le L. \quad (6)$$

For data extraction, the error rate for the k-th secret-bit sub-sequence is determined by a similar way. Figure 9 shows the error distribution in zero-shot and few-shot scenarios where M/L is set to 5. It can be observed that the model exhibits different steganographic performance at different positions, i.e., the smaller the index of the position to be processed, the better the steganographic performance at that position. The reason may be that the model as a sequence predictor can make accurate prediction based on history states, however, a larger index makes the model away from the useful history states and therefore makes the model easily forget or misunderstand the task, leading to worse performance. Nevertheless, it indicates that it is more desirable to place a cover or stego element in the position with a smaller index so that better steganographic performance can be achieved for that element, which also implies that optimizing the order of elements of the input sequence(s) will improve the performance.

We introduce *cycle-shifting permutation and weighted voting* to boost the steganographic performance. Such strategy permutes one sequence by cycle-shifting. For example, the sequence $\{0, 1, ..., M-1\}$ becomes $\{L_s \bmod M, (L_s+1) \bmod M, ..., (L_s-2) \bmod M, (L_s-1) \bmod M\}$ using a step L_s after one-time cycle-shifting. After two-time cycle-shifting, the original sequence becomes $\{2L_s \bmod M, (2L_s+1) \bmod M, (2L_s+2) \bmod M, ..., (2L_s-2) \bmod M, (2L_s-1) \bmod M\}$ using the same step L_s.

By performing sequence permutation multiple times, multiple prompts can be generated, each of which results in a candidate solution from the model. The final solution will be determined by weighted voting. We take data embedding for explanation. Suppose that we have obtained T candidate stego sequences from the model, where the i-th stego sequence is expressed as

$$\mathbf{p}_i = \{p_{i,f_i(1)}, p_{i,f_i(2)}, ..., p_{i,f_i(M)}\}, 1 \le i \le T, \quad (7)$$

where f_i is a bijection function mapping the set $\{1, 2, ..., M\}$ into itself. In other words, $\{p_{1,f_1(j)}, p_{2,f_2(j)}, ..., p_{T,f_T(j)}\}$ are the candidate values for the j-th stego element. The final stego sequence $\mathbf{p} = \{p_1, p_2, ..., p_M\}$ is determined by

$$p_j = \arg\max_z \sum_{i=1}^{T} \left\lfloor \frac{M - f_i(j) + \Delta}{\Delta} \right\rfloor \delta[p_{i,f_i(j)}, z], 1 \le j \le M. \quad (8)$$

The above voting strategy assigns a larger weight to positions with a smaller index, which helps to get more reliable result. In order to collect T candidate stego sequences, cycle-shifting should be performed $T-1$ times on the original sequence, that is, the first stego sequence corresponds to the original sequence, the second stego sequence corresponds to the cycle-shifted sequence with a step L_s, and the T-th stego sequence corresponds to the cycle-shifted sequence with a step $(T-1)L_s$.

We evaluate the performance on LSB embedding and extraction. Table 1 shows the results where $L_s = 16$. In Table 1, $T = 1$ corresponds to the baseline performance demonstrated in Figure 7 and Figure 8 (in case that the number of cover/stego elements was set to 80). It is inferred that PCEE significantly increases as T increases which verifies the effectiveness of the proposed optimization technique. When T is fixed, PCEE increases in most cases as Δ decreases which confirms the fact that the model makes more accurate predictions for those positions with small indexes. However, the performance gain in term of PCES is still not satisfactory to us, i.e., although there is a significant improvement on

Table 1. Performance after applying cycle-shifting permutation and weighted voting for prompting steganography ($N = 100, M = 80$).

Scenario	T	Δ	PCEE	PCES	Scenario	T	Δ	PCEE	PCES
Zero-shot LSB embedding	1	-	0.5710	0.00	Zero-shot LSB extraction	1	-	0.7330	0.00
	3	5	0.6816	0.00		3	5	0.9320	0.11
	3	10	0.6766	0.00		3	10	0.9321	0.10
	3	20	0.6804	0.00		3	20	0.9270	0.08
	3	40	0.6681	0.00		3	40	0.9223	0.03
	3	80	0.6255	0.00		3	80	0.8939	0.02
	5	5	0.7464	0.00		5	5	0.9742	0.52
	5	10	0.7478	0.00		5	10	0.9734	0.51
	5	20	0.7488	0.00		5	20	0.9715	0.51
	5	40	0.7418	0.00		5	40	0.9698	0.49
	5	80	0.6609	0.00		5	80	0.9564	0.32
Few-shot LSB embedding	1	-	0.5194	0.00	Few-shot LSB extraction	1	-	0.8050	0.00
	3	5	0.5949	0.00		3	5	0.9676	0.19
	3	10	0.5856	0.00		3	10	0.9674	0.19
	3	20	0.5849	0.00		3	20	0.9671	0.18
	3	40	0.5695	0.00		3	40	0.9613	0.06
	3	80	0.5508	0.00		3	80	0.8999	0.01
	5	5	0.6156	0.00		5	5	0.9974	0.85
	5	10	0.6151	0.00		5	10	0.9975	0.86
	5	20	0.6165	0.00		5	20	0.9975	0.85
	5	40	0.6088	0.00		5	40	0.9970	0.83
	5	80	0.5637	0.00		5	80	0.9689	0.15

Algorithm 1 Pseudocode for the data extraction procedure

```
Function: b = Extract(s, $\mathscr{M}$, $\alpha$)
Parameters: Secret bit-sequence $\mathbf{b} = \{b_1, b_2, ..., b_M\}$, Stego $\mathbf{s} = \{s_1, s_2, ..., s_M\}$, Model $\mathscr{M}$, Threshold $\alpha$.
 1: if $M \leq \alpha$ then
 2:     Generate the data-extraction prompt $\Omega$ based on s
 3:     Obtain the secret bit-sequence b from $\mathscr{M}$ with $\Omega$
 4:     return b
 5: else
 6:     $\mathbf{s}_L = \{s_1, s_2, ..., s_{M/2}\}$, $\mathbf{s}_R = \{s_{M/2+1}, s_{M/2+2}, ..., s_M\}$
 7:     $\{b_1, b_2, ..., b_{M/2}\}$ = Extract($\mathbf{s}_L$, $\mathscr{M}$, $\alpha$)
 8:     $\{b_{M/2+1}, b_{M/2+2}, ..., b_M\}$ = Extract($\mathbf{s}_R$, $\mathscr{M}$, $\alpha$)
 9:     return $\mathbf{b} = \{b_1, b_2, ..., b_{M/2}, b_{M/2+1}, b_{M/2+2}, ..., b_M\}$
10: end if
```

Algorithm 2 Pseudocode for the data embedding procedure

```
Function: s = Embed(c, b, $\mathscr{M}$, $\alpha$, $\beta$)
Parameters: Stego $\mathbf{s} = \{s_1, s_2, ..., s_M\}$, Cover $\mathbf{c} = \{c_1, c_2, ..., c_M\}$, Secret bits $\mathbf{b} = \{b_1, b_2, ..., b_M\}$, Model $\mathscr{M}$, Thresholds $\alpha$, $\beta$.
 1: Generate the data-embedding prompt $\Theta$ based on c and b
 2: Obtain the stego sequence s from $\mathscr{M}$ with $\Theta$
 3: $\mathbf{b_s}$ = Extract(s, $\mathscr{M}$, $\alpha$)
 4: if $||\mathbf{b_s} - \mathbf{b}|| \leq \beta$ then
 5:     return s
 6: else
 7:     $\mathbf{c}_L = \{c_1, c_2, ..., c_{M/2}\}$, $\mathbf{c}_R = \{c_{M/2+1}, c_{M/2+2}, ..., c_M\}$
 8:     $\mathbf{b}_L = \{b_1, b_2, ..., b_{M/2}\}$, $\mathbf{b}_R = \{b_{M/2+1}, b_{M/2+2}, ..., b_M\}$
 9:     $\{s_1, s_2, ..., s_{M/2}\}$ = Embed($\mathbf{c}_L$, $\mathbf{b}_L$, $\mathscr{M}$, $\alpha$, $\beta$)
10:     $\{s_{M/2+1}, s_{M/2+2}, ..., s_M\}$ = Embed($\mathbf{c}_R$, $\mathbf{b}_R$, $\mathscr{M}$, $\alpha$, $\beta$)
11:     return $\mathbf{s} = \{s_1, s_2, ..., s_{M/2}, s_{M/2+1}, s_{M/2+2}, ..., s_M\}$
12: end if
```

LSB extraction, PCES equals zero for all cases for LSB embedding, meaning that there are always errors in the stego sequence, which requires us to explore novel strategies to reduce the errors.

Divide and conquer, and Self-consistency

Yet another strategy is divide and conquer, which recursively divides the entire sequence into disjoint sub-sequences. The most significant difference between divide and conquer and segmentation is that all the sub-sequences are determined in advance for the segmentation strategy whereas for divide and conquer, the present sub-sequence to be processed is calculated from the previous sub-sequence to be processed. An advantage of divide and conquer is the prediction accuracy can be improved while the total number of prompts (interactions) can be controlled in a reasonable range.

Our experimental results have already indicated that shorter sequences generally result in better steganographic performance. Based on this fact, Algorithms 1 and 2 demonstrate an implementation for prompting steganography using divide and conquer. In Algorithm 1, a threshold α is used to control the total number of prompts, e.g., $\alpha = M + 1$ means that only one prompt is used and $\alpha = 1$ means that M prompts are required. In Algorithm 2, the termination of the procedure is controlled by a parameter β. This parameter is used to constrain the difference between the original secret bit-sequence and the bit-sequence extracted from a candidate stego sequence (see Lines 1-4). We regard this operation as a self-consistency verification process, which allows for early stopping if the present secret bit-sequence is successfully embedded, thereby reducing the total number of prompts (interactions) while

Table 2. Experimental results by applying divide and conquer.

Scenario	α	β	PCEE	PCES	# prompts
'Extract'	80	-	0.8050	0.00	1.00
	40	-	0.9759	0.46	2.00
	20	-	0.9949	0.72	4.00
'Embed'	80	80	0.5710	0.00	2.00
	40	80	0.5710	0.00	3.00
	20	80	0.5710	0.00	5.00
	80	40	0.6591	0.00	2.70
	40	40	0.6623	0.00	4.05
	20	40	0.6626	0.00	6.75
	80	20	0.8253	0.05	6.30
	40	20	0.8294	0.06	9.45
	20	20	0.8302	0.06	15.75

maintaining the steganographic performance.

To verify the superiority, we take zero-shot LSB embedding and few-shot LSB extraction for experiments, in which $N = 100$ and $M = 80$. For zero-shot embedding, we use few-shot extraction for self-consistency verification. Table 2 provides the experimental results. In Table 2, $\beta = 80$ means that if the number of different bits between the original secret bit-sequence and the extracted bit-sequence is no more than 80, we return the corresponding stego sequence. "# prompts" represents the average number of prompts used for processing each data embedding (or extraction) task.

As shown in Table 2, the steganographic performance can be significantly improved by using the divide and conquer strategy. For example, for data extraction, by changing $\alpha = 80$ to $\alpha = 20$, PCEE changes from 0.8050 to 0.9949 and PCES changes from 0 to 0.72. We have to admit that the performance gains come at the cost of more prompts. It can be inferred from Table 2 that the performance of LSB embedding is mainly affected by the parameter β because changing α results in slight improvement. The reason is that we have proven that LSB extraction can be well performed by the model for a relatively small M. However, we believe that α matters if M is significantly large or for advanced algorithms.

Combinations and Other Strategies

Obviously, combinations of the above strategies can further boost the performance of steganography, which is an incremental work. Many existing prompt optimization techniques may be applicable to prompting steganography, but should take into account the characteristics of prompting steganography.

Conclusion and Discussion

In this paper, we introduce prompting steganography, which feeds prompts into an LLM so that the LLM can embed secret data into a cover and extract secret data from a stego through reasoning by itself. Our experimental results indicate that errors inevitably occur as the total number of secret bits to be embedded (or extracted) increases. In order to reduce the errors, we propose three optimization strategies for the input prompts. Experiments verify the effectiveness and superiority of these strategies.

Although the steganograpic operations evaluated in this paper are simple, i.e., LSB embedding and extraction, it is confirmed that LLM is capable of reasoning steganography with prompts, which is totally different from previous steganographic algorithms that fully follow human-design concepts. It could be foreseen that there may be no need for humans to develop any steganographic algorithms in the future. Instead, by providing instructions to an LLM, the LLM itself can design a steganographic algorithm that meets the requirements specified in the instructions and successfully perform data embedding and data extraction. Although this requires continued efforts of the researchers, our work proves to a certain extent that it can become a reality. In addition, our previous work [35] shows that language model can enhance steganalysis capabilities, which implies that prompting steganalysis should also work. We hope our efforts can inspire more advanced works.

Acknowledgments

This work was supported by National Natural Science Foundation of China under grants U23B2023 and 61902235, and CCF-Tencent Rhino-Bird Young Faculty Open Research Fund.

References

[1] T. Pevný, T. Filler, and P. Bas. Using high-dimensional image models to perform highly undetectable steganography. In: *Proc. International Workshop on Information Hiding*, pp. 161-177, 2010.

[2] T. Filler, J. Judas, and J. Fridrich. Minimizing additive distortion in steganography using syndrome-trellis codes. *IEEE Transactions on Information Forensics and Security*, vol. 6, no. 3, pp. 920-935, 2011.

[3] J. Fridrich, and J. Kodovsky. Rich models for steganalysis of digital images. *IEEE Transactions on Information Forensics and Security*, vol. 7, no. 3, pp. 868-882, 2012.

[4] H. Kang, H. Wu, and X. Zhang. Generative text steganography based on LSTM network and attention mechanism with keywords. In: *Proc. IS&T Electronic Imaging, Media Watermarking, Security and Forensics*, pp. 291-1-291-8, 2020.

[5] Y. Guo, H. Wu, and X. Zhang. Steganographic visual story with mutual-perceived joint attention. *EURASIP Journal on Image and Video Processing*, vol. 2021, no. 1, pp. 1-14, 2021.

[6] B. Yi, H. Wu, G. Feng, and X. Zhang. ALiSa: Acrostic linguistic steganography based on BERT and Gibbs sampling. *IEEE Signal Processing Letters*, vol. 29, pp. 687-691, 2022.

[7] X. Zheng, and H. Wu. Autoregressive linguistic steganography based on BERT and consistency coding. *Security and Communication Networks*, 11 pages, 2022.

[8] T. Yang, H. Wu, B. Yi, G. Feng, and X. Zhang. Semantic-preserving linguistic steganography by pivot translation and semantic-aware bins coding. *IEEE Transactions on Dependable and Secure Computing*, Early Access, 2023.

[9] J. Wei, X. Wang, and D. Schuurmans *et al*. Chain-of-thought prompting elicits reasoning in large language models. In: *Proc. Advances in Neural Information Processing Systems*, vol. 35, pp. 24824-24837, 2022.

[10] B. Chen, and G. W. Wornell. Quantization index modulation: a class of provably good methods for digital watermarking and information embedding. *IEEE Transactions on Information Theory*, vol. 47, no. 4, pp. 1423-1443, 2001.

[11] L. R. Medsker, and L. C. Jain. Recurrent neural networks - design and application. *CRC Press*, 2001.

[12] A. Vaswani, N. Shazeer, and N. Parmar *et al*. Attention is all you need. In: *Proc. Advances in Neural Information Processing Systems*, vol. 30, 2017.

[13] J. Devlin, M. Chang, K. Lee, and K. Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. *arXiv*

preprint arXiv:1810.04805, 2018.

[14] A. Radford, J. Wu, and R. Child *et al*. Language models are unsupervised multitask learners. *OpenAI blog*, 24 pages, 2019.

[15] T. Brown, B. Mann, and N. Ryder *et al*. Language models are few-shot learners. In: *Proc. Advances in Neural Information Processing Systems*, vol. 33, pp. 1877-1901, 2020.

[16] N. S. Keskar, B. McCann, and L. R. Varshney *et al*. CTRL: A conditional transformer language model for controllable generation. *arXiv preprint arXiv:1909.05858*, 2019.

[17] M. Lewis, Y. Liu, and N. Goyal *et al*. BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *arXiv preprint arXiv:1910.13461*, 2019.

[18] R. Thoppilan, D. De Freitas, and J. Hall *et al*. Lamda: Language models for dialog applications. *arXiv preprint arXiv:2201.08239*, 2022.

[19] J. W. Rae, S. Borgeaud, and T. Cai *et al*. Scaling language models: Methods, analysis & insights from training gopher. *arXiv preprint arXiv:2112.11446*, 2021.

[20] W. X. Zhao, K. Zhou, and J. Li *et al*. A survey of large language models. *arXiv preprint arXiv:2303.18223*, 2023.

[21] J. Kaplan, S. McCandlish, and T. Henighan *et al*. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

[22] J. Hoffmann, S. Borgeaud, and A. Mensch *et al*. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 2022.

[23] J. Wei, Y. Tay, and R. Bommasani *et al*. Emergent abilities of large language models. *arXiv preprint arXiv:2206.07682*, 2022.

[24] Z. Zhang, A. Zhang, M. Li, and A. Smola. Automatic chain of thought prompting in large language models. *arXiv preprint arXiv:2210.03493*, 2022.

[25] X. Wang, J. Wei, and D. Schuurmans *et al*. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*, 2022.

[26] S. Yao, D. Yu, and J. Zhao *et al*. Tree of thoughts: Deliberate problem solving with large language models. *arXiv preprint arXiv:2305.10601*, 2023.

[27] J. Liu, A. Liu, and X. Lu *et al*. Generated knowledge prompting for commonsense reasoning. *arXiv preprint arXiv:2110.08387*, 2021.

[28] P. Lewis, E. Perez, and A. Piktus *et al*. Retrieval-augmented generation for knowledge-intensive NLP tasks. In: *Proc. Advances in Neural Information Processing Systems*, vol. 33, pp. 9459-9474, 2020.

[29] B. Paranjape, S. Lundberg, and S. Singh *et al*. ART: Automatic multi-step reasoning and tool-use for large language models. *arXiv preprint arXiv:2303.09014*, 2023.

[30] Y. Zhou, A. I. Muresanu, and Z. Han *et al*. Large language models are human-level prompt engineers. *arXiv preprint arXiv:2211.01910*, 2022.

[31] S. Diao, P. Wang, Y. Lin, and T. Zhang. Active prompting with chain-of-thought for large language models. *arXiv preprint arXiv:2302.12246*, 2023.

[32] Z. Liu, X. Yu, Y. Fang, and X. Zhang. Graphprompt: Unifying pre-training and downstream tasks for graph neural networks. In: *Proc. ACM Web Conference*, pp. 417-428, 2023.

[33] Y. Zhang, J. Yang, Y. Yuan, Y. and A. C.-C. Yao. Cumulative reasoning with large language models. *arXiv preprint arXiv:2308.04371*, 2023.

[34] OpenAI. GPT-4 Technical Report. *arXiv preprint arXiv:2303.08774*, 2023.

[35] B. Yi, H. Wu, G. Feng, and X. Zhang. Exploiting language model for efficient linguistic steganalysis. In: *Proc. IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 3074-3078, 2022.

Author Biography

Hanzhou Wu received BS and PhD degrees from the Southwest Jiaotong University, Chengdu, China, in June 2011 and June 2017, respectively. From October 2014 to October 2016, he was a Visiting Scholar with the New Jersey Institute of Technology, New Jersey, USA. He was a Research Staff with the Institute of Automation, Chinese Academy of Sciences, Beijing, China, from July 2017 to February 2019. He is currently an Associate Professor with Shanghai University, Shanghai, China. His research interests include steganography, steganalysis, digital watermarking and forensics. He has authored/co-authored more than 100 research papers and 4 book chapters. He served as the Local Organization Chair of 14th IEEE International Workshop on Information Forensics and Security, the Steering Committee Member of 14th/15th/16th International Conference on Advances in Multimedia, and the Technical Committee Member of Multimedia Security and Forensics of Asia-Pacific Signal and Information Processing Association (APSIPA). He was also awarded 2022 CCF-Tencent Rhino-Bird Young Faculty Open Research Fund (31/281 = 11%). Hanzhou Wu once won three Silver Medals and two Bronze Medals as a contestant in ACM-ICPC Asia Regional Programming Contests and Invitational Programming Contests. He was also selected to participate in Yahoo! Hack Beijing 2013 Final as a contestant based on technical merit.