Near-imperceptible Neural Linguistic Steganography via Self-Adjusting Arithmetic Coding

Jiaming Shen, Heng Ji, Jiawei Han
Department of Computer Science, University of Illinois Urbana-Champaign, IL, USA
{js2, hengji, hanj}@illinois.edu

Abstract

Linguistic steganography studies how to hide secret messages in natural language cover texts. Traditional methods aim to transform a secret message into an innocent text via lexical substitution or syntactical modification. Recently, advances in neural language models (LMs) enable us to directly generate cover text conditioned on the secret message. In this study, we present a new linguistic steganography method which encodes secret messages using self-adjusting arithmetic coding based on a neural language model. We formally analyze the statistical imperceptibility of this method and empirically show it outperforms the previous state-of-the-art methods on four datasets by 15.3% and 38.9% in terms of bits/word and KL metrics, respectively. Finally, human evaluations show that 51% of generated cover texts can indeed fool eavesdroppers.[1]

1 Introduction

Privacy is central to modern communication systems such as email services and online social networks. To protect privacy, two research fields are established: (1) *cryptography* which encrypts secret messages into codes such that an eavesdropper is unable to decrypt, and (2) *steganography* which encodes messages into cover signals such that an eavesdropper is not even aware a secret message exists (Westfeld and Pfitzmann, 1999; bin Mohamed Amin et al., 2003; Chang and Clark, 2014). One useful cover signal for steganography is natural language text because of its prevalence and innocuity in daily life.

Traditional linguistic steganography methods are mostly edit-based, i.e., they try to directly edit the secret message and transform it into an innocent text that will not raise the eavesdropper's suspicious eyes. Typical strategies include synonym substitution (Topkara et al., 2006), paraphrase substitution (Chang and Clark, 2010), and syntactic transformation (Safaka et al., 2016), applied to various text media such as Email (Tutuncu and Hassan, 2015) and Twitter (Wilson et al., 2014). Although being able to maintain the grammatical correctness of output text, those edit-based methods cannot encode information efficiently. For example, the popular CoverTweet system (Wilson and Ker, 2016) can only encode two bits of information in each tweet on average.

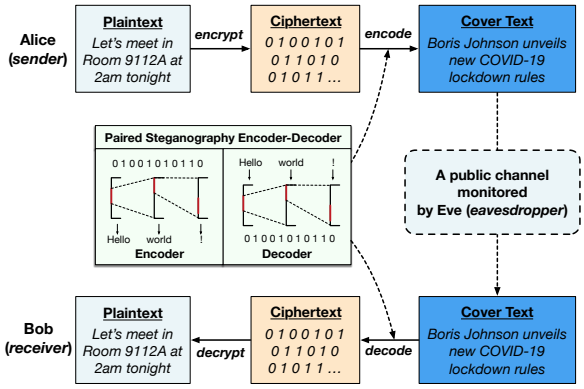

Figure 1: Linguistic steganography pipeline.

Recent advances in neural language models (LMs) (Józefowicz et al., 2016; Radford et al., 2019; Yang et al., 2019a) have enabled a diagram shift from edit-based methods to generation-based methods which directly output a cover text by encoding the message reversibly in the choices of tokens. Various encoding algorithms (Fang et al., 2017; Yang et al., 2019b; Ziegler et al., 2019) have been proposed to leverage neural LMs to generate high-quality cover texts in terms of both fluency and information hiding capacity. However, most of the existing methods do not provide explicit guarantees on the imperceptibility of generated cover text (i.e., to what extent the cover text is indistinguishable from natural texts without hidden messages). One recent exception is the work (Dai

[1]Code and datasets are available at https://github.com/mickeystroller/StegaText.

Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, pages 303–313,
November 16–20, 2020.

and Cai, 2019) which shows the imperceptibility of the method in Fang et al. (2017). Nevertheless, for other more advanced steganography methods (Yang et al., 2019b; Ziegler et al., 2019), their imperceptibilities still remain unknown.

In this work, we propose a new linguistic steganography method with guaranteed imperceptibility. Our new method is built based on the previous study (Ziegler et al., 2019) which views each secret message as a binary fractional number and encodes it using arithmetic coding (Rissanen and Langdon, 1979) with a pretrained neural LM. This method generates cover text tokens one at a time (c.f. Fig. 2). At each time step t, it computes a distribution of the t-th token using the given LM; truncates this distribution to include only top K most likely tokens, and finally outputs the t-th token based on the secret message and the truncated distribution. In their study, this hyperparameter K is the same across all generation steps. We analyze this method's imperceptibility and show it is closely related to the selected K. Specifically, increasing K will improve the method's imperceptibility at the cost of a larger probability of generating rarely-used tokens and slower encoding speed. When the cover text token distribution is flat and close to the uniform distribution, we need a large K to achieve the required imperceptibility guarantee. When the cover text token distribution is concentrated, we can use a small K to avoid generating rarely-used tokens and to increase encoding speed. As different generation steps will witness different underlying cover text token distributions, using a static K is clearly sub-optimal.

To address this issue, we propose a new algorithm SAAC[2] which automatically adjusts K by comparing the truncated cover text token distribution with the original LM's output at each generation step and selects the minimal K that achieves the required imperceptibility. We theoretically prove the SAAC algorithm is near-imperceptible for linguistic steganography and empirically demonstrate its effectiveness on four datasets from various domains. Furthermore, we conduct human evaluations via crowdsourcing and show 51% of cover texts generated by SAAC can indeed fool eavesdropper.

Contributions. This study makes the following contributions: (1) We formally analyze the imperceptibility of arithmetic coding based steganography algorithms; (2) We propose SAAC, a new near-imperceptible linguistic steganography method that encodes secret messages using self-adjusting arithmetic coding with a neural LM; and (3) Extensive experiments on four datasets demonstrate our approach can on average outperform the previous state-of-the-art method by 15.3% and 38.9% in terms of bits/word and KL metrics, respectively.

2 Background

2.1 Linguistic Steganography

We consider the following scenario where Alice (sender) wants to send Bob (receiver) a secret message (plaintext) through a public text channel (e.g., Twitter and Reddit) monitored by Eve (eavesdropper). This is also known as the "prisoner problem" (Simmons, 1984). Eve expects to see fluent texts in this public channel and will suspect every non-fluent text of concealing some hidden messages. Therefore, Alice's goal is to transform the plaintext into a fluent cover text that can pass through Eve's suspicious eyes while ensuring that only Bob can read the secret message.

To achieve this goal, Alice could take the "encrypt-encode" approach (c.f. Fig. 1). Namely, she first encrypts the plaintext into a ciphertext (i.e., a bit sequence indistinguishable from a series of fair coin flips) and then encodes the ciphertext into the cover text using an encoder f. When Bob receives the cover text, he first decodes it into the ciphertext using the decoder f^{-1} and then decrypts the ciphertext into the plaintext. Linguistic steganography research focuses on the encoding/decoding steps, i.e., how to design the encoder that transforms the bit sequence into a fluent cover text and its paired decoder that maps the cover text back to the original bit sequence. Note here we introduce the middle ciphertext for two purposes. First, it increases communication security as more advanced encryption/decryption methods (e.g., AES, RSA, *etc.*) can be used on top of the steganography encoder/decoder. Second, it enlarges the output cover text space by removing the unnecessary restriction that the cover text must be transformed from the original plaintext.

2.2 Statistical Imperceptibility

Notations. A vocabulary $\mathcal{V}$ is a finite set of tokens[3]. A language model (LM) inputs a token

[2] SAAC is short for Self-Adjusting Arithmetic Coding.

[3] Each token can be a single word, a subword unit, or even a character, depending on the tokenizer choice.

sequence $\mathbf{x} = [x_1, x_2, \ldots, x_n]$ and returns the joint probability $\mathbf{P}_{LM}(\mathbf{x})$. From this joint probability, we can derive the conditional probability $\mathbf{P}_{LM}(x_{t+1}|x_1, \ldots, x_t)$ which enables us to sample a text $\mathbf{x}$ by drawing each token $x_t, t = 1, 2, \ldots$, one at a time.

A steganography encoder f inputs a language model $\mathbf{P}_{LM}$ as well as a length-L ciphertext $\mathbf{m} \sim \text{Unif}(\{0,1\}^L)$, and outputs its corresponding cover text $\mathbf{y} = f(\mathbf{m}; \mathbf{P}_{LM})$. To ensure the receiver can uniquely decode the cover text, this encoder function f must be both deterministic and invertible. Moreover, this encoder f, together with the ciphertext distribution and the input LM, implicitly define a distribution of cover text $\mathbf{y}$ which we denote as $\mathbf{Q}(\mathbf{y})$. When cover texts are transmitted in the public channel, this distribution $\mathbf{Q}(\mathbf{y})$ is what an eavesdropper would observe.

Imperceptibility. To avoid raising eavesdropper's suspicion, we want the cover text distribution $\mathbf{Q}$ to be similar to the true natural language distribution (i.e., what this eavesdropper would expect to see in this public channel). Following (Dai and Cai, 2019), we formulate "imperceptibility" using the total variation distance (TVD) as follows:

$$\text{TVD}(\mathbf{P}^*_{LM}, \mathbf{Q}) = \frac{1}{2}\|\mathbf{Q} - \mathbf{P}^*_{LM}\|_1, \tag{1}$$

where $\mathbf{P}^*_{LM}$ denotes the true language distribution. As we approximate $\mathbf{P}^*_{LM}$ using a LM $\mathbf{P}_{LM}$ (e.g., OpenAI GPT-2 (Radford et al., 2019)), we further decompose $\text{TVD}(\mathbf{P}^*_{LM}, \mathbf{Q})$ as follows:

$$\text{TVD}(\mathbf{P}^*_{LM}, \mathbf{Q}) \leq \frac{1}{2}\|\mathbf{P}^*_{LM} - \mathbf{P}_{LM}\|_1 + \frac{1}{2}\|\mathbf{P}_{LM} - \mathbf{Q}\|_1, \tag{2}$$

where the first term measures how good this LM is and the second term, that is the main focus of this study, indicates the gap induced by the steganography encoder. Even without knowing the first term, we can still obtain a relative imperceptibility guarantee based on the second term, which enables us to compare different steganography algorithms.

Using Pinsker's inequality (Fedotov et al., 2003), we set the upper-bound for the total variation distance using the KL divergence[4]:

$$\frac{1}{2}\|\mathbf{P}_{LM} - \mathbf{Q}\|_1 \leq \sqrt{\frac{\ln 2}{2} D_{KL}(\mathbf{Q}\|\mathbf{P}_{LM})}. \tag{3}$$

Then, we further decompose the right hand side of the above inequality based on the additivity of KL divergence and obtain the following result:

$$\frac{1}{2}\|\mathbf{P}_{LM} - \mathbf{Q}\|_1 \leq \sqrt{\frac{\ln 2}{2} \sum_{t=1}^{\infty} D_{KL}(\mathbf{Q}(\cdot|\mathbf{y}_{<t})\|\mathbf{P}_{LM}(\cdot|\mathbf{y}_{<t}))}, \tag{4}$$

where $\mathbf{y}_{<t} = [y_1, \ldots, y_{t-1}]$ is a cover text prefix. $\mathbf{P}_{LM}(\cdot|\mathbf{y}_{<t})$ and $\mathbf{Q}(\cdot|\mathbf{y}_{<t})$ are distributions over the next token y_t conditioned on the prefix $\mathbf{y}_{<t}$ before and after the steganography encoding algorithm, respectively. This inequality provides a formal framework to analyze the imperceptibility of a steganography encoder. Moreover, it implies that in order to achieve the near-imperceptibility, we must guarantee the encoder's output $\mathbf{Q}(\cdot|\mathbf{y}_{<t})$ being close to its input $\mathbf{P}_{LM}(\cdot|\mathbf{y}_{<t})$ at all steps.

3 Self-Adjusting Arithmetic Coding

In this section, we first introduce the general arithmetic coding and discuss its practical limitations. We then present SAAC, a self-adjusting arithmetic coding algorithm and analyze its imperceptibility.

3.1 Arithmetic Coding

Arithmetic coding is a method initially proposed to compress a string of elements sampled from a known probability distribution (Rissanen and Langdon, 1979). For data compression, arithmetic coding is asymptotically optimal in the sense that it can compress information within a long string to its entropy. In practice, it also outperforms the better-known Huffman coding method (Huffman, 1952) because it does not partition the input string into blocks. Traditionally, arithmetic coding encodes a string of elements into a bit sequence. To use such a coding for linguistic steganography, we follow (Ziegler et al., 2019) and reverse the encoding order. Namely, we *encode* a bit sequence (ciphertext) into a string of tokens (cover text) and *decode* a cover text to its original ciphertext.

Encoding. During the encoding stage, we view the bit sequence $\mathbf{m} = [m_1, m_2, \ldots, m_L]$ as the binary representation of a single number $B(\mathbf{m}) = \sum_{i=1}^{L} m_i \times 2^{-i}$. For example, if $\mathbf{m} = [1, 0, 1]$, we have $B(\mathbf{m}) = 1 \times 2^{-1} + 1 \times 2^{-3} = 0.625$.

The encoder generates the cover text token one at a time. At each time step t, the encoder has access to an underlying language model $\mathbf{P}_{LM}$ and considers three things: (1) the number $B(\mathbf{m})$, (2) the cover text prefix $\mathbf{y}_{<t}$, and (3) the current interval $[l_t, u_t)$ (at the beginning of the encoding process, this interval $[l_1, u_1)$ is set to [0, 1), but it will

[4] We will consistently compute KL divergence in base 2.

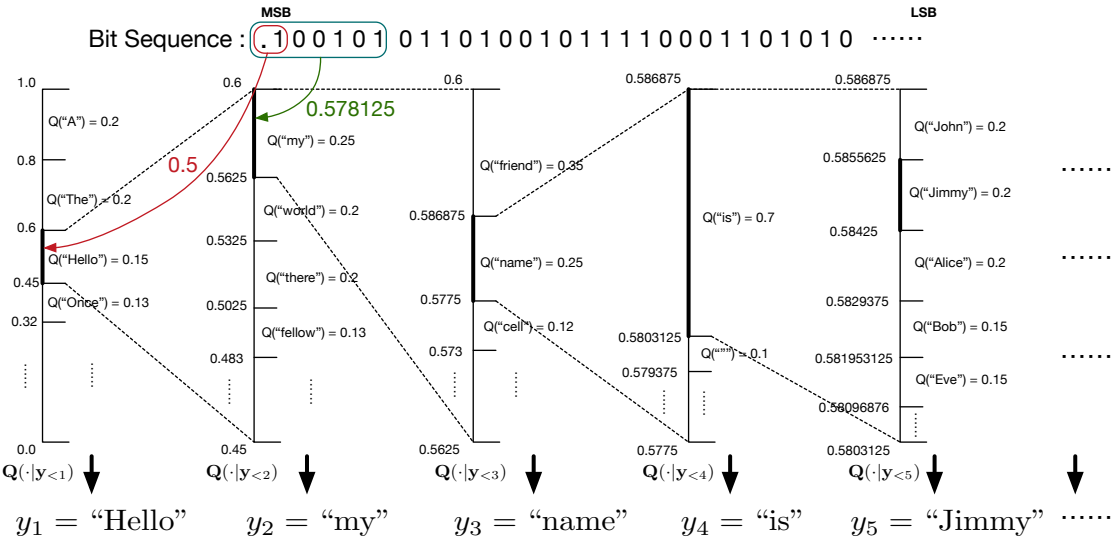

Figure 2: A running example of arithmetic coding. We input a bit sequence (i.e., the ciphertext) with the most significant bit (MSB) at the left and output the encoded cover text.

change). Based on the LM and cover text prefix, the encoder first computes the conditional distribution of the next token $\mathbf{Q}(y_t|\mathbf{y}_{<t})$. Then, it divides the current interval $[l_t, u_t)$ into sub-intervals, each representing a fraction of the current interval proportional to the conditional probability of a possible next token. Whichever interval contains the number $B(\mathbf{m})$ becomes the interval used in the next step (i.e., $[l_{t+1}, u_{t+1})$) and its corresponding token becomes the cover text token y_t. The encoding process stops when all $\mathbf{m}$-prefixed fractions fall into the final interval, that is, the generated cover text unambiguously defines the bit sequence $\mathbf{m}$. Before we discuss and analyze the concrete design of $\mathbf{Q}(\cdot|\mathbf{y}_{<t})$ in the next section, we first present a running example in Figure 2.

Suppose we want to encode a bit sequence $\mathbf{m} = [1, 0, 0, 1, 0, 1, \ldots]$. This bit sequence represents a fraction $B(\mathbf{m}) \in [0.58425, 0.58556)$. At the time step $t = 1$, we divide the initial interval $[0, 1)$ and find $B(\mathbf{m})$ falling into the sub-interval $[0.45, 0.6)$ which induces the first cover text token $y_1 =$ "Hello". At the time step $t = 2$, we further divide the interval $[0.45, 0.6)$ and observe that $B(\mathbf{m})$ belongs to the range $[0.5625, 0.6)$ corresponding to the second cover text token $y_2 =$ "my". We repeat this process until the final interval covers all binary fractions starting with $\mathbf{m}$ and output the generated cover text by then.

Decoding. During the decoding stage, we are given a cover text $\mathbf{y} = [y_1, \ldots, y_n]$ as well as the same language model $\mathbf{P}_{LM}$ used in the encoding stage, and aim to recover the original ciphertext $\mathbf{m}$. We achieve this goal by reversing the encoding process and gradually narrowing the range of possible bit sequences. At each time step t, the decoder first generates the conditional distribution $\mathbf{Q}(y_t|\mathbf{y}_{<t})$. Then, it divides the current interval $[l_t, u_t)$ (initialized to $[0, 1)$) into sub-intervals based on $\mathbf{Q}(y_t|\mathbf{y}_{<t})$ and the one corresponding to y_t becomes the interval used in the next step, that is, $[l_{t+1}, u_{t+1})$. The decoding process stops after we process the last cover text token y_n and outputs the decoded ciphertext to be the shared common prefix of the binary representations of l_{n+1} and u_{n+1}.

3.2 Imperceptibility Analysis

One important issue remained in the general arithmetic coding procedure is how to design the conditional distribution $\mathbf{Q}(\cdot|\mathbf{y}_{<t})$. As we discussed in Section 2.2, this distribution should be close to the underlying model LM. Ideally, we may just set $\mathbf{Q}(\cdot|\mathbf{y}_{<t})$ to be the same as $\mathbf{P}_{LM}(\cdot|\mathbf{y}_{<t})$. However, this naïve design has several problems. First, it may generate a rarely-used cover text token because we are actually reading off the tokens based on the ciphertext, instead of really sampling the LM. This could harm the cover text fluency and raises the eavesdropper's suspicion. Second, $\mathbf{P}_{LM}(\cdot|\mathbf{y}_{<t})$ is a distribution over the entire vocabulary $\mathcal{V}$ (with a full rank $|\mathcal{V}|$) and using it to divide the $[0, 1)$ interval will quickly encounter the precision problem, even if we implement the coding scheme using a fixed precision binary fractions (Witten et al., 1987). Finally, this design further slows down the coding speed and the slow speed is the major weak-

ness of arithmetic coding compared to its rival Huffman method (Duda, 2013).

Due to the above reasons, people in practice will truncate the LM distribution to include only top K most likely tokens (Ziegler et al., 2019), which leads to the following distribution:

$$\mathbf{Q}(y_t|\mathbf{y}_{<t}) \propto \begin{cases} \mathbf{P}_{LM}(y_t|\mathbf{y}_{<t}) & \text{if } y_t \in \mathcal{T}_K(\mathbf{y}_{<t}) \\ 0 & \text{otherwise} \end{cases} \quad (5)$$

where $\mathcal{T}_K(\mathbf{y}_{<t}) = \texttt{argtopK}_{y'}\, \mathbf{P}_{LM}(y'|\mathbf{y}_{<t})$. Accordingly, we have the imperceptibility of one generation step to be:

$$D_{KL}(\mathbf{Q}(y_t|\mathbf{y}_{<t})\|\mathbf{P}_{LM}(y_t|\mathbf{y}_{<t})) = -\log Z_K,$$
$$Z_K = \sum_{y' \in \mathcal{T}_K(\mathbf{y}_{<t})} \mathbf{P}_{LM}(y'|\mathbf{y}_{<t}), \quad (6)$$

where Z_K is essentially the cumulative probability of top K most likely tokens. From this equation, we can see that the imperceptibility of arithmetic coding depends crucially on how the underlying LM distribution concentrates on its top K predictions. Previous study uses the same K across *all* generation steps and ignores the different distribution characteristics in different steps. This strategy is sub-optimal because in some steps, the predefined K is too small to achieve good imperceptibility, while in the other steps, the same K is too large and slows down the encoding speed.

In this study, we propose a new *self-adjusting* arithmetic coding algorithm SAAC to remedy the above problem. The idea is to dynamically select the most appropriate K that satisfies a pre-defined per-step imperceptibility guarantee. Specifically, the sender can set a small per-step imperceptibility gap $\delta \ll 1$ and at time step t, we set the K_t as:

$$K_t = \min(\{K| \sum_{y' \in \mathcal{T}_K(\mathbf{y}_{<t})} \mathbf{P}_{LM}(y'|\mathbf{y}_{<t}) \geq 2^{-\delta}\}). \quad (7)$$

This selected K_t is essentially the smallest K that can achieve the imperceptibility guarantee. As we later show in the experiment, this selected K varies a lot in different steps, which further confirms the sub-optimality of using a static K.

The above method guarantees that each step incurs no more additional imperceptibility than the threshold δ. This makes the imperceptibility of *an entire sequence* dependent on the length of bit sequence. To achieve a length-agnostic imperceptibility bound, we may choose a convergent series for per-step threshold. For example, if we set $\delta_t = \frac{\delta_0}{t^2}$

Dataset	Drug	News	COVID-19	Random
Num. of Sentences	3972	6437	2000	3000
Avg. Num. of Words	19.01	14.30	24.21	—
Avg. Num. of Bits	289.75	211.08	308.65	256

Table 1: Datasets statistics.

and based on the inequality 4 we will have:

$$\frac{1}{2}\|\mathbf{P}_{LM} - \mathbf{Q}\|_1 \leq \sqrt{\frac{\ln 2}{2}\sum_{t=1}^{\infty}\frac{\delta_0}{t^2}} = \sqrt{\frac{\pi^2 \ln 2}{12}\delta_0}. \quad (8)$$

This result shows our proposed SAAC algorithm is near-imperceptible for linguistic steganography.

4 Experiments

4.1 Experiment Setups

Datasets. We conduct our experiments on four datasets from different domains: (1) **Drug** (Ji and Knight, 2018), which contains a set of Reddit comments related to drugs, (2) **News**, which includes a subset of news articles in the CNN/DailyMail dataset (Hermann et al., 2015), (3) **COVID-19**, which is a subset of research papers related to COVID-19 in the CORD-19 dataset (Wang et al., 2020), and (4) **Random**, which is a collection of uniformly sampled bit sequences. The first three datasets contain natural language texts and we convert them into bit sequences[5] following the same process in Ziegler et al. (2019). Table 1 summarizes the dataset statistics.

Compared Methods. We compare the following linguistic steganography methods.

1. Bin-LM (Fang et al., 2017): This method first splits the vocabulary $\mathcal{V}$ into 2^B bins and represents each bin using a B-bit sequence. Then, it chunks the ciphertext into $\lceil L/B \rceil$ blocks and encodes the t-th block by taking the most likely token (determined by the underlying LM) that falls in the t-th bin.
2. RNN-Stega (Yang et al., 2019b): This method first constructs a Huffman tree for top 2^H most likely tokens at each time step t according to $\mathbf{P}_{LM}(\cdot|\mathbf{y}_{<t})$. Then, it follows the bits in ciphertext to sample a cover text token y_t from the constructed Huffman tree. It improves the above Bin-LM method by encoding one or more bits per generated cover text token.
3. Patient-Huffman (Dai and Cai, 2019): This method improves RNN-Stega by explicitly checking if the KL divergence between the

[5] This is essentially the encryption step in Fig. 1.

Methods	Drug		News		COVID-19		Random	
	Bits/Word ↑	D_{KL} ↓	*Bits/Word* ↑	D_{KL} ↓	*Bits/Word* ↑	D_{KL} ↓	*Bits/Word* ↑	D_{KL} ↓
Bin-LM ($B = 1$)	1	1.864	1	1.922	1	1.838	1	1.185
Bin-LM ($B = 2$)	2	2.358	2	2.385	2	2.346	2	2.374
Bin-LM ($B = 3$)	3	2.660	3	2.680	3	2.659	3	2.664
RNN-Stega ($H = 3$)	2.370	1.015	2.387	1.015	2.368	0.999	2.378	0.991
RNN-Stega ($H = 5$)	3.399	0.628	3.393	0.628	3.368	0.624	3.370	0.630
RNN-Stega ($H = 7$)	4.202	0.424	4.202	0.426	4.197	0.426	4.163	0.422
Patient-Huffman ($\epsilon = 0.8$)	1.835	0.269	1.834	0.269	1.844	0.270	1.847	0.271
Patient-Huffman ($\epsilon = 1.0$)	2.147	0.360	2.154	0.361	2.142	0.357	2.148	0.358
Patient-Huffman ($\epsilon = 1.5$)	2.596	0.524	2.583	0.522	2.579	0.519	2.584	0.520
Arithmetic ($K = 300$)	3.497	0.203	3.491	0.209	3.510	0.191	3.466	0.189
Arithmetic ($K = 600$)	4.247	0.162	4.240	0.166	4.289	0.146	3.599	0.160
Arithmetic ($K = 900$)	4.376	0.149	4.358	0.152	4.414	0.131	3.669	0.147
SAAC ($\delta = 0.1$)	4.262	0.153	4.232	0.157	4.301	0.133	4.225	0.136
SAAC ($\delta = 0.05$)	4.451	0.134	4.441	0.138	4.519	0.114	4.419	0.117
SAAC ($\delta = 0.01$)	**4.862**	**0.109**	**4.784**	**0.117**	**4.851**	**0.093**	**4.778**	**0.099**

Table 2: Quantitative performance of linguistic steganography methods across all datasets. Each method has one parameter controlling various tradeoffs (c.f. detailed discussions in Compared Method subsection) and we indicate them in the parentheses. Larger *bits/word* values ↑ and smaller D_{KL} values ↓ indicate better performance.

LM distribution and the Huffman distribution is smaller than a specified threshold ϵ. If the KL divergence is larger than ϵ, it samples from the base LM distribution and *patiently* waits for another opportunity.

4. Arithmetic (Ziegler et al., 2019): This method also uses the arithmetic coding to generate cover text tokens. At each time step t, it truncates the $\mathbf{P}_{LM}(\cdot|\mathbf{y}_{<t})$ distribution to include only top K most likely tokens and samples one cover text tokens from the truncated distribution.
5. SAAC: This method is our proposed Self-Adjusting Arithmetic Coding algorithm which automatically adjusts $\mathbf{P}_{LM}(\cdot|\mathbf{y}_{<t})$ to achieve the required imperceptibility guarantee δ.

Evaluation Metrics. We follow previous studies and evaluate the results using two metrics:

1. *Bits/word*: This metric is the average number of bits that one cover text token can encode. A larger bits/word value indicates the algorithm can encode information more efficiently.
2. D_{KL}: This metric is the KL divergence between the LM distribution and the cover text distribution. A smaller D_{KL} value indicates the model has better imperceptibility (c.f. Section 2.2).

Implementation Details. We implement all compared methods based on the codebase in (Ziegler et al., 2019). All the code and data are publicly available[6]. Specifically, we use PyTorch 1.4.0 and the pretrained OpenAI GPT-2 medium model in the Huggingface library as the underlying LM for all methods. This LM includes 345M parameters and there is no additional parameter introduced by steganography encoding algorithms. For baseline method Bin-LM, we choose its block size B in $[1, 2, 3, 4, 5]$. For RNN-Stega method, we vary the Huffman tree depth H in $[3, 5, 7, 9, 11]$. For Patient-Huffman method, we change the patience threshold ϵ in $[0.8, 1.0, 1.5]$. For Arithmetic method, we select its hyperparameter K ranging from 100 to 1800 with an increment 300 and fix its temperature parameter $\tau = 1$. Finally, we choose the imperceptibility gap δ in our SAAC method in $[0.01, 0.05, 0.1]$. For both Arithmetic and SAAC methods, we implement the arithmetic coding using a fixed 26-bits precision binary fractions. We do not perform any hyperparameter search and directly report all the results in the main text.

Discussions on LM Sharing. We note that all compared methods require the employed LM to be shared between the sender and the receiver beforehand. Therefore, in practice, people typically use a popular public language model (e.g., GPT2) available to everyone. This allows two parties to directly download the same LM from a centroid place (e.g., an OpenAI hosted server) and removes the necessity of sending the LM though some communication channel.

4.2 Experiment Results

Overall Performance. Table 2 shows the overall performance. First, we can see all variable

[6] https://github.com/mickeystroller/StegaText

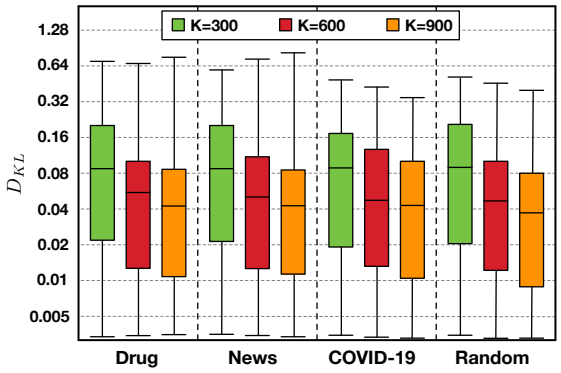

Figure 3: D_{KL} for static arithmetic coding with different Ks. Note that the Y axis is in the log scale.

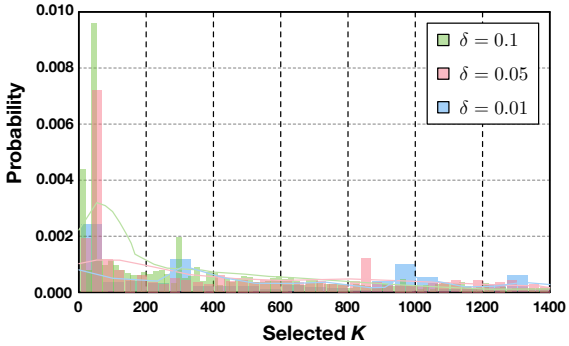

Figure 4: Histogram of selected Ks in our SAAC method's next token distribution $\mathbf{Q}(y_t|\mathbf{y}_{<t})$.

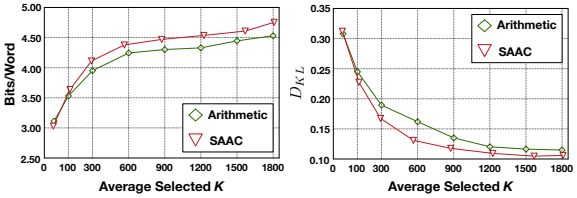

Figure 5: Comparison of baseline Arithmetic method with SAAC across (roughly) the same average Ks. Larger *bits/word* values ↑ and smaller D_{KL} values ↓ indicate better performance.

length coding algorithms (i.e., RNN-Stega, Patient-Huffman, Arithmetic, SAAC) outperform the fixed length coding algorithm Bin-LM. The Bin-LM method achieves worse imperceptibility (i.e., larger D_{KL}) when it encodes message bits at higher compression rate (i.e., larger *Bits/Word*), which aligns with the previous theoretical result in (Dai and Cai, 2019). Second, we observe that Patient-Huffman method improves RNN-Stega as it achieves smaller D_{KL} when *Bits/Word* is kept roughly the same. Third, we find the arithmetic coding based methods (i.e., Arithmetic and SAAC) outperform the Huffman tree based methods (i.e., RNN-Stega and Patient-Huffman). Finally, we can see our proposed SAAC method can beat Arithmetic by automatically choosing the most appropriate K values and thus achieves the best overall performance.

Comparison with Arithmetic Baseline. We further analyze where SAAC's gains over the Arithmetic baseline method come from. Fig. 3 shows the KL divergence between LM's distribution $\mathbf{P}_{LM}$ and steganography encoder's distribution $\mathbf{Q}$ across all time steps. We can see that although most of KL values are less than 0.08, the 95th percentiles are all above 0.32, which means even for large predefined $K = 900$, five percent of generation steps induce KL values larger than 0.32. Fig. 4 shows three histograms of SAAC selected Ks, one for each required imperceptibility bound δ. We observe that these histograms have several modes with one (largest) mode around 50 and one mode larger than 300. This indicates that for a majority of generation steps, choosing a $K < 50$ is enough to guarantee the required imperceptibility bound and thus fixing a static $K = 300$ is a big waste for those steps. Meanwhile, the LM distributions at some generation steps are too "flat" and we indeed need to use a larger K to achieve the required imperceptibility bound δ. Finally, we vary the imperceptibility bound δ and calculate the average K selected by SAAC. Fig. 5 compares the baseline Arithmetic method (of different predefined Ks) with SAAC method that has the (roughly) same average selected K. We can see that using about the same Ks, our SAAC method can clearly outperform the Arithmetic baseline method in terms of both *Bits/word* and D_{KL} metrics.

Efficiency Analysis. We run all our experiments on a machine with one single RTX 8000 GPU and 80 Intel Xeon Gold 6230 CPUs. On average, encoding one sentence takes Bin-LM 2.361 second, RNN-Stega 1.617 second, Arithmetic 2.085 second, Patient-Huffman 4.443 second, and our proposed SAAC method 1.722 second. This result shows dynamic selection of step-wise K will not introduce many computational overhead and can sometimes even improve the efficiency of the static arithmetic coding method.

Case Studies. We show some concrete examples of generated cover texts in Fig. 6. Following (Ziegler et al., 2019), we use an introductory context c for generating the first cover text token (i.e., replace $\mathbf{Q}(\cdot|\mathbf{y}_{<1})$ with $\mathbf{Q}(\cdot|[c;\mathbf{y}_{<1}])$). This strategy helps to improve the cover text quality and will later also be used in the human evaluation. We can see that those generated cover texts are fluent and grammatically correct. Besides, they are

Plaintext	Context	Generated Cover Text
Phylogenetic analysis showed that Bat-SARS-CoV formed a distinct cluster with SARS-CoV. <eos>	The first tally is in. The HEROES Act, passed the House of Representatives Friday evening, would reduce federal revenue by a net total of \$883 billion between 2020 and 2030, according to the Joint Committee on Taxation (JCT). It is highly unlikely that the bill will get signed into law as is, given the White House's veto threat and Senate Republican's view of it as hardly salvageable.	bipartisan bill would require a \$13 billion appropriation at the end of the current fiscal year. Under the Act, you would not collect federal taxes on drugs or make drug-related appropriation if you were a major manufacturer of cannabis. The proposal will likely give Trump the opportunity to only fund the 10 types of confiscated marijuana that the federal government has been conducting a current drug .
molly ultra caps capped at 180mgs will have you flying for hrs clean come down 99 of the time . <eos>	Washington received his initial military training and command with the Virginia Regiment during the French and Indian War. He was later elected to the Virginia House of Burgesses and was named a delegate to the Continental Congress, where he was appointed Commanding General of the nation's Continental Army. Washington led American forces, allied with France, in the defeat of the British at Yorktown.	Confederate troops were assigned a plaque near Berrien\'s Mill, a creek south of New York City. His monument of Lafayette\'s power to bear arms became the very flag of the Union government. Washington returned to Pennsylvania in 1788 when the navy introduced Continental forces to Britain. Five years later the \"Black Ships\" were commissioned

Figure 6: Cover text examples generated by our SAAC method. The context is used for generating the first cover text token (c.f. $\mathbf{Q}(\cdot|\mathbf{y}_{<1})$ in Fig. 2). We can see that those generated cover texts are fluent and effectively hide messages in the original plaintexts.

Step *t*	Already Generated Cover Text $y_{<t}$	Select *K*	Generated Next Token y_t
t=1	""	838	"Following"
t=2	"**Following**"	502	"the"
t=3	"Following **the**"	1036	"retreat"
t=4	"Following the **retreat**"	10	"of"
t=5	"Following the retreat **of**"	399	"the"
t=6	"Following the retreat of **the**"	1059	"British"
t=7	"Following the retreat of the **British**"	138	","
t=8	"Following the retreat of the British **,** "	243	"Washington"
t=9	"Following the retreat of the British , **Washington**"	585	"\'s"
t=10	"Following the retreat of the British , Washington **\'s**"	1563	"comrades"
t=11	"Following the retreat of the British , Washington \'s **comrades**"	28	"said"
...	.....	...	...

Figure 7: One step-by-step example of cover text generation. When less variety exists in the next token distribution $\mathbf{Q}(\cdot|\mathbf{y}_{<t})$, we will choose a smaller K (lines in blue color). Otherwise, we select a larger K (lines in pink color).

topically similar to the provided introductory context and effectively hide messages in the original plaintexts. In Fig. 7, we further show a step-by-step generation example. We can see that in step 4, the next token distribution $\mathbf{Q}(\cdot|\mathbf{y}_{<4})$ following word "retreat" exhibits less variety, and thus we select a small $K = 10$. On the other hand, in step 6, the next token distribution $\mathbf{Q}(\cdot|\mathbf{y}_{<6})$ following word "the" has more variety and we use a larger $K = 1059$ to satisfy the required imperceptibility.

4.3 Human Evaluation

We conduct human evaluation to test whether generated cover texts can indeed fool human eavesdroppers via crowdsourcing. First, we select 100 news articles from the CNN/DM dataset and treat each article's first 3 sentences as the context. Next, we sample 100 ciphertexts uniformly at random and pair each of them with the above 3 sentence context. Then, for each ⟨context, ciphertext⟩ pair, we generate a cover text using different steganography methods, including RNN-Stega with Huffman tree depths 3, 5, 7, arithmetic coding with top Ks 300, 600, 900, and SAAC with imperceptibility gaps 0.1, 0.05, 0.01. Finally, we gather all the generated cover texts; mix them with the original human-written sentences (i.e., the 4th sentence in each news article), and send them to crowd accessors on Amazon Mechanical Turk.

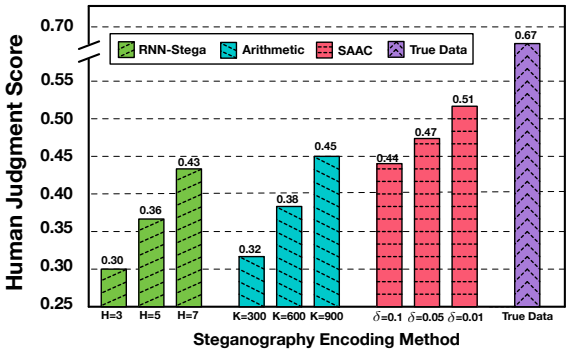

Figure 8: Human evaluation results. Y axis shows the percentage of cover texts (generated by one method) that are considered by humans to be a proper continuation of the context.

In each HIT, the assessor is given one context paired with one sentence and is asked "Given the start of a news article: <context>, is the following a likely next sentence: <sentence>? Yes or No?". We explicitly ask assessors to consider whether this sentence is grammatically correct, contains no factual error, and makes sense in the given context. To ensure the quality of collected data, we require crowd assessors to have a 95% HIT acceptance rate, a minimum of 1000 HITs, and be located in the United States or Canada. Moreover, we include a simple attention check question in 20% of HITs and filter out the results from assessors who do not

pass the attention check.

Fig. 8 shows the human evaluation results. First, we can see this test itself is challenging as only 67% of time people can correctly identify the true follow-up sentence. Second, more encouragingly, we find the cover texts generated by our SAAC algorithm can indeed fool humans 51% of times. For those cover texts that do not pass the human test, we analyze crowd assessor's feedbacks and find they are rejected mostly because they contain some factual errors. Thus, we believe improving the generation factual accuracy is an important direction for future linguistic steganography research.

5 Related Work

Early steganography methods (Marvel et al., 1999; Gopalan, 2003) use image and audio as the cover signal because they have a high information theoretic entropy. However, sending an image or audio recording abruptly though a public channel will likely cause the eavesdropper's suspicion. Thus, linguistic steganography methods are proposed to leverage text as the cover signal because natural language is prevalent and innocuous in daily life.

Linguistic steganography methods can be categorized into two types, edit-based or generation-based (Bennett, 2004). Edit-based methods try to directly edit the secret message and transform it into an innocent text. Typical transformations are synonym substitution (Topkara et al., 2006; Chang and Clark, 2014), paraphrase substitution (Chang and Clark, 2010; Ji and Knight, 2018), and syntactic transformation (Thamaraiselvan and Saradha, 2015; Safaka et al., 2016). Instead of editing all words in the secret message, (Zhang et al., 2014, 2015) take an entity-oriented view and focus on encoding/decoding morphs of important entities in the message. Finally, some work (Grosvald and Orgun, 2011; Wilson et al., 2014) allows human agents to assist the cover text generation process.

One major limitation of edit-based methods is that they cannot encode information efficiently. (Wilson and Ker, 2016) show the popular CoverTweet system (Wilson et al., 2014) can encode only two bits information in each transformed tweet on average. To address this limitation, generation-based methods try directly output the cover text based on the secret message. Early study (Chapman and Davida, 1997) utilizes a generative grammar to output the cover text. More recently, people leverage a neural language model for linguistic steganography. One pioneering work by (Fang et al., 2017) divides the message bits into equal-size blocks and encodes each block using one cover text token. (Yang et al., 2019b) improves the above method by constructing a Huffman tree and encoding the message in variable length chunks via a Huffman tree. (Dai and Cai, 2019) presents the first theoretical analysis of the above two methods and proposes a modified Huffman algorithm. The method most related to this study is (Ziegler et al., 2019) where the arithmetic coding algorithm is introduced for steganography. In this study, we present a more formal analysis of arithmetic coding based steganography method and propose a better self-adjusting algorithm to achieve the statistical imperceptibility.

6 Discussions and Future Work

This work presents a new linguistic steganography method that encodes secret messages using self-adjusting arithmetic coding. We formally prove this method is near-imperceptible and empirically show it achieves the state-of-the-art results on various text corpora. There are several directions we will further explore in the future. First, we may combine the edit-based steganography with generative steganography method by first transforming the original plaintext in a semantics-preserving way and then encoding the transformed plaintext. Second, we will study whether this current method is still effective when a small-scale neural LM (e.g., distilGPT-2) is applied. Finally, this study assumes a passive eavesdropper who does not modify the transmitted cover text. Adapting the current methods to be robust to an active eavesdropper who may alter the cover text is another interesting direction.

Acknowledgements

Research was sponsored in part by US DARPA SocialSim Program No. W911NF-17-C-0099, NSF IIS-19-56151, IIS-17-41317, IIS 17-04532, and IIS 16-18481, and DTRA HDTRA11810026. Any opinions, findings or recommendations expressed herein are those of the authors and should not be interpreted as necessarily representing the views, either expressed or implied, of DARPA or the U.S. Government. We thank anonymous reviewers for valuable and insightful feedback.

References

Krista Bennett. 2004. Linguistic steganography: Survey, analysis, and robustness concerns for hiding information in text.

Ching-Yun Chang and Stephen Clark. 2010. Linguistic steganography using automatically generated paraphrases. In *HLT-NAACL*.

Ching-Yun Chang and Stephen Clark. 2014. Practical linguistic steganography using contextual synonym substitution and a novel vertex coding method. *Computational Linguistics*, 40:403–448.

Mark Chapman and George I. Davida. 1997. Hiding the hidden: A software system for concealing ciphertext as innocuous text. In *ICICS*.

Falcon Z. Dai and Zheng Cai. 2019. Towards near-imperceptible steganographic text. In *ACL*.

Jarek Duda. 2013. Asymmetric numeral systems: entropy coding combining speed of huffman coding with compression rate of arithmetic coding.

Tina Fang, Martin Jaggi, and Katerina J. Argyraki. 2017. Generating steganographic text with lstms. In *ACL*.

Alexei A. Fedotov, Peter Harremoës, and Flemming Topsøe. 2003. Refinements of pinsker's inequality. *IEEE Trans. Inf. Theory*, 49:1491–1498.

Kaliappan Gopalan. 2003. Audio steganography using bit modification. In *International Conference on Multimedia and Expo*.

Michael Grosvald and C Orhan Orgun. 2011. Free from the cover text: a human-generated natural language approach to text-based steganography. *Journal of Information Hiding and Multimedia Signal Processing*, 2(2):133–141.

Karl Moritz Hermann, Tomás Kociský, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. 2015. Teaching machines to read and comprehend. In *NIPS*.

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

Heng Ji and Kevin Knight. 2018. Creative language encoding under censorship. In *COLING*.

Rafal Józefowicz, Oriol Vinyals, Mike Schuster, Noam Shazeer, and Yonghui Wu. 2016. Exploring the limits of language modeling. *ArXiv*.

Lisa M Marvel, Charles G Boncelet, and Charles T Retter. 1999. Spread spectrum image steganography. In *IEEE Transactions on image processing*.

Muhalim bin Mohamed Amin, Mazleena bt. Salleh, Subariah Ibrahim, Mohd. Rozi b. Katmin, and M. Z. I. Shamsuddin. 2003. Information hiding using steganography. *4th National Conference of Telecommunication Technology, 2003. NCTT 2003 Proceedings.*, pages 21–25.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners.

Jorma Rissanen and Glen G. Langdon. 1979. Arithmetic coding. *IBM J. Res. Dev.*, 23:149–162.

Iris Safaka, Christina Fragouli, and Katerina J. Argyraki. 2016. Matryoshka: Hiding secret communication in plain sight. In *FOCI*.

Gustavus J Simmons. 1984. The prisoners' problem and the subliminal channel. In *Advances in Cryptology*, pages 51–67. Springer.

R. Thamaraiselvan and A. Saradha. 2015. Text-based steganography using cover text free human-generated natural language (hgnl) approach.

Umut Topkara, Mercan Topkara, and Mikhail J. Atallah. 2006. The hiding virtues of ambiguity: quantifiably resilient watermarking of natural language text through synonym substitutions. In *Multimedia and Secruity*.

Kemal Tutuncu and Abdikarim Abi Hassan. 2015. New approach in e-mail based text steganography. *International Journal of Intelligent Systems and Applications in Engineering*, 3:54–56.

Lucy Lu Wang, Kyle Lo, Yoganand Chandrasekhar, Russell Reas, Jiangjiang Yang, Darrin Eide, Kathryn Funk, Rodney Michael Kinney, Ziyang Liu, William. Merrill, Paul Mooney, Dewey A. Murdick, Devvret Rishi, Jerry Sheehan, Zhihong Shen, Brandon Stilson, Alex D. Wade, Kuansan Wang, Christopher Wilhelm, Boya Xie, Douglas M. Raymond, Daniel S. Weld, Oren Etzioni, and Sebastian Kohlmeier. 2020. Cord-19: The covid-19 open research dataset. *ArXiv*, abs/2004.10706.

Andreas Westfeld and Andreas Pfitzmann. 1999. Attacks on steganographic systems. In *Information Hiding*.

Alex Wilson, Phil Blunsom, and Andrew D. Ker. 2014. Linguistic steganography on twitter: hierarchical language modeling with manual interaction. In *Electronic Imaging*.

Alex Wilson and Andrew D. Ker. 2016. Avoiding detection on twitter: embedding strategies for linguistic steganography. In *Media Watermarking, Security, and Forensics*.

Ian H. Witten, Radford M. Neal, and John G. Cleary. 1987. Arithmetic coding for data compression. *Commun. ACM*, 30:520–540.

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime G. Carbonell, Ruslan Salakhutdinov, and Quoc V. Le. 2019a. Xlnet: Generalized autoregressive pretraining for language understanding. In *NeurIPS*.

Zhong-Liang Yang, Xiaoqing Guo, Zi-Ming Chen, Yongfeng Huang, and Yu-Jin Zhang. 2019b. Rnn-stega: Linguistic steganography based on recurrent neural networks. *IEEE Transactions on Information Forensics and Security*, 14:1280–1295.

Boliang Zhang, Hongzhao Huang, Xiaoman Pan, Heng Ji, Kevin Knight, Zhen Wen, Yizhou Sun, Jiawei Han, and Bülent Yener. 2014. Be appropriate and funny: Automatic entity morph encoding. In *ACL*.

Boliang Zhang, Hongzhao Huang, Xiaoman Pan, Sujian Li, Chin-Yew Lin, Heng Ji, Kevin Knight, Zhen Wen, Yizhou Sun, Jiawei Han, and Bülent Yener. 2015. Context-aware entity morph decoding. In *ACL*.

Zachary M. Ziegler, Yuntian Deng, and Alexander M. Rush. 2019. Neural linguistic steganography. In *EMNLP*.