Meteor: Cryptographically Secure Steganography for Realistic Distributions

Gabriel Kaptchuk
Boston University
Boston, United States
kaptchuk@bu.edu

Tushar M. Jois, Matthew Green, Aviel D. Rubin
Johns Hopkins University
Baltimore, United States
{jois,mgreen,rubin}@cs.jhu.edu

ABSTRACT

Despite a long history of research and wide-spread applications to censorship resistant systems, practical steganographic systems capable of embedding messages into realistic communication distributions, like text, do not exist. We identify two primary impediments to deploying universal steganography: (1) prior work leaves the difficult problem of finding samplers for non-trivial distributions unaddressed, and (2) prior constructions have impractical minimum entropy requirements. We investigate using generative models as steganographic samplers, as they represent the best known technique for approximating human communication. Additionally, we study methods to overcome the entropy requirement, including evaluating existing techniques and designing a new steganographic protocol, called Meteor. The resulting protocols are provably indistinguishable from honest model output and represent an important step towards practical steganographic communication for mundane communication channels. We implement Meteor and evaluate it on multiple computation environments with multiple generative models.

CCS CONCEPTS

• **Security and privacy** → **Cryptography**; **Network security**; **Pseudonymity, anonymity and untraceability**.

KEYWORDS

Steganography; Applied Cryptography; Generative Models; Censorship Resistance

ACM Reference Format:
Gabriel Kaptchuk and Tushar M. Jois, Matthew Green, Aviel D. Rubin. 2021. Meteor: Cryptographically Secure Steganography for Realistic Distributions. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security (CCS '21), November 15–19, 2021, Virtual Event, Republic of Korea.* ACM, New York, NY, USA, 21 pages. https://doi.org/10.1145/3460120.3484550

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

CCS '21, November 15–19, 2021, Virtual Event, Republic of Korea

© 2021 Copyright held by the owner/author(s). Publication rights licensed to ACM.

ACM ISBN 978-1-4503-8454-4/21/11...$15.00
https://doi.org/10.1145/3460120.3484550

1 INTRODUCTION

The past several years have seen a proliferation of encrypted communication systems designed to withstand even sophisticated, nation-state attackers [1, 2]. While these systems maintain the confidentiality of plaintext messages, the data transmitted by these tools is easily identifiable as encrypted communication. This makes these protocols easy targets for repressive regimes that are interested in limiting free communication [3, 4]: for example, using network censorship techniques such as those practiced by countries like China [5–7]. Concrete attempts to suppress the encrypted communication technologies used to evade censors are now underway. For example, China's Great Firewall (GFW) not only prevents users from accessing content deemed subversive, but it also actively detects and blocks encryption-based censorship circumvention technologies such as Tor [8–10].

In regimes where cleartext communication is expected, the mere *use* of encryption may be viewed as an indication of malicious or subversive intent. To work around blocking and avoid suspicion, users must make their communications look mundane. For instance, Tor users in China have begun to leverage steganographic techniques such as ScrambleSuit/obfs4 [11], SkypeMorph [12], StegoTorus [13], TapDance [14, 15], and Format-Transforming Encryption [16]. These techniques embed messages into traffic that censors consider acceptable.

While the current generation of steganographic tools is sufficient to evade current censorship techniques, these tools are unlikely to remain a sustainable solution in the future. While some tools provide strong cryptographic guarantees [12, 17, 18], this is achievable only because they encode messages into (pseudo-)random covertext channels, *i.e.*, replacing a random or encrypted stream with a chosen pseudorandom ciphertext. Unfortunately, there is no guarantee that such channels will continue to be available: a censor can systematically undermine such tools by preventing the delivery of encrypted traffic for which it does not have a suitable trapdoor, (*i.e.*, an access mechanism), or by selectively degrading the quality of encrypted channels. An audacious, repressive regime could even consider *all encryption* to be subversive, and drop all packets not explicitly recognizable as meaningful plaintext. Rigorous studies of the capabilities of the current GFW focus on other techniques [19–22], but there is anecdotal evidence that encryption suppression has begun to occur [23], including the blocking of some TLS 1.3 connections [24].

Steganography for Realistic Communication Channels. To combat extreme censorship, there is a need for steganographic protocols that can produce *stegotext* (the steganographic equivalent of ciphertext) that closely mimics real, innocuous communication.

With such techniques, it would be impossible for a censor to *selectively* repress communications, as subversive messages could hide in benign communication. For instance, if dissidents could encode secret messages into mundane appearing emails, web-forum posts, or other forms of "normal" human communication, censorship would be impractical. The ideal tool for this task is universal steganography: schemes which are able to securely hide sensitive information in arbitrary *covertext channels* (the steganographic term for communication channels). Even if the censor suspects something, the secret message cannot be found — nor is there any statistical evidence of its existence.

A key challenge in this setting is to identify a generator of some useful distribution where sampling will produce symbols that are identical (or at least close) to ordinary content present in a communications channel. Given such a generator, numerous *universal* steganographic constructions have been proposed that can sample from this distribution to produce a stegotext [25–31]. Unfortunately, identifying useful generators is challenging, particularly for complex distributions such as natural language text. To our knowledge, the only practical attempts to achieve practical steganography such natural communication channels have come from the natural language processing (NLP) community [32–44]. While the resulting text is quite convincing, these works largely rely on insecure steganographic constructions that fail to achieve formal definitions [45–50]. In this work, we focus our attention on constructing provably secure steganography for the kinds of distributions that would be difficult for a censor block without suffering significant social repercussions. To do so, we identify and overcome the barriers to using steganographic techniques as practical tools to combat network censorship.

Overcoming Shortcomings of Existing Steganographic Techniques. Steganographic schemes that are able to encode into any communication channel have been the subject of significant theoretical work, *e.g.*, [25–31]. Generally, constructions rely on the existence of an efficient *sampler* functionality that, on demand, outputs a *token* (sometimes referred to as a *document*) that could appear in the covertext channel. These tokens are then run through a hash function that maps the token to a small, fixed number of bits. Using rejection sampling, an encoder can find a token that maps to some specific, desired bits, usually the first few bits of a pseudo-random ciphertext. By repeatedly using this technique, a sender can encode an entire ciphertext into a series of tokens, and a receiver can recover the message by hashing the tokens and decrypting the resulting bits. Security of these approaches relies on the (pseudo-)randomness of the ciphertext and carefully controlling the bias introduced by rejection sampling.

There are two significant barriers to using universal steganographic systems for censorship-resistant communication: (1) the lack of appropriate samplers for real, desirable covertext channels, like English text, and (2) the minimum entropy bounds required to use existing techniques.

(1) Generative Models as Steganographic Samplers. Existing work leaves samplers as an implementation detail. However, finding a suitable sampler is critical to a practical construction. Sampling is straightforward for simple covertext channels for which the instantaneous probability distribution over the next token in the channel can be measured and efficiently computed: draw random coins and use them to randomly select an output from the explicit probability distribution. Natural communication channels — the most useful targets for practical steganography — are generally too complex for such naïve sampling techniques. For example, it is infeasible to perfectly measure the distribution of the English language, and the usage of English continues to evolve and change.

Without access to perfect samplers, we explore steganographic samplers that *approximate* the target channel. While this relaxation introduces the risk that an adversary can detect a steganographic message by distinguishing between the real channel and the approximation, *this is the best we can do* when perfect samplers cannot be constructed. In this work, we propose to use *generative models* as steganographic samplers, as these models are the best technique for approximating complex distributions like text-based communication. While these models are still far from perfect, the quality of generated content is impressive [51, 52] and continues to improve, raising concerns about the disastrous societal impact of misuse [53].

Generative models operate by taking some context and model parameters and outputting an explicit probability distribution over the next token (for example, a character or a word) to follow that context. During typical use, the next token to add to the output is randomly sampled from this explicit distribution. This process is then repeated, updating the context with the previously selected tokens, until the output is of the desired length. Model creation, or training, processes vast amounts of data to set model parameters and structure such that the resulting output distributions approximate the true distributions in the training data.

The use of generative models as steganographic samplers facilitates the creation of stegotext that are provably indistinguishable from honest model output, and thus good approximations of real communication (although not indistinguishable from real communication). We show that the nature of generative models, *i.e.* a shared (public) model and explicit probability distribution, can be leveraged to significantly increase concrete efficiency of steganographic schemes. Our key insight is that a sender and receiver can keep their models synchronized, and thus recover the *same* explicit probability distribution from which each token is selected, a departure from traditional steganographic models. This allows the receiver to make inferences about the random coins used by the sender when *sampling* each token. If the message is embedded into this randomness (in an appropriately protected manner), the receiver can use these inferences to extract the original message.

(2) Steganography for Channels with High Entropy Variability. The second barrier is the channel *entropy* requirements of most existing schemes. Specifically, most universal steganographic schemes are only capable of encoding messages into covertext channels if that channel maintains some *minimum entropy*, no matter the context. Real communication channels often encounter moments of low (or even zero) entropy, where the remaining contents of the message are fairly proscribed based on the prior context. For instance, if a sentence generated by a model trained on encyclopedia entries begins with "The largest carnivore of the Cretaceous period was the Tyranosaurus" with overwhelming probability the next token will be "Rex", and any other token would be very unlikely. In many existing steganographic proposals, if the hash of this next token

(*i.e.* Hash("Rex")) does not match the next bits of the ciphertext, no amount of rejection sampling will help the encoder find an appropriate token, forcing them to restart or abort. Thus, to ensure that the probability of this failure condition is small, most classical constructions impose impractical entropy requirements. We investigate overcoming this problem in two ways. First, we evaluate the practicality of known techniques for public-key steganography, in which an arbitrary communication channel is compiled into one with sufficient entropy. Second, we leverage the structure of generative models to create a new, symmetric key steganographic encoding scheme called Meteor. Our key observation is that the best way to adapt to variable entropy is to fluidly change the encoding rate to be proportional to the instantaneous entropy. Together, these could be used to build hybrid steganography, where the public-key scheme is used to transmit a key for a symmetric key scheme.

Contributions. In this work we explore the use of modern generative models as samplers for provably secure steganographic schemes. This provides the groundwork for steganography that convincingly imitates natural, human communication once the differences between generative models and true communication become imperceptible. In doing so, we have the following contributions:

- **Evaluation of Classical Public-Key Steganography in Practice.** We evaluate the use of a classical public-key steganographic scheme from [54]. We investigate adapting this scheme to work with generative models, and show that known techniques introduce prohibitively high overhead.
- **Meteor.** We present Meteor, a new symmetric-key, stateful, provably secure, steganographic system that naturally adapts to highly variable entropy. We provide formalization for the underlying techniques so that they can be easily applied to new generative models as they are developed.
- **Implementation and Benchmarking.** Additionally, we implement Meteor and evaluate its performance in multiple computing environments, including on GPU, CPU, and mobile. We focus primarily on English text as our target distribution, but also investigate protocol generation. To the best of our knowledge, our work is the first to evaluate the feasibility of a provably secure, universal steganographic using text-like covertext channels by giving concrete timing measurements.
- **Comparison with Informal Steganographic Work.** In addition to the constructive contributions above, we survey the insecure steganographic techniques present in recent work from the NLP community [32–44]. We discuss modeling differences and give intuition for why these protocols are not provably secure.

Limitations. We want to be clear about the limitations of our work.

- Our work does not address how well a machine learning model can approximate an existing, "real" communication channel. Answering this question will be crucial for deployment and is the focus of significant, machine learning research effort [51, 52]. Regardless of the current state of generative models and how well they imitate real communication, our work is valuable for the following reasons:
  (1) The ever-changing and poorly defined nature of real communication channels makes sampling an inherently hard problem; channels of interest are impossible to perfectly measure and characterize. This means the imperceptibility of steganography for these channels will always be bounded by the accuracy of the available *approximation* techniques. The best approximation tool available in the existing literature is generative modeling [55], and thus we focus on integrating them into steganographic systems.
  (2) We prepare for a future in which encrypted and pseudorandom communications are suppressed, breaking existing tools. As such, the current inadequacies of generative models should not be seen as a limitation of our work; the quality of generative models has steadily improved [52] and is likely to continue improving. Once the techniques we develop are necessary in practice, there is hope that generative models are sufficiently mature to produce convincingly real output.
  (3) Finally, there already exist applications in which sending model output is normal. For instance, artificial intelligence powered by machine learning models regularly contribute to news articles [56, 57], create art [58, 59], and create other digital content [60, 61]. Theses channels can be used to facilitate cryptographically secure steganographic communication using our techniques today.
- In Meteor, we assume that the sender and receiver (along with the censor) access the same generative model. While this requirement might seem like a limitation, we reiterate that the security of the scheme does not require that the model remain private. As such, this model is similar to the common random string model common in cryptography. Additionally, it is common practice to share high quality models publicly [51, 52, 62], and these models would outperform anything an individual could train. As such, we believe that this assumption is reasonable and show it yields significant performance gains.

Deployment Scenario. Our work focuses on the following scenario: Imagine a sender (*e.g.* news website, compatriot) attempting to communicate with a receiver (*e.g.* political dissident) in the presence of a censor (*e.g.* state actor) with control over the communications network. We assume that the sender and receiver agree on any necessary key information out of band and select an appropriate (public) generative model. Although we focus on English text in this work, the generative model could be for any natural communication channel. The sender and receiver then initiate communication over an existing communication channel, using a steganographic encoder parameterized by the generative model to select the tokens they send over the channel. The censor attempts to determine if the output of the generative model being exchanged between the sender and receiver is subversive or mundane. We note that practical deployments of these techniques would likely incorporate best practices to achieve forward secrecy, post compromise security, and asynchronicity, possibly by using parts of the Signal protocol [1].

Organization. In Section 2, we give background and assess related work on classical steganographic techniques from the cryptographic community, how steganography is currently used in practice, and generative models. In Section 3, we give formal definitions for steganography. In Section 4, we explore using existing techniques and steganographic schemes to build public-key steganography for English text distributions. In Section 5, we give a construction of a new, symmetric key steganographic system,

Algorithm 1: Public-Key Encoding Scheme from [54]

Input: Plaintext Message m, Distribution $\mathcal{D}$, Sampling Bound k, public-key pk
Output: Stegotext Message c
$x \leftarrow \mathsf{PseudorandomPKEncrypt}(pk, m)$
Let $x_0||x_1||\dots||x_{|x|} \leftarrow x$
$c \leftarrow \varepsilon$
for $i < |x|$ **do**
  $c_i \leftarrow \mathsf{Sample}(\mathcal{D})$
  $j \leftarrow 0$
  while $f(c_i) \neq x_i$ *and* $j < k$ **do**
    $c_i \leftarrow \mathsf{Sample}(\mathcal{D})$
    $j \leftarrow j + 1$
  $c \leftarrow c||c_i$
Output c

Figure 1: The public-key steganography scheme from [54]. PseudorandomPKEncrypt **is the encryption routine for a pseudorandom, public-key encryption scheme.** Sample **randomly selects an token from the covertext space according to the distribution** $\mathcal{D}$.

Meteor, and analyze its efficiency and security. In Section 6, we give implementation details for Meteor and evaluate the efficiency of using Meteor on different systems. Finally, in Section 7 we discuss existing work from the NLP community and show why it is insecure.

2 BACKGROUND AND RELATED WORK

Classical Steganography. Since Simmons' first formalization of steganographic communication [25], significant effort has been devoted to theoretical steganography. Early work focused on achieving information-theoretic security [26, 27, 63, 64] before moving on to cryptographic [28–30] and statistical [65–67] notions of steganography. The are many symmetric-key constructions [27, 28, 68], public-key constructions [29, 30, 69, 70], and even identity based constructions [71]. Relatively little on formal steganography has been in the last 15 years, although there are recent works considering the boundaries of steganography [72], the related problem of backdoor resistance [73] and keyless steganography [74].

In general, the steganographic schemes presented in the literature rely on rejection sampling to find randomly selected elements of the covertext distribution that hash to desired bits. Given space constrains, we cannot describe and compare to all prior work. For a representative example, consider the public-key steganographic scheme from [29, 54] presented in Algorithm 1. First, the encoder uses a pseudorandom, public-key encryption scheme to encrypt the message. Then, one bit x_i at a time, the encoder uses rejection sampling to find a token c_i in the covertext distribution $\mathcal{D}$ such that $f(c_i) = x_i$, where f is a perfectly unbiased function over $\mathcal{D}$. We omit the formal description of the simple decoding algorithm, in which the receiver simply computes $f(c_i)$ for all i, concatenates the bits, and decrypts the result.

Security for such schemes is simple to see: each bit of the encrypted message is random, by the pseudorandomness of the cipher, and each token in the stegotext is randomly sampled from the true distribution, with no bias introduced by the hash function (by definition). As such, the distribution of the stegotext matches the covertext exactly. However, if no unbiased hash function exists, as none do for infinitely many distributions [54], a universal hash function can be used instead, and the bias it introduces must be carefully controlled.

These rejection sampling algorithms fail when the distribution has very low entropy. In such cases, it is unlikely an unbiased hash function will exist, so a universal hash function must be used. One of two possible problems is likely to occur. (1) During sampling, it is possible that the sampling bound k may be exceeded without finding an acceptable token, after which the encoder simply appends a randomly sampled token. Importantly, the receiver *can not detect* that this error has occurred, or indeed how many such errors are contained in the message, and will just get a decryption error during decoding. (2) If k is set very high, it may be possible to find a token that hashes to the correct value, at the cost of introducing noticeable bias in the output distribution. As such, it is critical that the distribution maintain some minimum amount of entropy. To our knowledge, only two prior works [31, 54] build stateful steganographic techniques that avoid the minimum entropy requirement. Focusing on asymptotic performance, both rely on error correcting codes and have poor practical performance.

In the closest related work, the authors of [75] theoretically analyze the limitations of using Markov Models as steganographic samplers. The prove that any sampler with limited history cannot perfectly imitate the true covertext channel. Our work overcomes this limitations by considering the output of the model the target covertext distribution.

In our work we consider more powerful machine learning models and allow the sender and receiver to share access to the same public model. This is a departure from prior steganographic work, motivated by the public availability of high quality models [51, 52, 62] and because this relaxation introduces significant efficiency gains. As there has been, to our knowledge, no work testing the practical efficiency of secure steganographic constructions for complex channels, no other work considers this model.

Current Steganography in Practice. The main contemporary use for steganography is to connect to Tor ([8–10]) without being flagged by the plethora of surveillance mechanisms used by censors [19]. Steganographic techniques include protocol obfuscation, *e.g.*, obfs4/ScrambleSuit [11], domain fronting [76], or mimicry, *e.g.*, SkypeMorph [12], FTEProxy [16], StegoTorus [13], CensorProofer [17], and FreeWave [18]. Although these tools allow users to circumvent censors today, they are quite brittle. For example, protocol obfuscation techniques are not cryptographically secure and rely on censors defaulting open, *i.e.*, a message should be considered innocuous when its protocol cannot be identified. Protocol mimicry techniques, encoding one protocol into another, are not always cryptographic and often fail when protocols are under-specified or change without warning [77].

Modern steganographic techniques that are cryptographically secure include tools like SkypeMorph [12], CensorProofer [17], and FreeWave [18], that tunnel information through Voice-Over-IP (VoIP) traffic, which is usually encrypted with a pseudorandom cipher. Once encrypted communication has started, a sender can replace the normal, VoIP encrypted stream with a different encrypted stream carrying the secret message. By the security of the cipher, a censor cannot detect that the contents of the encrypted channel have been replaced and the communication looks like normal, encrypted VoIP traffic. If access to encrypted or pseudorandom communication channels were suppressed, these tools would no longer work.

There have been small-scale tests [78] at deploying cryptography secure steganographic tagging via ISP level infrastructure changes, as suggested in Telex [14] and TapDance [15]. These tags indicate that a message should be redirected to another server, but stop short of hiding full messages. These tags also critically rely on the presence of (pseudo-)random fields in innocuous protocol traffic.

Practical work has been done in the field of format-transforming encryption (FTE), such as [79–82]. These approaches require senders to explicitly describe the desired covertext channel distribution, an error-prone process requiring significant manual effort and is infeasible for natural communication. None of these applications, however, provide any kind of formal steganographic guarantee. Recently, there has also been work attempting to leverage machine learning techniques to generate steganographic images, *i.e.* [83–88], but none of these systems provide provable security.

Generative Neural Networks. Generative modeling aims to create new data according to some distribution using a model trained on input data from that distribution. High quality language models [51, 52], are generative neural networks, which use neural network primitives. The model itself contains a large number of "neurons" connected together in a weighted graph of "layers", which "activate" as the input is propagated through the network. Unlike traditional feed-forward neural networks used in classification tasks, generative networks maintain internal state over several inputs to generate new text. Training these models typically ingests data in an effort to set weights to neurons, such that the model's output matches the input data distribution; in other words, the network "learns" the relationships between neurons based on the input. The first practical development in this field was the creation of long short-term memory (LSTM) networks [89]. LSTM networks are found in machine translation [90, 91], speech recognition, and language modeling [55]. The transformer architecture [92], exemplified by the GPT series of models [51, 52], is also becoming popular, with results that are increasingly convincing [53].

After training, the model can be put to work. Each iteration of the model proceeds as follows: the model takes as input its previous state, or "context". As the context propagates through the network, a subset of neurons activate in each layer (based on previously trained weights), up until the "output layer". The output layer has one neuron for output token, and uses the activated neurons to assign each token a weight between 0 and 1. The model uses its trained weights and the context input to generate a distribution of possible tokens, each with a probability assigned. The model uses random weighted sampling to select a token from this distribution, returning the chosen token as output. Finally, the returned token is appended to the context and the next iteration begins.

We note there is work focusing on differentiating machine-generated text from human-generated text [93–95]. It has yet to be seen if these techniques will remain effective as machine learning algorithms continue to improve, setting the stage for an "arms race" between generative models and distinguishers [96].

3 DEFINITIONS

3.1 Symmetric Steganography

The new construction in this work is symmetric-key stenography, so for completeness we include symmetric-key definitions. The definitions for public-key steganography are a straightforward adaptation of the definitions provided here and can be found in [54].

A symmetric steganographic scheme $\Sigma_{\mathcal{D}}$ is a triple of possibly probabilistic algorithms, $\Sigma_{\mathcal{D}} = (\mathsf{KeyGen}_{\mathcal{D}}, \mathsf{Encode}_{\mathcal{D}}, \mathsf{Decode}_{\mathcal{D}})$ parameterized by a covertext channel distribution $\mathcal{D}$.

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

We use the history notation that is used in a number of previous works [28, 29], but not universally adopted. The history input to the encode and decode functions capture the notion that covertext channels may be stateful. For instance, members of the ordered set $\mathcal{H}$ could be text messages previously exchanged between two parties or the opening messages of a TCP handshake.

Correctness. A steganographic protocol must be correct, *i.e.* except with negligible probability an encoded message can be recovered using the decode algorithm. Formally, for any $k \leftarrow \mathsf{KeyGen}_{\mathcal{D}}(1^{\lambda})$,

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

Security. We adopt a symmetric-key analog of the security definitions for a steganographic system secure against a chosen hiddentext attacks in [29], similar to the real-or-random games used in other cryptographic notions. Intuitively, a steganographic protocol $\Sigma_{\mathcal{D}}$ is secure if all ppt. adversaries are unable to distinguish with non-negligible advantage if they have access to encoding oracle $\mathsf{Encode}_{\mathcal{D}}(k, \cdot, \cdot)$ or a random sampling oracle $O_{\mathcal{D}}(\cdot, \cdot)$ that returns a sample of the appropriate length. This ensures that an adversary wishing to block encoded messages will be forced to block innocuous messages as well. We allow the adversary to not only have a sampling oracle to the distribution (as in [28]), but also have the same distribution description given to the encoding algorithm. More formally, we write,

Definition 1. We say that a steganographic scheme $\Sigma_{\mathcal{D}}$ is secure against *chosen hiddentext attacks* if for all ppt. adversaries $\mathcal{A}_{\mathcal{D}}$,

$k \leftarrow \mathsf{KeyGen}_{\mathcal{D}}(1^{\lambda})$,

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

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

3.2 Ranged Randomness Recoverable Sampling Scheme

To construct Meteor, we will need a very specific property that many machine learning algorithms, like generative neural networks, possess: namely, that the random coins used to sample from the distribution can be recovered with access to a description of the distribution. If it is possible to *uniquely* recover these random coins, steganography is trivial: sample covertext elements using a pseudorandom ciphertext as sampling randomness and recover this ciphertext during decoding. However, generative machine learning models do not achieve unique randomness recovery.

Meteor requires a sampling algorithm with a randomness recovery algorithm that extracts the *set* of all random values that would yield the sample. Because this set could possibly be exponentially large, we requiring that the set be made up of polynomial number[1] of continuous intervals, *i.e.* it has a polynomial space representation that can be efficiently tested for membership. We call schemes that have this property *Ranged Randomness Recoverable Sampling Schemes,* or RRRSS. The formal interface for RRRSS schemes is parameterized by an underlying distribution $\mathcal{D}$, from which samples are to be drawn and has two ppt. algorithms. Additionaly, we make the size of length of the randomness explicit by requiring all random values to be selected from $\{0,1\}^{\beta}$. The two algorithms are defined below:

- $\mathsf{Sample}_{\mathcal{D}}^{\beta}(\mathcal{H}, r) \to s$. On history $\mathcal{H}$ and randomness $r \in \{0,1\}^{\beta}$, sample an output s from its underlying distribution $\mathcal{D}$
- $\mathsf{Recover}_{\mathcal{D}}^{\beta}(\mathcal{H}, s) \to \mathcal{R}$. On history $\mathcal{H}$ and sample s, output the set $\mathcal{R} = \{r \in \{0,1\}^{\beta} | \mathsf{Sample}_{\mathcal{D}}(\mathcal{H}, r) = s\}$.

Note that our sampling scheme takes in a history, making it somewhat stateful. This allows for conditioning sampling on priors, a key property we require to ensure that Meteor is sufficiently flexible to adapt to new covertext distributions. For example, consider character-by-character text generation: the probability of the next character being "x" is significantly altered if the prior character was a "e" or a "t."

The notion of *randomness recovery* has been widely studied in cryptography, primarily when building $\mathsf{IND-CCA2}$ secure public-key cryptography, *e.g.* [97, 98]. These works define notions like *unique randomness recovery* and *randomness recovery*, in which the recover algorithm run on some s returns a single value r such that $f(k,r) = s$ for an appropriate function f and key k. Unlike the definitions in prior work, we require a sample scheme over a some distribution and the extraction of intervals.

4 ADAPTING CLASSICAL STEGANOGRAPHIC SCHEMES

Characterizing Real Distributions. In this section, we focus on adapting classical steganographic techniques to English language distributions using generative models, specifically the GPT-2 [51] language model. As noted in Section 2, existing steganographic schemes require a certain, minimum amount of entropy for each sampling event. Any positive value, no matter how small, is sufficient for a channel to be "always informative," *i.e.*, theoretically permit the generation of stegotext. In practice, as we will see, an always informative channel with trivial entropy will yield extraordinarily long stegotext, a problem in practice.

Practical covertext channels, on the other hand, may not be always-informative, let alone have non-trivial entropy. Figure 2a depicts several representative runs of the entropy over time for a sample of tokens from the GPT-2 model. Each data point reflects the amount of entropy in the model after sampling x characters from the model. The entropy varies wildly between sampling events, and there is no clear consistency state of entropy over several tokens. Moreover, the entropy occasionally drops close to zero. As such, existing steganographic techniques will fail; in our testing, Algorithm 1 from [54] has a 100% failure rate when encoding a 16-byte message using GPT-2.

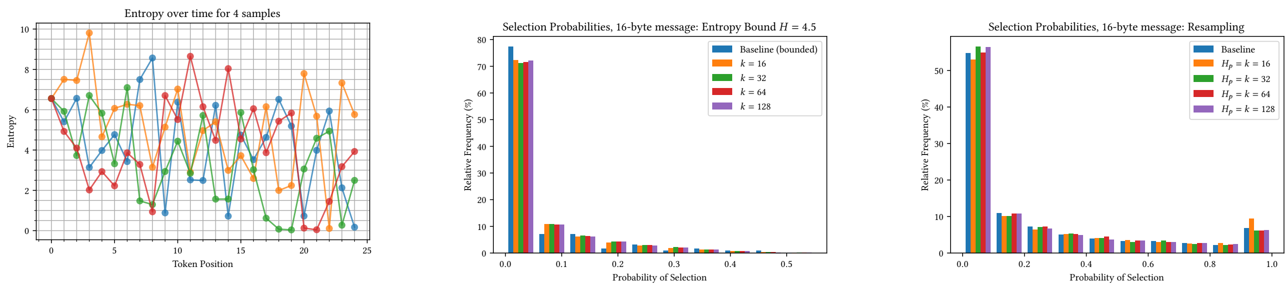
Adaptation 1: Entropy Bounding. A natural adaptation to periods of low entropy would be to not attempt to encode information while the entropy in the channel is too low. Both the sender and receiver have access to the distribution, meaning they can both detect periods of low entropy and skip them. This means that only "high-entropy" events are utilized for sampling, fixing a minimum entropy that is used in the steganographic protocol. In effect, this entropy bounding creates a sort of channel-within-a-channel that meets the always entropy requirement.

While this does increase the success rate (this method achieved 0–10% failure rate in our tests), it also introduces a new problem: significant bias in the sampled tokens. Figure 2b is a histogram showing the probability that the selected token from the distribution would be sampled (*i.e.* the probability weight of the selected tokens). In the figure, entropy bounding for different numbers of tries are shown (k), along with a baseline sample. The baseline is also "bounded" here: it represents the probabilities of normally-sampled tokens when the distribution entropy was above $H = 4.5$.

As the figure depicts, the entropy bounding method introduces significant bias by including a disproportionate number of tokens in certain bins. This is because the hash function used is not unbiased, so repeated rejection sampling from the same distribution exacerbates the bias of the hash function. In short, there is still not *enough* entropy to hide the bias introduced by the hash function. Thus, an adversary can distinguish between an encoded message and an innocuous one by seeing if the selection probabilities of the messages are different.

Adaptation 2: Variable Length Samples. An alternative method to handle low-entropy periods, as proposed by [54], is to compile the channel into one with sufficient entropy. If a channel is always

[1] In practice, we will be working with schemes for which there is a single set, continuous set of random values that result in the same output.



(a) Entropy of GPT-2 output distributions. Each datapoint computed as Shannon entropy of the output distribution after sampling a certain number of tokens. Then, a random token is sampled from that distribution and appended to the context. Different colors represent different runs starting with the same context and different randomness.

(b) Binned probability of selecting the tokens included in the final stegotext using entropy bounding with a value of 4.5 and the GPT-2 model. The stegotext tokens clearly come from a different distribution. Note that baseline tokens were only sampled from events above the entropy bound.

(c) Binned probability of selecting the tokens included in the final stegotext variable length sampling. Although there is slight variation in the distributions, there is no clear difference between the stegotext and the baseline. Moreover, this method is proved secure in [54].

Figure 2: Investigations into adapting classical steganographic techniques.

Table 1: Performance results for model load encoding using the method of [54] and resampling, averaged over 30 runs. The message being encoded is the first 16 bytes of Lorem Ipsum.

Parameters	Samples (Tokens)	Time (Sec)	Stegotext Len. (KiB)	Overhead (Length)
$H_p = k = 16$	502.8	42.69	2.3	149.4x
$H_p = k = 32$	880.4	128.41	4.1	261.8x
$H_p = k = 64$	1645.0	361.28	7.5	482.1x
$H_p = k = 128$	2994.6	765.40	13.6	870.7x

informative, meaning it always has some $\epsilon > 0$ entropy, this can be done by sampling some fixed number ℓ elements together, such that the resulting channel has at least $\ell \times \epsilon$ entropy. By setting ℓ appropriately, the entropy in the compiled channel is guaranteed to be high enough. However, in real communications channels, the entropy in the channel may not always be non-zero. As such, a naïve application of this approach will fall short.

We overcome this by sampling a *variable* number of tokens in each sampling event, such that the cumulative entropy of the distributions from which the tokens come surpasses the minimum requirement. More specifically, instead of sampling one token at a time in the `while` loop of Algorithm 1, this method samples p tokens until the sum of the entropy of the distributions from which those tokens were sampled meets a minimum threshold H_p. Intuitively, this approach "collects" entropy before attempting to encode into it, boosting success rate while avoiding the issues of low entropy.

Figure 2c shows a selection probabilities graph, with different values of H_p compared against a baseline measurement of normal sampling from the GPT-2 (note this baseline includes all sampled tokens, unlike in Figure 2b). In the figure, each set of runs of the model sets $\lambda = k$, i.e., the entropy required to encode is equivalent to the number of tries to encode. There are differences between the probabilities, but here is no clear pattern – this variation can be attributed to sampling error. [54] proved that for this approach to be secure, H_p must be strictly larger than $\log(k)$; to achieve useful security parameters, we need $H_p = k \approx 2\times\lambda$, where λ is the security parameter.

While provably secure, variable length sampling results in unreasonably large stegotext and long encoding times. Table 1 shows the length of stegotext and encoding times when encoding a 16 byte plaintext message using adaptation 2 on our Desktop/GPU test environment using the GPT-2 model (refer to Section 6 for hardware details). Each row corresponds to 30 runs of the model for that set of parameters. As H_p (and thereby k) increase, the length of the stegotext also increases: the higher resampling entropy requirement means that more tokens must be sampled, which takes more time. We note that these results include GPU acceleration, so there is little room for performance boosts from hardware.

5 METEOR: A MORE EFFICIENT SYMMETRIC-KEY STEGANOGRAPHIC SCHEME

We now design a symmetric-key steganographic scheme that is more practical than the techniques above. A more efficient symmetric-key approach would allow for hybrid steganography, in which a sender encodes a symmetric key using the public-key steganography and then switches to a faster and more efficient encoding scheme using this symmetric key. We note that while symmetric-key approaches have been considered in the past, *e.g.* [28, 68], they also rely on the entropy gathering techniques highlighted above. Our approach's intuition to accommodate high entropy variability is to fluidly change the encoding rate with the instantaneous entropy in the channel. As will become clear, Meteor does this *implicitly*, by having the *expected* number of bits encoded be proportional to the entropy.

5.1 Intuition

Suppose we have, for example, a generative model $\mathcal{M}$ trained to output English text word-by-word. Each iteration takes as input all previously generated words $\mathcal{H}$ and outputs a probability distribution $\mathcal{P}$ for the next word, defined over all known words $\mathcal{T}$. This is done by partitioning the probability space between 0 and 1 (represented at some fixed precision) into continuous intervals $r_0, r_1, \ldots, r_m$ corresponding to each valid word. For instance, if the precision is 5 bits, r_0 might be interval $[00000, 00101)$, r_1 might be $[00101, 10000)$, and so on. The algorithm then generates a uniform random value $r \in [00000, 11111]$, finds the interval r_i into which r falls, and outputs the corresponding word. In the example, if $r = 01110$, then the word corresponding to r_1 would be chosen. In practice, these values all have much higher precision, for example $r \in \{0,1\}^{32}, r_i \in \{0,1\}^{32} \times \{0,1\}^{32}$.

Meteor embeds messages into the random number r used to sample from the model, as illustrated in Figure 3. Consider the information that a potential receiver with access to the model might learn from a single output of the generative model. Because the receiver has access to $\mathcal{M}$, they can recover the interval r_i into which r must have fallen. Note that a r_i might contain a huge — possibly exponential — number of possible values that would all yield the same sample, meaning the receiver cannot uniquely recover the true value of r. However, because the intervals are *continuous*, all such values may share a prefix, effectively fixing the first few bits of r in the view of the receiver. In this example above, all values in r_1 are contained in the first half of the distribution space, so the receiver can conclude the first bit of r must have been a 0. Similarly, if the word corresponding to r_0 had been chosen, the first bits of r must have been 00. Another example can be seen in Figure 3, in which the interval corresponding to the word "The" shares the prefix 01, so a receiver can recover these bits. In this way, if r is a function of the hidden message, the receiver can potentially recover bits of information about the message with each output of the model. Because the sender and receiver share the description of the distribution, the sender can determine how many bits will be recoverable, and then discard those bits before repeating the process.

The key challenge in this setting is keeping the message hidden from the adversarial censor with access to the same distribution. Clearly, using the bits of the message as the randomness is insecure, as a censor with the same model could extract the message. Encrypting the message with a pseudorandom cipher, as in the public-key solution above, is also insufficient because it is possible that the encoder will be forced to *reuse randomness.* For example, consider a probability distribution in which the values of the interval containing r have no shared prefix, but 90% of the values in that interval begin with a 0. Because no bits are transmitted and the next iteration will use the same value of r. The censor now knows that with 90% likelihood, r in the second sampling event begins with zero. Over enough trials, a censor could detect this bias and distinguish between honestly sampled output and stegotext.

To avoid the reuse of randomness, Meteor generates a fresh mask for r each time the sender samples an output. This is done using a PRG, keyed with state shared by the sender and receiver, and applied using XOR. The receiver recovers as many bits of r as possible and then unmasks them with the corresponding XOR mask to recover bits of the message. Conceptually, this can be seen as repeatedly encrypting the message with a stream cipher, facilitating bit-by-bit decryption. This novel encoding technique means the number of bits that can be transmitted in each sampling event is not fixed. In practice, this is a huge advantage, as the expected number of bits transmitted is proportional to the entropy in the channel without requiring any explicit signaling (see Section 5.2). Finally, it is intuitively clear why this approach yields a secure scheme: (1) each sampling event is performed with a value of r that appears independent and random and (2) all bits that can be recovered are obscured with a one-time pad.

5.2 Meteor

For notation, let λ be a security parameter, ϵ be the empty string, and $\|$ represent concatenation or appending to an ordered set, as appropriate. We adopt Python-like array indexing, in which $x[a : b]$ includes the elements of x starting with a and ending with b, exclusive. Finally, we use two subroutines $\mathsf{LenPrefix}^{\beta}(\cdot)$ and $\mathsf{Prefix}^{\beta}(\cdot)$, presented in Algorithm 2 and Algorithm 3, respectively. The first gives the length of the longest shared bit prefix of elements in the set, and the second returns this bit prefix explicitly.

Pseudorandom Generators. Our construction leverages a pseudorandom generator PRG [99]. For a more formal treatment of the security notions of PRGs, see [100] and the citations contained therein. We adopt the notation used in stateful PRGs. Specifically, let the PRG have the functionalities $\mathsf{PRG.Setup}$ and $\mathsf{PRG.Next}$. The setup algorithm generates the secret state material, which we will denote k_{prg} for simplicity, and the next algorithm generates β pseudorandom bits. We require that the PRG satisfy at least the real-or-random security games.

Construction. Meteor consists of three algorithms, parameterized by a bit precision β and a model $\mathcal{M}$ that supports a RRRSS. We use a generative model $\mathcal{M}$ as our instantiation of the distribution $\mathcal{D}$ for an RRRSS as defined in Section 3. The key generation algorithm $\mathsf{KeyGen}^{\beta}_{\mathcal{M}}$ is presented in Algorithm 4, the encoding algorithm $\mathsf{Encode}^{\beta}_{\mathcal{M}}$ is presented in Algorithm 5, and the decoding algorithm $\mathsf{Decode}^{\beta}_{\mathcal{M}}$ is presented in Algorithm 6.

The precision $\beta \in \mathbb{Z}, \beta > 0$ controls the maximum number of bits that can be encoded in each iteration. β should be the accuracy of the underlying sampling scheme. Most models in our implementation give probability distributions accurate to 32 bits, so we set $\beta = 32$. In our tests, it is incredibly unlikely that 32 bits will successfully be encoded at once, meaning using a lower β is likely acceptable.

Because the model used in sampling is a generative one, the model maintains state on its previous inputs. Each distribution generated by the model is dependent on the values sampled from previous distributions. Additionally, the model requires an initial state to begin the generative process. This state is abstracted by the history parameter $\mathcal{H}$ passed to instances of Encode and Decode. This allows the distributions generated by each successful sampling of a covertext token c_i to remain synchronized between the two parties. We assume that the entire history $\mathcal{H}$ is maintained between the parties, including the initial state that primes the model.

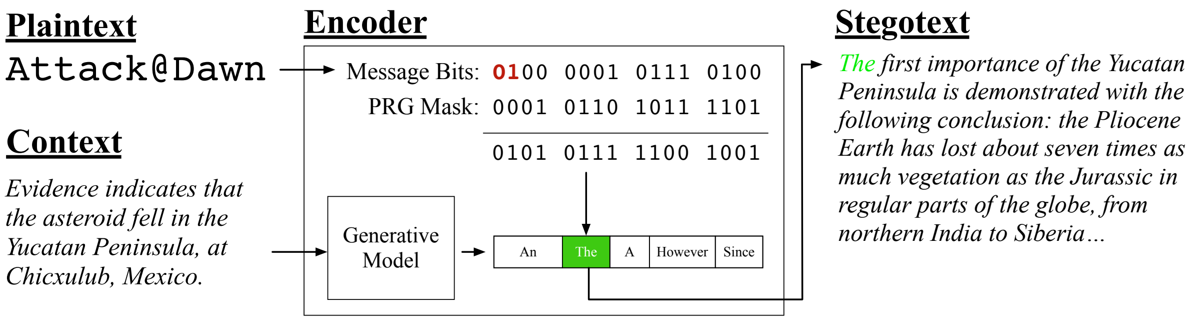


Figure 3: An overview of the encoding strategy for Meteor. In each iteration of Meteor, a new token (shown in green) is selected from the probability distribution created by the generative model. Depending on the token selected, a few bits (shown in red) can be recovered by the receiver. The stegotext above is real output from the GPT-2 model.

The encoding algorithm loops through three stages until the entire message has been successfully encoded: (1) generating and applying the mask, (2) sampling a next output to append to the covertext, and (3) updating the state of the algorithm based on the output of the sampling event. In the first stage, the mask is computed as the output of a pseudorandom generator and is applied with the XOR operation. The resulting value, r is distributed uniformly in $[0, 2^{\beta+1})$, as each bit of r is distributed uniformly in $\{0, 1\}$. This random value is then used in step (2) to sample the next output of the sampling scheme. To determine the number of bits this sampling event has successfully encoded, the encoding algorithm uses the $\mathsf{Recover}^\beta$ functionality of the RRRSS and calls $\mathsf{LenPrefix}$ on the resulting (multi-)set. Finally, the algorithm then updates the β bits that will be used in the next iteration, and updates its other state as appropriate.

The decoding algorithm performs these same three stages, but with the order of the first two reversed. With knowledge of the output of each sampling stage c_i, the first algorithm calls $\mathsf{Recover}^\beta$ and Prefix to recompute some (possibly zero) leading bits of the r. Then, it calculates the mask that was used by the encoder for those bits and removes the mask. The bits recovered in this way make up the message.

Note that we do not discuss reseeding the PRG. Most PRGs have a maximum number of bits that can be extracted before they are no longer considered secure. Because the PRG secret information is shared by the sender and receiver, they can perform a rekeying or key ratcheting function as necessary.

Proof of Security. We sketch the proof of security, as the formalities of this simple reduction are clear from the sketch. Consider an adversary $\mathcal{A}$ which has non-negligible advantage in the security game considered in Definition 1. We construct an adversary $\hat{\mathcal{A}}$ with non-negligible advantage in the PRG real-or-random game, with oracle denoted $R(\cdot)$. To properly answer queries from $\mathcal{A}$, $\hat{\mathcal{A}}$ runs the encoding algorithm in Algorithm 2 with an arbitrary input message, but queries the $R(\cdot)$ to obtain the mask required for sampling. Additionally, $\hat{\mathcal{A}}$ keeps a table of all queries sent by $\mathcal{A}$ and the responses. When $\mathcal{A}$ queries the decoding algorithm, $\hat{\mathcal{A}}$ checks its table to see if the query matches a previous encoding query, and responds only if it is an entry in the table. Note that if $R(\cdot)$ implements a true random function, the encoding algorithm simply samples a random message from the distribution. When $\mathcal{A}$ terminates, outputting a bit b, $\hat{\mathcal{A}}$ outputs b as well.

As the message is masked by the queries $\hat{\mathcal{A}}$ sends to $R(\cdot)$, $\mathcal{A}$ must be able to distinguish between a true-random output and the xor of a message with a one-time pad. Because XOR preserves the uniformly-random distribution of the pad, this is not possible with non-negligible probability.

Efficiency. The asymptotic, expected throughout of Meteor is proportional to the entropy in the communication channel. To see this, note that the expected throughput for each sampling event can be computed as $\sum_{i \in |\mathcal{P}|} p_i \mathsf{Exp}(p_i)$, where $\mathcal{P}$ is the distribution in the channel for the sampling event, p_i is the probability of each individual outcome, and $\mathsf{Exp}(\cdot)$ is the expected number of shared prefix bits for some continuous interval of size p_i. Thus, if $\mathsf{Exp}(p_i)$ is proportional to $-\log_2(p_i)$, Meteor is asymptotically optimal (recall that entropy, the information-theoretic boundary for information transmission, is computed as $-\sum_{i \in |\mathcal{P}|} p_i \log(p_i)$). We show in Appendix A that $\mathsf{Exp}(p_i) \geq \frac{1}{2}(-\log_2(p_i) - 1)$ for $p_i \leq \frac{1}{2}$ by carefully observing the behavior of the $\mathsf{LenPrefix}$ function when evaluated on a fixed sized interval with a random starting point between $[0, 2^{\beta+1})$.

6 EVALUATION OF METEOR

In this section we discuss our implementation of Meteor and evaluate its efficiency using multiple models. We focus on evaluating Meteor, not a hybrid steganography system using the public key stegosystem in Section 4, because it is significantly more efficient. Moreover, the efficiency of a hybrid stegoanography system is determined by the efficiency of its constituent parts; the cost of such a scheme is simply the cost of transmitting a key with the public key scheme (see Section 4) plus the cost of transmitting the message with Meteor. An interactive online demonstration of our system is available at https://meteorfrom.space.

Implementation details. We implemented Meteor using the PyTorch deep learning framework [101]. We realize the PRG functionality with `HMAC_DRBG`, a deterministic random bit generator defined

Algorithm 2: $\mathsf{LenPrefix}^{\beta}$

Input: Set of Bit Strings $\mathcal{R} = \{r_1, r_2, \ldots r_n\}$
Output: Length ℓ
$\ell \leftarrow 1$
while $\ell < \beta$ **do**
  if $\exists\, i, j \in \{1, \ldots, n\}$ *such that* $r_i[0:\ell] \neq r_j[0:\ell]$ **then**
    Output $\ell - 1$
  $\ell \leftarrow \ell + 1$
Output ℓ

Algorithm 3: Prefix^{β}

Input: Set of Bit Strings $\mathcal{R} = \{r_1, r_2, \ldots r_n\}$
Output: Bit String s
Output $r_1[0 : \mathsf{LenPrefix}^{\beta}(\mathcal{R})]$

Algorithm 4: $\mathsf{KeyGen}^{\beta}_{\mathcal{M}}$

Input: 1^{λ}
Output: Key k_{prg}
Output $k_{prg} \leftarrow \mathsf{PRG.Setup}(1^{\lambda})$

Algorithm 5: $\mathsf{Encode}^{\beta}_{\mathcal{M}}$

Input: Key k_{prg}, Plaintext Message m, History $\mathcal{H}$
Output: Stegotext Message c
$c \leftarrow \varepsilon, n \leftarrow 0$
while $n < |m|$ **do**
  $mask \leftarrow \mathsf{PRG.Next}(k_{prg})$
  $r \leftarrow m[n : n + \beta] \oplus mask$
  $c_i \leftarrow \mathsf{Sample}^{\beta}_{\mathcal{M}}(\mathcal{H}, r)$
  $\mathcal{R} \leftarrow \mathsf{Recover}^{\beta}_{\mathcal{M}}(\mathcal{H}, c_i)$
  $n_i \leftarrow \mathsf{LenPrefix}^{\beta}(\mathcal{R})$
  $c \leftarrow c \| c_i, n \leftarrow n + n_i,\ \mathcal{H} \leftarrow \mathcal{H} \| c_i$
Output c

Algorithm 6: $\mathsf{Decode}^{\beta}_{\mathcal{M}}$

Input: Key k_{prg}, Stegotext Message c, History $\mathcal{H}$
Output: Plaintext Message m
$x \leftarrow \varepsilon$
Parse c as $\{c_0, c_1, \ldots, c_{|c|-1}\}$
for $i \in \{0, 1, \ldots, |c| - 1\}$ **do**
  $\mathcal{R} \leftarrow \mathsf{Recover}^{\beta}_{\mathcal{M}}(\mathcal{H}, c_i)$
  $x_i \leftarrow \mathsf{Prefix}^{\beta}(\mathcal{R})$
  $mask \leftarrow \mathsf{PRG.Next}(k_{prg})$
  $x \leftarrow x \| (x_i \oplus mask[0 : |x_i|])$
  $\mathcal{H} \leftarrow \mathcal{H} \| c_i$
Output x

Figure 4: Algorithms for Meteor

in NIST SP 800-90 A Rev. 1 [102]. The implementation supports any type of binary data, such as UTF-8-encoded strings or image files, as input.

Algorithm 7: $\mathsf{Sample}^{\beta}_{\mathcal{M}}$ for the GPT-2 model.

Input: Randomness $r \in \{0,1\}^{\beta}$, History $\mathcal{H}$
Output: Token *next*
$\mathcal{T}, \mathcal{P} \leftarrow \mathsf{Next}_{\mathcal{M}}(\mathcal{H})$
$cuml \leftarrow 0$
for $i \in \{0, 1, \ldots, |\mathcal{T}| - 1\}$ **do**
  $cuml \leftarrow cuml + \mathcal{P}[i]$
  if $cuml > r$ **then**
    Output $next \leftarrow \mathcal{T}[i]$
Output $next \leftarrow \mathcal{T}[|\mathcal{T}| - 1]$

Algorithm 8: $\mathsf{Recover}^{\beta}_{\mathcal{M}}$ for the GPT-2 model.

Input: History $\mathcal{H}$, Sample s
Output: Randomness set $\mathcal{R}$
$\mathcal{T}, \mathcal{P} \leftarrow \mathsf{Next}_{\mathcal{M}}(\mathcal{H})$
$cuml \leftarrow 0$
for $i \in \{0, 1, \ldots, |\mathcal{T}| - 1\}$ **do**
  if $\mathcal{T}[i] = s$ **then**
    Output
    $\mathcal{R} \leftarrow \{r \in \{0,1\}^{\beta} \mid cuml \leq r < cuml + \mathcal{P}[i]\}$
  $cuml \leftarrow cuml + \mathcal{P}[i]$
Output $\mathcal{R} \leftarrow \emptyset$

Figure 5: RRRSS algorithms for GPT-2 model. $\mathcal{T}$ is an array of possible next tokens and $\mathcal{P}$ is the probability associated with each of these tokens.

To illustrate Meteor's support for different model types, we implemented the algorithm with the weakened version of the GPT-2 language model released by OpenAI and two character-level recurrent neural networks (RNN) that we train. The GPT-2 model [51] is a generative model of the English language. It parses language into a vocabulary of words and generates words when given previous context. Meteor encodes stegotext into these generated words. The character-level models generate ASCII characters in each iteration. These models output lower-quality English text, but are more generalizable. Character-level models work with any data that can be represented as text, including other languages and non-text protocols, whereas word-level models are specific to the English language models.

Our GPT-2 codebase builds upon that of [44]. We note that the next-generation GPT language model, GPT-3, has been published by OpenAI [52]; however, at the time of this writing, the codebase for the GPT-3 has not been released. The GPT-3 interface is the same as the GPT-2, meaning integration will be automatic, increasing stegotext quality while maintaining security guarantees. Example stegotext generated with the GPT-2 model can be found in Appendix C.

Figure 5 shows how to instantiate the $\mathsf{Sample}^{\beta}_{\mathcal{M}}$ and $\mathsf{Recover}^{\beta}_{\mathcal{M}}$ algorithms from Section 3 with the distribution represented as a generative model $\mathcal{M}$ (in discussion of classical steganography, we used $\mathcal{D}$). Both algorithms use $\mathsf{Next}_{\mathcal{M}}(\mathcal{H})$, which generates an array of possible next tokens $\mathcal{T}$ and an array of probabilities associated with each token $\mathcal{P}$ using the model's internal structure. The $\mathsf{Sample}^{\beta}_{\mathcal{M}}$

Table 2: Performance measurements for Meteor on the GPT-2 by device for a shorter context. Times are provided in seconds.

Device	Load	Encode	Decode	Overhead (time)
GPU	5.867	6.899	6.095	1×
CPU	5.234	41.221	40.334	4.6×
Mobile	1.830	473.58	457.57	49.5×

for generative networks accumulates the probabilities and selects the first token for which the cumulative probability exceeds the randomness supplied. This is equivalent to multinomial sampling, and is the unmodified method of sampling normally from the GPT-2 model. In the unmodified (i.e., non-Meteor) case, the GPT-2 chooses a true random value r instead of a PRG as in Meteor. $\text{Recover}^{\beta}_{\mathcal{M}}$ inverts the process, returning the entire set of random values that would yield the target sample s.

In addition to the GPT-2 variant, we trained two character-level RNN models to test with Meteor, using the code of [103] with locally trained models. Each model uses long short term memory (LSTM) cells to store state [89]. The first model, named "Wikipedia", was trained on the Hutter Prize dataset [104], which consists of a subset of English Wikipedia articles. The data from this model contains English text structured with Wiki markup. The output of this model is good, but its character-level nature makes its outputs less convincing human text than GPT-2 output. The second model, named "HTTP Headers", consist of the headers for 530,128 HTTP GET requests from a 2014 ZMap scan of the internet IPv4 space [105, 106]. This highly structured dataset would facilitate hiding messages amongst other HTTP requests. We note that the flexibility of character-level models allows us to generalize both text-like channels and protocol-esque channels [55]. Both models have three hidden layers. The Wikipedia model has a hidden layer size 795 and was trained for 25,000 epochs. The HTTP headers model has size 512 and was for 5,000 epochs, due to its more structured nature. The two models were trained at a batch size of 100 characters and learning rate 0.001. Example output from the Wikipedia character-level model can be found in Appendix C.

Evaluation hardware. To measure performance across different hardware types, we evaluate Meteor on 3 systems: (1) *Desktop/GPU*, a Linux workstation with an Intel Core i7-6700 CPU, NVIDIA TITAN X GPU, and 8 GiB of RAM, (2) *Laptop/CPU*, a Linux laptop with an Intel Core i7-4700MQ CPU, no discrete GPU, and 8 GiB of RAM, and (3) *Mobile*, an iPhone X running iOS 13. The Desktop ran benchmarks on the GPU, while the Laptop machine ran on the CPU; as such, the Laptop is more representative of consumer hardware. We evaluate Meteor on both the Desktop and Laptop using each of the three models discussed above. Additionally, we evaluate reordering and native compression optimizations (see below). The results are summarized in Table 3. We discuss mobile benchmarks separately at the end of this section.

Model performance. The capacity, or number of bits encoded per token, is much higher for the GPT-2 model examples than for the Wikipedia and HTTP Headers models. Intuitively, the word-level nature of GPT-2 means there is usually more entropy in each

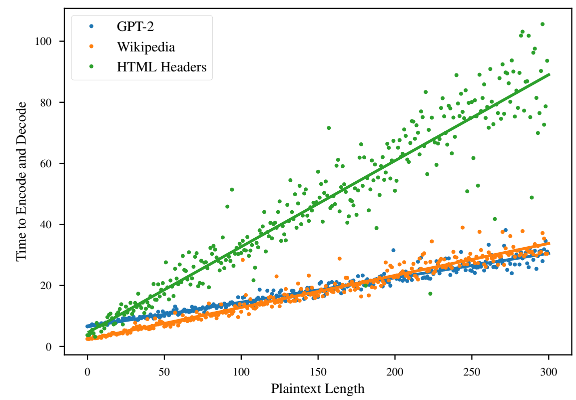


Figure 6: Comparison of plaintext length versus time to run encoding and decoding for different Meteor models. R = **0.9745 (GPT-2), 0.9709 (Wikipedia), 0.9502 (HTTP Headers)**

distribution, whereas the character-level models have, at most, 100 printable ASCII characters from which to sample; this pushes the capacity of a single token to be much higher as a result. The stark difference in capacity between the capacities of Wikipedia and HTTP Headers can be attributed to the difference in structure of the training data. The Wikipedia dataset, although structured, is mostly English text. On the other hand, the HTTP Headers dataset is based on the HTTP protocol, which is rigid in structure — variation only exists in fields that can change, such as dates and URLs.

Encoding statistics. Our next suite of benchmarks measures the relationship between the length of message and the time it takes to produce a stegotext. We generated plaintexts randomly and encoded them, incrementing the length of the message by one in each run. The results are plotted in Figure 6, which shows a clear linear relationship between the two variables. It is also apparent from the plot that the variance in encoding time increases as the length increases. This is because as tokens are selected, the model state can diverge; in some of these branches, the entropy may be very low, causing longer encoding times. This is amplified in the HTTP Headers model, as the baseline entropy is already very low.

Heuristic optimizations. In addition to implementing Meteor, we also evaluated two heuristic optimizations that could yield shorter stegotext. The first optimization is deterministically reordering the model's output distribution intervals to maximize expected throughput. Because this deterministic process does not change the relative sizes of the interval, it does not impact the distribution of the stegotext. However, because the placement of the intervals is usually arbitrary, it is possible to move large intervals that would normally have no shared prefix to a starting location where there is a shared prefix, potentially increasing throughput. A more thorough discussion of this technique can be found in Appendix B.

We evaluate this optimization for all three of our models (see Table 3). For the GPT-2 model, we see a marked (24.8%) increase in capacity as well as a proportional reduction in stegotext length as a result of reordering the model outputs. The reordering does induce computational overhead, as the distribution over which the heuristic is performed is large (max 50,256 tokens). Reordering induces a 0.5% overhead in the Laptop/CPU, where updating the model is slow, and 69.0% overhead in the Desktop/GPU, where updating the model is fast. For the lower entropy models, the reordering algorithm we use is significantly faster, but yields mixed results. We believe these mixed results are an artifact of our choice of greedy reordering algorithms, which may perform poorly with heavily biased distributions.

The second optimization is to use the model itself as a compression function when encoding with an English language model, as in [44]. This technique leverages the fact that all known words in the model's vocabulary are internally represented by a unique number, taking fewer bits than its normal ASCII representation. Before encoding, the secret message can be tokenized and each token can be replaced by its unique identifier. These identifiers are then parsed as bits and encoded as normal. When implemented with GPT-2, we see a 47.77% decrease in time spent on CPU, and an associated 52.5% decrease in stegotext size. While powerful, this technique can only be used to encode English language messages into English language models. Compressing the plaintext message using traditional compression (*e.g.*, GZip) would yield similar results.

Mobile benchmarks. Because Meteor is intended for censorship resistance, it is natural to benchmark it on mobile devices, where most sensitive communication happens. We implement Meteor on iOS using the CoreML framework, utilizing an existing GPT-2 iOS implementation as a base [107]. To our knowledge, our work represents the first evaluation of a neural network-based steganographic system on a mobile device. Our implementation, in Swift, employs an even smaller version of the GPT-2 model which fits on mobile devices as it uses smaller size context. An example of the output from this experiment can be found in Appendix C.

Our results are summarized in Table 2. The Mobile benchmark in the table was performed on the iPhone X Simulator, as we wished to instrument and profile our tests. We separately confirmed that simulator runtimes were similar to those of actual iPhone X hardware. While Laptop/CPU is 4.6× slower than Desktop/GPU, the Mobile runtime is a massive 49.5× slower than the baseline case. While deep learning is supported on mobile platforms like iOS, the intensive, iterative computations required by Meteor and other neural stegosystems are not performant on mobile systems. Nonetheless, our proof-of-concept demonstrates that Meteor could be used in a mobile context, and hardware improvements [108] would allow for secure communication between users even when available communication platforms do not offer end-to-end encryption, such as WeChat.

7 COMPARISON TO NLP-BASED STEGANOGRAPHY

Noting the appeal of hiding sensitive messages in natural text, researchers in the field of natural language processing (NLP) have recently initiated an independent study of steganography. Unfortunately, this work does not carefully address the security implications of developing steganographic systems from NLP models. Instead, the results employ a variety of ad-hoc techniques for embedding secret messages into the output of sophisticated models. The resulting papers, often published in top NLP conferences, lack rigorous security analyses; indeed, existing work cannot be proven secure under the definitions common in the cryptographic literature. Highlighting this weakness, there is a concurrent line of work in the same conferences showing concrete attacks on these schemes, *e.g.*, [45–50].

The first wave of steganographic techniques in the NLP community leverages synonyms and grammatical reorganization for encoding, *e.g.*, [32–36, 42]. The key observation in this work is that natural variation in linguistic patterns can be used to hide information. For instance, if one of two synonyms can be used in a sentence, each with probability .5, then the selection conveys a bit of information. Similarly, comma usage or word order can be used to encode small amounts of information. Because not all possible linguistic variations occur with equal likelihood, some of these works adapt a Huffman encoding scheme to facilitate variable length encoding, *e.g.*, [32, 36]. These approaches rely on linguistic idiosyncrasies and are therefore not generalizable.

More recently, researchers found ways to use the structure of these models to steganographically encode information, including LSTMs [37], Generative Adversarial Networks [38], Markov Models [39], and other forms of Deep Neural Networks [40, 41, 43, 44]. Rather than give an exhaustive description of the encoding techniques used in these works, we give a brief description of the most important techniques.

Early constructions directly modified the distributions. One such construction [37] organized the distribution into "bins," each representing a short bitstring, and randomly selected an output from the bins corresponding to the message.[2] Building on this intuition, other research [41, 43] uses Huffman coding to encode variable numbers of bit in each iteration. More recent work has attempted to use the message itself as the sampling method, a method known as "arithmetic coding" [44]. This method attempts to convert a plaintext message into a deterministic stegotext based on its contents, iteratively using bits from the message to sample into the distribution. The first two constructions heavily modify the output distribution, rendering stegotext easily detectable. The arithmetic construction is also insecure, since it reuses randomness in multiple sampling events, a problem similar to the one that Meteor is designed to overcome.

The relaxed adversarial models considered in the NLP community lead to significantly less robust constructions. For instance, the adversaries in the NLP literature do not have access to the model [37, 41, 43, 44], significantly limiting the attacks they can mount. Without this assumption, an adversary can clearly differentiate between a stegotext and covertext by identifying biases in the output distribution. The adversary compares the candidate output to random samples from the model, easily distinguishing when a stegosystem is being run and defeating the purpose entirely.

The NLP threat model folds in the face of an advanced, persistent adversary who can always exfiltrate the model through other means. Moreover, recent advanced in adversarial machine learning have demonstrated how even the "secret" parameters of a black-box model can be extracted by seeing enough output [109–111], unlike that of encryption keys or pseudorandom functions. This pervasive requirement that the model remains private information is therefore unreasonable. Unable to achieve cryptographic

[2] A similar, but secure, partition based approach is investigated in [27]

Table 3: Model statistics for encoding a 160-byte plaintext. Timing results reflect model load, encoding, and decoding combined.

Mode	Desktop/GPU (sec)	Laptop/CPU (sec)	Stegotext Length (bytes)	Overhead (length)	Capacity (bits/token)
GPT-2	18.089	82.214	1976	12.36×	3.09
GPT-2 (Reorder)	30.570	82.638	1391	8.69×	4.11
GPT-2 (Compress)	11.070	42.942	938	3.39×	3.39
Wikipedia	19.791	46.583	2002	12.51×	0.64
Wikipedia (Reorder)	15.515	39.450	1547	9.67×	0.83
HTTP Headers	49.380	103.280	6144	38.4×	0.21
HTTP Headers (Reorder)	57.864	127.759	7237	45.23×	0.18

Table 4: Comparative distribution statistics for samples from neural steganography algorithms in prior NLP work, with random sampling as a baseline. "N/A" indicates that a metric is not relevant for an algorithm.

Algorithm	Perplexity	KL-Divergence	Capacity	Entropy	Secure?
Meteor (this)	21.60	0.045	3.09	6.30	✓
Arithmetic [44]	29.22	0.082	4.82	6.66	✗
Huffman [41, 43]	8.85	0.851	2.31	N/A	✗
Bins [37]	50.82	2.594	3.00	N/A	✗
Random Sample	13.82	0.040	N/A	5.36	N/A

security, these constructions evaluate their work by measuring the statistical difference between the output produced by the encoding scheme and real text. Highlighting the weaknesses of these schemes, numerous attack papers have been published, *e.g.*, [45–50]. These attacks use machine learning techniques to detect the presence of encoded messages generated with some of the works listed previously. Ad-hoc and non-cryptographic security is insufficient to provide security against powerful and determined adversaries, especially nation-state adversaries.

Comparative Analysis. We assess Meteor against the following previous solutions: (1) bins [37], (2) Huffman coding [41], and (3) arithmetic coding [44]. We compare standard NLP language statistics for these with a regular, random sample from the model, and provide our results in Table 4. Note that we mark entropy as "N/A" for Huffman and bins because these methods use a binning algorithm which prevents us from calculating entropy meaningfully. The random sample is a control distribution, and is not encoding anything thereby having "N/A" capacity.

Of particular note in our results is the Kullback-Leibler (KL) divergence across algorithms, which in this case compares the distribution of the model to the output distribution of the algorithm. The KL-divergence for Meteor is very close to that of the random sample, as Meteor merely changes the randomness to steganographically-encoded randomness. As discussed previously, algorithms that modify distributions from the model have high biases, and this is reflected in the KL-divergence of Huffman and bins being much higher than the rest. The arithmetic algorithm has a lower KL-divergence than the rest of the NLP algorithms, as it does not modify the distribution; however, it has a higher value than Meteor because it reuses randomness, while Meteor uses fresh randomness like the baseline random sample does.

We also note that the security properties of Meteor do not hamper the capacity metric significantly. Arithmetic output has a higher capacity, but we note that the insecurity of this system makes this additional capacity moot; modifying the parameters to Huffman or bins could have yielded the same capacity with the same security vulnerabilities. Table 4 also includes perplexity and entropy statistics, that show Meteor is competitive in performance with the insecure primitives proposed previously.

8 CONCLUSION

In this work we present an analysis of the practical limitations of using cryptographically secure steganography on real, useful distributions, identifying the need for samplers and impractical entropy requirements as key impediments. We show that adapting existing public key techniques is possible, but produces stegotext that are extremely inefficient. We then present Meteor, a novel symmetric key steganographic system that dramatically outperforms public key techniques by fluidly adapting to changes in entropy. We evaluate Meteor, implementing it on GPU, CPU, and mobile, showing that it is an important first step for universal, censorship-resistant steganography. Finally, we compare Meteor to existing insecure steganographic techniques from the NLP literature, showing it has comparable performance while actually achieving cryptographic security.

ACKNOWLEDGEMENTS

The first author is supported by the National Science Foundation under Grant #2030859 to the Computing Research Association for the CIFellows Project and DARPA under Agreement No. HR00112020021. Part of this work was completed while the first author was at Johns Hopkins University. The second and fourth authors are funded by the NSF under awards CNS-1329737 and CNS-1955172. The third author is supported by NSF under awards CNS-1653110 and CNS-1801479, the Office of Naval Research under contract N00014-19-1-2292, DARPA under Contract No. HR001120C0084, and a Security and Privacy research award from Google. Significant elements of this work were conducted while the first, second, and fourth authors were supported by NSF award CNS-1330491. The first author would like to thank Adam Poliak for his early help shaping this project. Additionally, the first author would like to thank Eric Wustrow for his insight into censorship resistance techniques. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do

not necessarily reflect the views of the United States Government or DARPA.

REFERENCES

[1] T. Perrin and M. Marlinspike, "he double ratchet algorithm." Available at https://whispersystems.org/docs/specifications/doubleratchet/.

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

[3] K. Conger, "Whatsapp blocked in brazil again." https://techcrunch.com/2016/07/19/whatsapp-blocked-in-brazil-again/, Jul 2016.

[4] T. Fish, "Whatsapp banned: Countries where whatsapp is blocked mapped." https://www.express.co.uk/life-style/science-technology/1166191/whatsapp-ban-map-which-countries-where-whatsapp-blocked-censorship-china-banned, Aug 2019.

[5] A. Shahbaz, "Freedom on the net 2018." Available at https://freedomhouse.org/report/freedom-net/freedom-net-2018/rise-digital-authoritarianism, 2018.

[6] Freedom House, "Freedom on the net 2018 map." https://freedomhouse.org/report/freedom-net/freedom-net-2018/map, 2018.

[7] R. S. Raman, A. Stoll, J. Dalek, R. Ramesh, W. Scott, and R. Ensafi, "Measuring the deployment of network censorship filters at global scale," in *Network and Distributed Systems Security (NDSS) Symposium 2020*, 2020.

[8] Tor Project, "The tor project: Privacy and freedom online." https://www.torproject.org/.

[9] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: The second-generation onion router," in *Proceedings of the 13th Conference on USENIX Security Symposium - Volume 13*, SSYM'04, (Berkeley, CA, USA), pp. 21–21, USENIX Association, 2004.

[10] M. G. Reed, P. F. Syverson, and D. M. Goldschlag, "Anonymous connections and onion routing," *IEEE Journal on Selected Areas in Communications*, vol. 16, pp. 482–494, May 1998.

[11] P. Winter, T. Pulls, and J. Fuß, "Scramblesuit: A polymorph network protocol to circumvent censorship," *CoRR*, vol. abs/1305.3199, 2013.

[12] H. M. Moghaddam, B. Li, M. Derakhshani, and I. Goldberg, "SkypeMorph: protocol obfuscation for Tor bridges," in *ACM CCS 2012* (T. Yu, G. Danezis, and V. D. Gligor, eds.), pp. 97–108, ACM Press, Oct. 2012.

[13] Z. Weinberg, J. Wang, V. Yegneswaran, L. Briesemeister, S. Cheung, F. Wang, and D. Boneh, "StegoTorus: a camouflage proxy for the Tor anonymity system," in *ACM CCS 2012* (T. Yu, G. Danezis, and V. D. Gligor, eds.), pp. 109–120, ACM Press, Oct. 2012.

[14] E. Wustrow, S. Wolchok, I. Goldberg, and J. A. Halderman, "Telex: Anticensorship in the network infrastructure," in *Proceedings of the 20th USENIX Security Symposium*, Aug. 2011.

[15] E. Wustrow, C. M. Swanson, and J. A. Halderman, "Tapdance: End-to-middle anticensorship without flow blocking," in *23rd {USENIX} Security Symposium ({USENIX} Security 14)*, pp. 159–174, 2014.

[16] K. P. Dyer, S. E. Coull, T. Ristenpart, and T. Shrimpton, "Protocol misidentification made easy with format-transforming encryption," in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pp. 61–72, ACM, 2013.

[17] Q. Wang, X. Gong, G. T. K. Nguyen, A. Houmansadr, and N. Borisov, "CensorSpoofer: asymmetric communication using IP spoofing for censorship-resistant web browsing," in *ACM CCS 2012* (T. Yu, G. Danezis, and V. D. Gligor, eds.), pp. 121–132, ACM Press, Oct. 2012.

[18] A. Houmansadr, T. J. Riedl, N. Borisov, and A. C. Singer, "I want my voice to be heard: IP over voice-over-IP for unobservable censorship circumvention," in *NDSS 2013*, The Internet Society, Feb. 2013.

[19] M. C. Tschantz, S. Afroz, Anonymous, and V. Paxson, "Sok: Towards grounding censorship circumvention in empiricism," in *2016 IEEE Symposium on Security and Privacy (SP)*, pp. 914–933, May 2016.

[20] R. Ensafi, P. Winter, A. Mueen, and J. R. Crandall, "Analyzing the great firewall of china over space and time," *PoPETs*, vol. 2015, pp. 61–76, Jan. 2015.

[21] B. Marczak, N. Weaver, J. Dalek, R. Ensafi, D. Fifield, S. McKune, A. Rey, J. Scott-Railton, R. Deibert, and V. Paxson, "An analysis of china's "great cannon"," in *5th {USENIX} Workshop on Free and Open Communications on the Internet ({FOCI} 15)*, 2015.

[22] R. Ensafi, D. Fifield, P. Winter, N. Feamster, N. Weaver, and V. Paxson, "Examining how the great firewall discovers hidden circumvention servers," in *Proceedings of the 2015 Internet Measurement Conference*, pp. 445–458, 2015.

[23] M. Bevand, "My experience with the great firewall of china." http://blog.zorinaq.com/my-experience-with-the-great-firewall-of-china/, Jan 2016.

[24] K. Bock, iyouport, Anonymous, L.-H. Merino, D. Fifield, A. Houmansadr, and D. Levin, "Exposing and circumventing china's censorship of esni," 8 2020.

[25] G. J. Simmons, "The prisoners' problem and the subliminal channel," in *CRYPTO'83* (D. Chaum, ed.), pp. 51–67, Plenum Press, New York, USA, 1983.

[26] R. J. Anderson and F. A. Petitcolas, "On the limits of steganography," *IEEE Journal on selected areas in communications*, vol. 16, no. 4, pp. 474–481, 1998.

[27] C. Cachin, "An information-theoretic model for steganography." Cryptology ePrint Archive, Report 2000/028, 2000. http://eprint.iacr.org/2000/028.

[28] N. J. Hopper, J. Langford, and L. von Ahn, "Provably secure steganography," in *CRYPTO 2002* (M. Yung, ed.), vol. 2442 of *LNCS*, pp. 77–92, Springer, Heidelberg, Aug. 2002.

[29] L. von Ahn and N. J. Hopper, "Public-key steganography," in *EUROCRYPT 2004* (C. Cachin and J. Camenisch, eds.), vol. 3027 of *LNCS*, pp. 323–341, Springer, Heidelberg, May 2004.

[30] M. Backes and C. Cachin, "Public-key steganography with active attacks," in *TCC 2005* (J. Kilian, ed.), vol. 3378 of *LNCS*, pp. 210–226, Springer, Heidelberg, Feb. 2005.

[31] N. Dedic, G. Itkis, L. Reyzin, and S. Russell, "Upper and lower bounds on black-box steganography," in *TCC 2005* (J. Kilian, ed.), vol. 3378 of *LNCS*, pp. 227–244, Springer, Heidelberg, Feb. 2005.

[32] C. Grothoff, K. Grothoff, L. Alkhutova, R. Stutsman, and M. Atallah, "Translation-based steganography," in *International Workshop on Information Hiding*, pp. 219–233, Springer, 2005.

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

[34] Z. Yu, L. Huang, Z. Chen, L. Li, X. Zhao, and Y. Zhu, "Steganalysis of synonym-substitution based natural language watermarking," 2009.

[35] C.-Y. Chang and S. Clark, "Linguistic steganography using automatically generated paraphrases," in *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, HLT '10, (Stroudsburg, PA, USA), pp. 591–599, Association for Computational Linguistics, 2010.

[36] C.-Y. Chang and S. Clark, "Practical linguistic steganography using contextual synonym substitution and a novel vertex coding method," *Computational Linguistics*, vol. 40, p. 403–448, Jun 2014.

[37] T. Fang, M. Jaggi, and K. Argyraki, "Generating steganographic text with lstms," *Proceedings of ACL 2017, Student Research Workshop*, 2017.

[38] D. Volkhonskiy, I. Nazarov, B. Borisenko, and E. Burnaev, "Steganographic generative adversarial networks," 2017.

[39] Z. Yang, S. Jin, Y. Huang, Y. Zhang, and H. Li, "Automatically generate steganographic text based on markov model and huffman coding," 2018.

[40] L. Xiang, "Reversible natural language watermarking using synonym substitution and arithmetic coding," 2018.

[41] Z. Yang, X. Guo, Z. Chen, Y. Huang, and Y. Zhang, "Rnn-stega: Linguistic steganography based on recurrent neural networks," *IEEE Transactions on Information Forensics and Security*, vol. 14, pp. 1280–1295, May 2019.

[42] S.-Y. HUANG and P.-S. Huang, "A homophone-based chinese text steganography scheme for chatting applications.," *Journal of Information Science & Engineering*, vol. 35, no. 4, 2019.

[43] F. Dai and Z. Cai, "Towards near-imperceptible steganographic text," *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 2019.

[44] Z. M. Ziegler, Y. Deng, and A. M. Rush, "Neural linguistic steganography," 2019.

[45] Z. Yang, Y. Huang, and Y.-J. Zhang, "A fast and efficient text steganalysis method," *IEEE Signal Processing Letters*, vol. 26, pp. 627–631, 2019.

[46] Z. Yang, K. Wang, J. Li, Y. Huang, and Y. Zhang, "Ts-rnn: Text steganalysis based on recurrent neural networks," *IEEE Signal Processing Letters*, p. 1–1, 2019.

[47] Z. Yang, N. Wei, J. Sheng, Y. Huang, and Y.-J. Zhang, "Ts-cnn: Text steganalysis from semantic space based on convolutional neural network," 2018.

[48] A. Wilson, P. Blunsom, and A. Ker, "Detection of steganographic techniques on twitter," *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, 2015.

[49] J. Kodovsky, J. Fridrich, and V. Holub, "Ensemble classifiers for steganalysis of digital media," *IEEE Transactions on Information Forensics and Security*, vol. 7, pp. 432–444, April 2012.

[50] P. Meng, L. Huang, Z. Chen, W. Yang, and D. Li, "Linguistic steganography detection based on perplexity," in *2008 International Conference on MultiMedia and Information Technology*, pp. 217–220, Dec 2008.

[51] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," *OpenAI Blog*, vol. 1, no. 8, 2019.

[52] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language models are few-shot learners," 2020.

[53] O. Blog, "Better language models and their implications." Available at https://openai.com/blog/better-language-models/, February 2019.

[54] N. J. Hopper, "Toward a theory of steganography," tech. rep., CARNEGIE-MELLON UNIV PITTSBURGH PA SCHOOL OF COMPUTER SCIENCE, 2004.

[55] A. Karpathy, “The unreasonable effectiveness of recurrent neural networks.” https://karpathy.github.io/2015/05/21/rnn-effectiveness/, May 2015.
[56] A. van Dalen, “The algorithms behind the headlines,” *Journalism Practice*, vol. 6, no. 5-6, pp. 648–658, 2012.
[57] A. Graefe, “Guide to automated journalism,” 2016.
[58] D. Rockmore, “What happens when machines learn to write poetry,” Jan 2020.
[59] A. Mayne, “Ai|writer.” https://www.aiwriter.app/.
[60] A. Kind, “Talk to transformer.” https://app.inferkit.com/demo.
[61] “Ai writer.” http://ai-writer.com/.
[62] J. Y. Koh. https://modelzoo.co/.
[63] J. Zöllner, H. Federrath, H. Klimant, A. Pfitzmann, R. Piotraschke, A. Westfeld, G. Wicke, and G. Wolf, “Modeling the security of steganographic systems,” in *International Workshop on Information Hiding*, pp. 344–354, Springer, 1998.
[64] T. Mittelholzer, “An information-theoretic approach to steganography and watermarking,” in *International Workshop on Information Hiding*, pp. 1–16, Springer, 1999.
[65] K. Solanki, K. Sullivan, U. Madhow, B. S. Manjunath, and S. Chandrasekaran, “Provably secure steganography: Achieving zero k-l divergence using statistical restoration,” in *2006 International Conference on Image Processing*, pp. 125–128, Oct 2006.
[66] A. Sarkar, K. Solanki, and B. S. Manjunath, “Secure steganography: Statistical restoration in the transform domain with best integer perturbations to pixel values,” in *IEEE International Conference on Image Processing (ICIP)*, Sep 2007.
[67] K. Sullivan, K. Solanki, B. S. Manjunath, U. Madhow, and S. Chandrasekaran, “Determining achievable rates for secure, zero divergence, steganography,” in *ICIP*, pp. 121–124, IEEE, 2006.
[68] L. Reyzin and S. Russell, “Simple stateless steganography.” Cryptology ePrint Archive, Report 2003/093, 2003. http://eprint.iacr.org/2003/093.
[69] T. V. Le, “Efficient provably secure public key steganography.” Cryptology ePrint Archive, Report 2003/156, 2003. http://eprint.iacr.org/2003/156.
[70] T. V. Le and K. Kurosawa, “Efficient public key steganography secure against adaptively chosen stegotext attacks.” Cryptology ePrint Archive, Report 2003/244, 2003. http://eprint.iacr.org/2003/244.
[71] T. Ruffing, J. Schneider, and A. Kate, “Identity-based steganography and its applications to censorship resistance,” in *ACM CCS 2013* (A.-R. Sadeghi, V. D. Gligor, and M. Yung, eds.), pp. 1461–1464, ACM Press, Nov. 2013.
[72] S. Berndt and M. Liskiewicz, “On the gold standard for security of universal steganography,” in *EUROCRYPT 2018, Part I* (J. B. Nielsen and V. Rijmen, eds.), vol. 10820 of *LNCS*, pp. 29–60, Springer, Heidelberg, Apr. / May 2018.
[73] T. Horel, S. Park, S. Richelson, and V. Vaikuntanathan, “How to subvert backdoored encryption: Security against adversaries that decrypt all ciphertexts,” in *ITCS 2019* (A. Blum, ed.), vol. 124, pp. 42:1–42:20, LIPIcs, Jan. 2019.
[74] T. Agrikola, G. Couteau, Y. Ishai, S. Jarecki, and A. Sahai, “On pseudorandom encodings,” in *TCC 2020, Part III* (R. Pass and K. Pietrzak, eds.), vol. 12552 of *LNCS*, pp. 639–669, Springer, Heidelberg, Nov. 2020.
[75] A. Lysyanskaya and M. Meyerovich, “Provably secure steganography with imperfect sampling,” in *PKC 2006* (M. Yung, Y. Dodis, A. Kiayias, and T. Malkin, eds.), vol. 3958 of *LNCS*, pp. 123–139, Springer, Heidelberg, Apr. 2006.
[76] D. Fifield, C. Lan, R. Hynes, P. Wegmann, and V. Paxson, “Blocking-resistant communication through domain fronting,” *Proceedings on Privacy Enhancing Technologies*, vol. 2015, no. 2, pp. 46–64, 2015.
[77] S. Frolov and E. Wustrow, “The use of tls in censorship circumvention.,” in *NDSS*, 2019.
[78] S. Frolov, F. Douglas, W. Scott, A. McDonald, B. VanderSloot, R. Hynes, A. Kruger, M. Kallitsis, D. G. Robinson, S. Schultze, *et al.*, “An isp-scale deployment of tapdance,” in *7th {USENIX} Workshop on Free and Open Communications on the Internet ({FOCI} 17)*, 2017.
[79] D. Luchaup, K. P. Dyer, S. Jha, T. Ristenpart, and T. Shrimpton, “Libfte: A toolkit for constructing practical, format-abiding encryption schemes,” in *23rd USENIX Security Symposium (USENIX Security 14)*, (San Diego, CA), pp. 877–891, USENIX Association, 2014.
[80] K. P. Dyer, S. E. Coull, T. Ristenpart, and T. Shrimpton, “Protocol misidentification made easy with format-transforming encryption,” in *ACM CCS 2013* (A.-R. Sadeghi, V. D. Gligor, and M. Yung, eds.), pp. 61–72, ACM Press, Nov. 2013.
[81] K. P. Dyer, S. E. Coull, and T. Shrimpton, “Marionette: A programmable network traffic obfuscation system,” in *24th USENIX Security Symposium (USENIX Security 15)*, (Washington, D.C.), pp. 367–382, USENIX Association, 2015.
[82] J. Oakley, L. Yu, X. Zhong, G. K. Venayagamoorthy, and R. Brooks, “Protocol proxy: An fte-based covert channel,” *Computers & Security*, vol. 92, p. 101777, May 2020.
[83] S. Baluja, “Hiding images in plain sight: Deep steganography,” in *Neural Information Processing Systems*, 2017.
[84] D. Hu, L. Wang, W. Jiang, S. Zheng, and B. Li, “A novel image steganography method via deep convolutional generative adversarial networks,” *IEEE Access*, vol. 6, pp. 38303–38314, 2018.
[85] Harveyslash, “harveyslash/deep-steganography.” https://github.com/harveyslash/Deep-Steganography, Apr 2018.
[86] A. Sen, S. Alfeld, X. Zhang, A. Vartanian, Y. Ma, and X. Zhu, “Training set camouflage,” *Decision and Game Theory for Security*, p. 59–79, 2018.
[87] M. Chaumont, “Deep learning in steganography and steganalysis from 2015 to 2018,” 2019.
[88] P. Wu, Y. Yang, and X. Li, “Stegnet: Mega image steganography capacity with deep convolutional network,” *Future Internet*, vol. 10, p. 54, Jun 2018.
[89] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural computation*, vol. 9, pp. 1735–80, 12 1997.
[90] M. Costa-Jussa and J. Fonollosa, “Character-based neural machine translation,” in *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, pp. 357–361, 03 2016.
[91] Y. Kim, Y. Jernite, D. Sontag, and A. M. Rush, “Character-aware neural language models,” in *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, AAAI’16, pp. 2741–2749, AAAI Press, 2016.
[92] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, u. Kaiser, and I. Polosukhin, “Attention is all you need,” in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS’17, (Red Hook, NY, USA), p. 6000–6010, Curran Associates Inc., 2017.
[93] R. Aharoni, M. Koppel, and Y. Goldberg, “Automatic detection of machine translated text and translation quality estimation,” in *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, vol. 2, pp. 289–295, 2014.
[94] A. Bakhtin, S. Gross, M. Ott, Y. Deng, M. Ranzato, and A. Szlam, “Real or fake? learning to discriminate machine from human generated text,” 2019.
[95] S. Gehrmann, H. Strobelt, and A. M. Rush, “Gltr: Statistical detection and visualization of generated text,” 2019.
[96] Y. Zhu, S. Lu, L. Zheng, J. Guo, W. Zhang, J. Wang, and Y. Yu, “Texygen: A benchmarking platform for text generation models,” in *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*, pp. 1097–1100, ACM, 2018.
[97] D. Dachman-Soled, G. Fuchsbauer, P. Mohassel, and A. O’Neill, “Enhanced chosen-ciphertext security and applications,” in *PKC 2014* (H. Krawczyk, ed.), vol. 8383 of *LNCS*, pp. 329–344, Springer, Heidelberg, Mar. 2014.
[98] C. Peikert and B. Waters, “Lossy trapdoor functions and their applications,” in *40th ACM STOC* (R. E. Ladner and C. Dwork, eds.), pp. 187–196, ACM Press, May 2008.
[99] M. Blum and S. Micali, “How to generate cryptographically strong sequences of pseudo random bits,” in *23rd FOCS*, pp. 112–117, IEEE Computer Society Press, Nov. 1982.
[100] S. Ruhault, “SoK: Security models for pseudo-random number generators,” *IACR Trans. Symm. Cryptol.*, vol. 2017, no. 1, pp. 506–544, 2017.
[101] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, “Automatic differentiation in pytorch,” in *NIPS-W*, 2017.
[102] E. Barker and J. Kelsey, “Nist special publication 800-90a revision 1 recommendation for random number generation using deterministic random bit generators,” 2015.
[103] S. Robertson, “spro/char-rnn.pytorch.” https://github.com/spro/char-rnn.pytorch, Dec 2017.
[104] M. Hutter, “The human knowledge compression contest.” http://prize.hutter1.net/, 2006.
[105] Z. Durumeric, E. Wustrow, and J. A. Halderman, “Zmap: Fast internet-wide scanning and its security applications,” in *Presented as part of the 22nd USENIX Security Symposium USENIX Security 13)*, pp. 605–620, 2013.
[106] S. Cutler, “Project 25499 ipv4 http scans.” https://scans.io/study/mi.
[107] HuggingFace, “huggingface/swift-coreml-transformers.” https://github.com/huggingface/swift-coreml-transformers, Oct 2019.
[108] T. Simonite, “Apple’s latest iphones are packed with ai smarts.” https://www.wired.com/story/apples-latest-iphones-packed-with-ai-smarts/.
[109] S. J. Oh, B. Schiele, and M. Fritz, “Towards reverse-engineering black-box neural networks,” in *Explainable AI: Interpreting, Explaining and Visualizing Deep Learning*, pp. 121–144, Springer, 2019.
[110] A. Salem, Y. Zhang, M. Humbert, P. Berrang, M. Fritz, and M. Backes, “Ml-leaks: Model and data independent membership inference attacks and defenses on machine learning models,” *arXiv preprint arXiv:1806.01246*, 2018.
[111] M. Juuti, S. Szyller, S. Marchal, and N. Asokan, “Prada: protecting against dnn model stealing attacks,” in *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*, pp. 512–527, IEEE, 2019.

A EFFICIENCY OF METEOR

We now show that the asymptotic expected throughput of Meteor is proportional to the entropy in the communication channel. Recall that the entropy in a distribution $\mathcal{P}$ is computed as $-\sum_{i\in|\mathcal{P}|} p_i \log_2(p_i)$, where p_i is the probability of the i^{th} possible outcome of $\mathcal{P}$. Similarly, the expected throughput of Meteor can be computed as $\sum_{i\in|\mathcal{P}|} p_i \mathsf{Exp}(p_i)$, where $\mathsf{Exp}(\cdot)$ is the expected number of shared prefix bits for some continuous interval of size p_i. Thus, the remaining task is to compute a concrete bound on $\mathsf{Exp}(\cdot)$.

We will make the simplifying assumption that the start of an interval p_i is placed randomly between $[0, 2^{\beta+1})$. Note that interval i will never start after $2^{\beta+1} - p_i$ in practice, so we the number of prefix bits in this case to be 0, so this simplification will lead to an expected throughput strictly less than the true value. Additionally, the starting locations for each interval are not independent in practice, as they each depend on $p_{j\neq i}$. However, this independence assumption also leads to equal or lower expected throughput, as the starting point for larger intervals will actually be more biased towards the middle of the distribution, where $\mathsf{Exp}(\cdot)$ will be lower, and smaller distributions will be biased to start near the edges of the distribution, where $\mathsf{Exp}(\cdot)$ will be higher.

By way of example, consider an interval i such that $p_i = \frac{1}{4} - \epsilon$, for some small ϵ (see Figure 7). If i starts between $[0, \epsilon)$, then it is contained completely before the prefix 01 begins, and thus would transmit 2 bits. The following p_i starting points all transmit only 1 bit, as the only shared prefix for the interval would be 0. If i starts between $[\frac{1}{4}2^{\beta+1}, (\frac{1}{4}+\epsilon)2^{\beta+1})$, the entire interval shares the prefix 01, so 2 bits can be transmitted. In $[(\frac{1}{4}+\epsilon)2^{\beta+1}, \frac{1}{2}2^{\beta+1})$, there is no shared prefix, as some of the samples that would land in that interval start with a 0 and others start with 1. The analysis continues in this way for the remainder of the starting points.

More generally, the expected throughput of an interval with size p is the average of these different sets of starting points with different length shared prefixed, weighted by size. More explicitly, let $g(p) = \lfloor -\log_2(p) \rfloor$, then

$$\mathsf{Exp}(p) \geq \begin{cases} 0 & ,p > 1/2 \\ g(p)(2^{-g(p)} - p)2^{g(p)} + p\sum_{j=1}^{g(p)-1}(j2^j) & ,p \leq 1/2 \end{cases}$$

The first part of the expression corresponds to the starting points where the interval has the most shared bits, *e.g.* the points in Figure 7 where the throughput is 2. There are $2^{g(p)}$ of these sets, each of which has size $(2^{-g(p)} - p)$, the difference between p and the nearest power of two less than 2. The sum corresponds to the when the interval transmits fewer bits, *e.g.* the points in Figure 7 where the throughput is 1 or 0. Each of these terms counts the $p2^j$ starting points where the number of bits transmitted is j.

Note that $\mathsf{Exp}(p) \geq \frac{1}{2}(-\log_2(p) - 1)$ for small enough p. To see this, note that $g(p) \geq -\log_2(p) - 1$, because of the rounding. Then, just consider the first term

$$\begin{aligned} g(p)(2^{-g(p)} - p)2^{g(p)} &\geq (-\log_2(p) - 1)(1 - p2^{-\log_2(p)-1}) \\ &= \frac{1}{2}(-\log_2(p) - 1). \end{aligned}$$

While this bound is not tight, it illustrates that $\mathsf{Exp}(p)$ asymptotically acts like $\log_2(p)$, meaning $\sum_{i\in|\mathcal{P}|} p_i \mathsf{Exp}(p_i)$, grows proportionally to the entropy in $\mathcal{P}$, $-\sum_{i\in|\mathcal{P}|} p_i \log_2(p_i)$. Thus, the expected throughput of Meteor is asymptotically optimal.

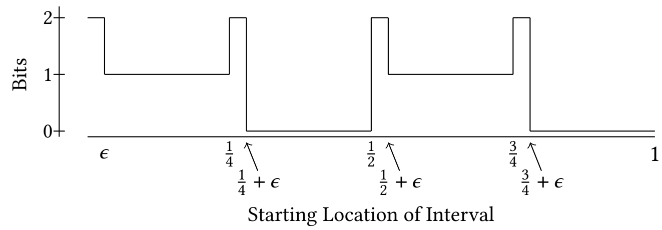


Figure 7: Bits of throughput by starting location for an interval i with size $p_i = \frac{1}{4} - \epsilon$, for some small ϵ. The expected throughput can be computed as the average of this function, *i.e.* $\mathsf{Exp}(p_i) \geq (2)(\frac{1}{4}-\epsilon)(0) + (2)(\frac{1}{4}-\epsilon)(1) + (2^2)(\epsilon)(2) = \frac{1}{2} - 6\epsilon$

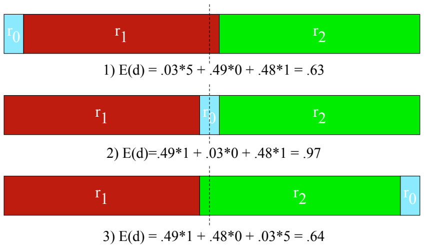
B HEURISTIC OPTIMIZATIONS

In evaluating Meteor, we also implement two heuristic optimizations that could lead to better performance without compromising security. Note that while they increases the *expected* throughput of scheme, it is not guaranteed to do so. Making any change to the output selected in a given sampling event might unintentionally push the model down a lower entropy branch of the covertext space, yielding more sampling iterations overall. The first optimization is performing a deterministic reordering operation of the model distribution, reduces the number of calls to the generative model by 20%-25%, and in some cases results in more efficient encoding and decoding times. The second optimization is an adaptation from the NLP literature that uses the generative model's internal word representation to compress English language messages.

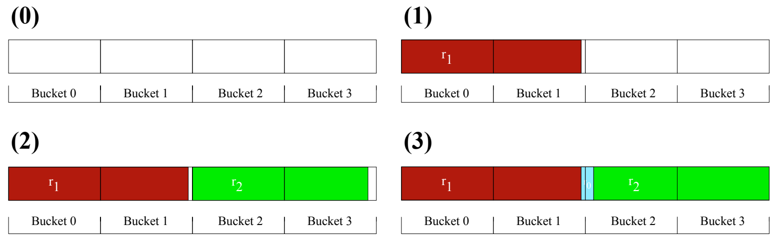
Before proceeding to the optimizations themselves, recall the intuition provided for Meteor in Section 5. In each iteration of the encoding algorithm, the sender extracts a probability distribution $\mathcal{P}$ from the generative model. $\mathcal{P}$ is subdivided into a series of continuous intervals $r_0, r_1, \ldots r_m$, the size of which determines the probability that the model would select the corresponding token is the next output. Meteor then generates a random sampling value $r = mask \oplus m$ and determines the interval r_i into which r falls. The number of bits encoded is computed as $\mathsf{LenPrefix}(r_i)$.

Optimization 1: Reordering the Distribution. We note that while we cannot manipulate $|r_i|$ without compromising the security of scheme, we are able to impact $\mathsf{LenPrefix}(r_i)$ by permuting the order of $r_0, r_1, \ldots, r_m$. It is clear there exists some such permutation that maximizes the expected throughput of Meteor, although finding this permutation proves to be difficult.

The distribution $\mathcal{P}$ is generally output by the model in some sorted or lexicographic order. This might yield to some orderings of r_i that are incredibly unfavorable to $\mathsf{LenPrefix}(\cdot)$. Consider an illustrative example in Figure 8a. If an interval r_i contains values on either side of the middle of the distribution, then $\mathsf{LenPrefix}(r_i) = 0$. When a large interval does so, as in cases (1) and (3), this severely decreases the expected number of bits that the distribution can encode. While this example is clearly contrived, it illustrates the impact correctly ordering $\mathcal{P}$ can have on the expected throughput – in this example an increase of over 50%. Importantly, we can use any reorganization procedure on the distribution provided (1) the same resulting permutation can be computed by both the sender and the receiver and (2) the size of r_i remains the same for all r_i.



(a) The impact of reorganizing a distribution.



(b) An overview of our reorganization algorithm.

Figure 8: (a) r_0 has 3% of the total probability density, while r_1 and r_2 have 48% and 49% respectively. Because $2^{-6} < .03 < 2^{-5}$, r_0 can encode 5 bits of information when located at the beginning or end of the distribution. In orderings (1) and (2), one of the larger intervals crosses the 50% line, meaning $\mathsf{LenPrefix}(\cdot) = 0$. When the smallest interval is placed in the middle, the total expected throughput of the distribution rises. (b) To reorder this distribution we create $2^2 = 4$ buckets, because the entropy is 1.16. In (1), we place the largest interval r_1 into bucket 0, overflowing its value through most of bucket 1. Note that r_1 could have been placed in bucket 2; in general, we break ties by taking the earlier bucket. In (2), r_2 can be placed either in bucket 1, overflowing into the following buckets, or placed in bucket 2, overflowing into bucket 3. To maximize $\mathsf{LenPrefix}(r_2)$, we place it in bucket 2. Finally, in (3), we note that r_0 will not fit in bucket 3, so it must be placed in bucket 1, pushing r_2 to make space.

Finding the optimal permutation of $\mathcal{P}$ proves to be a difficult task. Intuitively, each interval r_i, must be placed as a continuous block somewhere between 0 and 1 such that it does not overlap with other intervals. We take inspiration from approximation algorithms and design a greedy algorithm with pretty good performance, and we leave formal analysis and bounds proving of this algorithm for future work. A simple algorithm would be to find a "starting point" to place each interval, starting with the largest, that maximizes $\mathsf{LenPrefix}(r_i)$. However, there are 2^β possible starting points, meaning a linear search will be prohibitively expensive. Instead we generate $2^{\lceil H(\mathcal{P}) \rceil}$ buckets with capacity $\frac{\sum_i (r_i)}{2^{\lceil H(\mathcal{P}) \rceil}}$, where $H(\mathcal{P})$ is the entropy in the distribution. These buckets represent potential "starting points" that each r_i can be placed. Note that the entropy represents an upper bound on the possible value of the expected throuhput $E(\mathcal{P})$ and if each interval r_i could perfectly fit into one of these bins, $E(\mathcal{P}) = H(\mathcal{P})$.

Starting with the largest r_i, we find the bin that will maximize $\mathsf{LenPrefix}(r_i)$ when r_i is appended to that bucket. As buckets become full, they are no longer options for placement. Note that r_i may exceed the remaining capacity of a bucket, or even the total capacity of a bucket. When this is the case, we "overflow" the remainder into the following buckets. Occasionally, this overflowing remainder may cause a chain reaction, requiring other, already placed intervals be "pushed" to make space. We give a simple example of our reorganization algorithm in Figure 8b, using the same distribution given in Figure 8a. Step (3) gives an example of overflow that causes one of these chain reactions. Once each interval has been placed into a bin, the final ordering can be recovered by appending the contents of the bins.

The runtime of this algorithm is $O(2^{\lceil H(\mathcal{P}) \rceil} m)$, where m is the number of intervals; in our experiments, $\lceil H(\mathcal{P}) \rceil$ is typically less than 7, so this is close to $O(m)$, which is unsurprising given its similarities to bin-sorting. When n is very large, however, this algorithm is prohibitively expensive. In those cases, we use this algorithm to place the "big" intervals, and then simply place the smaller intervals into the first bucket with space. As we discuss in §6, reordering the distributions increases capacity by 20%-25%.

C MODEL OUTPUTS

This appendix contains stegotext outputs as generated by Meteor using several different model types. The plaintext associated with all of these outputs is the first 160 bytes of Lorem Ipsum. Figure 10 shows a truncated output for a stegotext generated using the Wikipedia model, which seems to have generated some kind of Wiki-markup contents page. Figures 11 and 12 are GPT-2 outputs for different contexts provided as input. Each output reads like a news article or book chapter. Representative output for the HTML headers model has been omitted due to space constraints. Finally, Figure 9 is a screenshot of Meteor running on the iPhone Simulator, generating stream-of-consciousness news text. Note that the context is shorter on the iPhone, as it can hold less state.

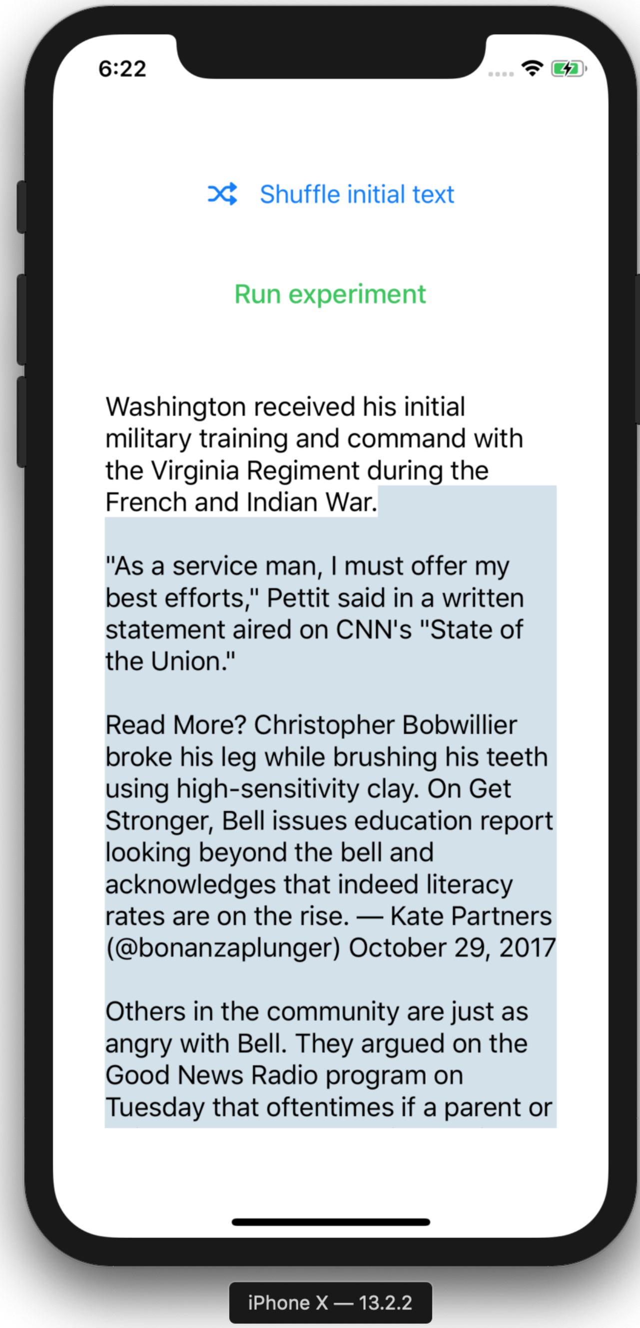


Figure 9: iPhone X screenshot of Meteor encoding of the first 160 bytes of Lorem Ipsum as generated by the GPT-2 model. Generated text is highlighted, and context is unhighlighted.

```
Haired the latter expand of the legal instance of the Imperial State of the American foal bridge, it is
↪  suspective that he was also notable to ensure that they produced a consolidate [[electricity]], the actual
↪  psychological cabinet [[Greece]] was the same time. It was born in many in the second [[tuak]] and [[timber]]
↪  at the idea of [[computer account|computer science]], which provides sales might therefore be in popular
↪  writings such as the European Investigation.

==Video games==
* [[Brown Town]]
* [[Sagan in the United States|Commonwealth of Western Technology]]
* [[Template International Society of Europe]]
* [[Commonwealth of Russia]]
* [[Commerce coverage in Sweden]]
* [[List of Constitution by Westman of the United Kingdom|British Politics]]
* [[List of California Court in 1981]]
* [[International Party of Diabetos Integrated Policy]]
* [[United Kingdom]]

===Overseas Roman Empire===
* [[Communist Council of Ireland]]
* [[Confucian States of the South Edition]]
* [[Conservative Party of the United Kingdom]]

[[Category:American states]]
[[Category:Conferences of India]]
```

Figure 10: Snippet of Meteor encoding of the first 160 bytes of Lorem Ipsum as generated by the Wikipedia model. The output was truncated to fit.

Context: The Alvarez hypothesis posits that the mass extinction of the dinosaurs and many other living things during the Cretaceous-Paleogene extinction event was caused by the impact of a large asteroid on the Earth. Prior to 2013, it was commonly cited as having happened about 65 million years ago, but Renne and colleagues (2013) gave an updated value of 66 million years. Evidence indicates that the asteroid fell in the Yucatan Peninsula, at Chicxulub, Mexico. The hypothesis is named after the father-and-son team of scientists Luis and Walter Alvarez, who first suggested it in 1980. Shortly afterwards, and independently, the same was suggested by Dutch paleontologist Jan Smit.
Stegotext: The picture in The Pale I HCR scientists' discussion now spans three dimensions. The first importance of the Yucatan Peninsula is demonstrated with the following conclusion: the Pliocene Earth has lost about seven times as much vegetation as the Jurassic in regular parts of the globe, from northern India to Siberia. Even since 1976, the continent has received or is already experiencing the worst mass extinction in recorded history since there were a million years last left from the extinction of Antarctic ice and the rapid spread of hydrocarbon-rich water as the Earth entered the Cretaceous Period. In turn, the world's oceans have been breathtakingly transformed and as a result the surrounding environments are especially vulnerable. The reason is obvious: there was no geological record of the presence of the Yucatan Peninsula in the Late Miocene epoch, which is the line that runs from the southern edge of South America to Siberia. The commander-in-chief of the Yucatan Peninsula, Frederick Nystrom, has determined that the area is to stay as barren as possible. As a result, individuals living on lake beds and on coastal areas have witnessed the loss of about 90 percent of their habitat. The Yucatan Peninsula consists of four zones, with two different habitats separated, each of which has experienced inseam damage. In one zone, along the northern shore of Lake Shemal, the retreating Tarahumara Ocean has been melted into a deep, seafloor called Nova Ravine, which south-east of the Yucatan Peninsula flows into Lake Isthmus, where there is an abundance of turtle life. A second beach, which lies at the far end of the peninsula, has been spewed down by a sea wall supporting Madagascar's Great Ocean Earthquake, 9,000 feet in magnitude and caused large numbers of deaths. The third zone, along the coast of Cancun and in Asuncion, is less severely affected. An estimated 16 percent of the continent is protected from the destruction of oceanic winds and floods. All three zones are in a state of catastrophic destruction. According to the definition provided by the National Commission on the extinction of the dinosaurs (Infection and Immunization in the Ind

Figure 11: The "Dinosaur" context and associated Meteor encoding of the first 160 bytes of Lorem Ipsum as generated by GPT-2. This is the context used in Figure 3.

Context:

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. Once victory for the United States was in hand in 1783, Washington resigned his commission.

Stegotext:

Washington was martyred in the battle of Cullman in 1788. Although down 19 men, yet his mission accomplished. He ended his life in 1896. When he died of disease, three of the men responsible said that Washington had been moved and was seen at the funeral placing him in his father's arms. Another said that Washington treated his wounds with "a study of a personal sense of injustice." (A 16th century German historian calls Washington a liar because of this website.) He was also a friend of Dr. Henry H. Jackson, who was notable as George Washington's physician and a collaborator, and recognizes him by name in the any books about him. The last surviving manuscript is from 1888 and contains a frank and truthful account of the Quakers' plight. One story states that while fighting in Whitesburg, Washington succumbed to pneumonia. He was 38 years old and according to a manuscript he got out the following year reports he grew old and fell in love. He also mentions a meeting with a woman who broke into his home and first went with him into a bath and gave him food and sleep. Three days later the woman left the room expecting him to eat her lunch and on that day he left home at 9:30 am in despair. He had not been to his bedside. On seeing this, he said a voice in him called out, "Your name is Jack. What is the girl?" Hamilton said the superior told him, "She was a layover in a bed and seven[Pg 209] feet below the bed where the general slept in very feminine attire. Nobody had time to look into her face. What was she to tell you about the general?"

A

Washington's Official Address to Congress with Americans May 17th, 1781

"I am the one to announce completely that I am a true Christian and an eloquent philosopher. I am not constrained

Figure 12: The "Washington" context and associated Meteor encoding of the first 160 bytes of Lorem Ipsum as generated by GPT-2. This is the encoding used throughout the benchmarks in Section 6.