International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 10 Issue: 12 | Dec 2023 www.irjet.net p-ISSN: 2395-0072

Automated Cover Text-based Linguistic Steganographic Model Using LSTM and Huffman Coding

Joshua J. Tom[1], Bukola A. Onyekwelu[2], Adigwe Wilfred[3], Folasade M. Aliu[4], Gabriel N. Odachi[5]

[1,2,4]*Department of Computer Science and Cyber Security, Elizade University, Ilara Mokin, Ondo State, Nigeria,*
[3]*Department of Computer Science, Delta State University of Science and Technology, Ozoro, Delta State, Nigeria,*
[5]*Admiralty University of Nigeria, Ibusa, Delta State, Nigeria.*

Abstract - *One of the methods used in text steganography is linguistic method. Carrier modification-based linguistic steganography is vulnerable to various attacks. Recent works in deep neural network based linguistic steganography utilize Recurrent Neural Network (RNN) to automatically generate the cover text. First, purely RNN-based steganography methods suffer from exposure bias and embedding deviation. Second, RNN struggles with long-range dependencies due to the vanishing gradient problem. Third, most RNN-based automatic covertext generation do not consider the context of the hidden message with respect to the covertext making the stegotext more vulnerable to detection by steganalysis techniques. In contrast, LSTM-based language models can capture complex and long-term dependencies between words in a sentence and also ability to generate more fluent and coherent text. In this paper, we propose an RNN-LSTM based linguistic steganography system using Huffman coding to hide the secret message. The developed model takes the meaning and context of the hidden message into account making the steganographic message to blend well with the cover text averting statistical analysis attack. It was also able to embed information in a way that was imperceptible to the human eye, making it nearly impossible to detect by anyone who was not aware of its existence. In addition, our model was able to handle large amounts of payload without risking detection.*

***Key Words*: linguistic steganography, automated text generation, LSTM, deep learning, word-level model, secret message hiding, information extraction, Huffman coding.**

1. INTRODUCTION

Steganography involves concealing information within an object, such as a letter, that appears innocuous to the untrained eye [1]. By this technique, a message can be securely sent from sender to receiver without interception or understanding by unauthorized parties. Steganography started in the ancient times during which people used wax tablets to write secret messages concealed with layers of insignificant text. In modern times, steganography is commonly used in digital media, such as embedding information in images, videos, or audio files. The main desire in using steganography is to render hidden messages undetectable by adversaries. A successful text steganographic technique requires careful selection of cover text to avoid detection. To achieve this, a chosen cover text must satisfy naturalness, coherency, and capable of not raising suspicion from a human reader. Furthermore, there is the need for both cover text and the hidden message to be semantically similar so that they are indistinguishable to avoid discoverability of the steganographic message. To automate the process of generating cover text, various machine learning algorithms can be used. These automatic text generation algorithms [2] can analyze large volumes of text and generate cover text that is both natural-sounding and has sufficient complexity to effectively hide the secret message. Among the popular algorithms used in text generation automation is the Markov Chain algorithm. Markov chains relies on the statistical properties of the input text to create a model that can be used to generate new sentences. The algorithm works by analyzing the source text to identify common word sequences used to construct a probabilistic model that assigns a probability to each possible word that follows a given sequence. The created model can then be used to generate new characters or words by selecting characters or words based on the predicted probability the previous characters or words in the generated sequence. Markov chains can be combined with deep neural networks. The technique involves mapping the secret information onto a sequence of transition probabilities between states in the Markov chain. The deep neural network is then used to encode the cover text while maintaining the original Markov chain probabilities. When the encoded text is decoded, the hidden message can be revealed by analyzing changes in the transition probabilities. Recurrent Neural Networks (RNN) are well-suited for encoding cover text. Leveraging the ability of RNN to process sequential data such as text, the context and meaning of a sentence can be captured considering the previous inputs [3]. Therefore, RNN is important in natural language processing (NLP) tasks as it is an efficient and effective way to encode cover text. But RNN is without limitations that impact negatively upon the quality of text generated. RNN is prone to the problem of vanishing or exploding gradients which causes RNN to struggle with long-range dependencies. Also, RNN-based steganography methods suffer from exposure bias and embedding deviation. LSTM (Long Short-Term

Memory) is designed to address the above mentioned RNN deficiencies of vanishing gradients, which can occur when traditional RNNs are trained on long or complex sequences. LSTM, a type of recurrent neural network, retains and utilizes information for a longer period. It is used in text steganography for cover text encoding due to its effectiveness in encoding the hidden information in a more complex and secure manner. This is made possible by presence of the long-term memory and dynamic nature of LSTM, allowing it to process and interpret the information that has long term dependencies. The deep neural network models chosen in this paper play a significant role in enhancing information hiding in linguistic steganography. RNNs process textual data sequentially and can learn features from different hidden states [4]. LSTM networks are capable of handling long-term dependencies and can selectively discard irrelevant information [5]. Hence, the duo would facilitate our model's ability to retain useful information and discard irrelevant data helping to protect the hidden message from being detected.

The main contribution of this research is the development of a method for embedding secret messages within automatically generated cover text using RNN/LSTM algorithms. This approach provides a novel way for secure communication as the secret messages are hidden within seemingly ordinary text. The study shows that the embedded messages can be successfully extracted with minimal loss in text quality, making it a promising approach for secure communication. The technique could be useful in fields such as military, finance, and communication where secure communication is critical.

2. LITERATURE REVIEW

Steganography a form of security through obscurity, as the hidden information is not easily found. The term "Steganography" is derived from two ancient Greek words: "Stegano" and "Graphy", both of which refer to "Cover Writing" [6]. The idea is to hide data within a larger file allows for data to be sent securely over an insecure network. The methods used to achieve steganography are varied, but typically involve some form of encryption or data obfuscation. Another method may involve using a special algorithm to scramble the data within the file, making it impossible to detect the true contents of the file. Traditional linguistic steganography involves hiding messages within natural language text, which typically involve the use of natural language elements. Nevertheless, traditional linguistic steganography has several drawbacks. First, its approaches lack automation, as embedding the hidden message is done manually and any errors committed during embedding may not be immediately noticeable. secondly, it lacks the ability to analyze the cover text and hidden text in terms of volume and accuracy. Hence, traditional linguistic steganographic techniques require a great deal of time and effort to be successful.

With the increasing prevalence and simplicity of Artificial Intelligence (AI) techniques, Steganography is rapidly shifting from conventional model building to AI models for Steganographic applications [7]. AI-based models have several advantages over their traditional Linguistic Steganographic counterparts. AI techniques are more automated as they provide greater accuracy in cover text and hidden text analysis, allows for faster analysis of large volumes of data with complex models and are more targeted and specific to the data they analyze.

2.1 Taxonomy of Steganography

Steganography is a technique of hiding information within a cover media without drawing attention to its existence. It can be categorized into five types: Text, where data is hidden within text files [8]; Audio, where information is hidden within audio files [9]; Image, where data is concealed within image files [10]; Video, where information is hidden within video files [11]; and Protocol, where covert communication is accomplished through protocol data units as shown in figure 1. Steganography is widely used for secret communication in various industries, including security, military, and intelligence. It helps protect sensitive information while keeping the communication private and confidential.

The focus of this paper is directed towards text steganography. The concepts and techniques involved in hiding secret messages within text are presented. The paper aims to provide a comprehensive understanding of text steganography and its potential applications. There are three categories in which text steganography can be classified, namely format-based approaches, statistical generation techniques, and linguistic methods shown in figure 1.

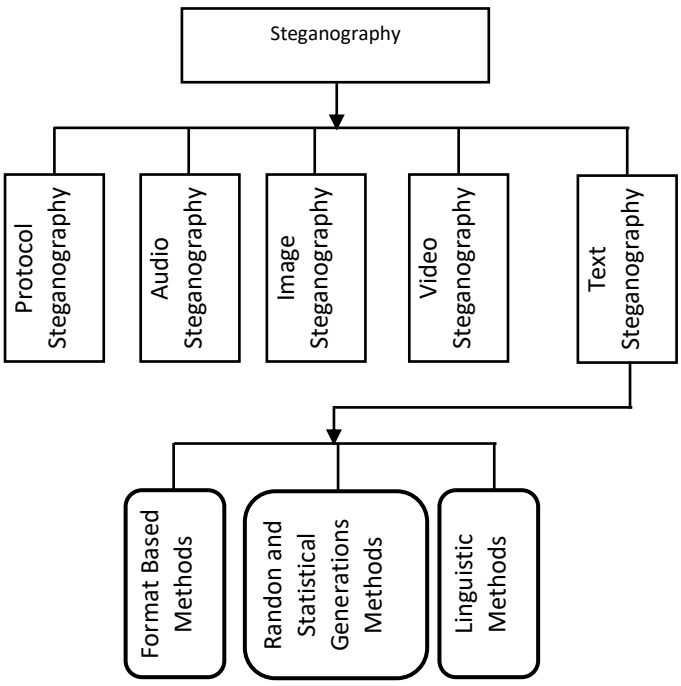

Fig -1: Classification of Steganography

Format-based methods involve hiding secret messages in the form of font size, type, or spacing. Statistical generation methods use algorithms to generate redundant data that contains hidden messages. In contrast, linguistic methods make use of semantic units, such as words or phrases, to conceal information. Each category has its own strengths and weaknesses for hiding information in text, and choosing the appropriate method depends on the specific use case. Linguistic steganography has a variety of use cases, ranging from espionage to personal privacy. Areas of application of linguistic steganography include but not limited to political or military context, individuals use to hide messages in social media posts, literary or artistic contexts. Basically, we also make a distinction between two types linguistic steganography. Two common types of linguistic steganography are carrier modification-based and carrier generation-based steganography see figure 2.

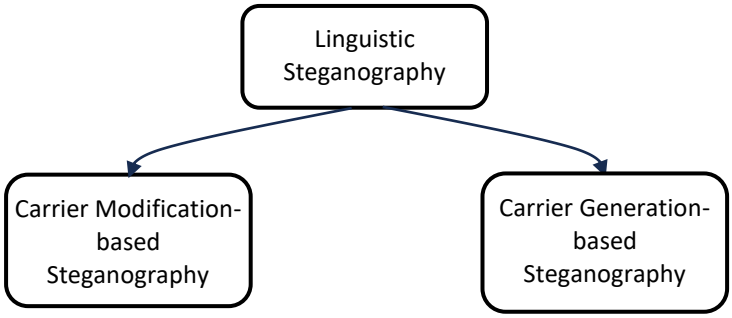

Fig -2: Types of Linguistic Steganography

Carrier modification-based steganography involves modifying an existing text to contain hidden messages, such as by changing the font, punctuation, or word choice. Carrier generation-based steganography, on the other hand, involves generating new text with hidden messages, such as by using specialized algorithms to construct sentences with hidden meanings. Both methods require a high degree of linguistic knowledge and skill to execute successfully.

2.2 Last Short-Term Memory Network

LSTM network offers a vivid solution to the vanishing gradient problem in RNNs by introducing a memory cell and three gates: input gate, forget gate, and output gate as shown in figure 3. These gates control the flow of information into and out of the memory cell, enabling the model to selectively retain or discard information over longer time periods. The cell state serves as a memory unit that carries information over time steps. The input gate controls the amount of information that is added to the cell state, while the forget gate controls the amount of information that is removed. Finally, the output gate determines the output of the LSTM based on the cell state and the input. The components in figure 3 are defined by the following set of mathematical formulas describing the interactions between them, allowing the LSTM to learn complex sequences of data:

$$\begin{cases} I_t = \sigma(W_i[h_{t-1}, x_t] + b_i), \\ F_t = \sigma(W_f[h_{t-1}, x_t] + b_f), \\ C_t = F_t C_{t-1} + I_t \tanh(W_c[h_{t-1}, x_t] + b_c), \\ O_t = \sigma(W_o[h_{t-1}, o_t] + b_o), \\ h_t = O_t \tanh(C_t), \\ y_t = f_y(W_y h_t + b_y) \end{cases} \quad (1)$$

where I_t indicates the input gate which control which controls what new data needs to be stored or not in the memory cell. F_t indicates the forget state that controls whether the stored information needs to be discarded. C_t is the memory cell collectively controlled by both the input gate and the forget gate. O_t is the output gate which takes care of whether the current hidden state needs to be influenced by the memory cell. x_t is the input vector, h_t is the hidden vector representing the output of the current block, and y_t is the final output, at time step t. W_* and b_* represent matrices and biases respectively, to be learned by the model. The + and x functions represent the element-wise addition and multiplication respectively. We need to

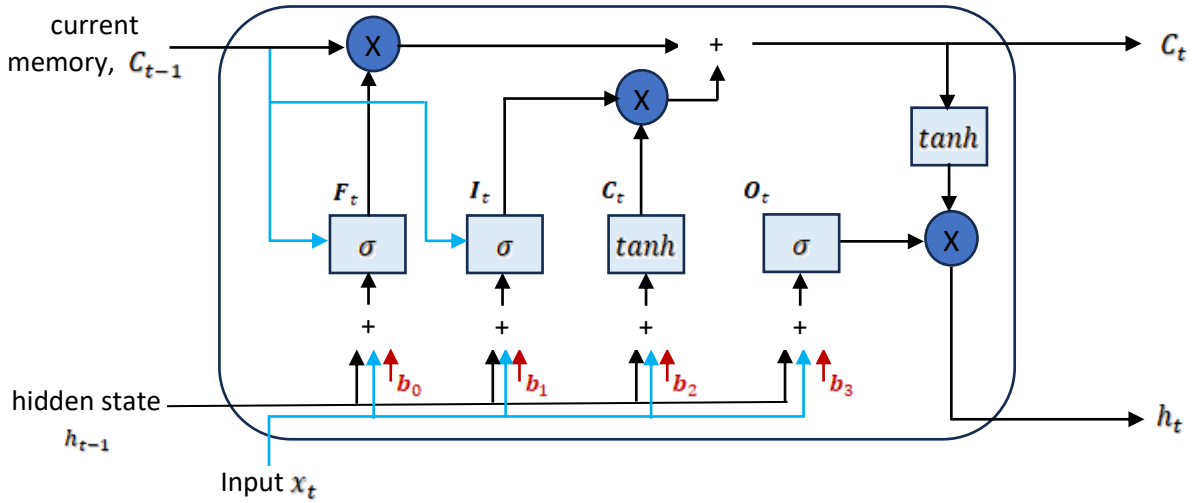

Fig. 3: LSTM Network with input, forget, and output gates

notice that when we calculate the output at time step t, the information we used based on the input vector at time step *t*, also includes the information stored in the cells at the previous *t*-1 moments. Therefore, we must ensure that when evaluating the output at time step t, the following formula is followed:

$$y_t = f_{LSTM}(x_t|x_1, x_2, \dots, x_{t-1}) \quad (2)$$

Note that the hyperbolic tangent (tanh) in the LSTM unit serves the purpose within the architecture of helping to control the flow of information and gradients thereby addressing the vanishing gradient problem that can occur in deep neural networks while the Sigmoid, σ maintains the values between 0 and 1 helping the network to update or forget the data. If the multiplication results in 0, the information is considered forgotten. Similarly, the information stays if the value is 1.

2.3 Text Generation for Linguistic Steganography

Deep learning algorithms can be employed to build a probabilistic model that describe a series of events where the probability of each event depends solely on the previous state. RNN-LSTM models are often used in Natural Language Processing for automatic text generation. By using the statistical properties of language, such as word frequency and sentence structure, deep learning-based language models can generate texts that is meaningful, fluent and close to natural language. The generated text can be used to as a cover text in steganographic models to hide secret messages. By using deep learning approach, researchers can examine the likelihood of certain outcomes and make predictions about future events. An RNN/LSTM effectively captures the complex patterns within the data. Statistical NLP involves the use of probability distributions to model individual words as follows:

$$p(S) = p(w_1, w_2, w_3, \dots, w_n) = p(w_1)p(w_2|w_1) \dots p(w_n|w_1, w_2, w_3, \dots, w_{n-1}) \quad (3)$$

where S is a sentence with n words consisting of the word sequence $w_1, w_2, w_3, \dots, w_n$ and $p(S)$ denotes the likelihood of the sentence. The joint probability of the whole sentence is calculated as the product of n conditional probabilities. The probability of all n words occurring together is determined by the product of the probability of each word given that the previous one(s) have occurred. By doing so, the overall probability of the entire sentence can be determined if the dependencies between all words are known. Hence for automatic text generation, we need a knowledge of the statistical language model of the training dataset so as to allow for accurate predictions. Based on equation 1, the probability distribution of the n^{th} word can be calculated given the $n-1$ words. Hence, this is similar to a time series event and the RNN-LSTM model is suitable for the purpose since a good estimate of the statistical language model of the training sample set is needed in this case. We relate the equation to the behavior of the LSTM network to prove its suitability for the problem. Suppose we are given a value space, $\beta = \{\mu_1, \mu_2, \mu_3 \dots, \mu_m\}$ and a stochastic variable sequence $Q = \{q_1, q_2, q_3 \dots, q_n\}$. Note that the value of Q are sampled from μ and for convenience, let the t^{th} state be μ^t, such that $q_t = \mu^t$ where $\mu^t \in \beta$. If $p(q_t|q_1, q_2, q_3, \dots, q_{t-1})$ is anything to go by, then we can build the model as follows:

$$P(q_t = \mu^t) = f(P(q_{t-1} = \mu^{t-1}), P(q_{t-2} = \mu^{t-2}), \dots, P(q_t = \mu^t))$$
$$s.t. \sum_{i=1}^{m} P(q_t = \mu_i) = 1, \forall \mu_i \in \beta$$

where f is the frequency transfer function, which can be represented as a matrix where the (i,j)[th] element represents the probability of transitioning from state *i* to state *j* after a given number of steps. The frequency transfer function is useful for analyzing the long-term behavior and stability of an LSTM based language model, as well as for predicting future states based on the current state. Hence, the probability of the whole stochastic variable sequence $Q = \{q_1, q_2, q_3 \dots, q_n\}$ can be stated as:

$$P(Q) = P(q_1, q_2, q_3 \dots, q_n) = P(q_1 = \mu^1)P(q_2 = \mu^2) \dots P(q_n = \mu^n) = P(q_1)P(q_2|q_1) \dots P(q_n|q_{n-1}, q_{n-2}, \dots, q_1) \quad (5)$$

Now, considering equations 1 and 3, we can deduce that μ^i in (3) represents the i^{th} word in the sentence and can be used to represent the conditional probability distribution in (1). Based on this analysis, RNN-LSTM is suitable for modeling text for automatic text generation in this paper. To generate automatic text the RNN-LSTM model is created with a training data set comprised of a considerable number of samples. Additionally, a dictionary of terms aligned with the training requirements must be developed. By doing so, the model can accurately predict the probability of certain words and phrases following each other, creating coherent and meaningful text. A large enough dictionary for such model can be constructed as follows:

$$D = word_{D_1}, word_{D_2}, word_{D_3}, word_{D_N} \quad (6)$$

where D is the dictionary, $word_{D_i}$ is the i^{th} word in the dictionary and N is the total number of words in the dictionary. Here, the dictionary, D is synonymous with the value space, β given already. A sentence can be input as a sequential signal with individual words, i^{th} word in the

sentence, S treated as the signal at a discrete time i, which can be represented as:

$$S = word_{s_1}, word_{s_2}, word_{s_3}, word_{s_L},$$
$$\text{s.t. } \forall\, word_{s_i} \in D \quad (7)$$

where $word_{s_i}$ is the i^{th} word in S and L is the number of words in S. The transition probability of each word in S is needed in the process of automatic text generation. Although Markov chain model can also be employed for generating text automatically, one major limitation of Markov models for text generation [12] is that they can only use previous words to make predictions, and thus are limited in their ability to capture longer-term patterns and dependencies in language. In contrast, RNN-LSTM models can use information about the entire history of a sequence to make predictions, allowing them to generate more coherent and contextualized text.

2.4 Related Works

In this section, we review the development in linguistic steganography using the RNN-LSTM model. The study focuses on the use of deep learning methods for text encoding and decoding. We examined various research efforts aimed at applying RNN-LSTM to linguistic steganography highlighting the strengths and limitations of each approach.

Gurunath et al. [7] examined the effectiveness of an artificial intelligence-based statistical language model in concealing secret messages within text through steganography. Through a series of experiments, the authors evaluated the ability of the system to embed hidden messages without significantly altering the original text's semantic meaning. Results demonstrated that the AI-based language model was successful in embedding covert messages with a high degree of accuracy and can serve as a viable tool for information security and privacy applications. In the paper, they used NLP-based Markov chain model for auto-generative cover text offering several positive aspects. It enables the generation of highly relevant and coherent text based on a given theme or topic, which can increase the perceived quality of the content.

Yang et al. [13] proposed a linguistic steganography technique based on Variational Auto-Encoder (VAE). In the proposed method, the plain text message is first encoded using a VAE to generate a compressed representation, which is then embedded into a cover text using a text-based method. Their technique ensures robustness, as the compressed representation of the message can tolerate some noise introduced during embedding. Some limitations of a linguistic steganography based on a Variational Auto-Encoder (VAE) include the need for a large amount of data to train the VAE effectively. Additionally, the generated text may not always be coherent or fluent, which can make detection easier. There is also a trade-off between the embedding capacity and the quality of the generated output. Finally, the VAE may not be robust enough to handle natural language variability and may struggle to maintain the meaning of the original text after embedding.

Using a non-machine learning approach, Yang et al. [14] developed new steganography method based on a unique sampling strategy. Their approach allows for more secure encryption of secret data within an image. The method involves sampling at specific locations within the image, determined by a pattern generated by an algorithm. The encrypted data is then embedded in the image's least significant bits at these locations, making it harder for attackers to detect and extract the secret data. The limitation of a steganography method based on the sampling strategy in their paper may not be as effective as other steganography methods in hiding the secret information. Furthermore, the new sampling strategy may be difficult to implement and may require a lot of computational resources, which can limit its practical applications.

To offer improved security and stealthiness in linguistic steganography, Xiang et al. [15] proposed a method which utilizes a Long Short-Term Memory (LSTM) based language model to embed the secret message into individual characters, making it more difficult to detect and extract the hidden information. The model addressed the low embedding capacity problem suffered by most existing linguistic steganographic methods. One major limitation of embedding secret information into characters rather than words using an LSTM based language model is that it decreases the capacity of the model to encode the nuances of natural language. The model may struggle to differentiate between the intended hidden message and noise generated by the character-level encryption.

Hwang et al. [16] proposed a new steganographic method called Steg-GMAN, which involves using a Generative Multi-Adversarial Network to increase the overall security of steganography. Their approach is designed to enhance the resilience of steganographic messages against attackers by creating a more complex and adaptive system that can better identify and thwart attempts to uncover hidden information. The proposed method outperforms both traditional steganography techniques and modern GAN-based steganographic methods according to comparative results in the paper. However, GMAN may introduce visible distortions in the image, which can make the steganographic message easily detectable.

Kumar et al. [17] proposed a method for steganography that involves combining LSB encoding with deep learning modules, which are trained using the Adam

algorithm. In their proposed work, the authors developed a deep learning-based method to hide secret images in plain sight by dispersing them throughout the bits of a cover image. The hiding network generates a stegotext by embedding secret messages into cover images using LSB encoding. The reveal network is then able to extract the hidden message from the stegotext. Their deep learning modules improve the efficiency and security of the steganography process. However, their approach has some shortcomings. Combining LSB encoding with deep learning modules may face some limitations that include the fact that LSB encoding is a low-level steganographic technique that is vulnerable to detection by image analysis techniques. Furthermore, attackers may use advanced techniques such as generative adversarial networks (GANs) to create adversarial examples that fool the deep learning model and reveal the hidden data.

Zhou et al. [18] proposed a new linguistic steganographic model that is based on adaptive probability distribution and generative adversarial network. Generative adversarial network is used to ensure that the stego-text remains indistinguishable from the original text. Their approach strikes a balance between usability and security, allowing for efficient communication while maintaining an impenetrable level of confidentiality. In their work, the steganographic generator uses a generative adversarial network to overcome exposure bias and create a candidate pool using a probability similarity function. However, their work stands to suffer from the same limitation as in the work of Kumar et al. [17] due to the use of the generative adversarial network which attackers may leverage to foul the model.

Ongoing developments in cryptology and encryption methods may strengthen the security of steganography in the future. In the light of this, Zheng & Wu [19] proposed a novel autoregressive LS algorithm that utilizes BERT and consistency coding to enhance embedding payload while maintaining system security. The proposed method achieves a better trade-off between these two goals, ensuring the highest possible level of protection while also increasing the amount of data that can be reliably encoded. BERT, while a state-of-the-art language model, still has limitations in understanding context-dependent and entity-specific information. In addition, consistency coding methods can be time-consuming and require significant human annotation and processing.

Leveraging the unique architecture of LSTM-based RNNs to maintain a kind of memory, which are crucial to accurate predictions, Buddana et al. [20] develop a generative model for automatic text generation. The model was trained on a large dataset of existing text, and used statistical algorithms to learn patterns and structures within the data. The trained model was able to generate novel and coherent text that is similar in style and content to the original dataset. The model could be a valuable tool for a variety of applications, including content creation, data analysis, and natural language processing. The paper describes how to generate text using word-level Long Short-Term Memory Recurrent Neural Networks (LSTM-RNN). The paper provides a step-by-step illustration of generating new text based on a given text corpus. Overall, this paper highlights the effectiveness of using LSTM-RNNs for text generation purposes.

3. METHODOLOGY

3.1 Proposed System Architecture

In this section, we present a new system architecture for automatic cover text-based LSTM model designed for linguistic steganography. The main objective of this architecture (figure 5) is to improve the hiding of confidential information within text by utilizing LSTM models. In developing the automatic cover text-based steganography, we divide the process into two phases: (i) Automatic Cover Text Generation and (ii) Secret Message Hiding and Recovery. In the first phase, the cover text is generated using the LSTM-based language model develop in this paper. The generated text is generated to be grammatically correct and semantically coherent, so that it does not alert anyone of its hidden contents. In the second phase, the automatically generated text is entered as input to this phase where the secret message is embedded into it by same LSTM model. The trained LSTM neural network block shown in Figure 5 and stego-key should be pre-shared between the sender and the receiver so that the secret data can be fully reconstructed.

3.2 Automatic Cover Text Generation

We expand the automatic carrier generation engine previously shown in figure 4 to show its internal structure and components. Figure 4 shows the LSTM based automatic cover text generator (ACG) model comprised of a series of LSTM cells. Sequences of words, $w_t, w_{t+1}, w_{t+2}, \ldots, w_N$ (N is the number of words in a sequence) are input at different time steps, *t, t+1, t+2*, into the model and the model generates $w_{t+1}, w_{t+2}, w_{t+3}, \ldots, w_{N+1}$ as output. The LSTM introduces memory cells that allow the model to retain information over long sequences, while the RNN performs temporal processing to capture dependencies. among the text elements. The combination of both makes the model effective in hiding and retrieving secrets. After data preprocessing to weed out punctuations, stopwords, set the dataset text to lowercase, etc.

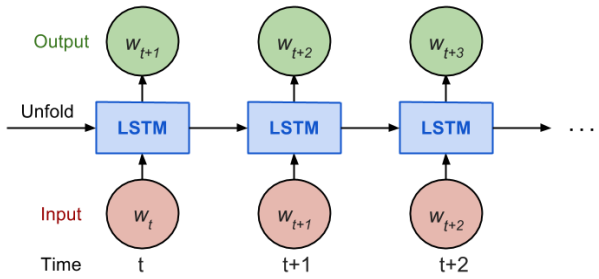

Fig. 4: LSTM based Automatic Cover Text Generator Architecture

2. convert the input words to a list of integers using the vocabulary dictionary. Also, convert the output words converted to a list of integers using the same vocabulary dictionary. A section of the first 10 generated input sequences of the input sequence list is as shown in figure

[[9, 11, 251, 3, 102, 26, 471, 6, 345, 72, 14, 382, 16, 1, 827, 2, 6, 346, 130, 3, 50, 131, 53, 592, 5, 19, 828, 61, 1, 320, 14, 382, 11, 829, 20, 4, 19, 41, 682, 53, 1453, 10, 4, 2, 27, 32, 1, 208, 6, 320], [11, 251, 3, 102, 26, 471, 6, 345, 72, 14, 382, 16, 1, 827, 2, 6, 346, 130, 3, 50, 131, 53, 592, 5, 19, 828, 61, 1, 320, 14, 382, 11, 829, 20, 4, 19, 41, 682, 53, 1453, 10, 4, 2, 27, 32, 1, 208, 6, 320, 56], [251, 3, 102, 26, 471, 6, 345, 72, 14, 382, 16, 1, 827, 2, 6, 346, 130, 3, 50, 131, 53, 592,

Fig. 7: A section of the first 10 generated input sequences of the input sequence list

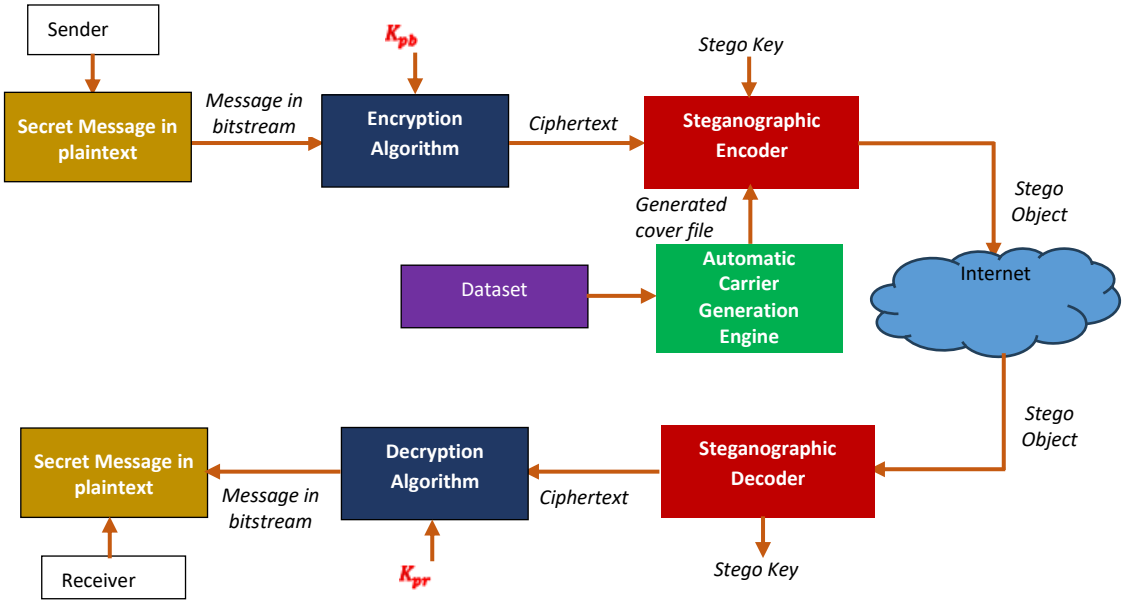

Fig. 5: Architecture of the proposed steganographic model

The procedure for cover text generation in this paper proceeds as follows:

1. create a vector of each sentence in the dataset using the index of words from our vocabulary dictionary as values. We present the first 106 entries in our model's dictionary generated using Python as shown in figure 6.

{'the': 1, 'and': 2, 'to': 3, 'it': 4, 'she': 5, 'of': 6, 'said': 7, 'you': 8, 'alice': 9, 'in': 10, 'was': 11, 'that': 12, 'as': 13, 'her': 14, 'at': 15, 'on': 16, 'all': 17, 'with': 18, 'had': 19, 'but': 20, 'for': 21, 'they': 22, 'so': 23, 'be': 24, 'not': 25, 'very': 26, 'what': 27, 'this': 28, 'little': 29, 'he': 30, 'out': 31, 'is': 32, 'one': 33, 'down': 34, 'up': 35, 'there': 36, 'if': 37, 'his': 38, 'then': 39, 'about': 40, 'no': 41, 'them': 42, 'know': 43, 'like': 44, 'were': 45, 'would': 46, 'went': 47, 'again': 48, 'herself': 49, 'do': 50, 'have': 51, 'when': 52, 'or': 53, 'could': 54, 'queen': 55, 'thought': 56, 'off': 57, 'time': 58, 'how': 59, 'me': 60, 'into': 61, 'see': 62, 'can': 63, 'well': 64, 'did': 65, 'who': 66, 'king': 67, 'your': 68, 'don': 69, 'now': 70, 'turtle': 71, 'by': 72, 'began': 73, 'my': 74, 'its': 75, 'll': 76, 'an': 77, 'm': 78, 'way': 79, 'hatter': 80, 'mock': 81, 'quite': 82, 'gryphon': 83, 'are': 84, 'think': 85, 'just': 86, 'their': 87, 'much': 88, 'say': 89, 'some': 90, 'first': 91, 'here': 92, 'rabbit': 93, 'head': 94, 'go': 95, 'only': 96, 'which': 97, 'thing': 98, 'more': 99, 'never': 100, 'voice': 101, 'get': 102, 'come': 103, 'oh': 104 , 'looked':

Fig. 6: A Section of the Generated Vocabulary Dictionary

3. Convert the input words into one-hot vectors for each word in the input.
4. Compute the hidden state layer comprising three LSTM cells with parameters as shown in figure 10 of section 4.2. To do this, we create three LSTM layers with 800 neurons each. We added a final dense layer to predict the index of the next word
5. Compute the output layer and then pass it through softmax to get the results as probabilities.
6. Feed the target word, w_t at time step (t) as the input word at time step (t + 1).
7. Go back to step 1 and repeat until we finish all the letters in the training dataset.

3.3 Secret Information Hiding Module

In the information hiding module, we assign unique code to each word based on their conditional probability distribution given as $p(w_n|w_1, w_2, w_3, \dots, w_{n-1})$ and use this to map the secret message represented as a binary bitstream to the word space. With a well-trained model, it is possible to have one feasible solution at each time step, hence the model also choses the top m words sorted in order of the prediction probabilities of all the words in the dictionary, D to build the candidate pool (CP). For instance, if we use C_i to represent the i-th word in CP then CP can be modeled as:

$$CP = [c_1, c_2, \dots c_m]$$

Given a suitable size of the candidate pool, any word, C_i selected as the output at a specific time step is still reasonable and ensures the semantics of the cover text is not affected. With the probability distribution of the current word, we create a candidate pool containing the m most likely words, sorted in descending order of probability. Next, we built a Huffman tree (Huffman, 1952) based on all the entries in the candidate pool. We use the Huffman tree to hide the secret message by first converting the secret message to a bitstream and then traversing the Huffman tree according to the bit pattern of the bitstream. The bitstream sequence corresponds to the pattern from the root to the leaf is hidden in that leaf, which returned as the encoded word. The model accomplishes this by reading in one bit of the secret bitstream at a time and if the bit is a "0", it turns to the left subtree else if the bit is a "1" it turns to the right subtree. The value of the leaf node is added to a previous output as the generated the stegotext. This process is This process is repeated until the end of the binary bitstream. The information hiding process is illustrated in figure 8.

3.4 Secret Information Extraction Module

We input the first word of the steganographic text received from the sender as the keyword into the LSTM model which then calculates the probability distribution of words at each subsequent time point in time, producing an output vector which shows how likely it is for a given word to appear after another one in the sentence. With the probability distribution of the current word, we create a Candidate Pool containing the m most likely words, sorted in descending order of probability and then use this to build a Huffman tree. The decoded bits are then extracted from the leaf node and propagated upwards towards the root node, following the path of that particular leaf. As each internal node is encountered in this process, a bit is added to form a longer codeword until we reach the root node which contains all the encoded information. This allows us to accurately extract the hidden data contained within each transmitted word. The information hiding and extraction algorithms implemented in this paper are as shown in figure 9.

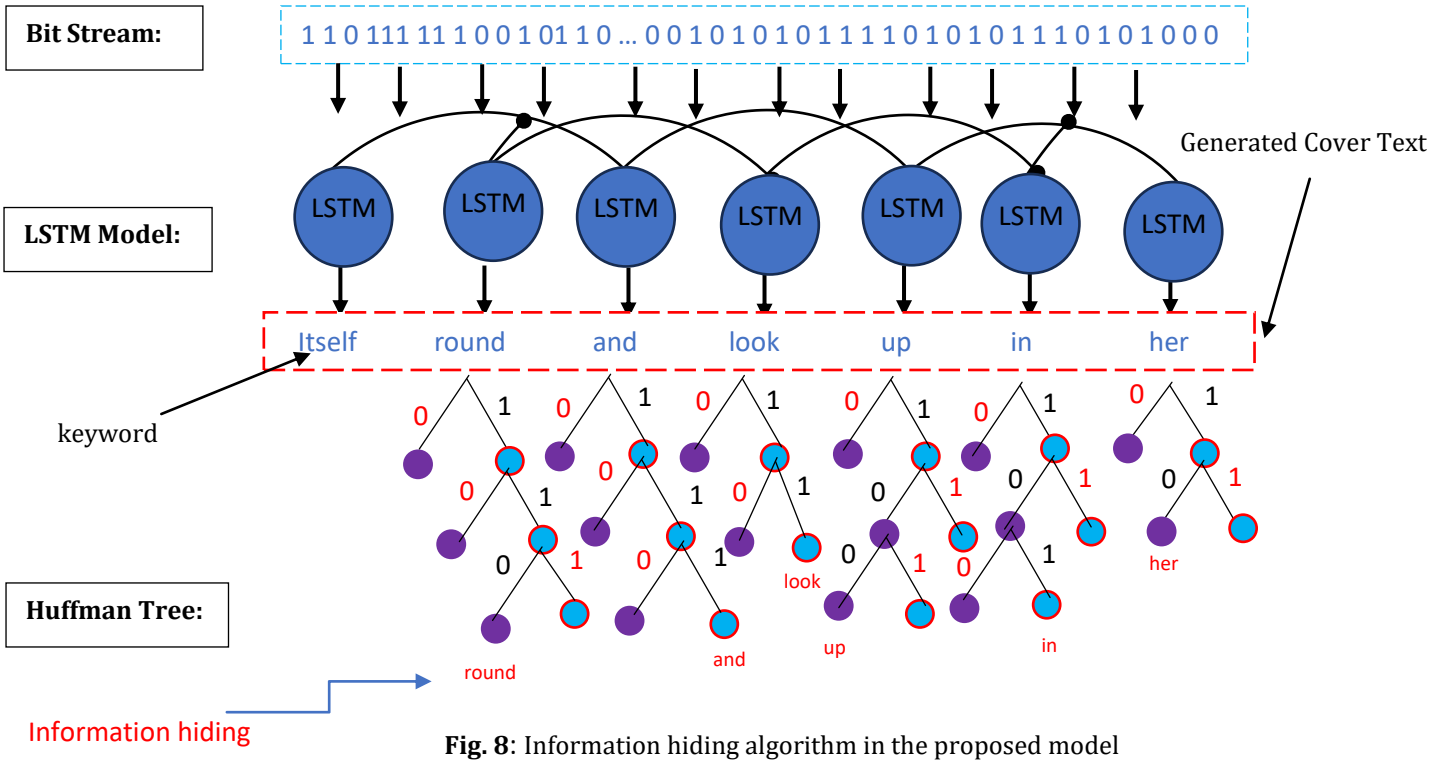

Fig. 8: Information hiding algorithm in the proposed model

Information Hiding Algorithm
Input:
Secret bitstream: $B = \{1,1,0,1,1,1,1,1,1,0,0,1,0,1,1,0\}$
Candidate Pool Size (CPS): m
Keyword list: $A = \{key_1, key_2, \ldots, key_F\}$
Output:
Multiple generated steganography text: $Text = \{S_1, S_2, \ldots, S_N\}$
1: Data preprocessing and LSTM training;
2: while not the end of B do
3: if not the end of current sentence, then
4: Calculate the probability distribution of the next word according to the previously generated words using the trained LSTM;
5: Descending the prediction probability of all the words and select the top m sorted words to form the Candidate Pool {CP};
6: Construct a Huffman tree(VLC) according to the probability distribution of each word in the CP and encode the tree;
7: Read the binary stream, and search from the root of the tree according to the encoding rules until the corresponding leaf node is found and output its corresponding word;
8: else
9: Random select a keyword key_i in the keyword list A as the start of the next sentence;
10: if not the end of current sentence, then
11: Select the word with the highest probability outside the Candidate Pool as the output of current time;
12: Select the word with the highest probability at each moment as output until the end of the sentence;
13: return Generated sentences

Information Extraction Algorithm
Input:
Multiple generated steganography text: $Text = \{S_1, S_2, \ldots, S_N\}$
Candidate Pool Size (CPS): m
Output:
Secret bitstream: $B = \{1,1,0,1,1,1,1,1,1,0,0,1,0,1,1,0\}$

1: for each sentence S in $Text$ do
2: Enter the first word of the sentence S into the trained RNN as the Key;
3: for each word in $Word_i$ sentence S do
4: Calculate the probability distribution of the next word according to the previously words using LSTM;
5: Descending the prediction probability of all the words and select the top $m = 2k$ sorted words to form the Candidate Pool {CP};
6: Using variable-length coding to encode words in the Candidate Pool;
7: if $Word_i$ in CP then
8: Based on the actual accepted word $Word_i$ at each moment, determine the path from the root node to the leaf node;
9: According to the tree coding rule, i.e., the left side of the child node is 0 and the right side is 1, extract the corresponding bitstream and append to B;
10: else
11: The information extraction process ends;
12: return Extracted secret bitstream B;

Fig. 8: Information hiding algorithm in the proposed model

4. EXPERIMENT AND ANALYSIS

4.1 Data Pre-Processing

For our model to automatically mimic and learn the writing patterns of humans, for experimental purposes, we choose *alice_in_the_wonderland.txt* as our dataset. For practical implementations, we recommend much larger data on Twitter (now X) particularly the Sentiment140 dataset. In the pre-processing step, we converted all words into lowercase to ensure consistency and eliminate any potential case-sensitivity issues. We removed the punctuations and special characters in order to remove noise and irrelevant information that may hinder the accuracy of the model. Lastly, we filtered low-frequency words to remove infrequently occurring words that may not add much value to the analysis. The details of the dataset after data preprocessing and cleansing are as given on Table 1.

Table -1: Parameters of the ***Alice_in_the_wonderland*** dataset before dataset cleansing

Dataset Name	Alice_in_the_wonderland
Number of Blank lines	3583
Number of Sentences	1629
Number of Words	34377
Total Number of Characters	148309
Total Number of Unique Words	25717
Total Patterns	148209

4.2 LSTM Model Training

As for the specifics of our framework, it is built using Python programming language. We have chosen Tensorflow and Keras as our primary machine learning libraries due to their ease of use, scalability, and wide range of pre-built functions for NLP tasks. Additionally, we have implemented various techniques for model optimization, such as early stopping, learning rate scheduling, and batch normalization, to improve the accuracy and efficiency of our model. Overall, our framework is well-suited for handling large-scale NLP tasks with high accuracy and speed. Our model consists of LSTM network with an input layer with 256 dimensions, two hidden layers containing 256 LSTM units each as shown in figure 10, and an output layer used to map the given words into a vector space.

```
2023-12-05 15:37:52.786684: I tensorflow/core/platform/cpu_feature_guard.cc
To enable the following instructions: SSE SSE2 SSE3 SSE4.1 SSE4.2 AVX AVX2 F
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 50, 50)            132150

 lstm (LSTM)                 (None, 50, 256)           314368

 lstm_1 (LSTM)               (None, 256)               525312

 dense (Dense)               (None, 256)               65792

 dense_1 (Dense)             (None, 2643)              679251

=================================================================
Total params: 1716873 (6.55 MB)
Trainable params: 1716873 (6.55 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
None
Epoch 1/100
```

Fig. 4: Configuration of the Proposed LSTM-based Model

During the training process, we used the dropout regularization technique to prevent overfitting by randomly subsetting or "dropping out" neurons in the LSTM network. For activation functions, we adopted the $softmax$ activation to get the probabilities for the next words given the previous words. For optimization, we used $Adam$. We set the learning rate to 0.001, batch size to 128 and used a dropout rate of 0.2. We run the model for 100 epochs and achieved a training accuracy of 86.66% on the training data. Figure 11 shows the first 10 epochs of 100 epochs.

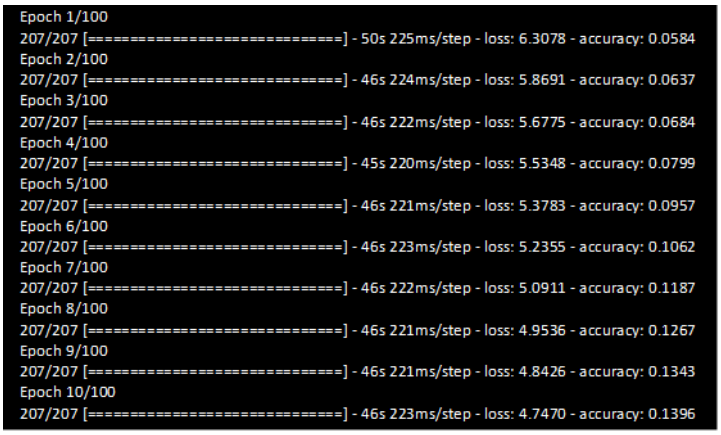

Fig. 4: Training of the Proposed LSTM-based Model

4.3 Imperceptibility and Payload Capacity Analysis

In this section, we analyzed the imperceptibility of information hidden in the automatically generated cover text by our LSTM based steganographic model. This allows us to measure the ability of our system to hide data within cover text without any changes being visible or detectable. We pay much attention to this evaluation parameter because it determines how difficult it would be for someone to detect that data has been hidden in cover text. To test for the imperceptibility of our model, we used a metric called *perplexity*, a standard metric for sentence quality testing. Perplexity can be thought of as a measure of uncertainty: higher perplexities indicate more uncertainty in predicting future outcomes from the same data set. The lower the perplexity, then the better our model's predictions are on average. We can compute perplexity as the average per-word log-probability on test texts as:

$$\begin{aligned} \text{perplexity} &= 2^{-\frac{1}{n}logP(s)} \\ &= 2^{-\frac{1}{n}logP(w_1,w_2,w_3,\ldots,w_n)} \\ &= 2^{-\frac{1}{n}logP(w_i|w_1,w_2,w_3,\ldots,w_{i-1})}, \end{aligned} \quad (8)$$

where $s = \{w_1, w_2, w_3, \ldots, w_n\}$ is the generated sentence, $p(s)$ denotes probability distribution over words in sentence, s and n is the number of words in s. A simple comparison between Equations 8 and 3 specifically reveals that $perplexity$ measures the complexity of a language model's predictions for a given text, by comparing their statistical distribution to that of the training data. A lower perplexity indicates a better fit between the model and the training data. We compared the performance of our model to two text steganography models. The first adopted Markov Chain method to generate automatic text and Huffman Coding technique to embed secret message in the generated cover text [21] and the second adopted LSTM algorithm both in automatic text generation and information hiding. As the embedding rate in language generation models rises, we notice a corresponding trend of higher perplexity. As the number of bits embedded in each word increases in a steganographic algorithm, the selected output word becomes more heavily influenced by the embedded bits in each iterative process. This makes it increasingly challenging to choose words that accurately reflect the statistical distribution of the training text, as the embedded bits can significantly alter the frequency and distribution of words in the output message. As a result, the steganographic algorithm may become less effective at concealing the hidden message within the cover text.

4.4 Results and Discussion

in this paper, we discussed the use of LSTM neural network to build a language model and use it to automatically generate cover texts for linguistic steganography. For hiding the secret information in the cover text, we adopted Huffman coding method. To evaluate the performance of our model, we evaluated performance of the model in terms of two metrics namely, imperceptibility and payload capacity. We measured the imperceptibility of the setgotext by calculating the perplexity of the stegotext and the payload capacity by analyzing how much information can be embedded in the generated texts and compared it with some other text steganography algorithms. Compared to other text steganography algorithms, our method demonstrated comparable or superior results in terms of payload capacity while maintaining a high level of imperceptibility. As pointed out in step 5 of the procedure for cover text generation in section 3.2, the primary goal of the probability distribution layer is to maximize the values of the hidden layer and minimize those of the input layer. By achieving this objective, we can successfully enhance the accuracy of our model because the hidden layer comprises the critical features in the data that help in making predictions. In contrast, the input layer consists of raw data that, if too large, can cause the model to overfit. By optimizing the probability distribution layer, we obtained a more balanced and accurate model that can make better predictions. Our approach involves incorporating a regularization technique known as dropout, which randomly drops out some nodes during training, preventing the model from relying too heavily on any one node. Through our experiments, we anticipate that the perceived relevance, accuracy, and coherence of the automatically generated cover text by our model will be on par with those produced by the state-of-the-art approaches proposed by [21] and [22]. Our goal is to create cover text that are both informative and engaging, while maintaining a high level of linguistic and semantic correctness. We believe that our proposed methodology, which combines advanced natural language generation techniques with domain-specific knowledge, will enable us to achieve this objective.

5. CONCLUSION AND FUTURE WORK

The automatic cover text based linguistic steganography model developed in this paper utilizes LSTM neural network and adopted Huffman coding technique to hide secret information. This holds great potential but also presents significant challenges. The high coding degree and low redundancy of text make it difficult to embed secret information without altering the natural flow and meaning of the text. However, by generating fluent text carriers specifically tailored to the secret information, this method offers a promising solution to this longstanding problem in steganography. As earlier stated, a higher embedding rate allows for increased payload capacity is desirable, but it may lead to detectable changes in the cover text's statistical properties. As a future work, we seek to finding a balance between capacity and stealthiness. We aim to strike a delicate balance between two crucial factors: capacity and stealthiness. While increasing capacity is essential for meeting the growing demands of data storage and transmission, it also raises concerns about signal vulnerability and interception. Therefore, our future work will focus on developing innovative techniques that maximize capacity without compromising stealthiness, ensuring secure and undetectable communication in various applications, from wireless networks to satellite systems. By achieving this balance, we can significantly advance the field of communication technology and provide more reliable and secure solutions for various industries and applications.

Declaration of Competing Interest

The authors hereby declare that for the paper entitled: Automated Covertext-based Linguistic Steganographic Model using LSTM and Huffman Coding, there is no conflict of interest

REFERENCES

[1] V. Kumar, S. Sharma, C Kumar, & A. K. Sahu. Latest Trends in Deep Learning Techniques for Image Steganography. International Journal of Digital Crime and Forensics (IJDCF) 2023, 15(1), 1-14.

[2] R. Goyal, P. Kumar & V. P. Singh. A Systematic survey on automated text generation tools and techniques: application, evaluation, and challenges. Multimed Tools App, 2023l.

[3] Liu, Q. Yang, J. Jiang, H. Wu, J. Peng, T. Wang, T. Wang, G., "When Deep Learning Meets Steganography: Protecting Inference Privacy in the Dark," IEEE INFOCOM 2022 - IEEE Conference on Computer Communications, London, United Kingdom, 2022, pp. 590-599.

[4] H. Dridi & K. Ouni. Towards Robust Combined Deep Architecture for Speech Recognition: Experiments on TIMIT. International Journal of Advanced Computer Science and Applications, 11, 2020.

[5] Azzouni, A., & Pujolle, G. A Long Short-Term Memory Recurrent Neural Network Framework for Network Traffic Matrix Prediction. *ArXiv, abs/1705.05690*, 2017.

[6] M. A. Majeed, R. Sulaiman, Z. Shukur, & M. K. Hasan. A Review on Text Steganography Techniques. Mathematics, 9(21), 2021.

[7] R. Gurunath, A. H. Alahmadi, D. Samanta, M. Z. Khan, and A. Alahmadi: "A Novel Approach for Linguistic Steganography Evaluation Based on Artificial Neural Networks," in IEEE Access, vol. 9, pp. 120869-120879, 2021, doi: 10.1109/ACCESS.2021.3108183.

[8] M. A. Ziang, R. Sulaiman, Z. Shukur, & M. K. Hasan. A Review on Text Steganography Techniques. Mathematics, 9(21), 2018.

[9] R. Huang, A. H. Alahmadi, D. Samanta, M. Z. Khan, and A. Alahmadi: "A Novel Approach for Linguistic Steganography Evaluation Based on Artificial Neural Networks," in IEEE Access, vol. 9, pp. 120869-120879, 2011, doi: 10.1109/ACCESS.2021.3108183.

[10] J. Wang, C. Yang, P. Wang, X. Song, J. Lu. Payload location for JPEG image steganography based on co-frequency sub-image filtering. Int. J. Distrib. Sens. Netw. 2020, 16, 1550147719899569.

[11] M. M. Sadek, A. Khalifa, M.G.M. Mostafa, Video steganography: A comprehensive review Multimed Tools Appl. 2015, 74, 7063–7094.

[12] Z. Yang, S. Jin, Y. Huang, Y. Zhang, H. Li. Automatically generate steganographic text based on Markov model and Huffman coding. arXiv, arXiv:1811.04720, 2018.

[13] Z. L. Yang, S. Y. Zhang, Y. T. Hu, Z. W. Hu, and Y. F. Huang, "VAE-Stega: Linguistic Steganography Based on Variational Auto-Encoder," in IEEE Transactions on Information Forensics and Security, vol. 16, pp. 880-895, 2021, doi: 10.1109/TIFS.2020.3023279.

[14] B. Yang, W. Peng, Y. Xue, P. Zhong. A Generation-based Text Steganography by Maintaining Consistency of Probability Distribution. KSII Transactions on Internet and Information Systems. Korean Society for Internet Information, 2021.

[15] L. Y. Xiang, L. Shuanghui, L. YuhangQian and C. Zhu. Novel Linguistic Steganography Based on Character-Level Text Generation. MDPI Journal of Mathematics 2020, 8, 1558; doi:10.3390/math8091558.

[16] D. Huang, W. Luo, M. Liu, W. Tang, and J. Huang. "Steganography Embedding Cost Learning with Generative Multi-Adversarial Network," in IEEE Transactions on Information Forensics and Security, 202. doi: 10.1109/TIFS.3318939.

[17] V. Kumar, S. Laddha, Aniket, N. Dogra. Steganography techniques using convolutional neural networks. Review of Computer Engineering Studies, Vol. 7, No. 3, pp. 66-73, 2020.. https://doi.org/10.18280/rces.070304.

[18] X. Zhou, W. Peng, B. Yang, J. Wen, Y. Xue, P. Zhong. "Linguistic Steganography Based on Adaptive Probability Distribution" in IEEE Transactions on Dependable and Secure Computing, vol. 19, no. 05, pp. 2982-2997, 2022. doi: 10.1109/TDSC.2021.3079957.

[19] X. Zheng, and H. Wu. "Autoregressive Linguistic Steganography Based on BERT and Consistency Coding", Security and Communication Networks, vol. 2022, Article ID 9092785, 11 pages, 2022. https://doi.org/10.1155/2022/9092785.

[20] H. V. K. S. Buddana, S. S. Kaushik, P. V. S. Manogna, and P. S. Shijin Kumar. "Word Level LSTM and Recurrent Neural Network for Automatic Text Generation," 2021 International Conference on Computer Communication and Informatics (ICCCI), Coimbatore, India, 2021, pp. 1-4, doi: 10.1109/ICCCI50826.2021.9402488.

[21] Z. Yang, S. Jin, Y. Huang, Y. Zhang, H. Li. Automatically generate steganographic text based on Markov model and huffman coding. 2018. arXiv, arXiv:1811.04720.

[22] Tina Fang, Martin Jaggi, and Katerina Argyraki. (2017). Generating Steganographic Text with LSTMs. In Proceedings of ACL 2017, Student Research Workshop, pages 100–106, Vancouver, Canada. Association for Computational Linguistics.

BIOGRAPHIES

Joshua J. Tom holds a Ph.D. in Computer Science from the Federal University of Technology, Akure, Nigeria, obtained in 2018. His research interests are Information and Cyber Security.

Dr. Bukola A. Onyekwelu holds a Ph.D. in Computer Science from the Federal University of Technology, Akure, Nigeria in 2015. She is a lecturer in the Department of Mathematics and Computer Science, Faculty of Basic and Applied Sciences of Elizade University, Ilara-Mokin, Nigeria.

Adigwe Wilfred is a PhD holder in Computer Science he specialized in Data Communication. He has been a lecturer for eighteen years. He is presently SIWES Director of Delta State University of Science and Technology.

Dr. Folasade Aliu holds a Ph.D. in Computer Science from the Federal University of Technology, Akure, Nigeria. With a rich background in Information Security, her research has consistently pushed the boundaries of knowledge in the field.

Dr. Gabriel N. Odachi holds a PhD in Computer and is a member of various professional bodies including NCS, CPN, etc. Lecturer in Cyber Security and the Dean, Faculty of Science, Admiralty University of Nigeria, Ibusa, Delta State.