Can You Spot the Hidden Message? Enhancing Mobile Privacy with Image Steganography

CRISTIAN HAIDAU, University of Twente, The Netherlands

As technology has rapidly advanced over the last decades, increasing threats have endangered people's privacy. While methods for hiding message content are widely used, techniques for concealing metadata such as who is communicating, when, or how have often been overlooked. One approach that aims to take a step in that direction is Image Steganography, the practice of hiding secret information within images. The research conducted by Dodeja (2022) has led to the development of "BaatCheet", a mobile messaging application that allows users to send secret messages embedded in pictures. This allows users to hide the message content and obscures the act of communication itself. The author uses a technique called LSB embedding to modify the pixels of the cover image. While basic LSB substitution is computationally efficient and easy to implement, it is also the most vulnerable to image steganalysis. More advanced variants like LSB matching provide improved resistance by introducing less noticeable modifications to the cover image. This research aims to explore steganographic embedding techniques such as LSB matching, edge-adaptive embedding, and matrix embedding. First, the research will experiment with LSB matching alone, then pair it with matrix embedding and edge-based embedding separately and finally combine all three. Each of these configurations will represent a distinct embedding system. Every configuration will be evaluated based on embedding and extraction time, extraction reliability, embedding capacity, and robustness against steganalysis. The paper aims to develop an embedding system suitable for use in a mobile messaging application. This research is particularly useful for developers and researchers working on secure image steganographic communication systems, especially in mobile environments where performance and privacy are critical.

Additional Key Words and Phrases: Image Steganography, LSB, LSB matching, Matrix Embedding, Edge Based steganography, steganalysis, mobile application

1 INTRODUCTION

The necessity of protecting personal information has grown significantly in recent years. Messaging applications have become essential tools for staying connected and are trusted to keep conversations secure through built-in encryption features. Platforms like WhatsApp, Telegram, and Facebook Messenger use end-to-end encryption (E2EE) [26] protocols to keep the content of conversations secure. However, they often fail to protect metadata such as who is communicating, when messages are sent, and how frequently communication occurs. Protecting metadata [9] is essential because it can reveal sensitive information about users' relationships, behaviours, and activities even without access to the actual message content. To address these privacy concerns Image Steganography [22] can be used. Image Steganography is the technique of hiding information within digital images. By embedding the message within an image, the act of communication itself becomes hidden. Transmitting images still generates metadata, however, steganography can obscure the hidden content, making it less obvious that a secret message is being sent. Image Steganography can be categorized into two main types: spatial domain steganography, which hides data by directly altering pixel values, and transform domain steganography, which embeds information by modifying frequency coefficients [22]. Although transform domain steganography offers higher resistance against image distortions, it comes at the cost of higher computational complexity [15]. Spatial domain steganography is more suitable for messaging platforms due to its computational simplicity and increased embedding capacity [15]. In 2022 Dodeja [8] conducted a research in which he developed "BaatCheet", a mobile messaging application that allows users to communicate by embedding messages in images. The app combines end-to-end encryption (E2EE) with image steganography by first encrypting the message and then embedding the encrypted data.

To achieve this, Dodeja [8] has utilized one of the simplest and most used steganographic embedding methods, the Least Significant Bit (LSB) [11] embedding technique to embed messages into the spatial domain of images. This involves replacing the least significant bits of an image with the bits of a secret message. These alterations are invisible to the human eye and introduce no significant impact on the image's appearance. However, basic LSB embedding approaches such as the one used in Dodeja's [8] work can compromise the secrecy of the hidden message by introducing statistical anomalies. These anomalies can later be detected through image steganalysis. Image steganalysis is the process of detecting hidden data within images. In the context of Dodeja's [8] application, steganalysis techniques can be used to reveal the presence of hidden messages in images. The paper "Classification of Steganalysis Techniques: A Study" by Nissar and Mir [25] categorizes steganalysis into two main types: signature steganalysis and statistical steganalysis. Signature steganalysis detects hidden data by examining certain patterns left by known steganographic tools. Statistical steganalysis, on the other hand, finds embedded information by looking for unusual patterns in the cover image's statistical properties. This study focuses on statistical steganalysis because it does not require prior knowledge of the tool used for embedding. This makes it more suitable for testing general LSB-based methods. The statistical tests adopted in this paper are RS analysis, Weighted Stego Analysis, Triples detection, and Sample Pair Analysis. These techniques are described in more detail in Section 7.1. In addition, the study also includes visual inspection of histograms and bit planes. Histogram analysis [18] can reveal patterns in the pixel value distributions. This can indicate a hidden payload. Furthermore, structural changes can also be observed in the bit planes of the stego images. This is done through bit plane analysis [30]. See section 7.1.1 for a detailed description of the image steganalysis techniques.

TScIT 43, June 29, 2025, Enschede, The Netherlands

To try to preserve the statistical properties of an image during embedding, the LSB matching (LSBM) algorithm was developed [23]. LSBM is a variation of LSB embedding that, instead of directly flipping bits, changes the pixel value by randomly adding or subtracting 1. This makes the changes less noticeable to statistical analysis. As a result, it offers improved security compared to basic LSB approaches such as the one used by Dodeja. However, LSBM still remains vulnerable to advanced steganalysis techniques, particularly those based on modern machine learning methods. Furthermore, LSBM can be paired with matrix embedding. Matrix embedding is a general coding technique that can be used in image steganography to reduce the number of changes made to an image during embedding [14]. Reducing modifications to the cover image makes it less prone to detection. As a final layer of security, edge-based embedding can be used. Instead of selecting random pixels to hide the secret data, Edge-based LSB techniques embed data in areas of an image that have high variance, such as edges or textured regions [3]. This method makes changes to the image less noticeable both visually and statistically. While more advanced edge-detection algorithms like Canny, exist, the Sobel operator is adopted in this implementation due to its computational simplicity. The algorithm is adaptive, meaning that it selects more pixels as edges based on the secret message length (see Section 5.3 for a detailed description).

The previously discussed steganographic techniques, namely LSBM, edge-based embedding, and matrix-embedding, each play a distinct role in different stages of the embedding process. Edge detection selects the regions used for embedding. Matrix-embedding decides how to embed the message bits with the minimum number of possible changes and LSBM applies these changes. As a result, these algorithms can be overlapped to achieve various trade-offs between execution time, embedding capacity, and security. This work explores all possible combinations of these algorithms and compares the advantages and drawbacks of each configuration. Each unique combination is also referred to in this work as an embedding system. The goal of this research is to provide a secure embedding system that can be used in a mobile messaging app. The proposed systems are evaluated based on their embedding and extraction times, embedding capacity, extraction reliability, and robustness against statistical steganalysis.

To support this research, a mobile application called "BaatCheet Embedder" was developed. The application has features for both message extraction and embedding. It should be noted that, due to time constraints, message extraction was only implemented for two systems, namely LSBM and LSBM coupled with Matrix embedding. The application operated as a testing and implementation environment for the various embedding systems. Its main objective was to evaluate each system's performance in terms of message extraction and embedding times in realistic mobile scenarios. Given that the aim of this work is the development of a secure embedding configurations, communication functionalities will not be incorporated into the mobile application because they are outside the scope of this study.

This research is novel due to its systematic exploration of LSB-based embedding techniques specifically designed for mobile environments. While individual methods like LSBM, matrix embedding, and edge-based embedding have been studied separately, this work evaluates their combined performance in multiple configurations, focusing on practical trade-offs between speed, capacity, and security. By implementing and testing these techniques within a mobile application, the study addresses a gap in current literature that often overlooks mobile-specific constraints for real-world deployment.

2 RESEARCH QUESTIONS

(1) How can different image steganography embedding techniques such as LSB Matching, Matrix Embedding, and Edge-based embedding be combined to reduce the detectability of steganographic images by statistical steganalysis?
(2) What is the robustness of the embedding systems explored in RQ1 against statistical and visual image steganalysis techniques?
(3) Which embedding system achieves the most favorable balance between time performance, extraction reliability, embedding capacity, and resistance to image steganalysis for use in a mobile messaging application?

3 FUNCTIONAL REQUIREMENTS

This work aims to assess which embedding configuration is most suitable for use in a steganographic mobile messaging application. To be considered suitable for deployment, an embedding system must meet the requirements described below. The first and most important requirement is robustness against image steganalysis. To ensure the secrecy of the hidden message, the embedding scheme must be resistant to statistical and visual steganalysis techniques. Consequently, the modifications introduced during embedding should not significantly alter the statistical properties of the image. According to Cachin [6], in a secure embedding scheme no statistical test can distinguish between cover and stego objects better than random guessing. Therefore, to consider the embedding schemes employed in this paper secure, the detection rate on stego images should approach the false positive rate. Furthermore, the embedding configuration must be computationally efficient. Miller [24] indicates that users are most comfortable when computer response times are below two seconds. Based on this, in a mobile environment, the embedding process and the extraction process should be completed within two seconds.

Additionally, the embedding system should embed messages of average conversational length. According to [28], only 3.49 percent of WhatsApp messages contain 20 or more words. Although most messages are shorter, supporting at least 30 words ensures coverage of both average and longer-than-average messages. [5] shows that the average word length in English is below five characters. Since ASCII uses one byte per character, 30 words of five characters each translate to 150 bytes of payload. Thus, the embedding capacity of each stego image should be no less than 150 bytes. The configurations that use Matrix embedding can only embed 3 secret bits per 7 cover bits. To embed 150 bytes of data in an image the minimum number of pixels needed is:

$$150 \times 8 = 1200 \text{ bits} \tag{1}$$

$$\frac{1200 \text{ bits}}{3 \text{ bits/group}} = 400 \text{ groups} \tag{2}$$

$$400\,\text{groups} \times 7\,\text{pixels/group} = 2800\,\text{pixels} \tag{3}$$

To ensure such a capacity, embedding configurations that rely on Matrix embedding are constrained to using images with a number of pixels greater than or equal to 2800 ($\approx 53 \times 53$ pixels). In contrast, the other configurations are able to embed 1 secret bit per pixel, thus the minimum number of pixels needed to embed 150 bytes of data is:

$$1200\,\text{bits} \times \frac{1\,\text{pixel}}{1\,\text{bit}} = 1200\,\text{pixels} \tag{4}$$

This translates to a minimum resolution of $\approx 35 \times 35$ pixels.

The final requirement addresses extraction reliability. The embedding systems are aimed for use in a dedicated steganographic app, where lossless communication and uncompressed image formats are expected. As a result, lossless extraction is required when performed in a controlled environment. While the reliability will also be evaluated under image compression and resizing, performance under these conditions will not be considered a formal requirement.

4 RESEARCH METHODOLOGY

To support the development of this research, a literature review was conducted. The review was divided into two main areas: embedding techniques, and image steganalysis methods. The first area was mainly focused on spatial domain steganography and LSB based embedding techniques. The second area delved into statistical steganalysis techniques. In addition, the review was also focused on studies related to the use of image steganography in mobile messaging applications. Keywords such as "image steganography", "LSB matching", "matrix embedding", "edge-based embedding" and "image steganalysis" were used to identify relevant studies.

To address RQ1, an Android-based mobile application named "BaatCheet Embedder" was developed. This app served as a platform for implementing the embedding algorithms and combining the embedding techniques into different configurations. The application was built using Android Studio [4], an integrated development environment (IDE) specifically designed for Android app development. The Java [1] code used to implement each embedding system was also utilized independently to generate steganographic images outside the Android Studio environment.

Moving to RQ2, a collection of 500 RGB PNG images was gathered to serve as cover images for embedding. RGB images were chosen instead of grayscale because they are more typical in the context of messaging platforms. Additionally, in color images, the visual impact of LSB changes is distributed across three channels compared to a single channel in the case of grayscale images. The images were sourced from DIV2K, a known image dataset [2]. Furthermore, all images were resized to a resolution of 600×600 pixels to reduce computational complexity and standardize input dimensions across all tests. The messages used for embedding were generated using Java's `Random` class [27] without specifying a fixed seed. As a result, the message content varied across runs. The length of each random message was calculated to match a given embedding rate. The embedding rate refers to the proportion of the cover image used to embed hidden data. For instance, a 10% embedding rate means that hidden data is embedded in 10% of the total pixels in the image.

Finally, each embedding system was run on the full set of 500 cover images at three different embedding rates: 10%, 20%, and 30%. For example, System 1 was used to embed payloads at 10% embedding rate into all 500 images, resulting in one set of 500 stego images. Then, System 1 was used again to embed data at a 20% and 30% embedding rates. This produced another two sets of 500 stego images. The same process was repeated for Systems 2, 3, and 4. This entire process resulted in a total of 12 distinct stego datasets. The 10%, 20%, and 30% embedding rates were chosen because they offer sufficient capacity even for low-resolution images. Since the minimum number of pixels required to embed 150 bytes of data (i.e. the requirement set in Section 3) at a 10% embedding rate is 12,000 (equivalent to a 120×100 pixel image), even the lowest embedding rate used in this paper offers sufficient capacity for the vast majority of images. Higher embedding rates were avoided because they are less likely to occur in the context of mobile messaging application. The resulting sets of stego images were tested against statistical steganalysis and visual steganalysis. The statistical tests consisted of RS analysis, Sample Pair Analysis, Weighted Stego, and Triples detection and were conducted using the Aleheia tool [21]. The visual attacks consisted of histogram analysis and bit plane analysis. A custom Python script was developed to generate the necessary histograms and bit planes.

To answer RQ3, the embedding execution time of each hybrid system was measured using the "BaatCheet Embedder" app. The execution time refers to the time elapsed from the start of the embedding process to the end of the embedding process. This excludes the time required to input a message or select an image, and focuses solely on the execution time of the embedding algorithm. All measurements were taken while the application was running on a mobile device. This was done in order to mimic mobile computing resources. The message extraction time was also measured. These measurements were conducted in the same manner as the measurements for the execution time of the embedding process. However, due to time constraints surrounding this research, the extraction functionality was only implemented and tested for two embedding systems, namely LSBM and LSBM combined with Matrix-embedding. Furthermore, the reliability of the extraction process was measured. This was done by resizing and cropping the stego images to simulate lossy conditions. Two sets of 500 images each of 600x600 pixels were created. The first set was resized to 595x595 pixels, and the second set was compressed to JPEG at 90% image quality. The resizing and cropping values were chosen to mimic minimal tampering. Furthermore, if the extraction remained successful, more aggressive values were planned on being used. The reliability was assessed by counting the number of successful extractions out of the two sets of tampered images.

Finally, by analyzing the embedding and extraction time performance, extraction reliability, embedding capacity, and robustness against statistical analysis (RQ2), this research assessed which embedding system provides the most suitable trade-off for use in a mobile messaging application. A methodology diagram can be seen in Figure 5.

5 ALGORITHMS INVOLVED

This section examines the algorithms used in the development of each embedding system. It offers a detailed explanation of the internal processes of each algorithm.

5.1 Least Significant Bit Matching (LSBM)

LSBM is a steganographic embedding technique that tries to improve upon the classic LSB replacement method [23]. In the basic LSB technique, if the cover image bit differs from the secret message bit, the cover bit is flipped to match the secret bit. This bit-flipping method can cause disruptions in the statistical properties of the least significant bits in the image. These changes make it easier for statistical steganalysis methods to detect hidden data. LSB Matching aims to improve on this by randomly increasing or decreasing the pixel value by one when a change is needed [23]. This behavior increases its resistance against statistical steganalysis by avoiding predictable patterns. While basic LSB flips only the least significant bit, LSBM modifies the entire pixel value. This can result in changes to multiple bit positions. Since pixel values range from 0 to 255, LSBM subtracts one when the value is 255 and adds one when the value is 0. This is done to ensure that pixel values remain within the valid range.

5.2 Matrix Embedding

Matrix embedding is a steganographic technique that uses linear error-correcting codes to embed message bits. This approach allows for fewer changes to the cover image during embedding [13]. In this approach, a (7,4) Hamming code is used to embed 3 message bits into 7 bits of the cover image. During the embedding process, a parity-check matrix is used to compute the syndrome of the difference between the cover bits and the message bits. Based on this syndrome, the algorithm determines which single bit, if any, needs to be flipped to ensure the resulting block embeds the secret message. By doing so, only one bit per 7-bit block is modified. During extraction, the same parity-check matrix is applied to each stego block to compute the syndrome, which directly yields the embedded message. This method achieves an average modification per secret bit of 0.125, compared to 0.5 when using basic LSB embedding. Since Matrix embedding uses a (7,4) hamming code to embed 3 secret bits in 7 cover bits, the relative embedding capacity is

$$\text{Capacity} = \frac{3 \text{ secret bits}}{7 \text{ cover bits}} = \frac{3}{7} \approx 0.4285 \, (42.85\%) \tag{5}$$

Therefore, the maximum number of bits that can be embedded using this method is equal to 42.85% out of the total number of pixels in the image.

5.3 Edge-detection

Edge-based embedding is a steganographic embedding technique that uses the edges in an image to hide secret information. Since edges naturally contain higher levels of noise and variation, modifications made in these regions are statistically harder to detect, making the embedding technique more secure against image steganalysis. Edge regions are identified using the Sobel operator [29] applied to the grayscale version of the image. This involves two 3×3 convolution kernels that detect horizontal and vertical intensity changes. Pixels with gradient magnitudes above a set threshold are classified as edges and selected for embedding. The implementation adopted in this work uses an adaptive threshold. After one iteration of the algorithm, if the number of pixels classified as edges is lower than the number of bits of the secret message, the threshold is decreased, and the algorithm runs again. This process continues until the algorithm reaches a threshold where the number of pixels classified as edges is equal to or greater than the number of bits in the secret message. This approach increases embedding capacity by adaptively selecting more edge regions. On the other hand, it can reduce security at higher payloads. This is because the inclusion of smoother areas makes the embedded data more prone to detection. To extract the hidden message, the receiver must be provided with the list of edges that were used for embedding.

6 PROPOSED SOLUTION

The algorithms discussed previously were used to develop four steganographic embedding systems, each aiming to achieve a different level of complexity, security, and embedding capacity. The four embedding systems are described below. Their performance in terms of time efficiency, security, extraction reliability, and embedding capacity will be evaluated in a later section. An activity diagram for each system is provided at the end of its corresponding section.

6.1 System 1: Sequential LSB Matching (LSBM)

In order to hide data, this system only makes use of the LSBM algorithm. The algorithm is applied sequentially to every pixel in the image, processing them in the order in which they appear from left to right, top to bottom until all secret bits are embedded. Each pixel in a color image has three channels available for embedding: red, green, and blue (RGB). According to [16], the blue channel tends to be less perceptually sensitive and introduces a marginally lower distortion based on PSNR and MSE. Thus, each system designed in this paper will embed in the blue channel. See Figure 1.

6.2 System 2: Sequential LSB Matching + Matrix Embeddingg (LSBM + ME)

This system makes use of both the LSBM and matrix-embedding algorithms. First, the cover image is divided into blocks of 7 pixels, sequentially, as described in system 1. From each block, the least significant bits of the blue channel of each pixel are extracted. The parity-check matrix is then applied to the LSBs to compute the syndrome difference, which determines which bit, if any, needs to be flipped. The bit to be flipped is then set to the correct value by applying the LSBM algorithm. See Figure 2.

6.3 System 3: LSB Matching + Edge-Based Embedding (LSBM + EB)

This system uses the edge regions of the image to embed the secret bits. First, the edge-detection algorithm is used to extract a list of pixels from the edge regions. The edge pixels are selected adaptively based on a threshold as described in Section 5.3. A lower threshold results in more pixels being labeled as edges. The list of edge pixels is then traversed, and LSBM is applied to each pixel to set the LSB of

Table 1. Encoding scheme for identifying the embedding system

Header Bits	Embedding System
00	LSBM
01	LSBM + ME
10	LSBM + EB
11	LSBM + EB + ME

the blue channel to match the corresponding secret bit. After this is done, the resulting list of stego pixels is written back into the image, replacing the original pixel values in their respective positions to produce the final stego image. See Figure 3.

6.4 System 4: LSB Matching + Edge-Based Embedding + Matrix Embedding (LSBM + EB + ME)

This approach makes use of all three steganographic algorithms. Similarly to system 3, the edge detection algorithm is run on the cover image to obtain the list of edge pixels. The edge-detection threshold is lowered until the length of the edge pixels list is sufficient to embed the secret message bits. Considering that matrix embedding is used, this means a minimum number of pixels of $n \times \frac{7}{3}$ where n is the length of the secret message in bits. This list is then divided into blocks of 7 pixels and fed to the matrix embedding algorithm. For each block, the matrix embedding algorithm outputs the bit that must be flipped. LSBM is then applied to the corresponding pixel to flip the LSB value of its blue channel. The resulting list of stego pixels is then written to the image. See Figure 4.

6.5 Message Extraction

In order to provide the necessary information for message extraction a 32-bit header is embedded in the beginning of each stego image. The first 2 bits of the header represent the embedding system used for hiding the message. The bits are set according to Table 1. The last 30 bits represent the length of the hidden message in bytes. The header is then encrypted with the AES-128 [7] encryption algorithm using a private key. The encrypted header is then embedded in the image using System 1. The receiver must be able to extract the header without prior information about the algorithm used for embedding. Thus, the embedding algorithm used to embed the header must remain constant. System 1 was chosen to embed the header to minimize execution time and reduce the number of pixels required for embedding. To begin the extraction process, the header is retrieved and decrypted using the same private key. By interpreting the bits of the header, the receiver can identify the length of the hidden message and the algorithm used for embedding. This information allows the receiver to reverse the embedding process and retrieve the secret message.

6.6 BaatCheet Embedder

The implementation of each embedding system has led to the development of the "BaatCheet Embedder" app. "BaatCheet Embedeer" is a mobile application that offers two main functionalities: message embedding and message extraction. See Snapshots 15,16,17,18,19 and 20.

6.6.1 Embedding. To embed a message the user is first required to select an image by pressing the "Select Image" button. The image selected must be a PNG image. The next step is message input. To input a message, the user must press the "Type message" button, which opens a dialog box for entering the message. Once the image is inserted the embedding method has to be selected. By pressing the "Embed" button another dialog box pops up where the user can select one of the four embedding systems. After a system is selected the user must press the "Embed" button inside the dialog box and the message will be embedded. Once this is done, the "Download Image" button can be pressed to download the image to the gallery.

6.6.2 Extraction. In order to extract a message the user must press the "Extract Message" button. When pressed the photo gallery appears on the screen and the user must select a stego image from the gallery. After the image is selected, the "Extract" button must be pressed and the secret message will be displayed on the screen.

7 RESULTS AND DISCUSSION

7.1 Image Steganalysis

This section presents a comprehensive evaluation of the four proposed systems. It begins with statistical steganalysis tests, including RS analysis, Sample Pair Analysis (SPA), Weighted Stego (WS), and Triples detection, followed by visual analysis techniques such as histogram analysis and bit-plane examination. The analysis includes quantitative results for the conducted statistical attacks as well as qualitative insights from bit-plane and histogram analysis

7.1.1 Statistical Steganalysis using Aletheia. The tool Aletheia [21] was used to conduct the statistical tests on the stego datasets (see Section 4 for a detailed description of how the datasets were generated). Aletheia is a steganalysis framework that implements several well-known statistical attacks such as RS analysis, Sample Pair Analysis (SPA), Weighted Stego (WS), and Triples detection [10, 19, 20, 31].

RS analysis, introduced by Fridrich et al. [12], is a statistical method used to detect hidden data by measuring the smoothness of small groups of pixels. The smoothness is measured using a discriminant function. Based on this function, the groups of pixels are classified as Regular if flipping pixel bits makes them less smooth, or Singular if it makes them smoother. In natural images, there are usually more Regular groups than Singular groups. However, LSB replacement tends to disturb this balance. By observing the shift in balance, RS analysis is able to estimate the size of the secret payload. The output is a number between 0 and 1 where 0 indicates no embedded data and 1 means that every LSB in the image was altered.

Sample Pair Analysis (SPA) [10] detects LSB steganography in an image by examining the distribution of adjacent pixel pairs. It identifies specific types of pairs such as *(2i, 2i+1)*, *(2i+1,2i)*, *(2i,2i)* and *(2i+1,2i+1)* and counts how often they occur. LSB modifications tend to alter the natural distribution of such pixel values. By observing these changes, SPA can estimate the length of the hidden message. Similar to RS analysis, the output is a number between 0 and 1 representing the percentage of hidden data embedded in the image.

Weighted Stego Attack (WSA) [20] is a statistical attack that assigns different importance weights to different areas in the image.

Table 2. Number of false positives detected by each statistical steganalysis method on a set of 500 cover images.

Statistical Attack			
WS	Triples	SPA	RS
26	46	86	80

Table 3. Number of images classified as stego by the Weighted Stego attack, across different embedding systems and embedding rates.

Embedding System	Embedding Rate		
	10%	20%	30%
LSBM	49	51	51
LSBM + ME	49	54	56
LSBM + EB	26	28	27
LSBM + EB + ME	27	27	25

The idea is that LSB replacement affects smooth regions of the image in a more noticeable manner than textured or noisy regions. WSA analyzes the image by dividing it into regions and calculating statistical measures such as pixel differences or smoothness. This is done while applying higher weights in areas where embedding is more likely to disturb the natural characteristics of the image. The output is also an estimate of the embedding rate with a value between 0 and 1.

Triples detection [19] tries to detect LSB modifications by analyzing patterns in triplets of adjacent pixel values. Some triplets patterns are statistically unlikely in natural images. These triplets are categorized based on the differences between neighboring pixel values and the parity of the first pixel in the group. In unaltered images, there is typically a balance between certain types of triplets. For example, those beginning with even versus odd pixel values. By observing the imbalance in such patterns, Triples Detection can estimate the embedding rate. Similar to RS Analysis, WSA, and SPA, the output is a number between 0 and 1.

For all detection methods, Aletheia applies a classification threshold of 0.05. If an attack outputs a value greater than 0.05, the image is classified as stego.

Every statistical test was run on the 12 image sets to assess the number of images flagged as stego. The results for every statistical attack are shown in Tables 3, 4, 5, and 6. Each entry in the table shows the number of images classified as stego across different systems and embedding rates. Additionally, the number of false positives was assessed for each statistical steganalysis method. This was achieved by running Weighted Stego, Triples detection, RS analysis, and Sample Pair Analysis on the original set of 500 cover images used for embedding. Table 2 shows the number of false positives for each statistical attack on the 500 image set.

The results indicate an overall low detection rate across all steganalysis techniques, suggesting that each embedding system offers a relatively high level of resistance against statistical attacks. System 1 (LSBM) and system 2 (LSBM+ME) display a similar level of detectability. Compared to System 2 (LSBM+ME), System 1 shows marginally better resistance against Weighted Stego (WS), Sample Pair Analysis (SPA), and the Triples attack (Tables 3, 4 and 5). This

Table 4. Number of images classified as stego by Triples comparison, across different embedding systems and embedding rates.

Embedding System	Embedding Rate		
	10%	20%	30%
LSBM	67	74	79
LSBM + ME	69	80	88
LSBM + EB	46	47	47
LSBM + EB + ME	45	47	48

Table 5. Number of images classified as stego by Sample Pair Analysis, across different embedding systems and embedding rates.

Embedding System	Embedding Rate		
	10%	20%	30%
LSBM	99	106	103
LSBM + ME	101	107	109
LSBM + EB	87	84	84
LSBM + EB + ME	85	85	83

Table 6. Number of images classified as stego by RS analysis, across different embedding systems and embedding rates.

Embedding System	Embedding Rate		
	10%	20%	30%
LSBM	104	112	116
LSBM + ME	91	100	106
LSBM + EB	84	83	81
LSBM + EB + ME	80	83	82

translates to 3 to 9 fewer detections out of 500 images or under 2%. On the other hand, System 2 (LSBM+ME) is more robust under RS analysis (Table 6) with 10 to 13 fewer detections out of 500 images. This variation in results across systems 1 and 2 between RS analysis and the other three tests could be explained by how the two systems distribute changes across the image. Although the use of matrix embedding allows for fewer changes in the cover image, it spreads those changes over a larger region. This broader distribution may be more susceptible to certain statistical attacks compared to the more localized changes in System 1 (LSBM).

Furthermore, Systems 3 (EB+LSBM) and 4 (LSBM+EB+ME) demonstrate very similar performance. Both show a clear improvement in stealth over Systems 1 (LSBM) and 2 (LSBM+ME) across all detection techniques. In particular, the number of flagged images for these systems remains consistently close to the false positive threshold in every statistical attack (Table 2). For example, under Sample Pair Analysis (SPA) (Table 5), which had 86 false positives on clean images, Systems 3 and 4 reported only 84 and 83 detections respectively at a 30% embedding rate. Similarly, under RS analysis (Table 6), both systems stayed within just 1–3 detections of the 80-image false positive 2 count. This implies that systems 3 (LSBM+EB) and 4 (LSBM+EB+ME) are nearly indistinguishable from clean images when tested under WS, Triples, SPA, and RS analysis. The improvement is due to the edge-based embedding adopted by both systems. The results suggest that by embedding

data in high-variance, noisy regions such as edges, modifications become less detectable statistically.

7.1.2 Bit plane analysis. Bit plane analysis is a technique used in digital image processing where an image is decomposed into its individual bit planes [30]. In an RGB image, each pixel is represented by a 24-bit number, implying that each image has 24-bit planes. However, the embedding systems proposed in this paper embed information in the least significant bits of the blue channel. As a result, the bit plane representing the least significant bits of the blue channel will be examined. Four bit planes were analyzed, each corresponding to a steganogram generated using one of the four embedding systems at an embedding rate of 30%. The image used as the cover image can be seen in Figure 21

The bit plane shown in Figure 6 displays a pattern in the top portion of the image. This is in the form of repeating vertical lines. This pattern is easily observable and suggests structural changes to the least significant bits in the blue channel of the image. As a result, system 1 appears to be insecure against bit plane analysis. The bit planes shown in Figures 7, 8, and 9 display no visible patterns. Their distribution of LSBs resembles random noise. This indicates that systems 2,3 and 4 are secure against visual bit plane analysis.

7.1.3 Histogram analysis. Histogram analysis is a commonly used technique in image steganalysis to detect hidden data by examining irregularities or patterns in the distribution of pixel intensity values [18]. The histograms depicted in Figures 11, 12, 13, and 14 show the distribution of blue channel pixel intensities across four stego images, each generated using a different embedding system. Figure 10 shows the distribution of blue channel pixel intensities for a clean image.

Compared to the histogram of the clean image (Figure 10), the histograms corresponding to System 1 and System 3 (Figures 11 and 13) show sudden spikes in intensity levels between neighboring pixels. The histograms corresponding to systems 2 and 3 appear to be more smooth compared to the ones belonging to systems 1 and 3. This can indicate the presence of a hidden payload in the image. Furthermore, the histograms corresponding to System 2 and 4 (Figures 12 and 14) do not display any visible patterns. This suggests minimal or no hidden payload. These results indicate that systems that incorporate matrix embedding are more robust against histogram analysis. This is most likely due to the reduced number of changes made to the cover image.

7.2 Embedding Time Efficiency

To evaluate the time efficiency of the proposed systems the 'BaatCheet Embedder' mobile application was used. Each embedding system was applied in order to embed a payload into an image at three embedding rates 10%, 20%, and 30%. All tests were conducted using the same 600×600-pixel image (See Figure 21) to ensure consistency. The `measureTime` function [17] was used to record the time elapsed from the beginning to the end of the embedding process. According to JetBrains [17], elapsed time is calculated using `TimeSource.Monotonic`, the most accurate timing source offered by the platform. To maintain consistency, all tests were conducted using a single mobile device: the OnePlus 9 Pro. Future work could involve testing across a broader range of devices for wider applicability. The results are displayed in Table 7. Each entry in the table represents the time elapsed in milliseconds.

Table 7. The embedding execution time in ms for each system at different embedding rates

Embedding System	Embedding Rate		
	10%	20%	30%
LSBM	21	38	56
LSBM + ME	65	128	187
LSBM + EB	145	165	191
LSBM + EB + ME	185	248	245

Table 5 illustrates a consistent increase in embedding times for each system as the embedding rate increases. Furthermore, embedding time increases progressively from System 1 to System 4, with System 1 exhibiting the shortest times and System 4 the longest across all embedding rates. This is expected behavior, given that each system introduces additional computational complexity compared to its predecessor. The results satisfy the requirements set in Section 3, as none of the embedding systems required more than 2 seconds to run, even at the highest embedding rate (30%).

7.3 Embedding Capacity

System 1 has an embedding capacity equal to the total number of pixels in the image. For an image of size $N \times M$ the embedding capacity is equal to $N \times M$ bits. System 2 offers an embedding capacity of $\frac{3}{7} \cdot (N \times M)$, since matrix embedding uses 7 cover bits to embed 3 secret bits. As a result, only approximately 42.85% of the available cover bits can be used for embedding. Similar to System 1, System 3 also offers an embedding capacity of $N \times M$ bits. This is due to the use of the adaptive edge-detection algorithm. If the message size in bits is equal to the total number of pixels in the image, the edge-detection algorithm ends up selecting all the pixels in the image as edges. This is because the threshold is lowered until enough pixels are selected to fit the hidden message (see Section 5.3). Systems 4 also incorporates the edge-detection algorithm, however due to the use of the matrix embedding algorithm, its embedding capacity is reduced to 42.85% of the total number of pixels in the image. Given that a header is embedded in the image for systems 1 and 2, their embedding capacity is lowered by 128 bits (i.e. the size of the header).

7.4 Extraction Time Efficiency

The execution times of the extraction process for systems 1 and 2 were measured in the same manner as the execution time of the embedding process (See Section 7.2). For both systems 1 and 2, an increase in execution times across embedding rates can be observed. This is an expected behavior and is caused by the increase in payload. However, the execution times for both systems at any embedding rate are low and negligible. Based on these results, the execution time of the extraction process proves to be well within the 2 seconds requirement set in Section 3.

Table 8. The message extraction execution time in ms for each system at different embedding rates

Embedding System	Embedding Rate		
	10%	20%	30%
LSBM	14	26	34
LSBM + ME	41	81	122

7.5 Extraction Reliability

In an ideal scenario, where no image tampering and loss of data occurs, the extraction process is entirely deterministic. This implies that no data is lost during message extraction. In the case of a steganographic messaging application, lossless formats and careful handling of image data are required to preserve the integrity of the embedded information. However, when dealing with steganographic images that were shared outside a controlled application environment (i.e., via social media platforms or general-purpose messaging applications), images might undergo compression or resizing. This can alter pixel data and corrupt the embedded message. In the extraction implementation used in this paper (for Systems 1 and 2), any alteration to the region of the image containing the header renders the extraction of the secret message impossible. To measure the reliability of the extraction process the methodology described in Section 4 was adopted. The number of images where the message was successfully extracted was recorded. For both the compressed and resized image sets, message extraction failed for all 500 images. This was due to the decryption process failing to interpret the header, which had been corrupted by compression and resizing. This implies a 0% extraction success rate at minimal image tampering. As a result, the extraction process proves highly unreliable in uncontrolled environments where even minimal image tampering can occur. Given this result, testing with more aggressive compression and resizing levels is unlikely to provide any useful insights.

7.6 Overall Comparison

The four embedding systems evaluated in this study demonstrate different levels of performance in terms of time efficiency, extraction reliability, embedding capacity, and security. In a dedicated steganographic messaging application, where no image compression and resizing occur, system 2 emerges as the most suitable solution. This conclusion is based on a comparison limited to systems 1 and 2, because these are the only configurations for which the extraction functionality is implemented. Although both systems offer similar performance under statistical steganalysis, system 2 is more secure under histogram analysis and bit plane inspection. In terms of time efficiency, both systems offer embedding and extraction times well under the 2-second requirement set in Section 3. The implementation of the extraction process for systems 3 and 4 is left as future work. However, assuming that both systems achieve extraction times below two seconds, system 4 would emerge as the most suitable solution. This is due to its increased level of security with no significant compromises in embedding capacity or embedding time. Although System 2 uses matrix embedding, which reduces its embedding capacity to approximately 42.85% of the image's pixel count, this reduced capacity remains sufficient. According to the calculations conducted in Section 3, in order to embed a payload of 150 bytes, an image with a resolution of a minimum of 53x53 pixels is needed.

8 CONCLUSIONS AND FUTURE WORK

This research explored the general problem of metadata privacy and investigated possible approaches to enhance its security using steganography. The research aimed to extend Dodeja's work by improving the version of LSB embedding used in "BaatCheet" with a more secure approach. The paper aimed to develop a secure embedding scheme by looking into existing steganographic techniques and investigating how such techniques can be used together in different configurations to enhance security. Furthermore, the security of each system was assessed using several steganalysis techniques such as statistical steganalysis and visual attacks. Additionally, the mobile application "BaatCheet Embedder" was developed to test the time complexity of each system in a real mobile environment. The embedding time, extraction time, extraction reliability, and embedding capacity of each embedding system were measured.

System 2 emerges as the most suitable solution for use in a steganographic mobile messaging applications. This is due to the inclusion of the extraction functionality that is featured in systems 1 and 2 only. However, compared to system 1, system 2 offers better performance against steganalysis.

Future extensions of this work could focus on developing a mobile messaging application that utilizes the embedding systems analyzed in this study. Another potential improvement is the implementation of the extraction functionality for systems 3 and 4. Additionally, the systems can be tested against machine learning steganalysis to assess their performance against more advanced steganalysis techniques.

REFERENCES

[1] 2025. Java. https://www.java.com/en/. Accessed: 29 June 2025.

[2] Eirikur Agustsson and Radu Timofte. 2017. NTIRE 2017 Challenge on Single Image Super-Resolution: Dataset and Study. In *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. 1122–1131. https://doi.org/10.1109/CVPRW.2017.150

[3] Hayat Al-Dmour and Ahmed Al-Ani. 2016. A steganography embedding method based on edge identification and XOR coding. *Expert Systems with Applications* 46 (2016), 293–306. https://doi.org/10.1016/j.eswa.2015.10.024

[4] Android Studio. 2025. Android Developers. https://developer.android.com/studio. [Online; accessed 5-May-2025].

[5] Vladimir Bochkarev, Anna Shevlyakova, and Valery Solovyev. 2012. Average word length dynamics as indicator of cultural changes in society. *Social Evolution and History* 14 (08 2012), 153–175.

[6] Christian Cachin. 2004. An information-theoretic model for steganography. *Information and Computation* 192, 1 (2004), 41–56. https://doi.org/10.1016/j.ic.2004.02.003

[7] Joan Daemen and Vincent Rijmen. 2002. *The Design of Rijndael.* Springer-Verlag, Berlin, Heidelberg.

[8] Ujjwal Dodeja. 2024. BaatCheet : Android chat application coupling End-to-End encryption and LSB substitution. http://essay.utwente.nl/98212/

[9] Marlon Cordeiro Domenech, André Ricardo Abed Grégio, and Luis Carlos Erpen de Bona. 2022. On Metadata Privacy in Instant Messaging. In *2022 IEEE Symposium on Computers and Communications (ISCC)*. 1–7. https://doi.org/10.1109/ISCC55528.2022.9912901

[10] Sorina Dumitrescu, Xiaolin Wu, and Nasir Memon. 2002. On steganalysis of random LSB embedding in continuous-tone images. *On Steganalysis of Random LSB Embedding in Continuous-tone Images* 3, 641 – 644 vol.3. https://doi.org/10.1109/ICIP.2002.1039052

[11] Rutvik Dumre and Aashka Dave. 2021. Exploring LSB Steganography Possibilities in RGB Images. In *2021 12th International Conference on Computing Communication and Networking Technologies (ICCCNT)*. 1–7. https://doi.org/10.1109/ICCCNT51525.2021.9579588
[12] Jessica Fridrich, Miroslav Goljan, and Rui Du. 2002. Reliable Detection of LSB Steganography in Color and Grayscale Images. *Proceedings of the 2001 Workshop on Multimedia and Security: New Challenges* (10 2002). https://doi.org/10.1145/1232454.1232466
[13] Jessica Fridrich, Petr Lisoněk, and David Soukal. 2007. On Steganographic Embedding Efficiency. In *Information Hiding*, Jan L. Camenisch, Christian S. Collberg, Neil F. Johnson, and Phil Sallee (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 282–296.
[14] J. Fridrich and D. Soukal. 2006. Matrix embedding for large payloads. *IEEE Transactions on Information Forensics and Security* 1, 3 (2006), 390–395. https://doi.org/10.1109/TIFS.2006.879281
[15] Mehdi Hussain, Ainuddin Wahid Abdul Wahab, Yamani Idna Bin Idris, Anthony T.S. Ho, and Ki-Hyun Jung. 2018. Image steganography in spatial domain: A survey. *Signal Processing: Image Communication* 65 (2018), 46–66. https://doi.org/10.1016/j.image.2018.03.012
[16] De Rosal Ignatius Moses Setiadi, Supriadi Rustad, Pulung Nurtantio Andono, Guruh Fajar Shidik, M. Arief Soeleman, Pujiono, and Purwanto. 2022. Impact Analysis of RGB Channels to the Quality of Imperceptibility in Image Steganography. In *2022 5th International Conference on Information and Communications Technology (ICOIACT)*. 343–348. https://doi.org/10.1109/ICOIACT55506.2022.9972240
[17] JetBrains. 2025. *measureTime - Kotlin API Reference*. https://kotlinlang.org/api/core/kotlin-stdlib/kotlin.time/measure-time.html Accessed: 2025-06-19.
[18] Ki-Hyun Jung. 2018. Comparative Histogram Analysis of LSB-based Image Steganography. *WSEAS Transactions on Systems and Control* 13 (01 2018), 103–112.
[19] Andrew Ker. 2005. A General Framework for Structural Steganalysis of LSB Replacement. *Information Hiding, Lecture Notes in Computer Science* 3227, 296–311. https://doi.org/10.1007/11558859_22
[20] Andrew D. Ker and Rainer Böhme. 2008. Revisiting weighted stego-image steganalysis. In *Security, Forensics, Steganography, and Watermarking of Multimedia Contents X*, Edward J. Delp III, Ping Wah Wong, Jana Dittmann, and Nasir D. Memon (Eds.), Vol. 6819. International Society for Optics and Photonics, SPIE, 681905. https://doi.org/10.1117/12.766820
[21] Daniel Lerch. 2025. *Aletheia — Truth-seeking tools*. https://daniellerch.me/tools-en/#aletheia Accessed: 2025-06-19.
[22] Pratap Chandra Mandal, Imon Mukherjee, Goutam Paul, and B.N. Chatterji. 2022. Digital image steganography: A literature survey. *Information Sciences* 609 (2022), 1451–1488. https://doi.org/10.1016/j.ins.2022.07.120
[23] J. Mielikainen. 2006. LSB matching revisited. *IEEE Signal Processing Letters* 13, 5 (2006), 285–287. https://doi.org/10.1109/LSP.2006.870357
[24] Robert B. Miller. 1968. Response time in man-computer conversational transactions. In *Proceedings of the December 9-11, 1968, Fall Joint Computer Conference, Part I* (San Francisco, California) *(AFIPS '68 (Fall, part I))*. Association for Computing Machinery, New York, NY, USA, 267–277. https://doi.org/10.1145/1476589.1476628
[25] Arooj Nissar and A.H. Mir. 2010. Classification of steganalysis techniques: A study. *Digital Signal Processing* 20, 6 (2010), 1758–1770. https://doi.org/10.1016/j.dsp.2010.02.003
[26] Nurhayati, Kastari, and Feri Fahrianto. 2022. End-To-End Encryption on the Instant Messaging Application Based Android using AES Cryptography Algorithm to a Text Message. In *2022 10th International Conference on Cyber and IT Service Management (CITSM)*. 01–06. https://doi.org/10.1109/CITSM56380.2022.9935963
[27] Oracle. 2014. Class Random - Java 8 API Documentation. https://docs.oracle.com/javase/8/docs/api/java/util/Random.html. Accessed: 2025-06-24.
[28] Avi Rosenfeld, Sigal Sina, David Sarne, Or Avidov, and Sarit Kraus. 2016. WhatsApp Usage Patterns and Prediction Models. *ICWSM/IUSSP Workshop on Social Media and Demographic Research* (05 2016).
[29] Irwin Sobel and Gary Feldman. 1973. A 3×3 isotropic gradient operator for image processing. *Pattern Classification and Scene Analysis* (01 1973), 271–272.
[30] P. Sujatha. 2014. Detection of Hidden Information With Bit Plane Analysis. Online; accessed 19 June 2025. *International Journal of Linguistics and Computational Applications* 1, 1 (2014), —. https://www.academia.edu/35486219/Detection_of_Hidden_Information_With_Bit_Plane_Analysis Available on Academia.edu.
[31] Andreas Westfeld and Andreas Pfitzmann. 2000. Attacks on Steganographic Systems. In *Information Hiding*, Andreas Pfitzmann (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 61–76.

A DIAGRAMS

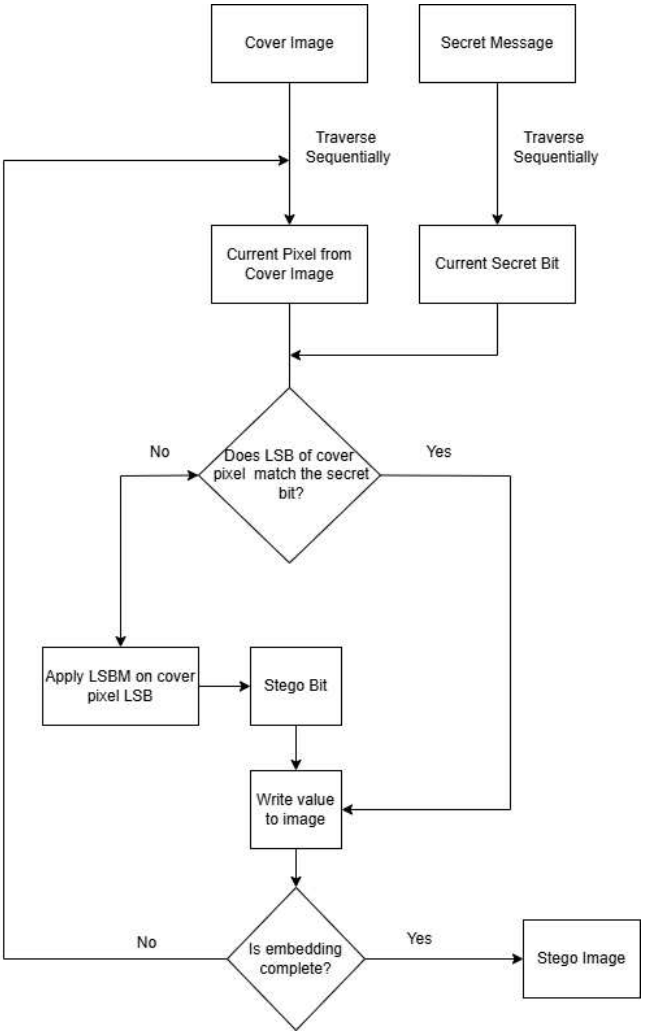

Fig. 1. System 1 diagram

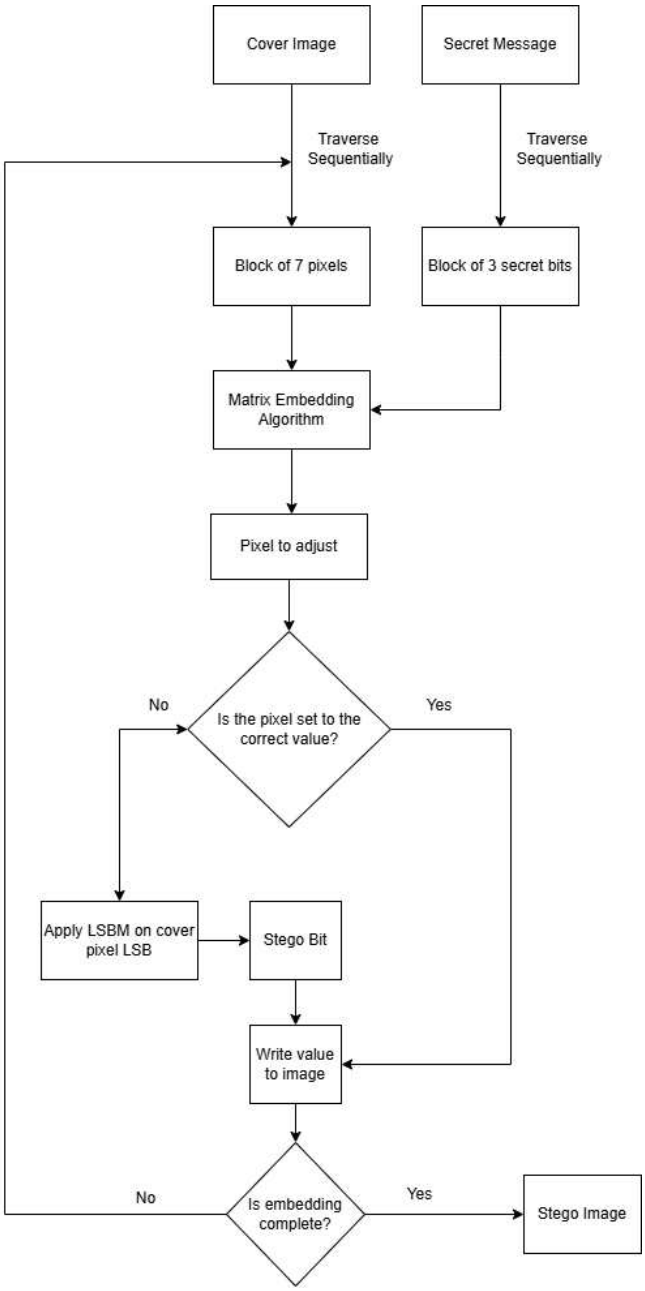

Fig. 2. System 2 diagram

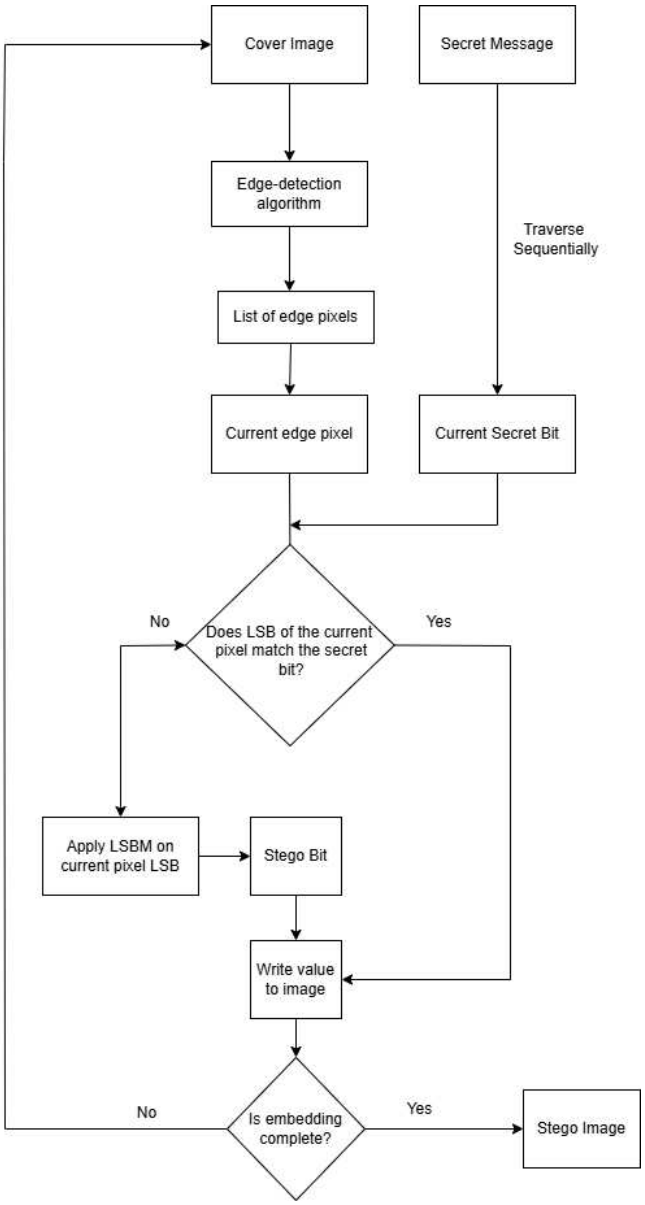

Fig. 3. System 3 diagram

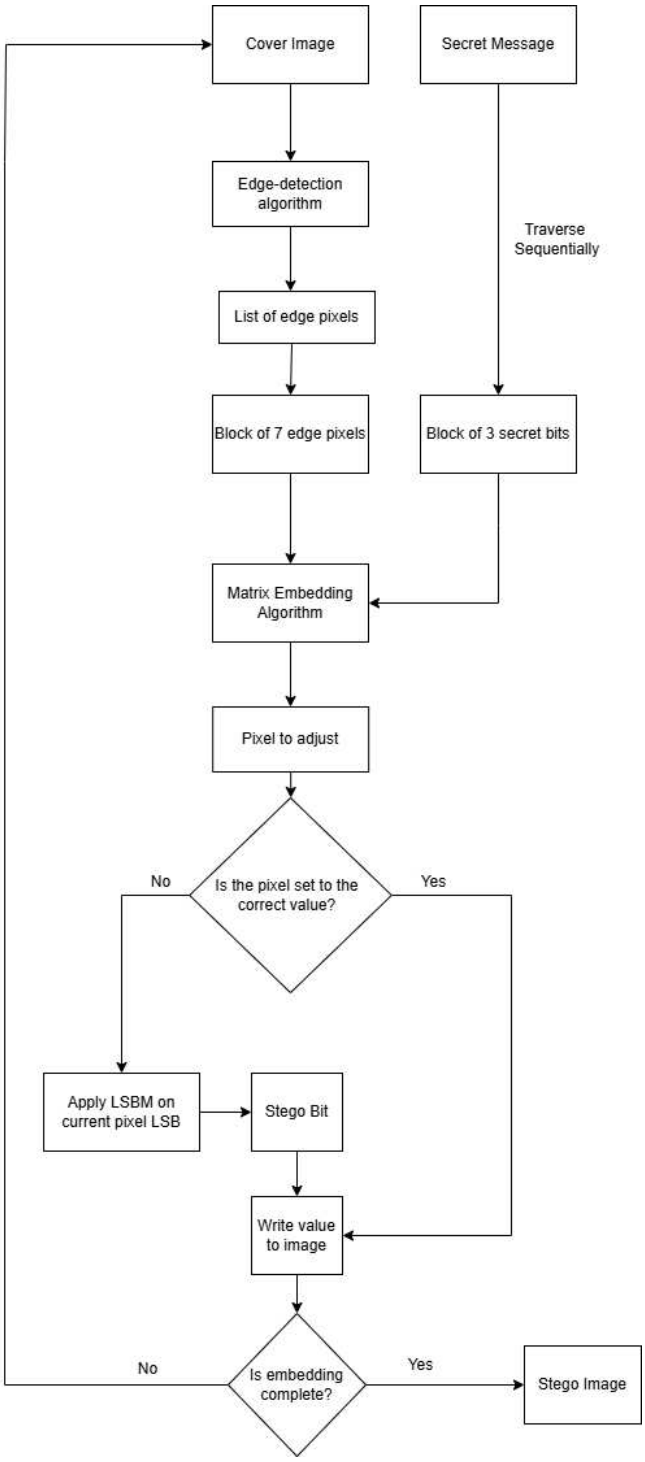

Fig. 4. System 4 diagram

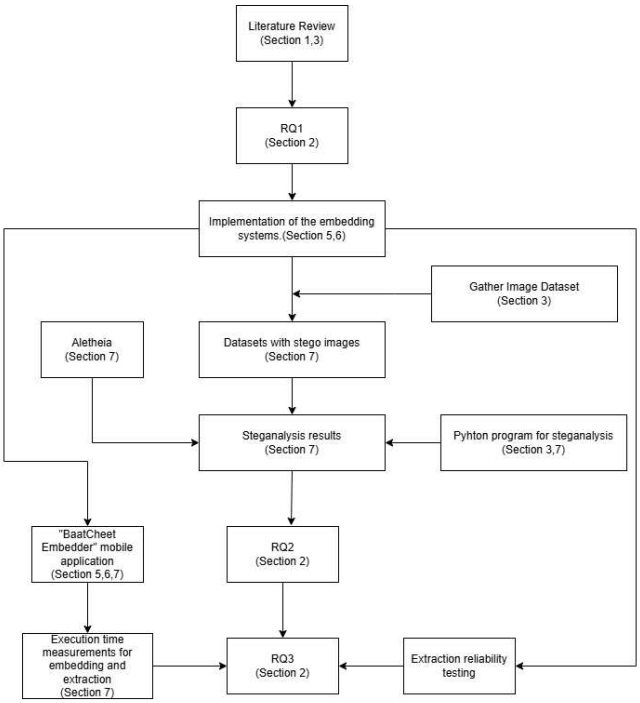

Fig. 5. Methodology diagram

B BIT PLANES

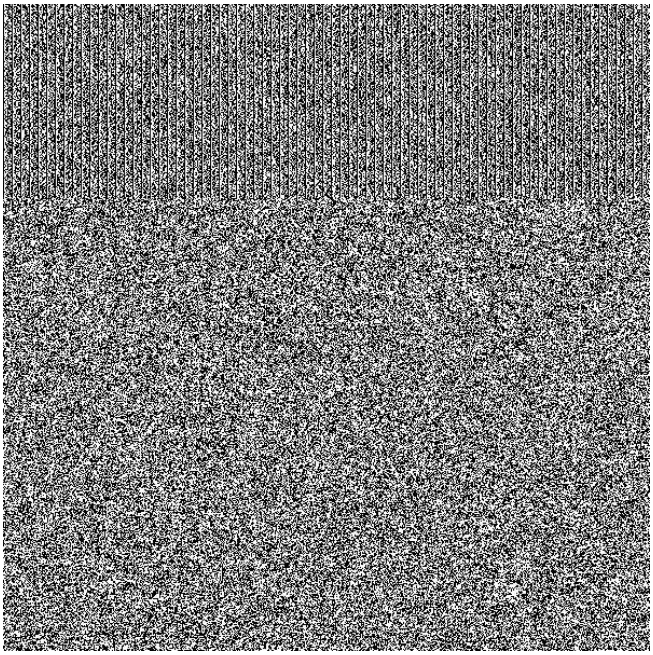

Fig. 6. Bit plane of the least significant bits in the blue channel for a stego image generated using System 1

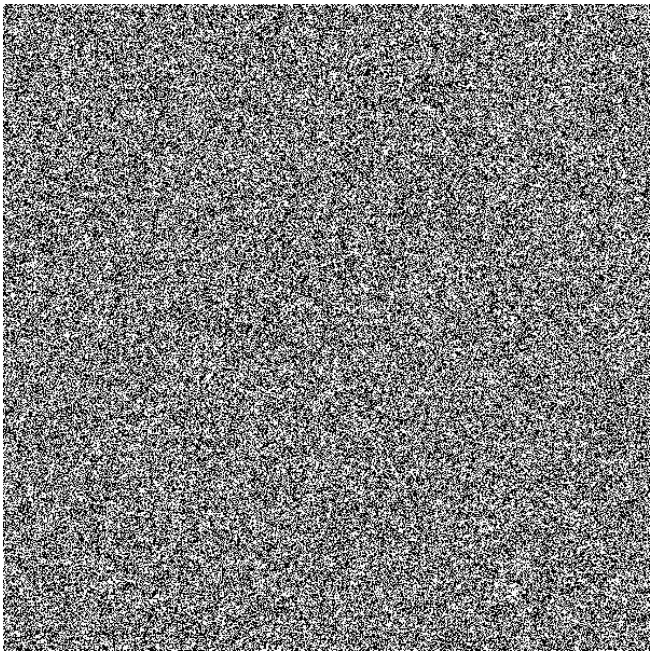

Fig. 7. Bit plane of the least significant bits in the blue channel for a stego image generated using System 2

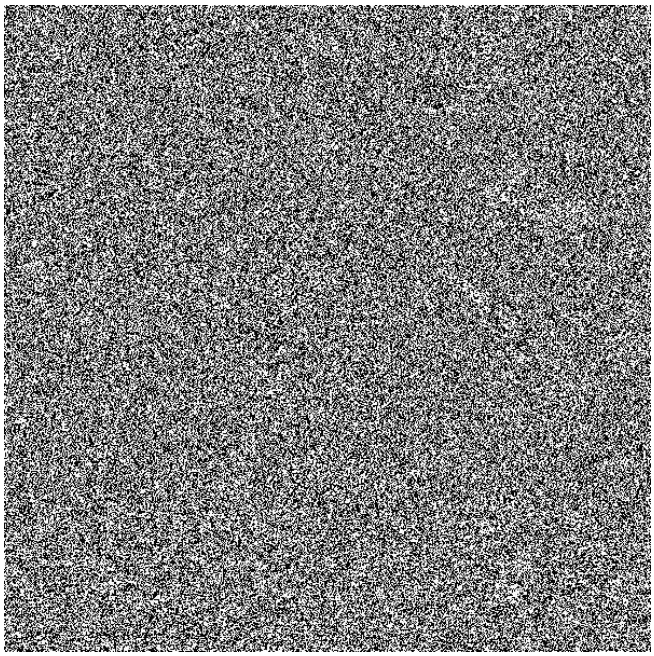

Fig. 8. Bit plane of the least significant bits in the blue channel for a stego image generated using System 3

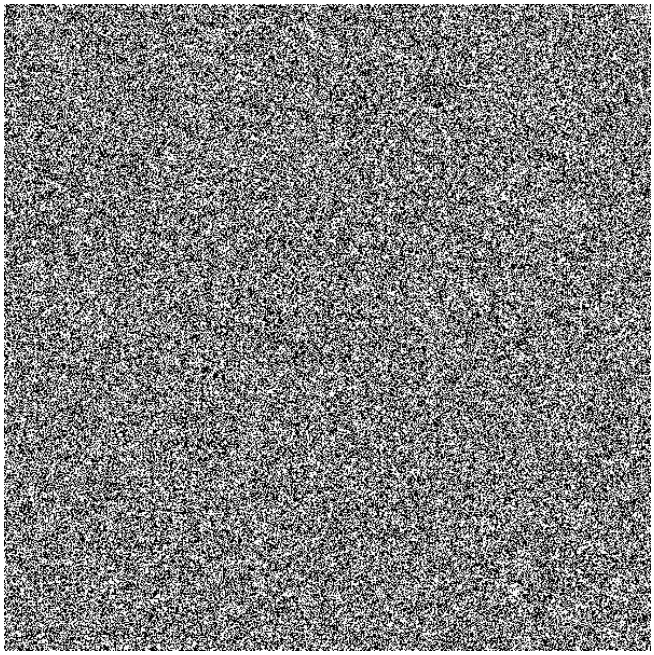

Fig. 9. Bit plane of the least significant bits in the blue channel for a stego image generated using System 4

C HISTOGRAMS

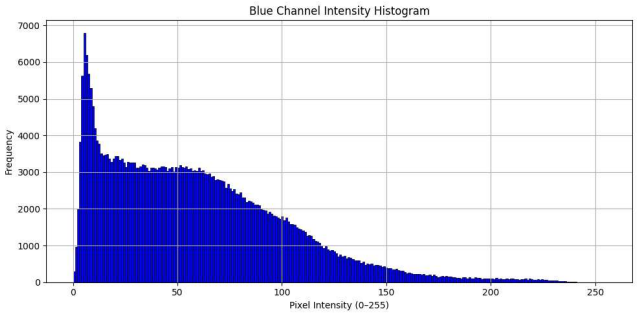

Fig. 10. Histogram of blue channel pixel intensities in the cover image

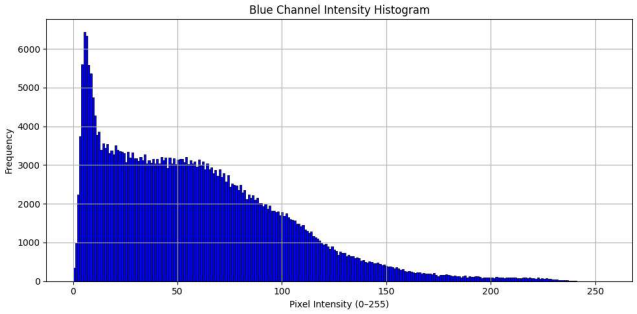

Fig. 11. Histogram of blue channel pixel intensities in a stego image generated using System 1

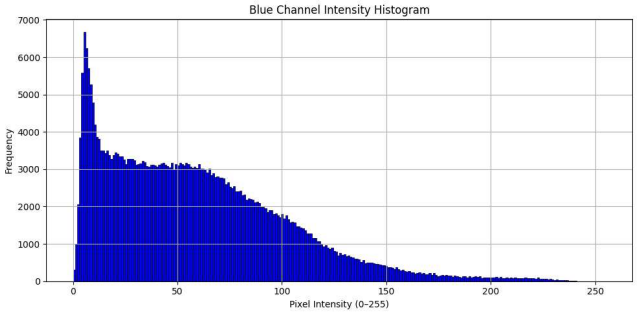

Fig. 12. Histogram of blue channel pixel intensities in a stego image generated using System 2

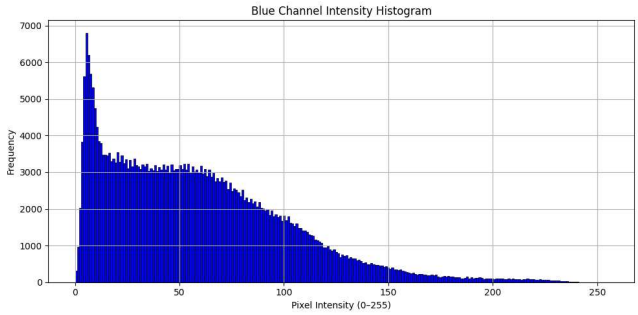

Fig. 13. Histogram of blue channel pixel intensities in a stego image generated using System 3

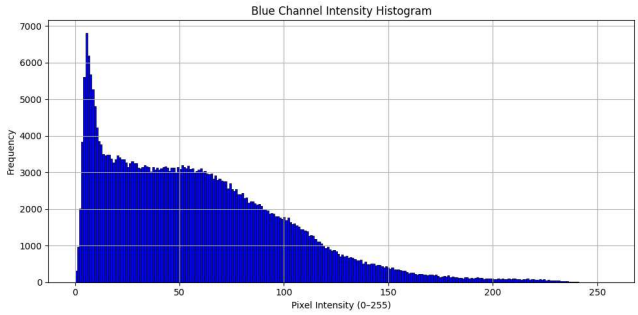

Fig. 14. Histogram of blue channel pixel intensities in a stego image generated using System 4

D BAATCHEET EMBEDDER SNAPSHOTS

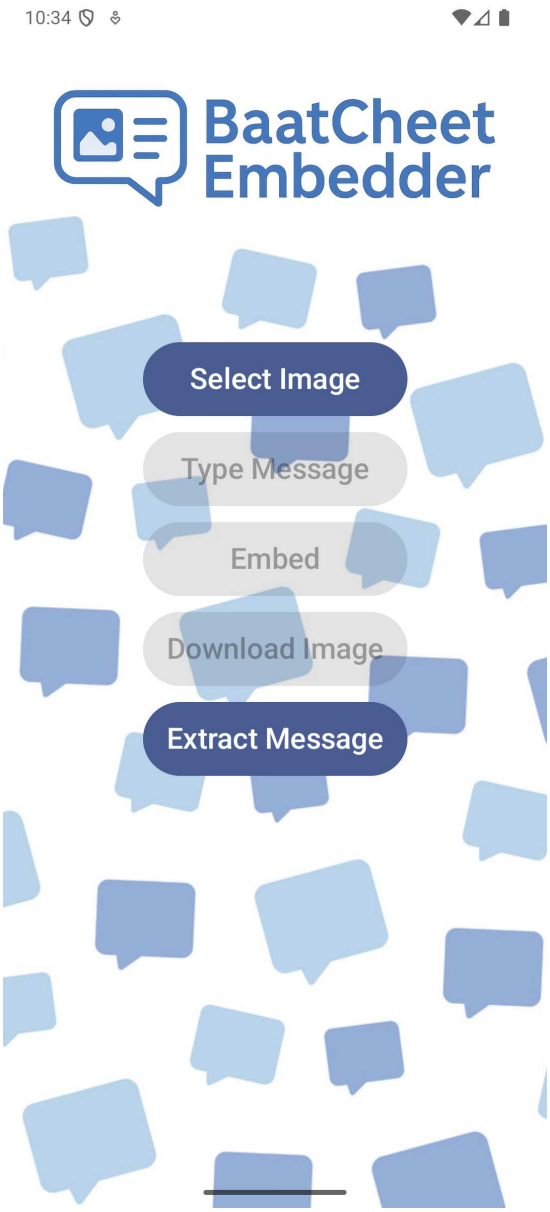

Fig. 15. BaatCheet Embedder

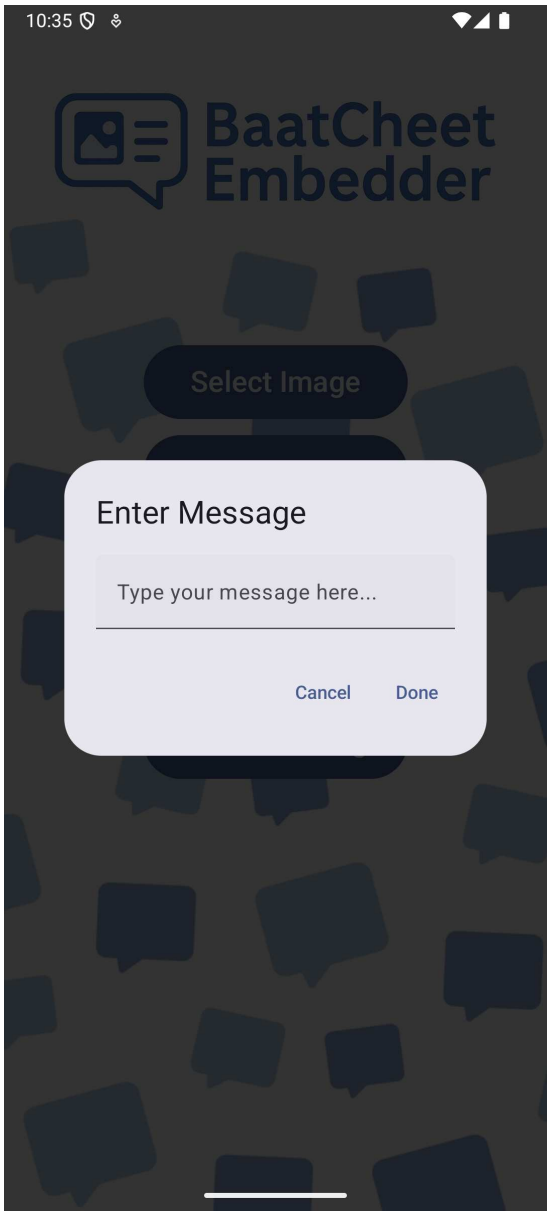

Fig. 16. BaatCheet Embedder

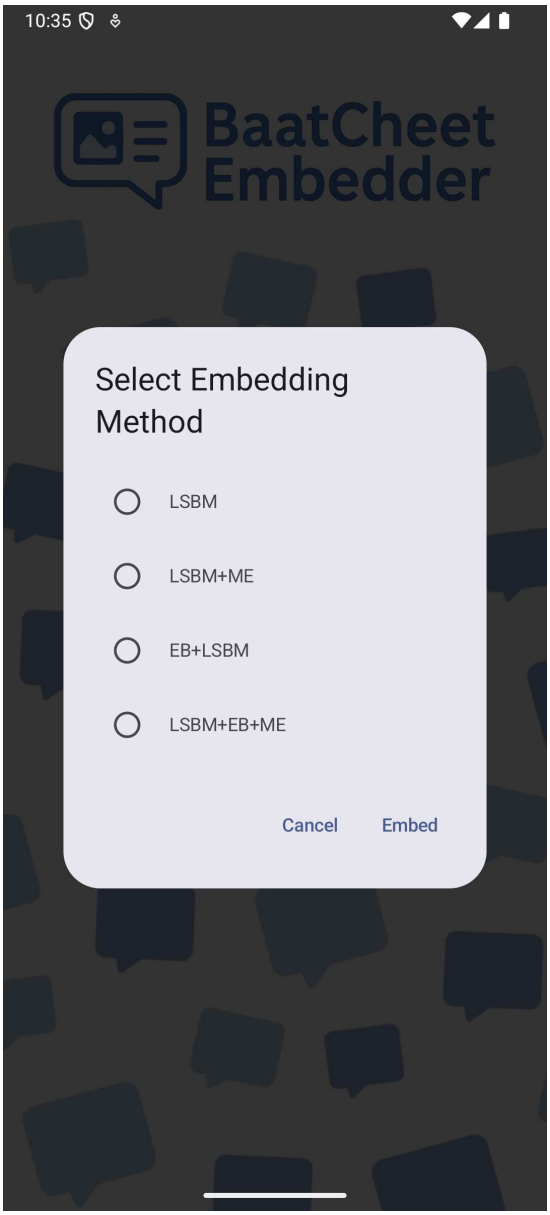

Fig. 17. BaatCheet Embedder

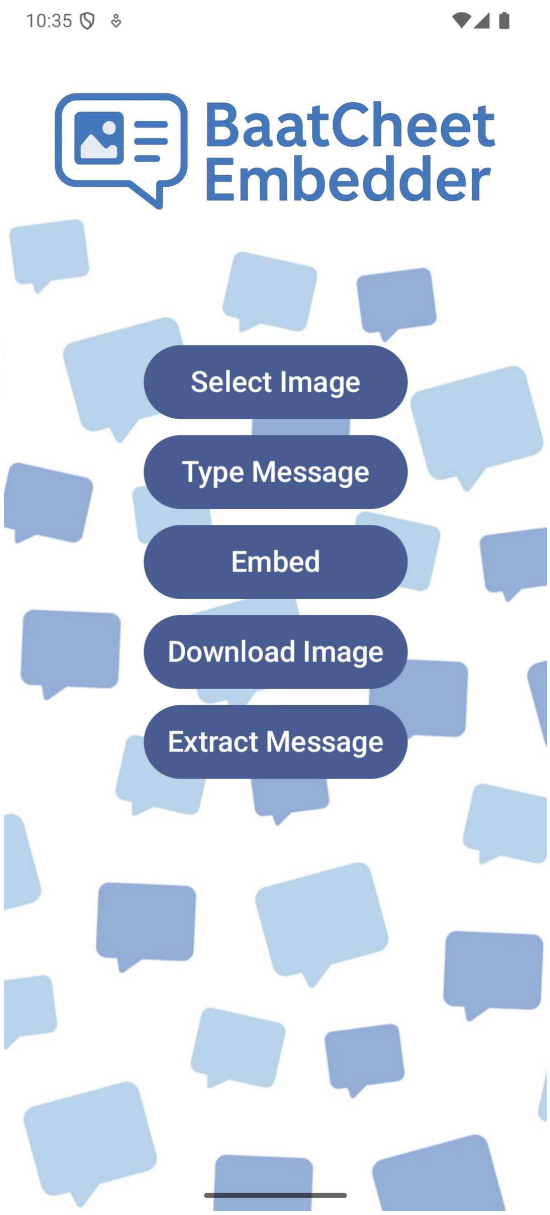

Fig. 18. BaatCheet Embedder

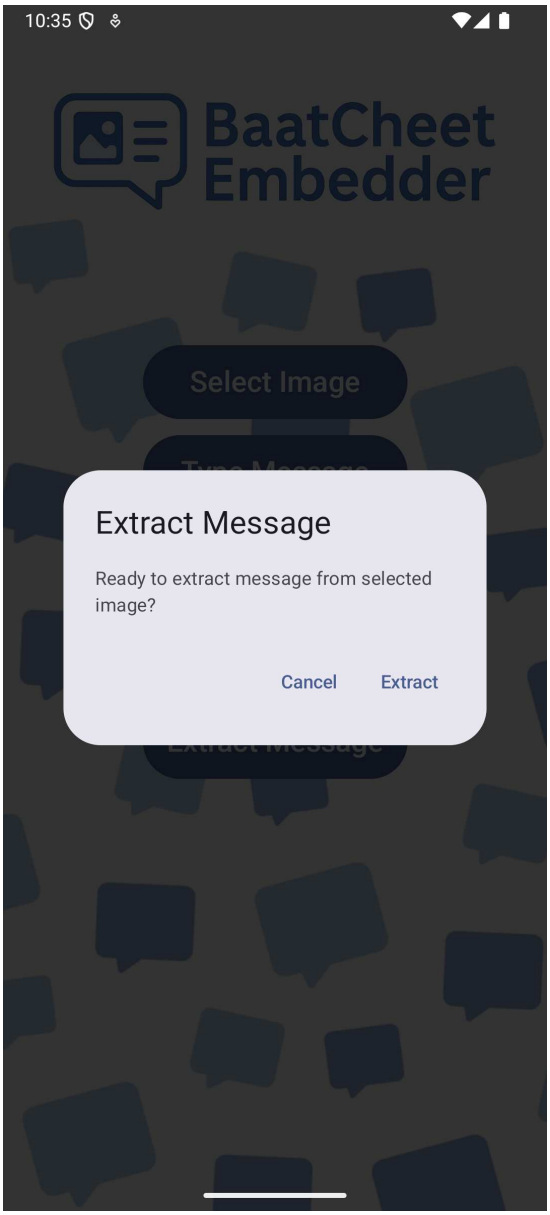

Fig. 19. BaatCheet Embedder

Fig. 20. BaatCheet Embedder

E COVER IMAGE

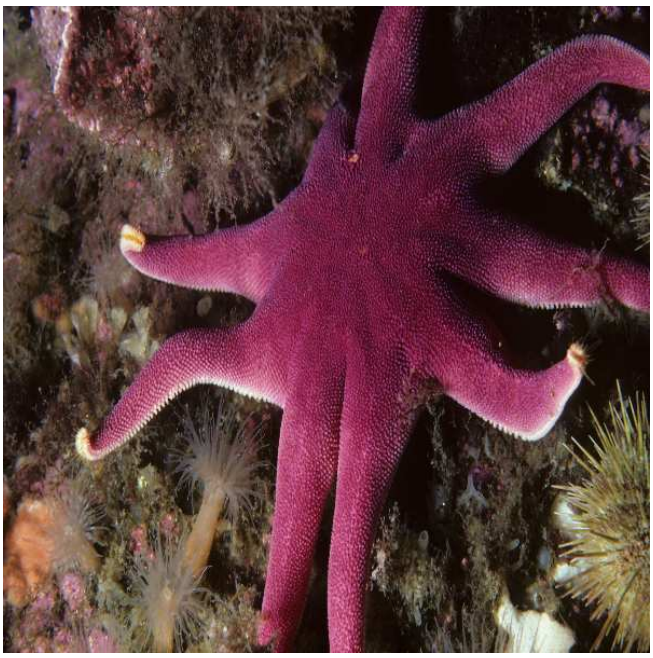

Fig. 21. Cover Image