Journal on Information Technologies & Communications
https://www.ictmag.vn **Volume 2023, Number 2**
September

TABLE OF CONTENTS

Proposals and Implementations of MDS Diffusion Layer Dynamic Algorithms for AES Block Cipher

Tran Thi Luong
Academy of Cryptography Techniques, No.141 Chien Thang road, Tan Trieu, Thanh Tri, Hanoi, Vietnam

Correspondence: Tran Thi Luong, luongtranhong@gmail.com
Communication: received 31 Jan 2023, revised 04 Apr 2023, accepted 26 Jun 2023
DOI: 10.32913/mic-ict-research.v2023.n2.1194

***Abstract*: Block ciphers are an interesting cryptographic topic that is widely used today. Therefore, the issue of improving the security of block ciphers is increasingly concerned today. There have been many research directions to animate block ciphers to improve their security against well-known strong attacks. In [1], we proposed two diffusion layer dynamic algorithms for SPN block ciphers. In this paper, we will execute these two algorithms with recursive MDS matrices of size 4, 8, and 16, then evaluate the cryptographic properties of the obtained dynamic MDS matrices including branch number, number of fixed points, coefficient of fixed points, number of XOR and Xtime operations. Then, we apply these two dynamic algorithms to dynamically modify the AES block cipher and evaluate the execution speed of the dynamic AES block ciphers. We propose two new diffusion layer dynamic algorithms, implement these two algorithms and modify dynamically AES block cipher according to these two algorithms, and evaluate the speed of dynamic AES block ciphers. We also evaluate the security and implementation resources for dynamically modified AES block ciphers. The proposed diffusion layer dynamic algorithms contribute to improving the security of the AES block cipher against many current strong attacks.**

Keywords: *MDS matrix, AES, dynamic diffusion layer.*

I. INTRODUCTION

Block ciphers are an area of special interest in cryptography. Proof of this is that the US opened two block cipher standard selection contests in two consecutive periods. The Russian Federation developed their own block cipher standard very early, the second time recently is the GOST R34.12-2015 block cipher standard [2-4]. Since NIST's AES [5-7] selection competition in 2000, the SPN block cipher [8-10] has received more and more research attention.

Along with the research work on SPN block ciphers, there have been many works focused on the highly secure direction for these block ciphers. That's why methods for making block ciphers dynamic were born. The most basic way of animating a block cipher is usually based on a given secret key. With the SPN block cipher, there are many ways of animating its components: substitution layer dynamics, diffusion layer dynamics, both substitution and diffusion layer dynamics, and several other types of dynamics. In the SPN block cipher dynamic at the substitution layer, dynamic substitution boxes (S-boxes) are made depending on a secret key used by the block cipher. The first studies of this type must include the studies of B. Schneier et al. in [11, 12]. Only the improvement of AES in this direction has had a lot of research work, for example in [13-20]. These works focused on constructing dynamic S-boxes that depend on a given secret key, then the authors used the found dynamic S-boxes to replace the original AES S-box, generating dynamic AES block ciphers.

The inclusion of dynamic S-boxes in block ciphers has been applied to many block ciphers today. However, the generation of a dynamic MDS matrix to create a dynamic diffusion layer for block ciphers is still an open research problem and attracts many cryptographers today. In this dynamic method, the SPN block cipher is animated at the diffusion layer. With this research direction, there are a number of related studies, such as in [1, 21, 22, 23, 24]. In [21], the authors used a dynamic MDS matrix based on scalar multiplication in the rows of the MDS matrix of AES and a complement key of m-bit. However, in [21] only studied introducing a 4×4 MDS matrix in to the Mixcolumn transformation of AES, but not studied for larger MDS matrices. In [22], the authors proposed a dynamic SPN block cipher based on AES, denoted DRAES, where the animated transformation is a rotation whose amount of rotation depends on the data (plaintext and ciphertext) in AddRoundKey and depends on the key in

the AES key scheme. However, in [22], the authors did not focus on MDS matrix and making AES dynamic based on Mixcolumn transformation. In [23], the authors proposed to use a dynamic MixColumn transformation based on key-dependent DNA structures and processes. However, the authors only studied to create dynamic 4×4 MDS matrices from the original AES matrix, but not for the larger MDS matrices. In [24], the authors created a private XOR table that depends on the 3D chaos mapping and an original secret key. The authors also used the dynamic MixColumn transformation for AES with a 4×4 dynamic MDS matrix generated using a 3D chaos mapping to permute the AES 4×4 MDS matrix. Unfortunately, their dynamic MDS matrix is not an MDS matrix.

In [1], we proposed two diffusion layer dynamic algorithms for SPN block ciphers in general, and these algorithms can be applied to MDS matrices of any size. However, we have not yet implemented these algorithms and have not applied them to a specific SPN block cipher. In [25], we studied the use of MDS matrices of the original SPN block cipher to efficiently compute the outputs of dynamic MDS matrices without directly calculating the dynamic MDS matrices. Dynamic MDS matrices are generated from the original MDS matrix using direct exponent and scalar multiplication transformations. However, we just came up with the idea of taking advantage of the original MDS matrices without any specific implementation. In this paper, we will execute the two dynamic diffusion layer algorithms proposed in [1] with recursive MDS matrices of size 4, 8, and 16 then evaluate the cryptographic properties of the obtained dynamic MDS matrices including branch number, number of fixed points, coefficient of fixed points, number of XOR and Xtime operations. Then, we apply these two dynamic algorithms to dynamically modify the AES block cipher and evaluate the execution speed of the dynamic AES block ciphers. We propose two new diffusion layer dynamic algorithms, implement these two algorithms and modify dynamically AES block cipher according to these two algorithms, and evaluate the speed of dynamic AES block ciphers. We also evaluate the security and implementation resources for dynamically modified AES block ciphers.

Thus, in [1], we just introduced the algorithms but have not implemented these algorithms to obtain specific dynamic MDS matrices, so it is impossible to evaluate the cryptographic properties of obtained dynamic MDS matrices. This is important for the security as well as the execution speed of future block ciphers. In addition, in [1] and [25], we have not applied the proposed algorithms to the AES block cipher modification. Modifying the AES block cipher in a way that animates the diffusion layer is a meaningful way to improve the security of AES against many current attacks such as linear attacks, differential attacks.

This article is organized as follows. In Section 2, we present some related works. Section 3 proposes new diffusion layer dynamic algorithms. Section 4 implements dynamic algorithms and applies them to the dynamic modification of AES. Section 5 gives security analysis and implementation resources evaluation for modified AES block ciphers. Section 6 is the conclusion.

II. RELATED WORKS

1. Direct exponentiation and scalar multiplication transformation

Direct exponentiation and scalar multiplication of MDS matrices were first given in [26]. However, in [27, 28, 29, 30], we presented many useful results about the two transformations. Furthermore, we also extend the concept of scalar multiplication to the MDS matrix.

Direct exponentiation with exponent e of a matrix $A = \left[a_{i,j}\right]_{m\times m}$, $a_{i,j} \in GF\left(p^r\right)$ where p is prime, r>=1, r is an integer, denoted A_{d^e} , where each of its elements is the result of the power e of the corresponding elements in A.

Scalar multiplication is defined as follows [28]: For $E = (e_1, e_2, \ldots, e_m)$, $F = (f_1, f_2, \ldots, f_m)$, (for $e_i f_i \in GF^*\left(p^r\right)$) and for $A = \left[a_{i,j}\right]_{m\times m}$, $a_{i,j} \in GF\left(p^r\right)$. We say that the product of (E, F) and A is a matrix denoted by $(E, F)(A) = \left[b_{i,j}\right]_{m\times m}$ determined by the following formula: $b_{i,j} = e_i f_j a_{i,j}$.

2. Take advantage of MDS matrices of the original SPN block cipher [26]

With dynamic MDS matrices generated from direct exponentiation and scalar multiplication, in [26], we studied an efficient method to compute the outputs of dynamic MDS matrices based on the original MDS matrices when its inputs are known, as well as the efficient calculation of the inputs of the dynamic MDS matrices based on the original MDS matrices when the outputs are known. That is, we can take advantage of the MDS matrix of the original SPN block cipher to compute the input and output of the dynamic MDS matrices of the dynamic SPN block cipher without having to specifically compute the dynamic MDS matrix.

Suppose both Alice and Bob have two initial MDS matrices, $A = \left[a_{i,j}\right]_{m\times m}$, $a_{i,j} \in GF\left(p^r\right)$ and $B = \left[b_{i,j}\right]_{m\times m}$, $b_{i,j} \in GF\left(p^r\right)$ $\left(B = A^{-1}\right)$. In [26], we make the following important remarks:

For direct exponentiation

With the secret parameter $k\ (1 \le k \le r)$, two dynamic MDS matrices are generated for encryption and decryption at the dynamic diffusion layer, denoted A_p and B_p (where $A_p = A_{dp^k}$ and $B_p = A_{p^{-1}}$).

Remark 1 [26]. Given any input vector $X \in (GF(p^r))^m$, Alice does not need to compute the matrix A_p and does not need to compute the output according to the formula $Y = A_p X$. Alice just needs to do presently:

- *Step 1: Calculate* $Z = X^{p^{r-k}}$.
- *Step 2: Calculate* $Y_1 = AZ$.
- *Step 3: Calculate* $Y = Y_1^{p^k}$ *and send* Y *to Bob.*

Remark 2 [26]. When Bob receives an encrypted vector $Y \in (GF(p^r))^m$, Bob does not need to compute the matrices A_p and B_p and also does not need to compute the plaintext using the formula $X = B_p Y$. Bob just need to do:

- *Step 1: Calculate* $Z' = Y^{p^{r-k}}$.
- *Step 2: Calculate* $Z = BZ'$.
- *Step 3: Calculate* $X = Z^{p^k}$.

For scalar multiplication

Alice and Bob agree on secret parameters $E = (e_0, e_1, \dots e_{m-1})$ and $F = (1, 1, \dots, 1)$. Then, two new dynamic MDS matrices are created for the encryption and decryption process at the dynamic diffusion layer, denoted A_M and B_M where $B_M = {A_M}^{-1}$.

Remark 3 [26]. Given any input vecto $X \in (GF(p^r))^m$, Alice does not need to compute the matrix A_M and does not need to compute the output according to the formula $Y = A_M X$. Alice just needs to do presently:

- *Step 1: Calculate* $Z = AX$.
- *Step 2: Calculate* $Y = (e_1 Z_1, e_2 Z_2, \dots, e_m Z_m)$ *and send* Y *to Bob.*

Remark 4 [26]. When Bob receives an encrypted vector $Y \in (GF(p^r))^m$, Bob does not need to compute the matrices A_M and B_M and also does not need to compute the plaintext using the formula $X = B_M Y$. Bob just need to do:

- *Step 1: Calculate* $Z = (Z_1, Z_2, \dots, Z_m)$ *where* $Z_i = {e_i}^{-1} Y_i$, $i = 1, 2, \dots, m$.
- *Step 2: Calculate* $X = BZ$.

Thus, in either case using direct exponentiation or scalar multiplication, neither Alice nor Bob need to compute the dynamic MDS matrices directly for encryption and decryption, but can take advantage of the original MDS matrices (A and B).

3. Diffusion layer dynamic algorithms [1]

In [1], we proposed two diffusion layer dynamic algorithms (Algorithm 1 and Algorithm 2) for SPN block ciphers based on two transformations namely direct exponentiation and scalar multiplication. Algorithm 1 generates a dynamic MDS matrix from an initial MDS matrix and a secret key. This dynamic matrix will be used for every round of the SPN block cipher. Algorithm 2 generates two dynamic MDS matrices from two initial MDS matrices and a secret key. These two dynamic matrices will be used alternately in the rounds of the SPN block cipher. In [1], we also provide a detailed analysis of improving the security of dynamic SPN block ciphers when using these dynamic algorithms. For more details see [1].

III. PROPOSING NEW DIFFUSION LAYER DYNAMIC ALGORITHMS

From the two diffusion layer dynamic algorithms proposed in [1], combined with the results (Remarks 1, 2, 3, 4) given in [26] about being able to take advantage of initial MDS matrices of the original SPN block cipher without having to compute the dynamic MDS matrices, in this section we propose two new diffusion layer dynamic algorithms (Algorithm 1*, Algorithm 2*) based on the idea of two Algorithms 1 and Algorithm 2 but apply the above results.

Algorithm 1* assumes that there are a given MDS matrix M and a secret key Key, the dynamic MDS matrix can be generated from the original matrix M , a secret key *Key*, and the direct exponential transformation (T_D) and the scalar multiplication transformation (T_S) . This dynamic MDS matrix will be used in the diffusion layer for every round of the block cipher. However, instead of directly calculating the dynamic MDS matrix, the algorithm below takes advantage of the original MDS matrix M as shown in [26]. Therefore, the dynamic algorithm below will only need to compute the secret parameters for the dynamic MDS matrix without having to specifically compute this dynamic MDS matrix.

algorithm 1*

Input: A secret key *Key*, a MDS matrix M of size $m \times m$ over $GF(2^r)$.

Output: A parameter $l\ (0 \le l \le r-1)$ or a secret vector $E = (e_0, e_1, \dots, e_{m-1})$ for $e_i \in GF(2^r)$.

Step 1: Take the first bit of the key *Key* to determine which matrix transformation is used in this algorithm:

- If it is 1, choose the transformation as.
- If it is 0, choose the transformation as .

Step 2:

If the result of step 1 is T_D *then do*:

Step 2.1. Take the next r bits of Key (and a is the corresponding decimal number). Calculate l=a mod r.

If the result of step 1 is T_S then do:

Step 2.2. Take consecutive non-zero bit segments (each segment with bits) from the key Key (i.e. if there is a segment where r bits are all zeros, shift right one bit until you get the segment with non-zero bits . Take m segments like that. Convert those *r-bit* strings to a non-zero element $e_i \in GF(2^r)$, $0 \le i \le m-1$.

Choose two vectors for scalar multiplication as $E = (e_0, e_1, \ldots, e_{m-1})$ and $F = (1, 1, \ldots, 1)$.

Encryption and decryption process

Both parties involved in the secret communication process will make the algorithm's parameters public and private as follows:

Public: Encryption algorithm, initial MDS matrix M (of size), and the dynamic diffusion algorithm (Algorithm 1*).

Private: The secret key Key.

The two communication parties need to implement Algorithm 1* and agree on a method to utilize the MDS matrices of the original SPN block cipher to find the secret parameters or for , and they don't need to calculate the dynamic MDS matrix M_K (where $M_K = M_{dp^k}$ hoặc $M_K = (E, F)\, M$). They can take advantage of the original MDS matrices M and M^{-1} to perform encryption and decryption at the diffusion layer (see Remarks 1, 2, 3, 4).

Algorithm 2*. In this dynamic algorithm, it is assumed that there are two given initial MDS matrices A and B and a secret key *Key*. Dynamic MDS matrices can be generated from such matrices by direct exponentiation or scalar multiplication and secret key *Key*. These dynamic MDS matrices will be used for the diffusion layer alternately in the rounds of the block cipher. However, instead of correctly calculating new key-dependent dynamic MDS matrices, we will make use of the original MDS matrices (A , A^{-1} and B , B^{-1}) for the encryption and decryption process of the diffusion layer.

Algorithm 2*

Input: A secret key *Key*, two MDS matrixs A, B of size $m \times m$ over $GF(2^r)$.

Output: Secret parameters l_1, l_2 $(0 \le l \le r-1)$ or secret vector $E = (e_0, e_1, \ldots, e_{m-1})$ and $E' = (e'_0, e'_1, ..., e'_{m-1})$ for $e_i, e'_i \in GF(2^r)$; Or secret parameters l $(0 \le l \le r-1)$ and $E = (e_0, e_1, \ldots, e_{m-1})$ for $e_i \in GF(2^r)$

Step 1: Find the first two bits of the key *Key* to determine which matrix transformation is used in this algorithm.

- If they are 01, choose the transformation as T_D .

- If they are 10, choose the transformation as T_S .

- If they are 00 or 11 then use both transformations as T_D, T_S.

Step 2:

If the result of step 1 is T_D then execute:

Step 2.1. Get the next r bits of Key (and a is the corresponding number). Calculate $l_1 = a \bmod r$. Take the last r bits of *Key* (and b is the corresponding number) and determine $l_2 = b \bmod r$.

If the result of step 1 is T_S then execute:

Step 2.2. Take the next r non-zero bits of *Key* and convert that sequence of r bits to the element $e_1 \in GF(2^r)$. Thus $e_1 \in GF^*(2^r)$. If r bits are all 0, then we shift right by bit of *Key* until we get a sequence of r bits such that $e_1 \ne 0 \in GF(2^r)$. Take the last r bits of *Key* and convert this string of r bits to the element $e_2 \in GF(2^r)$. If $e_2 = 0 \in GF(2^r)$ then we shift left 1 bit until we get the element $e_2 \ne 0 \in GF(2^r)$.

Step 2.3. Get the next r bits at the beginning of *Key* (after step 2.2). Then convert this string to an integer, denoted a, taking $i = a \bmod m$, so $0 \le i \le m-1$. Get the next r bits (in the right-to-left direction) at the end of the *Key* bit sequence (after step 2.2). Then convert this string to an integer, denoted by b, taking $j = b \bmod m$, so $0 \le j \le m-1$.

We specify that the rows and columns of the matrix A, B will be numbered from 0 to $m-1$.

Choose the vectors for multiplying by scalars as $E = (1, 1, \ldots, 1, e_1, 1, \ldots, 1)$, $F = (1, 1, \ldots, 1)$ where e_1 is the $(i+1)$th element of the vectors E and $E' = (1, 1, \ldots, 1, e'_2, 1, \ldots, 1)$, $F' = (1, 1, \ldots, 1)$ where e_2 is the $(j+1)$th element of the vector E'.

If the result of step 1 is both T_D and T_S then exxecute (apply T_D to and T_S to B):

Step 2.4. Get the next r bits of *Key* (and a is the corresponding number). Calculate $l = a \bmod r$. Take the last r bits of *Key* and convert this string of r bits to the element $e \in GF(2^r)$. If $e = 0 \in GF(2^r)$ then we shift left 1 bit until we get the element $e \ne 0 \in GF(2^r)$.

Step 2.5. Get the next r bits at the beginning of *Key* (after step 2.4). Then convert this string to an integer, denoted by c, taking $i = c \bmod m$, so $0 \le i \le m-1$.

Take the vectors multiplied by the scalar as $E = (1, 1, \ldots, 1, e, 1, \ldots, 1)$, $F = (1, 1, \ldots, 1)$ where e is the $(i+1)$th element of the vector E.

Note. In the case of initial MDS matrix of size ≥ 16 , in order for **Algorithms 1**, **Algorithms 2** to work, it is necessary to choose a secret key Key with bit length greater than 128, for example, with $m = 16$, it should be chosen the secret key *Key* has bit length of $n \ge 192$.

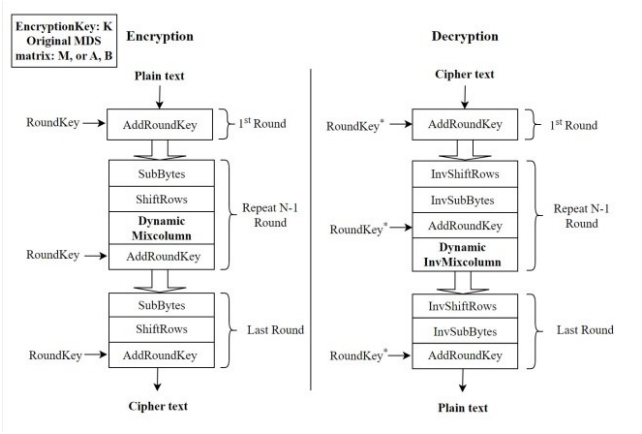


Figure 1. Structure of diffusion layer of dynamic modified AES Algorithms.

Encryption and decryption process

Both parties involved in the secret communication process will make the algorithm's parameters public and private as follows:

Public: Encryption algorithm, two initial MDS matrices A,B (of size), and the dynamic diffusion algorithm (Algorithm 2).

Private: The secret key Key.

The two parties need to implement Algorithm 2 and agree on a method to take advantage of the MDS matrices of the original SPN block cipher to find the secret parameters and they do not need to calculate the exact dynamic MDS matrices A_K, B_K. They can take advantage of the original MDS matrices A, A^{-1} and B, B^{-1} to perform encryption and decryption at the dynamic diffusion layer (see Remarks 1, 2 , 3, 4).

The **Algorithm 1*** and **Algorithm 2*** are very simple and the outputs are just parameters so the complexity is negligible (approximately $O(r)$).

IV. IMPLEMENTING DYNAMIC ALGORITHMS AND APPLICATION TO DYNAMICALLY MODIFIED AES

In this section, we implement the proposed diffusion dynamic algorithms including: Algorithm 1, Algorithm 2 in [1], and Algorithm 1*, Algorithm 2* in this paper. These dynamic algorithms are then applied to the dynamically modified AES block cipher, generating the corresponding dynamic versions of the AES block cipher.

Denote two dynamically modified AES block ciphers corresponding to two dynamic algorithms Algorithm 1 and Algorithm 2 AESD1 and AESD2 respectively. Two dynamic modified AES block ciphers correspond to two dynamic algorithms **Algorithm 1***, and **Algorithm 2***are **AESD1*** and **AESD2*** respectively.

Figure 1 shows the structure of dynamically modified AES block ciphers animated at the diffusion layer, specifically in the Mixcolumn transformation of AES.

The original MDS matrices in the original SPN are selected as recursive MDS matrices of sizes 4, 8, and 16 in Table I and Table II. These matrices are selected based on the following attributes: having small number of fixed points and small coefficient of fixed points ($D(A)$) and belongs to the class of matrices with a small number of *XORs* and *Xtimes*, in order to be efficient in performance.

1. Executing Algorithm 1 and Algorithm 2 [1]

In order to perform two dynamically modified AES block ciphers AESD1 and AESD2, we first implement two dynamic algorithms: Algorithm 1 and Algorithm 2 in [1] to find key-dependent MDS matrices or dynamic MDS matrices from the original MDS matrices. These dynamic MDS matrices are then fed into the AES Mixcolumn transformation (how these matrices are included in the Mixcolumn depends on Algorithm 1 and Algorithm 2).

a) Executing Algorithm 1

Conduct test implementation in C/C++ programming language, compiled on Visual Studio 2015 environment. To illustrate this algorithm, program execution is built with input matrices and keys as follows. The key pair corresponding to the **scalar multiplication** is:

+ 05060708090a0b0c0d0e0f1011121314 (for MDS matrices of size 4 and 8).

+05cb174fc9d78021321ad90348f5dabca23c45acb9b4c6da (for MDS matrices of size 16).

The result of Algorithm 1 implementation is that the corresponding key-dependent MDS matrices are generated. These matrices are also integrated into the diffusion layer of AES (AESD1) to analyze several cryptographic properties such as number of fixed points and coefficient $D(A)$, number of XOR and Xtime operations. The analysis results with the above two key pairs are described in Table III.

Remark 5. Analysis of the results in Table III shows that, the cryptographic properties as Branch number, number of fixed points and coefficient $D(A)$ of dynamically modified AES diffusion layer (AESD1) with the MDS matrix obtained from Algorithm 1 is completely equivalent to the diffusion layer's the same properties with the original MDS matrix selected. For Table 3, there exists an element $(e_i f_j)$ that is a generator element of the finite field, so the cycle of the transformation also reaches a maximum of 255 (see more in [28]).

TABLE I
ORIGINAL MATRIX M FOR ALGORITHM 1, ALGORITHM 1*; MATRIX A FOR ALGORITHM 2, ALGORITHM 2*

Matrix type	Original matrix representation (M or A)	Root element, cycle of T_p	Branch number, number of XOR, Xtime	Number of fixed points $F\_(A\_0)$	Coefficient $D(A)$
44 recursive MDS matrix	//0x169 0x29 0x46 0x5A 0x85 0x4F 0xD2 0x3D 0xA5 0xCB 0x87 0xA1 0xED 0x8B 0x87 0xE6 0xCA	$0x29, 8$	5, 48, 28	2^0	2.93874e-039 $(2^{-127.9999})$
88 recursive MDS matrix	//0x169 0xA3 0x2B 0x09 0x8A 0x19 0x2B 0xD3 0x75 0x8A 0x63 0x4D 0xA9 0xCA 0xD9 0x53 0xE9 0xF1 0x1E 0xB4 0x93 0x02 0x5E 0x35 0xB8 0x07 0xCD 0xC2 0x96 0x87 0x3E 0xF3 0x15 0xEC 0xE2 0x70 0xE1 0x12 0x62 0x7D 0x1D 0x50 0x38 0x17 0xCE 0xAD 0xC6 0xA6 0x80 0x88 0xFB 0x75 0x5D 0x70 0x07 0x45 0xFF 0x91 0x84 0x8A 0x7F 0x59 0x7C 0xB5 0xDF	$0x2b, 8$	9, 232, 56	2^0	2.93874e-039 $(2^{-127.9999})$
16×16 recursive MDS matrix	//0x169 4E A9 54 75 C7 61 62 D0 45 3D D6 1C 26 16 90 4E DB F9 61 88 43 67 13 FA 7B 8F 68 08 8B C6 3A 4B C4 B3 5C 75 DE 6F 96 A0 69 78 EF DA 21 25 E8 FE AD E5 68 B3 33 17 CD 9D E0 FD A5 4F C6 92 D3 45 AB 64 A3 38 82 83 68 D3 03 EC 14 BF EB AC 2A 78 61 EB 78 42 9C 1C 95 3B 41 FA C6 10 43 64 9C 4B C4 09 4E 6C 14 B0 ED 26 A8 42 9A 74 39 ED 4A 58 3B 8F B4 E3 54 94 D8 10 E7 87 06 A5 85 A4 80 71 8D 3E B5 33 A7 18 4B C4 3D A2 D4 FE 26 AC 49 0D BD BE 21 6E 74 98 30 FF 06 75 38 58 19 D8 E3 F4 93 4B BF 37 E8 CC 55 6D CF 69 C2 40 51 36 2A 70 C3 3F 25 49 B4 C5 71 99 05 B7 EC 26 E5 6E 4B E9 03 B7 C5 CE C2 3C 1F CF F6 49 AD B1 7A 3D 49 48 16 F9 EE 4E B8 4D 6B B5 93 B2 3A 3B F2 EE CA 5F B8 B9 81 61 F1 76 62 64 C6 0F 06 51 38 0D C8 72 5F 2F 7F 99 05 1E 0F 67 96 C4 4F DA B8 2B 39 97	$0x4e, 8$	17, 912, 112	2^0	2.93874e-039 $(2^{-127.9999})$

TABLE II
ORIGINAL MATRIX B FOR ALGORITHM 2, ALGORITHM 2*

Matrix type	Original matrix representation (B)	Root element, cycle of T_p	Branch number, number of XOR, Xtime	Number of fixed points $F_{(A_0)}$	Coefficient $D(A)$
44 recursive MDS matrix	//0x169 0x14 0xe6 0xa2 0xf6 0x86 0x20 0x9e 0x73 0xe3 0xda 0xa8 0x84 0x71 0x6d 0x88 0x95	$0x2a, 8$	5, 60, 28	2^0	2.93874e-039 $(2^{-127.9999})$
88 recursive MDS matrix	//0x169 0x50 0xbf 0xef 0x2b 0x57 0xbe 0x94 0x97 0xf9 0x0a 0x1c 0xf7 0x8c 0x9a 0xfc 0x06 0x89 0xc0 0xba 0xe6 0x6c 0x83 0x59 0x35 0xfc 0xb9 0x0c 0x9b 0x91 0x69 0xa2 0x17 0x7d 0xc2 0xfa 0xbf 0x83 0xb8 0xe4 0x16 0x2d 0xfc 0x6e 0x62 0xf0 0x14 0xa1 0xc7 0x1f 0xfd 0x33 0xf5 0xfa 0xe7 0x5f 0xca 0x34 0x02 0xd4 0x1e 0x65 0x2d 0xd7 0x58	$0xef, 8$	9, 304, 56	2^0	2.93874e-039 $(2^{-127.9999})$
16×16 recursive MDS matrix	//0x169 3a ce 17 d6 3a 0a 54 26 e1 ca aa c3 75 ec 24 27 90 05 99 89 46 25 6e ab c2 10 4c 2d ad 7b 5d fc c8 5b 34 81 41 41 56 3e 5a 42 67 a6 28 7d ba f1 a3 21 cb b6 22 34 2d e1 6f cc 11 fe 78 09 41 fc c8 68 42 d3 7e 25 47 7d 10 ef bb fb fb a8 c8 ed e9 13 05 bb 3a d3 e6 83 91 c4 b3 9c ad eb df e1 b8 de 9a b0 03 ef 5b e3 2e 99 16 24 64 a0 45 3b da e7 74 7f 6a c4 24 d7 ba 9d 10 2f c2 5b 5a f2 ed 08 4e e5 92 01 54 f9 cc 1b 59 a5 6e be 0b 75 93 68 0c fe 76 6b 02 4f 01 f4 f7 07 9f 4d 4f 65 88 c8 75 8b 76 2f 8c ab 4b a7 99 a7 1b 90 2e 83 96 5c 39 b7 1d cc 85 d2 77 41 f5 c5 61 4f a1 f3 d7 8a e2 7e 60 7c 08 7e 28 1c 2f 83 f1 f1 3b a9 6d 4c df 24 13 b7 ad 4e e6 c5 ec 87 f0 5a 8c d4 47 1c a5 06 63 6d 17 a9 50 7c 2d 1d dd 6e 9f 5c 26 b6 85 02 20 e9 e9 7e 5f b8 a5 ab 0d 5c bf aa	$0x3a, 8$	17, 976, 112	2^0	2.93874e-039 $(2^{-127.9999})$

b) Executing Algorithm 2

For **Algorithm 2**, an implementation is carried out using C/C++ programming language, compiled on Visual Studio 2015. To illustrate this algorithm, the test program is built

TABLE III
KEY-DEPENDENT MDS MATRICES - RESULTS OF ALGORITHM 1 AND AND THE SPEED OF AESD1

Original MDS Matrix (M)	The key-dependent MDS matrix (M_K) according to Algorithm 1 with the key pair is: *05060708090a0b0c0d0e0f1011121314* and *05cb174fc9d78021321ad90348f5dabca23 c45acb9b4c6da*	The transformation, Cycle	Branch number, number of XOR, Xtime	Number of fixed points $F_{(A_0)}$	Coefficient $D(A)$	Encryption/ Decryption speed of the AESD1 (Mb/sec)
44 recursive MDS matrix in Table 1	//0x169 0x29 0x29 0x7a 0x8e 0xfb 0xd2 0x1b 0x3f 0xcd 0xf6 0xa1 0x10 0x1f 0xaa 0x47 0xca	T_M, 255	5, 64, 28	2^0	2.93874e-039 $(2^{-127.9999})$	58.382
88 recursive MDS matrix in Table 1	//0x169 0xa3 0xb2 0x44 0xb9 0xa3 0x2c 0x28 0xd0 0xec 0x63 0xd1 0x78 0x84 0xe2 0x5e 0xa5 0x36 0xc4 0xb4 0x78 0x8d 0x6 0xae 0x85 0x19 0x3f 0x95 0x96 0x1e 0xe9 0xf0 0x37 0x84 0x1a 0xeb 0x85 0x12 0x78 0x37 0x23 0xb9 0x47 0x82 0x6f 0x87 0xc6 0x49 0xb0 0x24 0x9c 0xea 0xda 0xe9 0xa1 0x45 0x1f 0xe4 0x4e 0xd2 0xfb 0x1f 0x8d 0x6b 0xdf	T_M, 255	9, 246, 56	2^0	2.93874e-039 $(2^{-127.9999})$	55.323
1616 recursive MDS matrix in Table 1	//0x169 4E A9 54 75 C7 61 62 D0 45 3D D6 1C 26 16 90 4E DB F9 61 88 43 67 13 FA 7B 8F 68 08 8B C6 3A 4B C4 B3 5C 75 DE 6F 96 A0 69 78 EF DA 21 25 E8 FE AD E5 68 B3 33 17 CD 9D E0 FD A5 4F C6 92 D3 45 AB 64 A3 38 82 83 68 D3 03 EC 14 BF EB AC 2A 78 61 EB 78 42 9C 1C 95 3B 41 FA C6 10 43 64 9C 4B C4 09 4E 6C 14 B0 ED 26 A8 42 9A 74 39 ED 4A 58 3B 8F B4 E3 54 94 D8 10 E7 87 06 A5 85 A4 80 71 8D 3E B5 33 A7 18 4B C4 3D A2 D4 FE 26 AC 49 0D BD BE 21 6E 74 98 30 FF 06 75 38 58 19 D8 E3 F4 93 4B BF 37 E8 CC 55 6D CF 69 C2 40 51 36 2A 70 C3 3F 25 49 B4 C5 71 99 05 B7 EC 26 E5 6E 4B E9 03 B7 C5 CE C2 3C 1F CF F6 49 AD B1 7A 3D 49 48 16 F9 EE 4E B8 4D 6B B5 93 B2 3A 3B F2 EE CA 5F B8 B9 81 61 F1 76 62 64 C6 0F 06 51 38 0D C8 72	T_M, 255	17, 1028, 112	2^0	2.93874e-039 $(2^{-127.9999})$	47.894

with matrices and input key as follows:

- The input matrices of size 4×4,8×8,16×16 are all recursive MDS matrices. In which, the first input matrices (A) is the recursive matrices proposed in Table 1; The second input matrices (B) is also a recursive matrices, as listed in Table II.

- 02 encryption keys with length 128 bits respectively for matrices of size, and 01 key of length 192 bits for matrices of size 16×16 are:

0405060708090a0b0c0d0e0f10111213,

0708090a0b0c0d0e0f10111213141516,

0f101112131415161718191a1b1c1d1e1f20212223242526

The test result of Algorithm 2 is that the corresponding key-dependent MDS matrices are generated. These matrices are integrated into the diffusion layer of AES (AESD2) to analyze several cryptographic properties such as: number of fixed points, coefficient $D(A)$ number of XOR and Xtime operations. The results of the analysis are described in Table IV. And the speed of AESD2 is given here.

Remark 6. The analysis of the results in Table V shows that the cryptographic properties such as: Branch number, number of fixed points and coefficient $D(A)$ of dynamically modified AES diffusion layer (AESD2) with key-dependent MDS matrices A_K and B_K obtained from Algorithm 2 are completely equivalent to the diffusion layer with the original MDS matrices (A, B) selected.

2. Dynamic modification of the AES block cipher based on Algorithm 1 and Algorithm 2 [1]

This section applies two dynamic algorithms Algorithm 1 and Algorithm 2 to modify the AES block cipher, using the dynamic MDS matrices generated from these two algorithms to feed the Mixcolumn transformation of AES. Dynamically modified AES based on Algorithm 1 uses a dynamic matrix for Mixcolumn in every round of AES. Dynamically modified AES based on Algorithm 2 uses two dynamic matrices for Mixcolumn in alternating rounds of the AES block cipher.

After applying the above two algorithms to AES, we evaluate the performance speed of the dynamically modified AES block ciphers AESD1 and AESD2 when implementing Algorithms 1 and Algorithm 2.

AESD1 and AESD2 use new MDS matrices created by Algorithm 1 and Algorithm 2 from the original MDS matrices selected with different sizes as shown in Tables 1 and 2. AESD1 and AESD2 are implemented according to the component functions corresponding to the operations - transformations of the AES algorithm; Multiplications

TABLE IV
KEY-DEPENDENT MDS MATRICES - RESULTS OF ALGORITHM 2 AND THE SPEED OF AESD2

Original MDS Matrix (A, B)	The key-dependent MDS matrix (A_K adn B_K) with keys corresponding to sizes of 4×4,8×8,16×16 are: *0405060708090a0b0c0d0e0f10111213 0708090a0b0c0d0e0f10111213141516 0f101112131415161718191a1b1c1d1e1 f20212223242526*	The transformation, Cycle	Branch number, number of XOR, Xtime	Number of fixed points $F_{(A_0)}$	Coefficient $D(A)$	Encryption/ Decryption speed of the AESD2 (Mb/sec)
Recursive Matrices A,B of size 4×4 in Table 1, Table 2	A_K 0x8c 0x9b 0xa2 0xff 0xda 0x0c 0xf5 0x32 0x24 0xfb 0x22 0xfd 0xab 0xfb 0xb8 0x25	$T_P, 8$ $\left(A_K = A_{d^{2^1}}\right)$	5, 68, 28	2^0	2.93874e-039 $(2^{-127.9999})$	56.287
	B_K 0x14 0xcf 0x8e 0x96 0x49 0x20 0x9e 0x73 0xdb 0xda 0xa8 0x84 0xec 0x6d 0x88 0x95 (Multiply row 1 column 1)	$T_M, 15$ $(e = 0x13, f = 0x6a)$	5, 58, 28	2^0	2.93874e-039 $(2^{-127.9999})$	56.287
Recursive Matrices A,B of size 8×8 in Table 1, Table 2	A_K 0x26 0x88 0x41 0xaa 0x28 0x88 0x0d 0x3a 0xaa 0x47 0xde 0x62 0x25 0x49 0xe3 0xed 0xc4 0x3d 0x5a 0x82 0x04 0xb2 0xb5 0x0a 0x15 0x30 0x65 0x93 0xfb 0xf0 0xc0 0x78 0xfc 0xa8 0x2b 0xad 0x6d 0x46 0x7a 0x38 0xe6 0xe4 0x7c 0x35 0x72 0x75 0x37 0xee 0xae 0x80 0x3a 0xb7 0x2b 0x15 0x9e 0x90 0x86 0xfe 0xaa 0x7e 0xa7 0x7b 0x5b 0x5d	$T_P, 8$ $\left(A_K = A_{d^{2^1}}\right)$	9, 247, 56	2^0	2.93874e-039 $(2^{-127.9999})$	42.778
	B_K 0x50 0x81 0xac 0x98 0x4f 0x97 0x19 0x23 0xda 0x0a 0x1c 0xf7 0x8c 0x9a 0xfc 0x06 0xfd 0xc0 0xba 0xe6 0x6c 0x83 0x59 0x35 0xa9 0xb9 0x0c 0x9b 0x91 0x69 0xa2 0x17 0x16 0xc2 0xfa 0xbf 0x83 0xb8 0xe4 0x16 0x50 0xfc 0x6e 0x62 0xf0 0x14 0xa1 0xc7 0x11 0xfd 0x33 0xf5 0xfa 0xe7 0x5f 0xca 0xc4 0x02 0xd4 0x1e 0x65 0x2d 0xd7 0x58 (Multiply row 1 column 1)	$T_M, 85$ $(e = 0x16, f = 0x52)$	9, 248, 56	2^0	2.93874e-039 $(2^{-127.9999})$	42.778
Recursive Matrices A,B of size 16×16 in Table 1, Table 2	A_K de 9b 09 ac e1 aa c2 8f 0d 18 e5 bd ce 6f 80 de 5c 91 aa 3e 67 c0 6d f9 7d 55 10 bb 56 e0 73 dc 89 4c b2 ac 5e 7b ea 21 11 15 fe 5d a5 a6 95 fa 98 2c 10 4c c9 6e 33 39 2e 92 23 df e0 e9 e7 0d f2 a8 49 1a ec ed 10 e7 68 96 06 f4 fd 99 76 15 aa fd 15 66 38 bd 82 72 0e f9 e0 05 67 a8 38 dc 89 ba de 13 06 24 97 ce 9a 66 52 ad 1b 97 dd b1 72 55 27 46 09 83 34 05 45 ee 6a 23 87 22 85 af 3c 70 26 c9 4a be dc 89 18 48 8c fa ce 99 b5 b9 9d f5 a5 7a ad 3b a1 fb 6a ac 1a b1 bf 34 46 28 e8 dc f4 ca 95 32 08 12 5a 11 e3 0f 0b cb 76 ae e2 71 a6 b5 27 88 af 3a 02 4f 96 ce 2c 7a dc 94 68 4f 88 5b e3 19 d5 5a 41 b5 98 25 7c 18 b5 b4 6f 91 ff de 9f b6 78 26 e8 4d 73 72 42 ff 58 da 9f 9e 84 aa 2a c4 c2 a8 e0 d0 6a 0b 1a b9 31 c7 da 74 7e 3a 02 d4 d0 c0 ea 89 df 5d 9f 77 1b eb	$T_P, 8$ $\left(A_K = A_{d^{2^1}}\right)$	17, 1021, 112	2^0	2.93874e-039 $(2^{-127.9999})$	38.843
	B_K 3a a3 40 48 aa 15 30 d9 05 3b 2c 44 1b e2 95 ff c5 05 99 89 46 25 6e ab c2 10 4c 2d ad 7b 5d fc f8 5b 34 81 41 41 56 3e 5a 42 67 a6 28 7d ba f1 ce 21 cb b6 22 34 2d e1 6f cc 11 fe 78 09 41 fc f8 68 42 d3 7e 25 47 7d 10 ef bb fb fb a8 c8 ed 72 13 05 bb 3a d3 e6 83 91 c4 b3 9c ad eb df e1 bb de 9a b0 03 ef 5b e3 2e 99 16 24 64 a0 45 3b 79 e7 74 7f 6a c4 24 d7 ba 9d 10 2f c2 5b 5a f2 a6 08 4e e5 92 01 54 f9 cc 1b 59 a5 6e be 0b 75 9a 68 0c fe 76 6b 02 4f 01 f4 f7 07 9f 4d 4f 65 ef c8 75 8b 76 2f 8c ab 4b a7 99 a7 1b 90 2e 83 7b 5c 39 b7 1d cc 85 d2 77 41 f5 c5 61 4f a1 f3 59 8a e2 7e 60 7c 08 7e 28 1c 2f 83 f1 f1 3b a9 88 4c df 24 13 b7 ad 4e e6 c5 ec 87 f0 5a 8c d4 9c 1c a5 06 63 6d 17 a9 50 7c 2d 1d dd 6e 9f 5c 01 b6 85 02 20 e9 e9 7e 5f b8 a5 ab 0d 5c bf aa (Multiply row 1 column 1)	$T_M, 17$ $(e = 0x26, f = 0x35)$	17, 1029, 112	2^0	2.93874e-039 $(2^{-127.9999})$	38.843

by the elements of the MDS matrix are performed by looking up the finite field multiplication table as the usual implementation of Vincent Rijmen in [5]. The test results of AESD1 and AESD2 encryption/decryption speed evaluation with 128-bit key are described in Table 3 and Table 4 respectively.

To be able to compare the speed of dynamically modified AES block ciphers with AES, we also evaluate the execution speed of the AES block cipher in Table V.

Remark 7. Analysis of the results in Table 3, Table 4 and comparison with the results in Table 5 shows that the performance speed of dynamically modified AES algorithm

TABLE V
PERFORMANCE EVALUATION OF AES BLOCK CIPHER

		Optimal implementation according to Brian Gladman		Normal implementation			
	Matrix representation 4×4	Key length (bits)	Encryption/ Decryption Speed (Mb/sec)	Key length (bits)	Encryption/ Decryption Speed of AES_4_4 (Mb/sec)	Number of multiplications (look up table)/1 round	Number of additions/1 round
Original MDS Matrix	//0x11B 0x02 0x03 0x01 0x01 0x01 0x02 0x03 0x01 0x01 0x01 0x02 0x03 0x03 0x01 0x01 0x02	128	886.512	128	59.998	24	48
		192	753.311	192	59.820	24	48
		256	669.231	256	58.812	24	48

TABLE VI
PERFORMANCE RESULTS FOR ALGORITHM 1* AND SPEED OF AESD1*

Original Matrix	Key value	First two bits	Transformation	Set of secret parameters for transformation	Encryption/ Decryption Speed of AESD1* (Mb/s)
4×4 recursive MDS matrix in Table 1	27F12130405060708090A0B0C0D0E0F0 (128 bit)	01	T_P	$l = 4, r - l = 4$	38.898
	71F12130405060708090A0B0C0D0E0F0 (128 bit)	11	T_M	$(e_0, e_1, e_2, e_3) = (c5, 87, c4, 80)$	45.028
8×8 recursive MDS matrix in Table 1	EFFF0130405060708090A0B0C0D0E0F0 (128 bit)	01	T_P	$l = 7, r - l = 1$	31.215
	19FF0130405060708090A0B0C0D0E0F0 (128 bit)	10	T_M	$(e_0, e_1, e_2, e_3, e_4, e_5, e_6, e_7) = (c6, 37, c4, 80, 81, c1, 01, 42)$	36.305
16×16 recursive MDS matrix in Table 1	E0FF2130405060708090A0B0C0D0E0F 00111213241516171 (192 bit)	01	T_P	$l = 3, r - l = 5$	23.113
	71FF2130405060708090A0B0C0D0E0F 00111213241516171 (192 bit)	11	T_M	$(e_0, e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_8, e_9, e_{10}, e_{11}, e_{12}, e_{13}, e_{14}, e_{15}) = (c5, bf, c4, 80, 81, c1, 01, 42, 82, c2, 02, 43, 83, c3, 03, 44)$	20.878

AESD1 (Algorithm 1) is equivalent to AES algorithm, while the AESD2 algorithm (Algorithm 2) is not significantly slower. While the security of AESD1 and AESD2 is significantly increased compared to AES.

3. Executing Algorithm 1* and Algorithm 2* and dynamically modifying the AES block cipher

In order to perform two dynamically modified AES block ciphers AESD1* and AESD2*, we first perform two dynamic algorithms Algorithm 1* and Algorithm 2* to find out the secret parameters (number of direct exponent; scalar multiplier vectors) to include the encryption and decryption of AESD1* and AESD2* in a way that utilizes the MDS matrices of the original SPN block cipher.

In this section, the test results of AESD1* by Algorithm 1* and AESD2* by Algorithm 2* with the idea of taking advantage of the original MDS matrices are described as follows:

• The original matrices of size 4 , 8, 16 are the selected as recursive MDS matrices in Table I, Table II.

• Test results with different keys (length of 128 bits for matrices of size 4 and 8 , and length of 192 bits for matrices of size) according to Algorithm 1* and the speed of AESD1* are described in Table VI.

• Test results with different keys (length of 128 bits for matrices of size 4 and 8 , and length of 192 bits for matrices of size 16) according to Algorithm 2* and the speed of AESD2* are described in Table VII.

Remark 8. With AESD1* and AESD2* using Algorithm 1* and Algorithm 2* respectively, the encryption/decryption speeds of these dynamic block ciphers are shown in Table 6 and Table 7. It can be seen that the speed of these block ciphers is not significantly slower than the AESD1 and AESD2 using Algorithms 1 and Algorithm 2.

Table 8 gives a speed comparison between dynamically modified AES algorithms. Although the AESD1* and AESD2* algorithms are slower than the AESD1 and AESD2 algorithms, the advantage here is: when utilizing the original MDS matrices of the original SPN block cipher according to Algorithm 1* and Algorithm 2*, we do not need to compute, store and protect for dynamic MDS matrices. Especially in systems where many people are

TABLE VII
PERFORMANCE RESULTS FOR ALGORITHM 1* AND SPEED OF AESD2*

Original Matrix	Key value	First two bits	Trans formation	Set of secret parameters for transformation	Encryption/ Decryption Speed of AESD1* (Mb/s)
4×4 recursive MDS matrix in Table 1	17F12130405060708090A0B0C0D0E0F0 (128 bit)	10	T_P	$l_1 = 4, r - l_1 = 4$ và $l_2 = 7, r - l_2 = 1$	37.683
	2FF12130405060708090A0B0C0D0E0F0 (128 bit)	01	T_M	$(e_{10}, e_{11}, e_{12}, e_{13}) = (04, 01, 01, 01)$ $(e_{20}, e_{21}, e_{22}, e_{23}) = (0f, 01, 0f, 01)$	44.215
	3DF12130405060708090A0B0C0D0E0BE (128 bit)	11	T_P, T_M	$l_1 = 2, r - l_1 = 6(e_{10}, e_{11}, e_{12}, e_{13}) = (01, 01, eb, 01)$	40.231
8×8 recursive MDS matrix in Table 1	1EFE2130405060708090A0B0C0D0E0F0 (128 bit)	10	T_P	$l_1 = 0, r - l_1 = 8 v l_2 = 7, r - l_2 = 1$	30.124
	2EFE2130405060708090A0B0C0D0E0FC (128 bit)	01	T_M	$(e_{10}, e_{11}, e_{12}, e_{13}, e_{14}, e_{15}, e_{16}, e_{17} = (01, 01, 01, f8, 01, 01, 01, 01)$ $(e_{20}, e_{21}, e_{22}, e_{23}, e_{24}, e_{25}, e_{26}, e_{27} = (01, 01, 01, 01, 01, 01, cf, 01)$	35.415
	3DFE2130405060708090A0B0C0D0E0FC (128 bit)	11	T_P, T_M	$l_1 = 4, r - l_1 = 4(e_10, e_11, e_12, e_13, e_14, e_15, e_16, e_17 = (01, 01, 01, 01, 01, 01, 01, cf)$	31.895
16×16 recursive MDS matrix in Table 1	1BFF2130405060708090A0B0C0D0E 0F00111213241516171 (192 bit)	10	T_P	$l_1 = 4, r - l_1 = 4$ và $l_2 = 7, r - l_2 = 1$	17.905
	2DFF2130405060708090A0B0C0D0 E0F00111213241516171 (192 bit)	01	T_M	$(e_0, \ldots, e_2, \ldots, e_{15}) = (01, \ldots, \ldots, f4)(e_0, \ldots, e_14, e_{15}) = (01, \ldots, 0f, 01)$	20.625
	71FF2130405060708090A0B0C0D 0E0F00111213241516171 (192 bit)	11	T_P, T_M	$l_1 = 5, r - l_1 = 3(e_0, \ldots, e_12, \ldots, e_{15}) = (01, \ldots, 0f, \ldots, 01)$	18.830

TABLE VIII
SPEED COMPARISON OF DYNAMICALLY MODIFIED AES BLOCK CIPHERS

Original Matrix	Key value	First two bits	Trans formation	Set of secret parameters for transformation
4×4 recursive MDS matrix	58.382	56.287	38.898	37.683
8×8 recursive MDS matrix	55.323	42.778	31.215	30.124
16×16 recursive MDS matrix	47.898	38.843	23.113	17.905

secretly communicating with each other - the number of dynamic MDS matrices is very large, this method will be very useful. We can save storage space for dynamic MDS matrices, not having to calculate and protect these matrices anymore.

V. SECURITY ANALYSIS AND IMPLEMENTATION RESOURCES FOR DYNAMICALLY MODIFIED AES BLOCK CIPHERS

Regarding the security of dynamic AES block ciphers, we analyzed it in detail in [1]. Animating the diffusion layer makes cryptanalysis much more difficult because it is not known which MDS matrix is used in the Mixcolumn transformation. Shannon [31] showed that the unicity distance of the cryptosystem is calculated by the following formula: $U = \frac{H(K)}{D}$

where $H(K)$ is the entropy of the key, and D is the redundancy of the source in bits per character. Another definition of U is the average number of plaintext characters required for the redundancy (in bits) to exceed the key length. If the length of the encrypted message is less than this unicity distance, it is impossible to decide the unique key or plaintext from knowledge of the ciphertext alone (a lot of dummy keys will be found, without knowing which is the real key). It is quite possible that cryptosystems have an infinite value of U, and Shannon calls such cryptosystems ideal ones.

In the case of a static AES block cipher, the primary keyspace is the space of the original secret key used to encrypt the data. However, with dynamic AES block ciphers, the key space includes not only the secret key, but also the dynamic diffusion layer (or dynamic MDS matrix). Thus, the key space in the case of dynamic AES has been increased by the space of dynamic MDS matrices. In fact, if we use direct exponentiation, the number of dynamic matrices in $GF\left(2^8\right)$ is 8, if we use scalar multiplication,

the number of dynamic MDS matrices is about 255. Thus, the number of dynamic MDS matrices for the diffusion layer when combining these two transformations can be up to approximately 2^{11}. This number is not too large in theory, but it is very significant in practice.

For example, in order to perform strong attacks on block ciphers such as linear attacks, differential attacks, cryptanalysts must collect a lot of plaintext/ciphertext pairs, like DES algorithm, they must obtain 2^{47} clear code pair. Then if using dynamic block cipher algorithm, they have to collect about 2^{58} plaintext/ciphertext pairs. This is a very large number in practice, which will make it much more difficult for cryptanalysts to attack dynamic block ciphers.

Even cryptanalysts don't know when to use which dynamic algorithm: Algorithm 1, Algorithm 2, Algorithm 1* or Algorithm 2*. They also don't know which dynamic matrix will be used in which round. Besides, in order to perform strong attacks on block ciphers such as linear attacks, differential attacks, cryptanalysts must collect a lot of plain text/cipher text pairs, such as the DES algorithm, which they must obtain 2^{43} pairs of plain text/cipher text. In short, when we use dynamic algorithms, cryptanalysis will be many times more difficult than normal AES cryptanalysis. That will contribute to improving the security of the AES block cipher.

However, it should be recalled here that, when utilizing the MDS matrices of the original SPN block cipher according to Algorithm 1* and Algorithm 2*, there is no need to compute, store and protect the dynamic MDS matrices. Especially in systems where many people secretly communicate with each other - the number of dynamic MDS matrices is very large, this method is very useful.

The advantage of Algorithms 1*, 2* over Algorithms 1, 2 is: cryptographic parties do not need to specifically compute dynamic MDS matrices, nor do they have to store and protect the matrices. In particular, in a multi-user encryption system, if the static block cipher's version that utilizes MDS matrices is applied, the user does not have to compute and store and protect multiple dynamic MDS matrices at the same time. Also Algorithm 1* and Algorithm 2* are very simple, only need to calculate output as parameters, not dynamic MDS matrices, so Algorithm 1* and Algorithm 2* are faster than Algorithm 1 and Algorithm 2. However, their limitation is that for each round of encryption, we have to apply additional transformations on the input and output of the diffusion layer (in order to take advantage of the MDS matrices of static block ciphers), which will reduce the software execution of dynamic block ciphers, for example when applying them to AES modifications (AESD1*, AESD2*).

About implementation resources for dynamically modified AES block ciphers:

- For the optimal implementation method by reducing the table lookups, it is completely equivalent to AES (all uses 04 lookup tables of the same size).

- With the usual implementation method: For the application of the MDS matrix, the resource usage is completely equivalent to AES. For and MDS matrices, the amount of memory required is slightly larger because the size of the matrices is larger.

VI. CONCLUSION

In this paper, we execute the two dynamic diffusion layer algorithms proposed in [1] with recursive MDS matrices of size 4, 8, and 16 then evaluate the cryptographic properties of the obtained dynamic MDS matrices including branch number, number of fixed points, coefficient of fixed points, number of XOR and Xtime operations. Then, we apply these two dynamic algorithms to dynamically modify the AES block cipher and evaluate the execution speed of the dynamic AES block ciphers. We propose two new diffusion layer dynamic algorithms, implement these two algorithms and modify dynamically AES block cipher according to these two algorithms, and evaluate the speed of dynamic AES block ciphers. We also evaluate the security and implementation resources for dynamically modified AES block ciphers. The proposed diffusion layer dynamic algorithms contribute to improving the security of the AES block cipher against many current strong attacks.

REFERENCES

[1] Luong, T. T, "Building the dynamic diffusion layer for SPN block ciphers based on direct exponent and scalar multiplication", *Journal of Science and Technology on Information security*, 1(15), 38-45, 2022.

[2] Ishukova, E.A., Krasovki, A.V, Babenko L. K, "Security assessment of Kuznhyechik block cipher using key-related method", *basic research* (11-4) 698-703, 2016 (Russian).

[3] Tomanenko E. A, "Results of testing the possibility of using hybrid cryptography on the basis of symmetric algorithms", 22, 2022 (Russian).

[4] Dolmatov, V, "GOST R", *Block Cipher*" Kuznyechik", 34-12, 2015.

[5] Deamen, J., & Rijmen, "The design of rijndael", *In AES-The Advanced Encryption Standard*, Springer-Verlag, 2002.

[6] Daemen, J., & Rijmen, Aes proposal: Rijndael (version 2), nist aes website, 1999

[7] Zenzin O.X. Ivanov M.A, AES Encryption Standard. *Finite field. M.: Kudits-Obraz* 176, 15, 2002 (Russian)

[8] Ayoub.F, "Probabilistic completeness of substitution-permutation encryption networks". *IEE Proceedings E (Computers and Digital Techniques)*, 129(5), 195-199, 1982.

[9] Heys, H. M., & Tavares, S. E, "Avalanche characteristics of substitution-permutation encryption networks", *IEEE Transactions on Computers*, 44(9), 1131-1139, 1995.

[10] Heys. H. M., & Tavares. S. E, Substitution-permutation networks resistant to differential and linear cryptanalysis. *Journal of cryptology*, 9(1), 1-19, 1996.
[11] Schneier B., Kelsey J., Whiting D., Wagner D., Hall C. and Ferguson N, “Twofish: a 128-bit block cipher”, *NIST AES Proposal*, vol. 15, 1998.
[12] Schneier B., Kelsey J., Whiting D., Wagner D., Hall C. and Ferguson N, “The twofish encryption algorithm”, *Wiley*, 1999.
[13] Abd-ElGhafar, I., Rohiem, A., Diaa, A., & Mohammed. F, "Generation of AES key dependent S-boxes using RC4 algorithm", In *International Conference on Aerospace Sciences and Aviation Technology* (Vol. 13, No. AEROSPACE SCIENCES & AVIATION TECHNOLOGY, ASAT-13, May 26–28, 2009.
[14] Agarwal, P., Singh, A., & Kilicman. A, " Development of key-dependent dynamic S-boxes with dynamic irreducible polynomial and affine constant", *Advances in Mechanical Engineering*, 10(7), 2018.
[15] Assafli, H. T., & Hashim, I. A, "Generation and Evaluation of a New Time-Dependent Dynamic S-Box Algorithm for AES Block Cipher Cryptosystems", *In IOP Conference Series: Materials Science and Engineering*, 978, 1, 2020.
[16] Hosseinkhani, R., & Javadi, H. H. S, "Using cipher key to generate dynamic S-box in AES cipher system", *International Journal of Computer Science and Security (IJCSS)*, 6(1), 19-28, 2012.
[17] Juremi, J., Mahmod, R., Zukarnain, Z. A., & Yasin.S.M, "Modified AES s-box based on determinant matrix algorithm", *Int. J. Adv. Res. Comput. Sci. Softw. Eng*., 7(1), 110-116, 2017.
[18] Kazlauskas, K., & Kazlauskas. J, "Key-dependent S-box generation in AES block cipher system", *Informatica*, 20(1), 23-34, 2009.
[19] KHAMLICH. E, "Implementation of stronger AES by using Dynamic S-Box dependent of Master Key", *Journal of Theoretical and Applied Information Technology*, 53(2), 2013.
[20] Mahmoud, E. M., Abd, A., El Hafez, T. A. E., & El Hafez, T. A, Dynamic AES-128 with key-dependent S-box, 2013.
[21] Murtaza G., Khan A.A., Alam S.W. and Farooqi A, “Fortification of aes with dynamic mix-column transformation,” *IACR Cryptology ePrint Archive*, 2011.
[22] I.A. Ismil, Galal H. Galal - Edeen, Sherif Khattab and Mohamed Abd ElHamid I. Moustafa El Bahtity, "Performance examination of AES encrytion algorithm with constant and dynamic rotation", *International Journal of Reviews in Computing*, ISSN: 2076-3328, 31st December 2012. Vol. 12, 2012.
[23] Auday H. Al-Wattar, Ramlan Mahmod, Zuriati Ahmad Zukarnain and NurIzura Udzir, "A new DNA based approach of generating key dependent Mixcolumns transformation", *International Journal of Computer Networks & Communications (IJCNC)* Vol.7, No.2, 2015.
[24] Adnan Ibrahem Salih, Ashwak Alabaich, Ammar Yaseen Tuama, "Enhancing advance encryption standard security based on dual dynamic XOR table and mixcolumns transformation", *Indonesian Journal of Electrical Engineering and Computer Science* Vol. 19, No. 3, 2020.
[25] Luong, T. T., Cuong, N. N., & Tho, H. D, "On the calculation of input and output for dynamic MDS matrices in diffusion layer of SPN block ciphers", *In 2017 International Conference on Information and Communications (ICIC)* (pp. 281-287), 2017.
[26] Luong, T. T., Cuong, N. N., & Tho, H. D, "On the calculation of input and output for dynamic MDS matrices in diffusion layer of SPN block ciphers", *In 2017 International Conference on Information and Communications (ICIC)*, 2017
[27] Luong, T. T., & Cuong, N. N, "The preservation of good cryptographyic properties of mds matrix under direct exponent transformation", *Journal of Computer Science and Cybernetics*, 31(4), 291-291, 2015.
[28] Luong, T. T., & Cuong, N. N, "DIRECT EXPONENT AND SCALAR MULTIPLICATION TRANSFORMATIONS OF MDS MATRICES: SOME GOOD CRYPTOGRAPHIC RESULTS FOR DYNAMIC DIFFUSION LAYERS OF BLOCK CIPHERS", *Journal of Computer Science and Cybernetics*, 32(1), 1-17, 2016.
[29] Luong, T. T., Cuong, N. N., & Dung, L. T, "The preservation of the coefficient of fixed points of an MDS matrix under direct exponent transformation", *In 2015 International Conference on Advanced Technologies for Communications (ATC)* (pp. 111-116), 2015.
[30] Luong, T. T., Cuong, N. N., & Dung, L. T, "A new statement about direct exponent of an MDS matrix in block ciphers", *In 2015 Seventh International Conference on Knowledge and Systems Engineering (KSE)* (pp. 340-343), 2015
[31] Shannon C.E, “Communication theory of secrecy systems,” *Bell System Technical Journal*, vol. 28, no. 4, pp. 656-715, 1949.

Tran Thi Luong received Bachelor of Mathematics and Informatics of Ha Noi university of Sicence in 2006; Master degreee in cryptographic technique at Academy of Cryptographic Techniques in 2012; Ph.D degree in cryptographic technique at Academy of Cryptographic Techniques in 2019;
Recent research direction: Cryptogaphy, Coding theory and Information Security.
Email: luongtranhong@gmail.com

Research on Deep Learning and Its Application in Stock Price Prediction

Hoang Van Hai[1], Dang Thi Thu Hien[2]
[1]Thai Nguyen University of Economics and Business Administration, Tan Thinh ward, Thai Nguyen city, Vietnam
[2]Thuy Loi University, 175 Tay Son, Dong Da, Hanoi, Vietnam
Correspondence: Hoang Van Hai (hoanghai@tueba.edu.vn)
Communication: received 20 Nov 2022, revised 16 Feb 2023, accepted 31 Mar 2023
Digital Object Identifier: 10.32913/mic-ict-research.v2023.n1.1136

Abstract: The article studies the problem of forecasting the closing price of a stock based on historical data of a previous day. The paper uses and compares algorithms based on deep learning such as LSTM, BiLSTM, and CNN. The dataset includes data on price, trading volume and some technical indicators related to VCB, MSN, and HPG shares. The results show that CNN performs better for predicting the next day's closing price than the other architectures.

Keywords: *deep learning, stock price prediction, LSTM, BiLSTM, CNN.*

I. Introduction

The stock market has emerged since the 17th century and it is one of the most mercantile entities in the world. The mercantilist aspect of the stock market is the unpredictability of the companies 'stock prices because these prices are determined only by people who are willing to pay for shares of these companies. For attracting people to invest in a company by buying its stock, that company needs to have a good reputation in the public if there is any change in the way people perceive the company, it could positively or negatively affect its stock prices. Because of the general interest in making profits, researchers have tried for decades to predict price movements in the stock market.

Researches on stock market prediction mainly focuses on proposing methods of predicting stock trends effectively. Previous studies can be divided into three categories: feature-oriented methods, model- oriented methods and integration - oriented methods. Feature-oriented methods mainly use statistical-based techniques such as principal component analysis (PCA), independent components analysis (ICA), and information gains (IG) to select features efficiency [10][17] [21]. The performance of a given model will be improved if low relevant features in the input are removed. The model-driven approach focuses on improving the fitting ability of the model. Support vector machine (SVM) has been demonstrated to be very effective for stock market prediction [9]. In recent years, deep learning models have been shown many successful applications in computer vision and natural language processing (NLP) because of their powerful feature extraction capabilities. Therefore, deep learning has also been applied in stock market prediction [1][2][6][7][13][22][24]. Integration-oriented methods often combine various artificial intelligence models or statistical-based techniques to predict the stock market [11][18].

The article applies three deep learning methods, namely Long Short-Term Memory (LSTM), Bidirectional Long Short-Term Memory (BiLSTM), and Convolutional Neural Network (CNN) to predict the stock price of Joint Stock Commercial Bank for Foreign Trade of Vietnam (VCB), Masan Group Corporation (MSN) and Hoa Phat Group Joint Stock Company (HPG). Besides stock data including opening price, closing price, highest price, lowest price, and volume, the article added technical indicators to the analysis to be able to simulate cases of sharp price drops or price spikes due to the influence of exogenous shocks (such as epidemics, wars...). Data is downloaded from investing.com.

The article is organized as follows: part II briefly presents related works, part III presents the design of predictive models, part IV presents data processing and experimental results.

II. Related Works

Althelaya et al in [2] used some variations of LSTM, Gated Recurrent Unit (GRU) to forecast the daily closing price of the S&P500 index based on historical data of the previous 10 days. Specifically, the authors used a bidirectional LSTM (BiLSTM), a bidirectional GRU (BiGRU),

LSTM and GRU with 2 layers stacked to form S-LSTM and S-GRU architectures respectively. The data was taken from Yahoo Finance for the period from 01/01/ 2010 to 30/11/ 2017, and includes five input variables: closing price, opening price, lowest price, highest price, and volume. Data was preprocessed to remove the trend and seasonality by differencing and then normalized between 0 and 1 using min-max normalization. The authors then compared and evaluated the performance of these models. As a result, the S-LSTM architecture demonstrated the highest predictive performance. However, stabilizing time series data removes useful dynamic trends and relationships, and thus we may exclude valuable information. In addition, the study does not include technical indicators in the analysis while they are often referred to by investors to make decisions in buying/selling stocks.

Hoseinzade and Haratizadeh in [7] proposed two CNN-based methods to predict the next day's movement for the S&P 500, NASDAQ, DJI, NYSE and RUSSELL index using information from the previous 60 days. The first method, 2D-CNNpred, is based on the assumption that the actual mapping function from history to the future is an exact function for many markets. Therefore, a single model can predict the future of the market based on its own history. However, to extract the desired mapping function, that model needs to be trained by samples from different markets. In 2D-CNNpred, the input data was aggregated and fed to a specially designed CNN as a two-dimensional tensor consisting of the historical data and features dimension. In the second method, 3D-CNNpred, instead of training a single predictive model that can predict the future of each market based on its own historical data, the research used features from historical information of multiple markets to train a separate prediction model for each market. In 3D-CNNpred, the input data was aggregated and fed to a specially designed CNN as a three-dimensional tensor consisting of historical data, markets, and features. The dataset includes 82 variables for the period from January 2010 to October 2017. This set of variables can be classified into eight different groups, which are fundamental variables, technical indicators, world stock market indices, the exchange rate of the US dollar to other currencies, commodities, data of big US companies, future contracts, and other useful variables. The evaluation shows a significant improvement in the performance of the proposed models compared to baseline algorithms incorporated in the study. Since the research used commodity indices of the stock market of the United States, there will be elements in common among these indices. Therefore, using 2D-CNNpred with a single model that can predict the future of different market based on their own history or 3D-CNNpred with shared hyperparameters among different models for different markets are reasonable. The proof is that these two proposed models both offer better predictive results than the baseline algorithms included in the research. However, if the indexes are replaced by stocks of companies operating in different sectors of an economy, finding the optimal architecture for each stock may be the most effective method to improve prediction accuracy.

Sethia and Raut in [16] used LSTM, GRU, Artificial Neural Network (ANN), and SVM to predict stock prices for t+ 5th day based on historical data of the five previous trading days and thus provide a buying/selling strategy for the Standard's and Poor's 500 index. Data for the period from 03/01/2000 to 30/10/ 2017 taken from Yahoo finance includes opening price, highest price, lowest price, closing price, volume and a number of technical indicators. Independent Components Analysis (ICA) was used to reduce the size of this dataset. A performance comparison between the LSTM, GRU, ANN and SVM models was performed, and the LSTM and GRU models outperformed the remaining models. This result may come from the fact that the authors used ICA in data preprocessing, but because deep learning is very powerful in feature extraction, it is possible that this ICA technique does not contribute much to the results. The key may be that both LSTM and GRU are variants of Recurrent Neural Network (RNN) that is better suited for time series inputs than the other algorithms.

The commonalities of the aforementioned studies are assesments that made on indices, so a same model may not have the same performance if applied to a particular stock. In addition, the choice of window size in the studies has been neither mentioned nor supported by a statistical basis.

III. PROPOSED MODELS

The relationship between variables related to the stock is nonlinear. However, machine learning algorithms are not capable of learning all functions. This limits the problems that these algorithms can solve regarding the complex relationships between variables so stock prediction is a challenge with traditional machine learning methods [3]. Deep learning which is very powerful in feature selection and capable of learning any nonlinear function seems to be a promising approach to this challenge [3]. In deep learning, Artificial Neural Network (ANN), due to having no feedback loop, is only suitable for input data where the data points are independent of each other. Thus, it cannot capture sequential information in the input where a single data point depends on previous observations Therefore, it is difficult for ANN to give an accurate forecast as working with time series [3]. Recurrent Neural Network

(RNN) has a concept of memory which helps them to store the state or information of previous inputs to produce the next output of the sequence, thus more suitable for working with time series in general and stock data in particular [3]. Although RNN works well on time series data, it is difficult to avoid the long-term dependency matter due to the vanishing/exploding problems. Long short-term memory (LSTM) and Bidirectional long short-term memory (BiLSTM) are among the variants of RNN. Furthermore, LSTM was introduced as an efficient way to solve the vanishing/exploding gradient problem by having memory cells to retain time-related information [14].

The article uses an optimizer of stochastic gradient descent (SGD) to train LSTM and BiLSTM model with the expectation that higher performance and faster training can be achieved for related problems.

The formula of SGD as follows [19]:

$$\theta = \theta - \eta \nabla_\theta J\left(\theta; x^{(i:i+n)}; y^{(i:i+n)}\right) \tag{1}$$

Where θ is a vector whose components are the weights to be estimated; η is the learning rate; x^i, y^i is a single training data point and its corresponding label, respectively; n is batch size; and $\nabla_\theta j(\theta; x^{(i:i+n)}; y^{(i:i+n)})$ is the derivative of the loss function at θ. The article chooses n equal to 32.

However, one of the challenges faced by SGD is that when the objective function is not convex, it almost certainly converges to a local minimum. To overcome this limitation of the algorithm, the article uses SGD with momentum.

The formula of momentum as follows [19]:

$$v_{t+1} = \mu v_t - \varepsilon \nabla J(\theta_t) \tag{2}$$

$$\theta_{t+1} = \theta_t + v_{t+1} \tag{3}$$

Where v is velocity or momentum, ε is the learning rate, $\mu \in [0, 1]$ is the momentum coefficient with common values of 0.9 and 0.99, and $\nabla J(\theta_t)$ is the derivative at θ_t . The article chooses μ equal to 0.9.

Momentum along with the learning rate can improve the ability to detect a better set of weights in fewer training epochs. In practice, it is necessary to gradually reduce the learning rate over time. This is because SGD introduces a noise source that does not disappear even when we reach the minimum. Therefore, the paper uses a descending learning rate based on the number of epochs.

The formula for calculating the decreasing learning rate [5]:

$$learningrate = learningrate * 1/(1 + daycay * \#epoch) \tag{4}$$

$$decay = learningrate/epoch \tag{5}$$

The article chooses epoch equal to 200. The initial value of the learning rate will be determined through experiment.

The SGD algorithm with momentum solves the problem that gradient descent does not reach the global minimum, but only stops at the local minimum. However, when approaching the target, it still takes a long time to oscillate back and forth before coming to a complete stop, which is explained by the presence of momentum [20].

To overcome this drawback, depending on the characteristics of the data of each stock, Nesterov's Accelerated Gradient (NAG) can be used in the hope that it will help the algorithm become smarter to know where it proceeds to and slows down when approaching the global minimum. Whether or not to use NAG will be determined through experiment.

The Formula for calculating NAG [19]:

$$v_{t+1} = \mu v_t - \varepsilon \nabla J(\theta_t + \mu v_t) \tag{6}$$

$$\theta_{t+1} = \theta_t + v_{t+1} \tag{7}$$

NAG changes v in a faster and more responsible way, allowing it to behave more consistently than momentum in many situations, especially for higher μ values.

Early Stopping is used to help stop training when there is no improvement in the loss function value after a number of epochs.

Dropout can be used after each hidden layer. This is a fine-tuning method that can effectively reduce model overfitting and improve the model's prediction performance. Whether or not to use this technique and, if used, what the dropout rate is will be determined through experiment.

Convolutional neural network (CNN) is well suited to data with spatial relationships among observations [3], and thus can be suitable for time-relational data such as time series. In addition, CNN with the parameter sharing characteristic can learn from both the current timestep and previous timesteps, allowing them to capture both short-term and long-term patterns in the data. This makes this algorithm promising with time series data as input. Last but not least, CNN with suitable activation functions such as relu will become less sensitive to the vanishing/exploding gradient problem, which prevents deep models from being trained effectively. The article chooses a Multichannel CNN model, whereby each one-dimensional time series as input will be provided to the model as a separate channel. The model is compiled using the Adam optimizer with the learning rate determined through experiment and the loss function used is mean squared error.

IV. EXPERIMENTAL RESULTS

1. Experimental data

The article selects stocks of three companies representing three different industries in the economy according to the Global Industry Classification Standard: developed by Standard & Poor's and MSCI Barra. Russel Global Sectors, and Industry Classification Benchmark: co-developed by Dow Jones and FTSE. Specifically:

1. Joint Stock Commercial Bank for Foreign Trade of Vietnam (VCB)

2. Masan Group Corporation (MSN)

3. Hoa Phat Group Joint Stock Company (HPG)

VCB is in the financial sector, MSN is in the field of essential goods, and HPG is in the production of basic materials. All three companies were included in the VN30 index at the date of 11/03/2022.

Data to determine the performance of proposed methods is downloaded from investing.com. The original data includes opening price, closing price, highest price, lowest price, volume, VN-index, and the yield on 10-year government bond. Some technical indicators are added into the analysis since they are frequently used by traders to better understand the supply and demand of securities and market sentiment. Together, these indicators form the basis of technical analysis providing investors with buying and selling signals. Data of these features is built by Excel based on the original data aforementioned.

Thus, the dataset of each stock includes 21 features as follows:

Original data

1. open: Opening stock price of a day

2. close: Closing stock price of a day

3. high: Highest stock price in a day

4. low: Lowest stock price in a day

5. volume: Number of shares traded in a day

6. VNINDEX: VN-index

7. Y10_T: Yield on 10-year government bondTechnical Indicators

8. rsi: Relative Strength Indicator

$$RSI = 100 - \left[\frac{100}{1+\frac{Average\_gain}{Average\_loss}}\right].$$

- Average_gain: Average gains, Average_loss: Average losses.

- The average gains or losses used in this calculation is the average percentage of gains or losses over the lookback periods, which is 14 days in the article.

9. MACD: Moving Average Convergence Divergence;

$$MACD = EMA(12) - EMA(26)$$

EMA (12): Exponential moving average of 12 days;

EMA (26): Exponential moving average of 26 days.

10. ADX: Average Directional Index (see in [8])

11. % R: Williams Percent Range

$$\%R = \frac{Highest\_High - Close}{Highest\_High - Lowest\_Low} \times (-100);$$

Highest_High: The highest price during the lookback period, which is 14 days in the article.

Close: Most recent closing price.

Lowest_Low : The lowest price in the lookback period, which is 14 days in the article.

12. cci (14): Commodity Channel Index

$$cci(14) = \frac{Typical\_Price - MA}{0.015 \times Mean\_Deviation}$$

$$Typical\_Price = \sum_{i=1}^{P} ((High + Low + Close) \div 3)$$

$$p = 14$$

$$MA = (\sum_{i=1}^{P} Typical\_Price) \div P$$

$$Mean\_Deviation = (\sum_{i=1}^{P} |Typical\_Price - MA|) \div P$$

13. MA (5): Moving average of 5 days

14. MA (10): Moving average of 10 days

15. MA (20): Moving average of 20 days

16. ATR: Average True Range

$$ATR = \frac{1}{n}\sum_{i=1}^{n} TR_i$$

$$TR = Max\left[(H - L), \left|H - C_p\right|, \left|L - C_p\right|\right]$$

H: Today's highest price

L: Today's lowest price

C_p: Yesterday's closing price

$n = 14$

17. Ult Osc: Ultimate Oscillator

$$UltOsc = \left[\frac{A_7 \times 4 + A_{14} \times 2 + A_{28}}{4+2+1}\right] \times 100$$

A: Average

BP (Buying pressure) = Close – Min (Low, PC)

TR (True Range) = Max (High, PC) – Min (Low, PC)

PC: Prior Close

High: Today's highest price

Low: Today's lowest price

$$Average_7 = \frac{\sum_{p=1}^{7} BP}{\sum_{p=1}^{7} TR}$$

$$Average_{14} = \frac{\sum_{p=1}^{14} BP}{\sum_{p=1}^{14} TR}$$

$$Average_{28} = \frac{\sum_{p=1}^{18} BP}{\sum_{p=1}^{28} TR}$$

18. ROC: Price Rate of Change

$$ROC = \left(\frac{Close\_price_p - Close\_price_{p-n}}{Close\_price_{p-n}}\right) \times 100$$

Close_pricep: Closing price of most recent period

Close_pricep-n: Closing price n periods before most recent period

n = 13

19. STOCH (9,6): Stochastic oscillator

$\%K = \left(\frac{C-L9}{H9-L9}\right) \times 100$

C: The most recent closing price

L9: The lowest price traded of the 9 previous trading sessions

H9: The highest price traded during the same 9-day period %K: The current value of the stochastic indicator

STOCH (9,6): 6-period moving average of %K.

20. STOCHRSI (14):

$STOCHRSI = \frac{RSI - \min[RSI]}{\max[RSI] - \min[RSI]}$

RSI: Current RSI reading

min [RSI]: Lowest RSI reading over the last 14 periods

max [RSI]: Highest RSI reading over the last 14 periods

21. Bull/Bear: Bull/Bear Ratio (see in [12])

One problem that the article finds with the original data is that the sampling time is not the same. For example, we have the first date that is the first day of April (2015–04–01), the second one that is the second day of April (2015–04–02) and the third one that is the sixth day (2015–04–06).). The paper solves this problem in two steps based on the technique presented in [15]. Specifically, step 1 is to create a continuous time space from the start date to the end date. For example, with the aforementioned example, time space will include dates in the order 2015–04–01, 2015–04–02, 2015–04–03, 2015–04–04, 2015–04–05, 2015 –04–06. Step 2, at each time series, the missing value of a certain date will be filled with the value of the closest day available.

After using Excel to build technical indicators, with all three datasets, there are cells with no data in the first observations. To deal with this problem, with the dataset of VCB, observations from 12/03/2012 to 08/04/2012 are removed, with dataset of MSN, observations from 29/07/2013 to 25/08/2013 are not included, and with the HPG dataset, observations from 05/07/2013 to 01/08/2013 are excluded. The data processing results in the dataset of VCB including 3624 days during the period from 09/04/2012 to 11/03/2022, dataset of MSN including 3120 days for the period from 26/08/2013 to 11/03/2022, and the dataset of HPG including 3144 days for the period from 02/08/2013 to 11/03/2022.

Since data type of all features is numeric, the paper uses the Pearson correlation coefficient matrix in feature selection. If the absolute value of the Pearson correlation coefficient between a feature and the target variable of close is less than 0.05, that feature is not included in the proposed models. For VCB stock, Figure 4.1 shows that the correlation coefficient between close and STOCHRSI (14), ROC, and Bull_Bear is 0.011, 0.043, and -0.0032, respectively. Therefore, the three variables STOCHRSI (14), ROC, and Bull_Bear are not selected as inputs. For MSN stock, Figure 4.2 shows that the correlation coefficients between close and STOCHRSI (14), ADX, Ult Osc, and Bull_Bear are 0.0093, 0.034, -0.026, and -0.012, respectively. Therefore, the four variables STOCHRSI (14), ADX, Ult Osc, and Bull_Bear were not selected as inputs. For HPG stock, Figure 4.3 shows that the correlation coefficient between close and STOCHRSI (14), ADX, and Bull_Bear is -0.0066, -0.021, and -0.039, respectively. Therefore, the three variables STOCHRSI (14), ADX, and Bull_Bear are not chosen as inputs.

Thus, applying this technique to the data of three types of stocks, the article in turn obtains the following results:

- With VCB, the features included in the proposed models are close, open, high, low, volume, rsi, STOCH, MACD, ADX, %R, cci (14), MA (5), MA (10), MA (20), ATR, Ult Osc, VNINDEX, and Y10_T
- With MSN, the features included in the proposed models are: close, open, high, low, volume, rsi, STOCH, MACD, %R, cci (14), MA (5), MA (10), MA (20), ATR, ROC, VNINDEX, and Y10_T
- With HPG, the features included in the proposed models are close, open, high, low, volume, rsi, STOCH, MACD, %R, cci (14), MA (5), MA (10), MA (20). , ATR, Ult Osc, ROC, VNINDEX, and Y10_T.

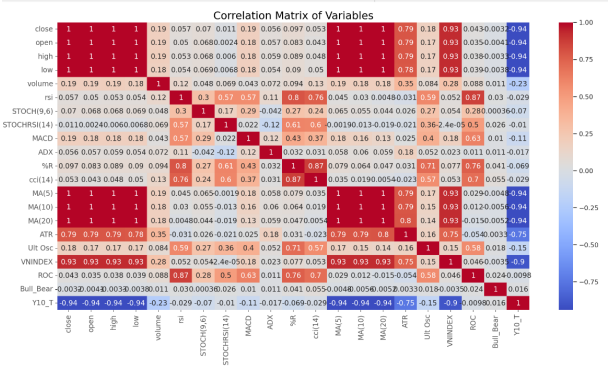


Figure 1. Correlation coefficients Matrix of VCB

To select the window size, the close time series is tested for autocorrelation. The article uses *stattools.arma_order_select_ic technique* with parameters *max_ar = 20, max_ma = 0, ic = ['bic'], trend = 'c'* for close time series of all 3 stocks. The results show that all three series in question have first order autocorrelation and so the window size is chosen to be 1. Thus, the length of

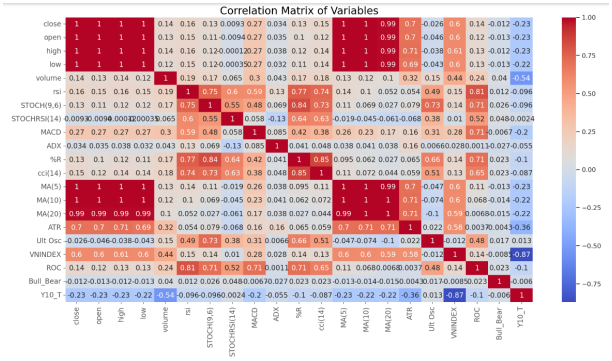


Figure 2. Correlation coefficients Matrix of MSN

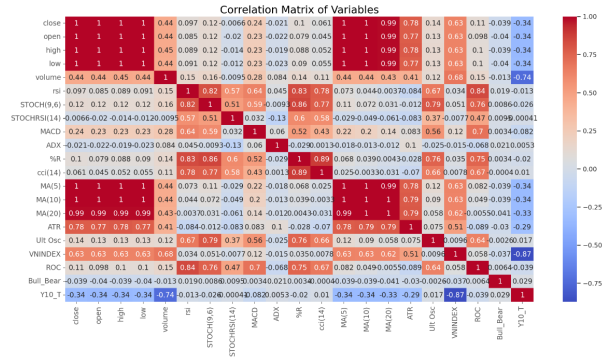


Figure 3. Correlation coefficients Matrix of HPG

the history is 1 day. In other words, for each prediction of the next day's closing price, the models use information from the previous day including the value of this variable (*close*) itself.

Variables measured at different scales do not contribute equally to the learned function of the model and may eventually produce bias. Therefore, to solve this potential problem, the z-score standardization method is applied to normalize the input data.

The input data is normalized according to the formula:

$$y_i = \frac{x_i - \overline{x}}{s} \tag{8}$$

where y_i is the normalized value, x_i is the input data, $\overline{x}$ is the mean value of the input data, and s is the standard deviation of the input data.

Covid-19 began to affect the Vietnamese stock market from the end of January 2020. Thus, in order for the training set to include observations during the period when the market is affected by this epidemic, it must contain pieces of data after January 2020. The goal of the paper is the training set to include as many records as possible during the Covid-19 period while still ensuring a large enough number of observations in the testing set in the hope of increasing the predictive performance of the algorithms. Derving from the data size of each stock (3624 days in the VCB's dataset, 3120 days in the MSN's dataset, and 3144 days in the HPG's dataset) and the last observation in the data of all three stocks is 11/03/2022, the paper takes first 90% of the dataset as the training set and the remaining 10

2. Evaluating methodology

We used two performance measures to assess which forecasting model accurately represents the actual data. The measures include Mean Absolute Error (MAE) and Root Mean Square Error (RMSE). The mathematical equations of the performance measures are defined as follows:

$$MAE = \frac{1}{N}\sum_{i=1}^{N}\left|y_{true}^{i} - y_{pred}^{i}\right| \tag{9}$$

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(y_{true}^{i} - y_{pred}^{i}\right)^2} \tag{10}$$

where N is the sample size, y_{true}^{i} is the observed value, and y_{pred}^{i} is the predicted value.

All methods are implemented using Python and Keras, an open-source learning library based on TensorFlow. All experiments are performed in the operating environment of Intel i5-1035G1 1.19 GHz, 8GBs of RAM, and Windows 10.

3. Experiment settings

The architectures proposed in the article are the best predictive performance ones among the experimented architectures.

a) Long short-term memory

With VCB, the architecture of LSTM includes: an LSTM hidden layer with 18 neurons, input shape = (1,18) where 1 is the window size and 18 is the number of features. Dropout with the parameter of 0.5 is used. In other words, 50% of the number of neurons in the hidden layer along with all its incoming and outgoing connections will be temporarily removed from the network after each iteration in the training process to deal with the overfitting problem. Finally, there is an output layer with 1 neuron to predict the closing price.

With MSN, the architecture of LSTM includes: an LSTM hidden layer with 16 neurons, input shape = (1,17) where 1 is the window size and 17 is the number of features. Dropout with the parameter of 0.1 is used. In other words, 10% of the number of neurons in the hidden layer along

```
model = Sequential()
model.add(LSTM(10, input_shape=(X_train.shape[1], X_train.shape[2])))
model.add(Dropout(0.5))
model.add(Dense(1))
epochs = 200
learning_rate = 0.1
decay_rate = learning_rate / epochs
mo= 0.9
sgd = SGD(learning_rate=learning_rate, momentum=mo, decay=decay_rate, nesterov=True)
model.compile(loss='mse', optimizer=sgd)
monitor = EarlyStopping(monitor='val_loss', min_delta=1e-3, patience=5,verbose=1, mode='auto', restore_best_weights=True)
model.fit(X_train, y_train, epochs=epochs, batch_size=32, validation_data=(X_test, y_test),  callbacks=[monitor], verbose=2, shuffle=False)
```

Figure 4. LSTM architecture corresponding to VCB

with all its incoming and outgoing connections will be temporarily removed from the network after each iteration in the training process to deal with the overfitting problem. Finally, there is an output layer with 1 neuron to predict the closing price.

With HPG, the architecture of LSTM includes: an LSTM hidden layer with 10 neurons, input shape = (1,18) where 1 is the window size and 18 is the number of features. Dropout with the parameter of 0.1 is used. In other words, 10% of the number of neurons in the hidden layer along with all its incoming and outgoing connections will be temporarily removed from the network after each iteration in the training process to deal with the overfitting problem. Finally, there is an output layer with 1 neuron to predict the closing price.

b) Bidirectional Long short-term memory

With VCB, the architecture of BiLSTM includes: two BiLSTM hidden layers with 21 neurons in each layer, input shape = (1,18) where 1 is the window size and 18 is the number of features. Dropout with the parameter of 0.1 is used between the two hidden layers and between the final hidden layer and the output layer. In other words, 10% of the number of neurons in each hidden layer along with all its incoming and outgoing connections will be temporarily removed from the network after each iteration in the training process to deal with the overfitting problem. Finally, there is an output layer with 1 neuron to predict the closing price.

With MSN, the architecture of BiLSTM includes: two BiLSTM hidden layers with 9 neurons in each layer, input shape = (1,17) where 1 is the window size and 17 is the number of features. Dropout with the parameter of 0.1 is used between the two hidden layers and between the final hidden layer and the output layer. In other words, 10% of the number of neurons in each hidden layer along with all its incoming and outgoing connections will be temporarily removed from the network after each iteration in the training process to deal with the overfitting problem. Finally, there is an output layer with 1 neuron to predict the closing price.

With HPG, the architecture of BiLSTM includes: two BiLSTM hidden layers with 11 neurons in each layer, input shape = (1,18) where 1 is the window size and 18 is the number of features. Dropout with the parameter of 0.5 is used between the two hidden layers and between the final hidden layer and the output layer. In other words, 50% of the number of neurons in each hidden layer along with all its incoming and outgoing connections will be temporarily removed from the network after each iteration in the training process to deal with the overfitting problem. Finally, there is an output layer with 1 neuron to predict the closing price.

c) Convolutional neural network

With VCB the architecture of CNN includes: A 1D convolution layer applying 72 filters with kernel size of 1, activation function is relu, input shape = (1,18) where 1 is window size and 18 is the number of features. Following the 1D convolutional layer is the max pooling layer with pool_size = 1 and the fully connected layer with 35 neurons and the activation function of relu. Finally, there is an output layer with 1 neuron to predict the closing price.

The model is compiled using the Adam optimizer with a learning rate of 0.0001 and the loss function used is the mean squared error.

With MSN the architecture of CNN includes: Two 1D convolution layer applying 16 filters with kernel size of 1, activation function is relu, input shape = (1,17) where 1 is window size and 17 is the number of features. Following the 1D convolutional layer is the max pooling layer with pool_size = 1 and the fully connected layer with 20 neurons and the activation function of relu. Finally, there is an output layer with 1 neuron to predict the closing price.

The model is compiled using the Adam optimizer with a learning rate of 0.01 and the loss function used is the mean squared error.

With HPG the architecture of CNN includes: A 1D convolution layer applying 56 filters with kernel size of 1, activation function is relu, input shape = (1,18) where 1 is

```
model = Sequential()
model.add(LSTM(16, input_shape=(X_train.shape[1], X_train.shape[2])))
model.add(Dropout(0.1))
model.add(Dense(1))
epochs = 200
learning_rate = 0.001
decay_rate = learning_rate / epochs
mo= 0.9
sgd = SGD(learning_rate=learning_rate, momentum=mo, decay=decay_rate, nesterov=False)
model.compile(loss='mse', optimizer=sgd)
monitor = EarlyStopping(monitor='val_loss', min_delta=1e-3, patience=5, verbose=1, mode='auto', restore_best_weights=True)
model.fit(X_train, y_train, epochs=epochs, batch_size=32, validation_data=(X_test, y_test),  callbacks=[monitor], verbose=2, shuffle=False)
```

Figure 5. LSTM architecture corresponding to MSN

```
model = Sequential()
model.add(LSTM(10, input_shape=(X_train.shape[1], X_train.shape[2])))
model.add(Dropout(0.1))
model.add(Dense(1))
epochs = 200
learning_rate = 0.1
decay_rate = learning_rate / epochs
mo= 0.9
sgd = SGD(learning_rate=learning_rate, momentum=mo, decay=decay_rate, nesterov=False)
model.compile(loss='mse', optimizer=sgd)
monitor = EarlyStopping(monitor='val_loss', min_delta=1e-3, patience=5,verbose=1, mode='auto', restore_best_weights=True)
model.fit(X_train, y_train, epochs=epochs, batch_size=32, validation_data=(X_test, y_test),  callbacks=[monitor], verbose=2, shuffle=False)
```

Figure 6. LSTM architecture corresponding to HPG

```
model = Sequential()
model.add(Bidirectional(LSTM(21, return_sequences = True,input_shape=(X_train.shape[1], X_train.shape[2]))))
model.add(Dropout(0.1))
model.add(Bidirectional(LSTM(21)))
model.add(Dropout(0.1))
model.add(Dense(1))
epochs = 200
learning_rate = 0.1
decay_rate = learning_rate / epochs
mo= 0.9
sgd = SGD(learning_rate=learning_rate, momentum=mo, decay=decay_rate, nesterov=False)
model.compile(loss='mse', optimizer=sgd)
monitor = EarlyStopping(monitor='val_loss', min_delta=1e-3, patience=5, verbose=1, mode='auto', restore_best_weights=True)
model.fit(X_train, y_train, epochs=epochs, batch_size=32, validation_data=(X_test, y_test),  callbacks=[monitor], verbose=2, shuffle=False)
```

Figure 7. BiLSTM architecture corresponding to VCB

```
model = Sequential()
model.add(Bidirectional(LSTM(9, return_sequences = True,input_shape=(X_train.shape[1], X_train.shape[2]))))
model.add(Dropout(0.1))
model.add(Bidirectional(LSTM(9)))
model.add(Dropout(0.1))
model.add(Dense(1))
epochs = 200
learning_rate = 0.1
decay_rate = learning_rate / epochs
mo= 0.9
sgd = SGD(learning_rate=learning_rate, momentum=mo, decay=decay_rate, nesterov=False)
model.compile(loss='mse', optimizer=sgd)
monitor = EarlyStopping(monitor='val_loss', min_delta=1e-3, patience=5, verbose=1, mode='auto', restore_best_weights=True)
history = model.fit(X_train, y_train, epochs=epochs, batch_size=32, validation_data=(X_test, y_test),  callbacks=[monitor], verbose=2, shuffle=False)
```

Figure 8. BiLSTM architecture corresponding to MSN

window size and 18 is the number of features. Following the 1D convolutional layer is the max pooling layer with pool size = 1 and the fully connected layer with 40 neurons and the activation function of relu. Finally, there is an output layer with 1 neuron to predict the closing price

The model is compiled using the Adam optimizer with a learning rate of 0.01 and the loss function used is the mean squared error.

```
model = Sequential()
model.add(Bidirectional(LSTM(11, return_sequences = True,input_shape=(X_train.shape[1], X_train.shape[2]))))
model.add(Dropout(0.5))
model.add(Bidirectional(LSTM(11)))
model.add(Dropout(0.5))
model.add(Dense(1))
epochs = 200
learning_rate = 0.1
decay_rate = learning_rate / epochs
mo= 0.9
sgd = SGD(learning_rate=learning_rate, momentum=mo, decay=decay_rate, nesterov=True)
model.compile(loss='mse', optimizer=sgd)
monitor = EarlyStopping(monitor='val_loss', min_delta=1e-3, patience=5, verbose=1, mode='auto', restore_best_weights=True)
history = model.fit(X_train, y_train, epochs=epochs, batch_size=32, validation_data=(X_test, y_test),  callbacks=[monitor], verbose=2, shuffle=False)
```

Figure 9. BiLSTM architecture corresponding to HPG

```
model = Sequential()
model.add(Conv1D(filters=72, kernel_size=1, activation='relu', input_shape=(n_timesteps,n_features)))
model.add(MaxPooling1D(pool_size=1))
model.add(Flatten())
model.add(Dense(35, activation='relu'))
model.add(Dense(n_outputs))
model.compile(loss='mse', optimizer= keras.optimizers.Adam(0.0001))
model.fit(train_x, train_y, epochs=200, batch_size=32, verbose=verbose)
```

Figure 10. CNN architecture corresponding to VCB

```
model = Sequential()
model.add(Conv1D(filters = 16, kernel_size=1, activation='relu', input_shape=(n_timesteps,n_features)))
model.add(MaxPooling1D(pool_size=1))
model.add(Conv1D(filters=16, kernel_size=1, activation='relu'))
model.add(MaxPooling1D(pool_size=1))
model.add(Flatten())
model.add(Dense(20, activation='relu'))
model.add(Dense(n_outputs))
model.compile(loss='mse', optimizer= keras.optimizers.Adam(0.01))
model.fit(train_x, train_y, epochs=200, batch_size=32, verbose=verbose)
```

Figure 11. CNN architecture corresponding to MSN

```
model = Sequential()
model.add(Conv1D(filters=56, kernel_size=1, activation='relu', input_shape=(n_timesteps,n_features)))
model.add(MaxPooling1D(pool_size=1))
model.add(Flatten())
model.add(Dense(40, activation='relu'))
model.add(Dense(n_outputs))
model.compile(loss='mse', optimizer= keras.optimizers.Adam(0.01))
model.fit(train_x, train_y, epochs=200, batch_size=32, verbose=verbose)
```

Figure 12. CNN architecture corresponding to HPG

d) Baseline algorithms

To demonstrate the effectiveness of the proposed methods, these methods are compared with ARIMA, a popular and widely used statistical method for time series forecasting, using the same training and testing data in the same operating environment. As a univariate model, the article runs ARIMA only on the closing price.

A standard notation used of ARIMA(p,d,q) where parameters are replaced with integer values to quickly indicate the particular ARIMA model in use. The parameters of the ARIMA model are determined as follows:

- p: A number of lag observations included in the

model, also called the lag order. Since there is an autocorrelation of 1 in the close series of all stocks, p is chosen to be 1 .

- d: A number of times that the raw observations are differenced, also called the degree of differencing to make the time series stationary. In the article, d equals 1 is enough to make the close time series in all stocks stationary.
- q: The size of the moving average window, also called the order of the moving average. The value of q is determined through experiment.

After conducting experiments with the data of the aforementioned three stocks, the paper finds the optimal ARIMA models corresponding to VCB, MSN, and HPG, are ARIMA(1,1,0) and ARIMA(1,1,1) respectively.

V. Resulting comparison

For each stock, all methods use the same training and testing set. All methods are implemented using Python and Keras, an open-source learning library based on TensorFlow. All experiments are performed in the operating environment of Intel i5-1035G1 1.19 GHz, 8GBs of RAM, and Windows 10.

Each prediction algorithm is executed multiple times. Averages of the results of all the experiments conducted per each technique are compared. Models that generate the minimum performance measures on the testing data are also selected and compared.

Tables I and II show the averages of RMSE and MAE for each stock's architecture. Comparisons between models generating the minimum RMSE and MAE of each stock are shown in Tables III and IV, respectively. The results **in bold** are the best results in the algorithms surveyed in the article. It is clear that in predicting the next day's closing price, the CNN models beat the rest including the ARIMA baseline models for all three stocks. This is proof that CNN is an effective and promising tool in stock price forecasting

The average comparison of the results in Tables I and II shows that the ARIMA baseline algorithm has the second highest prediction accuracy in all surveyed stocks. LSTM with prediction accuracy ranks 3rd in VCB, and 4th in MSN and HPG. BILSTM has the 3rd highest prediction accuracy in HPG and MSN, and 4th in VCB.

According to the results in table III and IV, we see that the ARIMA baseline algorithm has the second highest prediction accuracy in all surveyed stocks. LSTM with prediction accuracy ranks 3rd in VCB, 4th in MSN and HPG. BILSTM has the 3rd highest prediction accuracy in stocks including MSN and HPG, and 4th in VCB stocks.

TABLE I
RMSE IN AVERAGE OF ALL MODELS FOR ALL STOCKS

Models	VCB	MSN	HPG
LSTM	4877.394	31870.690	8238.961
BiLSTM	8183.190	29020.730	4252.388
CNN	860.530	1568.830	655.909
ARIMA	1126.409	2212.868	868.072

TABLE II
MAE IN AVERAGE OF ALL MODELS FOR ALL STOCKS

Models	VCB	MSN	HPG
LSTM	3771.708	28278.280	7246.288
BiLSTM	6672.860	26050.260	3472.885
CNN	647.325	1267.096	527.298
ARIMA	687.585	1396.499	541.981

TABLE III
MINIMUM RMSE OF ALL MODELS FOR ALL STOCKS

LSTM	4452.224	30768.223	5074.592
BiLSTM	5176.616	11960.404	3757.165
CNN	**635.660**	**1188.001**	**607.661**
ARIMA	1126.409	2212.868	868.072

TABLE IV
MINIMUM MAE OF ALL MODELS FOR ALL STOCKS

LSTM	3598.402	26753.147	4262.372
BiLSTM	4254.300	9736.850	3035.655
CNN	**472.443**	**905.085**	**460.292**
ARIMA	687.585	1396.499	541.981

VI. Conclusion

The article studies the problem of time series forecasting in finance, specifically the problem of forecasting the closing price of stocks based on historical data of a previous day. Besides stock data including opening price, closing price, highest price, lowest price, and volume, the article added technical indicators to the analysis to be able to simulate cases of sharp price drops or price spikes due to the influence of exogenous shocks (such as epidemics, wars....). Moreover, the paper also uses and compares algorithms such as LSTM, BiLSTM, CNN, and ARIMA. The results show that CNN performs better for predicting the next day's closing price than the remaining models. This is a proof that CNN is an effective and promising tool in stock price forecasting.

References

[1] Achkar, R., Elias-Sleiman, F., Ezzidine, H., & Haidar, N. (2018), "Comparison of BPA-MLP and LSTM-RNN for Stocks Prediction", *2018 6th International Symposium on Computational and Business Intelligence* (ISCBI). doi:10.1109/iscbi.2018.00019

[2] Althelaya, K. A., El-Alfy, E.-S. M., & Mohammed, S. (2018), "Stock Market Forecast Using Multivariate Analysis with Bidirectional and Stacked (LSTM, GRU)", *2018 21st*

Saudi Computer Society National Computer Conference (NCC). doi:10.1109/ncg.2018.8593076

[3] Aravindpai, P.: CNN vs. RNN vs. ANN – Analyzing 3 Types of Neural Networks in Deep Learning. https://www.analyticsvidhya.com/blog/2020/02/cnn-vs-rnn-vs-mlp-analyzing-3-types-of-neural-networks-in-deep-learning/

[4] Bengio, Y. (2012), "Practical Recommendations for Gradient-Based Training of Deep Architectures", *Neural Networks: Tricks of the Trade*, 437–478. doi:10.1007/978-3-642-35289-8_26

[5] Brownlee, J.:Using Learning Rate Schedules for Deep Learning Models in Python with Keras.https://machinelearningmastery.com/using-learning-rate-schedules- deep-learning-models-python-keras/

[6] Chen, W., Zhang, Y., Yeo, C. K., Lau, C. T., & Lee, B. S. (2017)," Stock market prediction using neural network through news on online social networks", *2017 International Smart Cities Conference (ISC2)*. doi:10.1109/isc2.2017.8090834

[7] Hoseinzade, E., & Haratizadeh, S. (2019), "CNNpred: CNN-based stock market prediction using a diverse set of variables", *Expert Systems with Applications*. doi:10.1016/j.eswa.2019.03.029

[8] Jay, A.: How to do Average Directional Index (ADX) in Excel: https://investsolver.com/how-to-do-average-directional-index-adx-in-excel/

[9] Kara, Y., Boyacioglu, M.A., Baykan, Ö.K.(2011), "Predicting direction of stock price index movement using artificial neural networks and support vector machines:The sample of the istanbul stock exchange", *Expert Syst. Appl*. 38 (5), 5311–5319.

[10] Lee, M.-C. (2009), "Using support vector machine with a hybrid feature selection method to the stock trend prediction", *Expert Syst. Appl*. 36 (8), 10896–10904.

[11] Luo, L., Chen, X. (2013), "Integrating piecewise linear representation and weighted support vector machine for stock trading signal prediction", *Appl. Soft Comput*.13 (2), 806–816.

[12] Mark, U.: How to Calculate the Elder Ray Technical Indicator using Excel: https://www.tradinformed.com/how-to-calculate-the-elder-ray-technical-indicator-in-excel/.

[13] Mehtab, S. & Sen, J. (2020), "Stock Price Prediction Using Convolutional Neural Networks on a Multivariate Timeseries", https://doi.org/10.48550/arXiv.2001.09769.

[14] Olah, C.: Understanding LSTM Networks http://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[15] Paialunga, P.: Noise cancellation with Python and Fourier Transform https://towardsdatascience.com/noise-cancellation-with-python-and-fourier-transform-97303314aa71.

[16] Ruder, S. (2017), "An overview of gradient descent optimization algorithms", https://doi.org/10.48550/arXiv.1609.04747.

[17] Sethia, A., & Raut, P. (2018), "Application of LSTM, GRU and ICA for Stock Price Prediction", *Smart Innovation, Systems and Technologies*, 479–487.

[18] Sonkiya, P. , Bajpai, V. and Bansal, A. (2021), "Stock price prediction using BERT and GAN", *In Proceedings of ACM Conference* (Conference'17). ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/.

[19] Sutskever, I., Martens, J., Dahl, G. and Hinton, G. (2013), "On the importance of initialization and momentum in deep learning", *Proceedings of the 30th International Conference on Machine Learning*, PMLR 28(3):1139-1147, 2013.

[20] Tran .T. T.: Optimizer- https://viblo.asia/p/optimizer-hieu-sau-ve-cac-thuat-toan-toi-uu-gdsgdadam-Qbq5QQ9E5D8.

[21] Tsai, C.-F., & Hsiao, Y.-C. (2010), "Combining multiple feature selection methods for stock prediction: Union, intersection, and multi- intersection approaches", *Decision Support Systems*, 50(1), 258–269. doi:10.1016/j.dss.2010.08.028.

[22] Zhang, K., Zhong, G., Dong, J., Wang, S., & Wang, Y. (2019)," Stock Market Prediction Based on Generative Adversarial Network", *Procedia Computer Science*, 147, 400–406. doi:10.1016/j.procs.2019.01.256.

[23] Zhang, Q., Wang, R., Qi, Y. and Wen, F. (2022)," A watershed water quality prediction model based on attention mechanism and Bi-LSTM", *Environ Sci Pollut Res (2022)*. https://doi.org/10.1007/s11356-022-21115-y.

[24] Zhou, X., Pan, Z., Hu, G., Tang, S., & Zhao, C. (2018), "Stock Market Prediction on High-Frequency Data Using Generative Adversarial Nets", *Mathematical Problems in Engineering*, 2018, 111. doi:10.1155/2018/4907423.

Hoang Van Hai received a B.Sc. degree in Mathematics from Thai Nguyen University of Education in 2001, completed M.Sc. degree in Public policy at Antwerp University in 2012, and has completed M.Sc. degree in Data science at Hanoi University of Sciences in 2023. He has currently been working at Faculty of Economics, University of Economics and Business Administration, Thai Nguyen University. His research interests are Deep learning, Regression, Machine learning.
Email: hoanghai@tueba.edu.vn

Dang Thi Thu Hien has currently been working at Faculty of Computer Science and Engineering, Thuy Loi University. Her research interests are Artificial intelligence, Neural networks, Deep learning, Regression, Data mining, Machine learning. .
Email: Hiendt@tlu.edu.vn

Proposing of Imaging Graph Neural Network with Defined Security Pattern for Improving Smart Contract Vulnerability Detection

Pham Trong Linh[1], Ta Minh Thanh[2]
[1,2]Le Quy Don Technical University, Ha Noi, Vietnam
Correspondence: Pham Trong Linh (tronglinhmta0611@gmail.com)
Communication: received 13 Feb 2023, revised 12 April 2023, accepted 15 May 2023
Digital Object Identifier: 10.32913/mic-ict-research.v2023.n2.1198

Abstract: A smart contract is a special set of protocols based on blockchain technology to implement the terms or agreements between the parties in the contract. Lots of smart contracts are built and deployed everyday. However, to ensure the safety of smart contracts is still a big challenge. Smart contracts are built to carry out transactions directly related to cryptocurrencies, therefore, the loss of security of smart contracts leads to huge financial losses. Common methods being used to check and to verify smart contract security are heavily dependent on hard rules defined by experts, leading to low detection accuracy and non-scalable, which can be bypassed by experienced attackers. In this paper, we propose to use the combination of Imaging Graph Neural Network With Defined Pattern to detect vulnerabilities in smart contracts. We construct a contract graph that shows the relationship between the main components in a smart contract. Then we extract graph features from normalized graphs, and combine graph features with defined security patterns to create combined features. Finally, we implemented normalization to gray scale image and feed it to the Convolutional Neural Network (CNN) to learn for vulnerability detection. Results show significantly improved accuracy compared to previous methods or other models. Specifically, 96,42%, 90,12%, 79% for reentrancy, timestamp dependence and infinite loop.

Keywords: *deep learning, stock price prediction, LSTM, BiLSTM, CNN.*

I. Introduction

1. Overview

Smart contracts (SC) are known as the programs that run on the blockchain [1]. It likes a digital contract that is bounded by a significant set of rules. Such rules are pre-defined by a set of computer code which must be copied and enforced in all nodes in the network. One of the most prominent platforms for SC definition and execution is Ethereum[1]. Ethereum is a blockchain-based decentralized network platform for deploying the smart contracts using the Turing-complete language. A smart contract when deployed to the blockchain is immutable, so if the vulnerabilities are not carefully checked before being deployed to the system, its consequences are extremely serious. One of the more famous recent attacks is the 3^{rd} of august 2022 event, thousands of Solana wallets were drained of nearly $8 million in total. The hackers' exploit was thought to have occurred due to complications in importing accounts (Solana wallets). Therefore, smart contracts (the program deployed on blockchain) need to be carefully checked for security before deploying to the blockchain network.

Common methods being used to detect smart contract vulnerabilities are Symbolic Execution Technologies [2], classical static analysis [3]. These methods have the weakness of being heavily dependent on predefined rules, leading to low vulnerability detection performance and being easily bypassed by experienced attackers. In addition, these methods consume a lot of time and computation resources.

Recently, studies on the application of deep learning neural networks to the detection of smart contract security vulnerabilities have yielded highly accurate results. In this paper, we propose a fully automated and scalable approach that can detect vulnerabilities at the function level. Specifically, we build a contract graph that shows the relationships between the components of a smart contract program in chronological order. Afterwards, we extract graph features from normalized graphs. Then, we extract the security pattern features from smart contracts code using the defined security patterns, then combine graph features with security pattern features to extract the special combined features

[1]https://ethereum.org/

for each pattern. Finally, we implement normalization to grayscale image and feed it to the Convolutional Neural Network (CNN) to learn for vulnerability detection.

2. Our contributions

In proposed approach, we focus on discovering the most suitable combined features for each pattern via graph neural network, then employ the imaging features to improve the efficiency of smart contract vulnerability detection. Our contributions can be listed as follows.

(1) Propose a extraction method that can generate the contract graph from source code of smart contract components of a smart contract in chronological order.

(2) Find out the appropriate algorithm to transform the combined features to gray scale image then feed such images to CNN. Our method can visualize the deference of each vulnerability of smart contract for classification.

(3) We focus on testing the smart contract code built in Solidity language running on the Ethereum platform. The final results show that the detection method has an accuracy of 96.64%, 90,12% ,79% for reentrancy, timestamp dependence,infinite loop. These results show that applying our model to the detection of security vulnerabilities yields outstanding results than state-of-the-art methods.

3. Roadmap

The rest of our paper is organized as follows. Sect. II describes the related works and the background of smart contracts. Then, Sect. III introduces the details of our proposed method. We extend the previous algorithm to propose the imaging graph neural network (IGNN) with defined security pattern for improving smart contract vulnerability detection. After that, Sect. IV gives the experimental results and discussion. Finally, Sect. V concludes this paper.

II. Related Works

1. Some previous works

The security of smart contracts has always been of special concern, to detect smart contract vulnerabilities can use many ways. Symbolic execution is one of the common techniques used to detect smart contract vulnerabilities. For example, Oyente [4] is the first symbolic execution tool for smart contracts. It supports detection for transaction order dependency (TOD), timestamp dependency, reentrancy, mishandled exception and integer overflow. Oyente [4] relies on simplified semantics and use templates to define specific properties. However, the tool lacks a semantic characterization. In addition, it reduces path explosion by limiting the number of loops, therefore it is difficult to detect variuos security defects. Securify [5] uses formal verification methods to detect the vulnerabilities of smart contracts, which uses certain characteristics to analyze

```
pragma solidity ^0.8.10;

contract Reentrancy {

    mapping (address => uint) private
        balances;

    function withdrawBalance() public {
        uint amountToWithdraw =
            balances[msg.sender];
        require(msg.sender.call.value(
            amountToWithdraw)());
        balances[msg.sender] = 0;
    }
}
```

Figure 1. Reentrancy vulnerability

whether the smart contract contains vulnerabilities or not. Securify generates a dependency graph by analyzing and displaying the bytecode of the smart contract. According to certain characteristics, it analyzes whether the semantic information of the contract satisfies or violates those characteristics, and evaluates whether there are security holes (loopholes) in the smart contract or not. It's input is the bytecode of smart contract and a bunch of templates. They are described in a domain-specific language for security flaws. The output is the location of specific security flaws and vulnerabilities. Such security patterns can be written by the user, then Security can be extensible.

On the other hand, some works tried to apply deep learning to smart contract vulnerability detection, such as using graph neural networks (GNNs) [6] for smart contracts code vulnerability detection which constructs a contract graph to represent both syntactic and semantic structures of a smart contract code, then use the pre-designed normalized model to highlight the major nodes. Finally, such method used a degree-free graph convolutional neural network (DR-GCN) and a novel temporal message propagation network (TMP) to learn security patterns from the normalized graphs for vulnerability detection. Different from these methods, in this work, we mainly focused on using Combination Imaging Graph Neural Network with Defined Security Pattern to detect Smart Contract Vulnerability.

2. Background of smart contract

a) Blockchain and Smart Contract

A blockchain is a ledger (as a distributed database) that is shared among the nodes of a computer network. The blockchain network stores information electronically in a

```
pragma solidity ^0.8.10;

contract TimestampDependence {
    uint constant TICKET_AMOUNT = 5;
    uint public pot;

    function play() payable {
        assert(msg.value ==
            TICKET_AMOUNT);
        pot += msg.value;
        var random = uint(sha3(block.
            timestamp)) % 3;
        if (random == 0){
            msg.sender.transfer(pot);
            pot = 0;
        }
    }
}
```

Figure 2. Timestamp dependence vulnerability

digital form[2], to maintain a secure and decentralized record of transactions. The innovation with blockchain is that it ensures the integrity and security of data records and creates trust without the need for a trusted third party.

Smart contracts (SC) are programs that are deployed on blockchain network. It works by following simple "if/when... then..." statements. That means the network of nodes execute the pre-determined actions when the pre-determined conditions have been met and verified beforehand. These actions may include disbursing funds to the appropriate parties, registering vehicles, sending notices or issuing tickets, etc. The blockchain is then updated when the transaction is complete. Note that, the transaction cannot be changed anymore, then only authorized parties can see the result. In a smart contract, there may be many provisions necessary to satisfy the participants that the task is satisfactorily completed. To establish such conditions, participants must pre-define how their transactions and data are represented on the blockchain, agreeing to "if/when...then.." rules that governs those transactions, uncovering all possible exceptions.

Smart contracts on Ethereum[3] are typically written in a high-level language, then compiled in EVM bytecode. The most widely used language is Solidity[4]. Solidity is a simple contract-oriented high-level programming language. Its syntax is similar to Javascript.

[2]https://en.wikipedia.org/wiki/Bitcoin
[3]https://ethereum.org/
[4]https://docs.soliditylang.org/

```
pragma solidity ^0.8.10;

contract InfiniteLoop {
    //...
    //...
    function () payable {
        count ++;

        while (count < 20) {
            withdraw();
        }
    }
}
```

Figure 3. Infinite loop vulnerability

b) Smart Contract Vulnerabilities

(i) Reentrancy

The Reentrancy vulnerability is caused by attackers withdrawing funds from the target by recursively calling the target withdrawal function. Attackers execute withdrawals several times before checking the balance. Whenever the attacker receives ETH, his/her contract automatically calls the fallback function, which calls the withdrawal function many times until become zero. At this point the attack enters a recursive loop, therefore, the contract cannot update the attacker's balance.

Figure. 1 shows an example of a re-entry vulnerability. The value of *call.value* (Line 9) implements the fallback function of the sender. It sends money, as much as the user's balance. If the fallback function in the sender's code calls the *drawBalance* function again, then a recurring deposit call loop will be generated multiple times before updating the balance status on Line 10. Thus, the smart contract of the victim will send unlimited money to the recipient. Reentrancy is a notorious vulnerability that has caused massive economic damage in the mining of DAOs (decentralized autonomous organizations). Following a DAO crash, safety development guidelines recommend completing state updates before performing any actions that may call other code.

(ii) Timestamp dependence

This timestamp dependency vulnerability in a smart contract occurs when it relies on the value of the block timestamp to perform an operation. The block timestamp value is generated by the node executing the smart contract, making it vulnerable to manipulation and attack.

The smart contract shown in Figure. 2 represents a game in which the user has to deposit an amount for the smart contract function *play()*. That amount must be equal to

the value of the TICKET_AMOUNT parameter. The smart contract then retrieves the actual time when the contract was executing, applies a formula to it, and stores the value in a random variable (see Line 10). The smart contract then checks if the value of the random variable is equal to "zero". If it is the user who submitted the transaction, it will become the winner. In order to achieve real-time, the contract smart use *v*ariable block.timestamp The value of the variable *block.timestamp* is provided by the node, so it can easily manipulate it by incrementing it for until receiving the result of Line 10 is "zero" which leads to a vulnerability in the smart contract.

(iii) Infinite loop

The computing power in the Ethereum blockchain is not free. Therefore, we need to pay Ether (ETH) to buy Gas[5] to get the amount of computing power to execute the transaction. Therefore, if you can reduce the number of calculation steps, you can save time and even save a lot of money. Adding loops to smart contracts is one way to increase gas costs. For example, an array already has a lot of loops. However, if the factors increase, then more integration is required to complete that particular loop. If an attacker can introduce infinite loops into the smart contract's process, then he or she can mine all the ETH Gas available in the user's wallet by himself.

Infinite loop vulnerability in smart contract is considered as a loop bug which unintentionally iterates looping. It makes to fail to jump out of the loop, then return an expected results. That means the attacker can influence the elements of the array length, which create a DOS (Denial-of-Service) issue and won't allow the system to jump out of the loop (as shown in Fig. 3). In additionally, it ensures that the contract is stalled as every contract comes with a Gas limit.

III. Our Proposed Method

Method overview. The overall architecture of our proposed method (shown in Fig. 4) consists of five phases: (1) graph contract generation and normalization phase, which create a contract graph to represent both syntactic and semantic structures of a smart contract function and normalize it, then (2) extract graph features from normalized graphs; (3) defined security pattern extraction phase, which extract security patterns predefined by experts from source code; (4) combination and grayscale image normalization phase, which combine graph features with security pattern features to create combined features. Finally, (5) vulnerability detection phase, which feeds grayscale normalized image to the Convolutional Neural Network (CNN) to learn for vulnerability detection.

[5]https://etherscan.io/gastracker

1. Smart Contract Graph Generation and Normalization

a) Smart Contract Graph Generation

The paper [7] shows that the smart contracts can be transformed into symbolic graph representations, which preserves the connection between the components in the program. Based on this idea, we build contract graphs from source code of smart contract by assigning it's components with different roles and calling those components nodes. To identify each link between the components in the program, we construct the edges in chronological order. Then, we determine the importance of the buttons and remove those that are not important.

We build three types of nodes representing 3 different important components in the program: *Main nodes*, *extra nodes* and *fallback nodes*. Such can be explained as follows:

Main nodes. The main node represents the key invitation and variables, which are the main factors that help determine if the program contains vulnerabilities. In particular, for reentrancy vulnerability, the main components used for vulnerability detection are the calls to *call.value*, the variables that represent the *balance*. In case of timestamp dependence vulnerability, the invocations to *block.timestamp*, and variables assigned by *block.timestamp*, and variables assigned by operations that use *block.timestamp*. In case of infinite loop of vulnerability, loop statements such as *for* and *while* can be failed, the loop condition variables, and recursive invocation. These components are considered to be the most important components for vulnerability detection.

Extra nodes. The extra nodes have the role of supporting vulnerability detection, it is combined with the main nodes to increase the accuracy of vulnerability detection. Specifically, reentrancy vulnerability, variables indirectly related to *user balance* or *bonus flag*. In case of timestamp dependence vulnerability, it is clear that invocations do not call *block.timestamp* and variables indirectly related to *block.timestamp*. These invocations and variables are defined as extra nodes E1, E2,..., En.

Fallback nodes. Fallback node: The fallback node F represents elements that directly interact with main nodes and extra nodes.

Edges. In the contract graph, we defined that the graph nodes are closely related to each other will be in chronological order. Therefore, we define edges to create a connection between the component nodes in the contract graph. Each edge represents the association between two component nodes in chronological order. Note that, the properties of graph edge are expressed as a tuple (Vs, Ve, o, t), where Vs and Ve represent its starting and ending of nodes; o represents its temporal order, and t the edge type. We

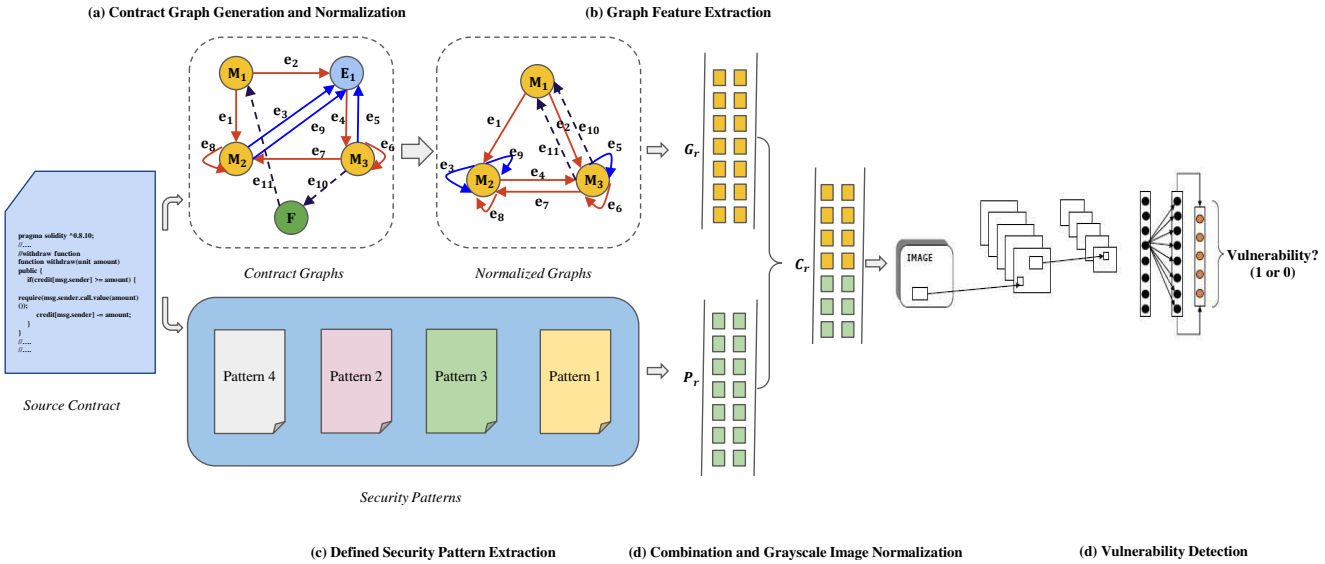


Figure 4. The overall architecture of our proposed method. (a) The contract graph generation and normalization phase; (b) the graph features extraction phase; (c) the defined security pattern extraction phase; (d) the combination and grayscale image normalization phase; (e) the vulnerability detection phase.

construct three categories of edges which are *control flow*, *data flow*, and *fallback edges*, respectively. The defined edges are shown in Table I.

b) Contract Graph Normalization

The nodes in the contract graph used to train the neural network have different roles. Moreover, each smart contract source code generates separate contract graphs, which makes it very difficult to train GNN. Therefore, we created a normalization process for the graph to remove the unimportant factors that interfere with the training process.

Nodes elimination. As explained in Sect. I), a contract graph consists of three components: main nodes $\{M_i\}_{i=1}^{|M|}$, extra nodes $\{E_i\}_{i=1}^{|E|}$ and fallback node F. We normalize the contract graph by removing the nodes that are less important in the graph, extra nodes E_i and fallback node F, and then move all the features to the nearest primary node. If extra nodes or fallback node have more than 2 neighbors, it will transfer all features to the nearest neighbor main node ensuring that the edges of the deleted node remain the same dimension and alignment order over time.

Feature of main nodes. After the extra nodes are removed from the graph, its properties are completely transferred to the neighboring main nodes, so that after normalization, the main nodes will update their features, new features of main nodes include three parts: (i) self-feature, called the feature of main-node M_i itself; (ii) in-features, called features of the extra-nodes $\{P_i\}_{j=1}^{|P|}$ that are merged to M_i and having a path pointing P_j to M_i; and (iii) out-feature, called features of the extra-nodes $\{Q_k\}_{k=1}^{|Q|}$ that are combined to M_i and have a path directed from Q_k from M_i. Fig. 5(e) illustrates the normalized graph of Fig. 5(d).

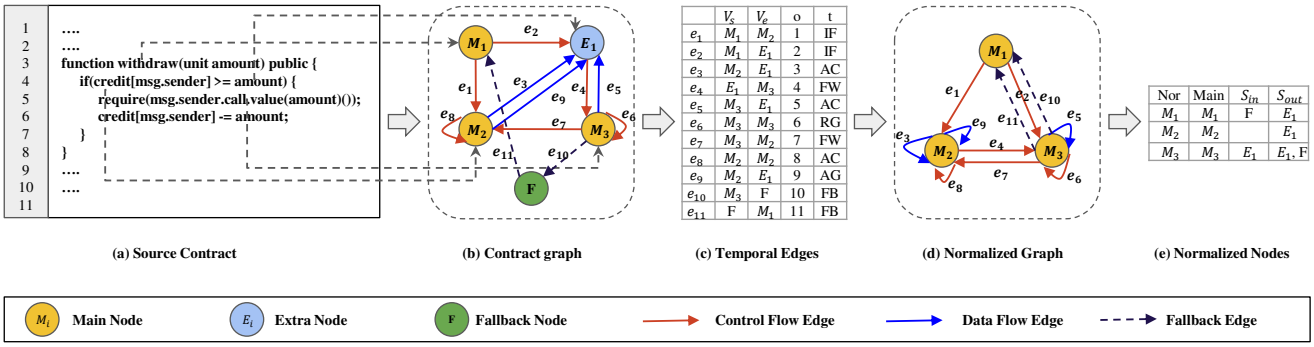
TABLE I
SEMANTIC EDGES SUMMARIZATION. ALL EDGES ARE CLASSIFIED INTO THREE CATEGORIES, NAMELY CONTROL-FLOW, DATA-FLOW, AND FALLBACK EDGES.

Symbol	Semantic Fact	Category
AH	assert{X}	Control-flow
RG	require{X}	
IR	if{...} revert	
IT	if{...} throw	
IF	if{X}	
GB	if{...} else {X}	
GN	if{...} then {X}	
WH	while{X} do{...}	
FR	for{X} do{...}	
FW	natural sequential relationships	
AG	assign{X}	Data-flow
AC	access{X}	
FB	interactions with fallback function	Fallback

2. Graph Feature Extraction

We use a temporal message propagation network. The features extraction process of the graph is divided into two stages, called message propagation phase, readout phase, respectively. Firstly, the network transmits information along its edges in chronological order. Then, it use the *readout* function to create the graph feature G_r, which contains final states of all nodes in the smart contract graph.

Message propagation phase. Let $G = \{V, E\}$ be the normalized contract graph, where V includes of the main



	V_s	V_e	o	t
e_1	M_1	M_2	1	IF
e_2	M_1	E_1	2	IF
e_3	M_2	E_1	3	AC
e_4	E_1	M_3	4	FW
e_5	M_3	E_1	5	AC
e_6	M_3	M_3	6	RG
e_7	M_3	M_2	7	FW
e_8	M_2	M_2	8	AC
e_9	M_2	E_1	9	AG
e_{10}	M_3	F	10	FB
e_{11}	F	M_1	11	FB

Nor	Main	S_{in}	S_{out}
M_1	M_1	F	E_1
M_2	M_2		E_1
M_3	M_3	E_1	E_1, F

Figure 5. The contract graph construction and normalization phase. (a) shows the source code of a smart contract; (b) visualizes the contract graph extracted from the code. Nodes M_i denote main nodes, nodes E_i denote extra nodes, and node F denotes fallback node; (c) illustrates the temporal edges in the extracted graph, where the types of edges are detailed in Table 1; (d) demonstrates the graph after normalization; (e) illustrates nodes in the normalized graph.

nodes, and E includes of all edges of that graph. Note that $E = \{e_1, e_2, \ldots, e_N\}$, where e_k is the k^{th} temporal edge of graph. The messages are propagated along the edges of the graph, one edge per time step. Firstly, initialize the hidden state h_i^0 for each node V_i of the graph. Hidden state is initialized for each node with its features. Afterwards, the message flows are through the k^{th} temporal edge e_k and updates the hidden state h_{ek} of the end node of e_k at the step k. In detail, the message m_k is firstly computed basing on the hidden state h_{sk} of the start node of e_k, and the edge type t_k. The calculation of each can be shown as follows:

$$x_k = h_{sk} \oplus t_k$$
$$m_k = W_k x_k + b_k,$$

where $\oplus$ is concatenation, W_k and b_k are network parameters. The original message x_k contains the information from the start node of e_k and edge e_k itself, which are continuously transformed into a vector embedding using matrix W_k and bias b_k.

The end node of e_k updates its hidden state after receiving the message. That means h_{ek} is updated as follows:

$$\hat{h} = tanh(Um_k + Xh_{ek} + b1),$$
$$h'_{ek} = softmax(R\hat{h}_{ek} + b2),$$

where the matrices U, X, R are network parameters, while $b1$ and $b2$ are bias vectors.

Readout phase. We proceed to extract features after traversing all edges in G by reading out the final states of all nodes. Suppose h_i^T is the final hidden state of the i^{th} node. In this case, the differences between the final hidden state h_i^T and the original hidden state h_i^0 are informative in the vulnerability detection problem. Therefore, the graph feature G_r is generated as follows:

$$s_i = h_i^T \oplus h_i^0$$
$$g_i = softmax(W_g^{(2)}(\tanh(b_g^{(1)} + W_g^{(1)} s_i)) + b_g^{(2)})$$
$$o_i = softmax(W_g^{(2)}(\tanh(b_o^{(1)} + W_o^{(1)} s_i) + b_o^{(2)})$$
$$G_r = FC(\textstyle\sum_{i=1}^{|V|} o_i \odot g_i),$$

where $\oplus$ denotes concatenation, $\odot$ denotes elementwise product. W_j , $b_j^{(1)}$, and $b_j^{(2)}$, $j \in \{g, o\}$ are network parameters.

3. Defined Security Pattern Extraction

For each vulnerability we design sub-patterns to represent the features used to detect the vulnerability. We use one-hot encoding to construct one-hot vectors representing each sub-pattern, then Check the occurrence of the feature in the source code and append 1/0 to the one-hot vector representing whether the sub-pattern is present in the source code or not. The vectors of sub-patterns related to a specific vulnerability are concatenated into a vector P_r, which are the pattern features. For example, reentrancy vulnerability have three sub-pattern are call.value invocation pattern, balance deduction pattern, enough balance pattern and three one-hot vectors corresponding to these three patterns are [1, 0, 0], [0, 1, 0], [0, 0, 1]. When the features corresponding to the sub-patterns of the reentrancy vulnerability are detected in the source code, append 1/0 to the end of the corresponding one-hot vectors. For example, when a call.value invocation is detected, it will append 1 to the end of the one-hot vector of the call.value invocation pattern to become a vector [1, 0, 0, 1].

4. Combination and Grayscale Image Normalization

Combining graph features G_r and pattern features P_r. After extracting graph features G_r and pattern features P_r. We construct a stage that combines G_r and P_r together to create a vector C_r. Specifically, we keep the features

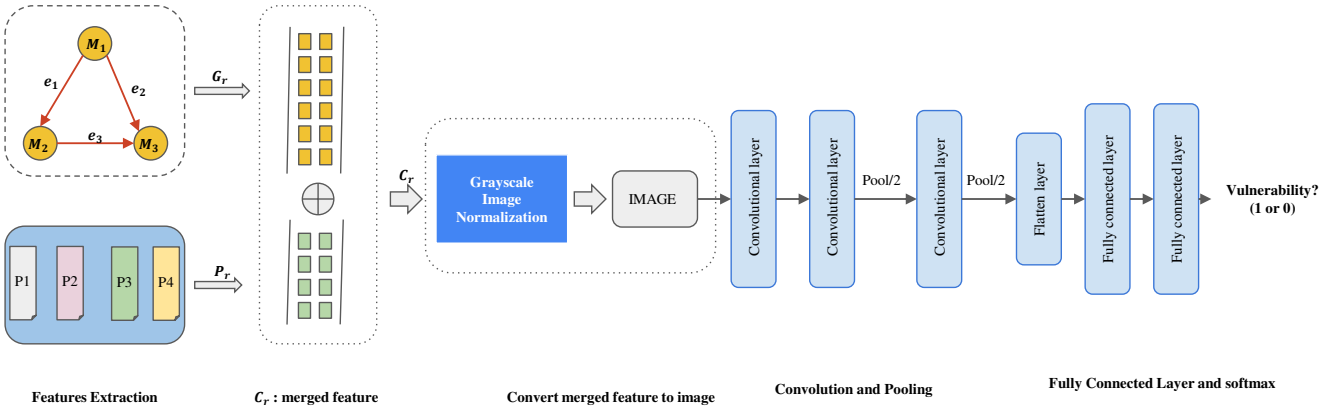
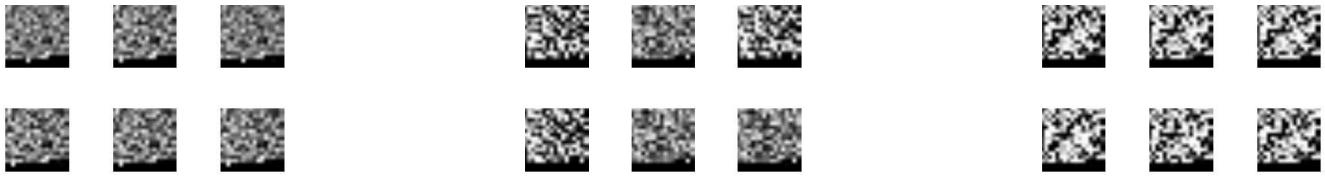


Figure 6. The process of vulnerability detection. First, extract features from smart contract source code. Then, combines G_r and P_r into the merged feature C_r. Then, the merged feature C_r is normalized to create gray image. Finally, Convolutional neural network is used to output the vulnerability detection results.



(a) Reentrancy (a) Timestamp dependence (a) Infinite loop

Figure 7. The source code after normalizing to a gray scale image of 3 vulnerabilities

of G_r as it is a 1-dim vector, at the same time reshape P_r into a 1-dim vector, which has a length equal to the sum of the lengths of G_r and P_r. The reason is that we convert vector P_r from 2-dimensional vector to 1-dimensional so that we can combine it with 1-dimensional vector G_r. After normalizing both vectors G_r and P_r to a 1-dimensional vector, we proceed to combine these two vectors by concatenating vector P_r to vector G_r to form vector C_r, which has a length equal to the sum of the lengths of G_r and P_r. After combining graph features G_r and pattern features P_r, we continue to append k features with zero value at the end of C_r so that the size of C_r is equivalent to the size of an mxn image after it is stretched.

Grayscale Image Normalization. After combining graph features G_r with pattern features P_r and appending 0 values to form a vector with the same size as an mxn image. However, the values of features of C_r still belong to different value domains, so we build a normalization procedure to convert the values of these features to the values of pixels [0,255]. Specifically, we use the Min-Max Scaling method [8] to transform the c_i values of C_r to the interval [0;1], then normalize the scaled C_r to C'_r in the interval [0;255] by multiplying by 255. Finally, reshape the vector C'_r into a matrix of size $m \times n$ corresponding to a grayscale image. Note that, the normalized grayscale images from the original raw data are the source code of the smart contract such as the solidity source code. Labeling will be carried out from the analysis of these source codes and after normalizing to gray scale images, the labels are also reassigned.

$$c'_i = \frac{c_i - min(C_i)}{max(C_i) - min(C_i)} \times 255$$

The images after normalizing the grayscale for each vulnerability will have the form as shown in Fig. 7. Obviously, the feature of transformed grayscale image can be used for classification.

5. Vulnerability Detection

After extracting graph features and pattern features from smart contract source code. We combine them and normalize grayscale images. We construct a CNN to train and detect smart contract vulnerability. This networks are fed with a large number of the normalized images generated from the functions of smart contract with the ground truth labels. After that, the trained models are employed to absorb the normalized image, then to yield a vulnerability detection label. We filter the image by multiple convolution layers and max pooling layers, then use a flatten layer, a fully connected layer and a softmax layer.

The convolution layer learns to select the different weights to the pixels, while the max pooling layer high-

lights the important pixels. Then flatten layer to flatten the n-dimensional matrix into a 1-dimensional matrix before feeding into the fully connected layer and softmax layer to give the final estimated label $\hat{y}$.

IV. EVALUATION

We perform model testing on two datasets which are real information such as ESC[6] is collected from Ethereum for reentrancy and timestamp dependence vulnerabilities, VSC[7] is collected from VNT Chain platforms for infinite loop vulnerability.

The ESC dataset has been collected 40,932 smart contracts from Ethereum [9], from which 307,396 functions have been extracted. Of the extracted data samples, about 5013 functions contain signs of reentrancy vulnerability, about 4833 functions contain signs of timestamp dependence vulnerability and about 56,800 functions contain signs of infinite loop vulnerability.

The VSC dataset [10]. has been collected 4170 smart contracts, from which 13761 functions have been extracted.

1. Experimental Setup

We use the datasets collected from blockchain networks to perform experiments. The dataset is large, so it has high requirements on CPU performance, hard disk space and memory size. To meet the needs of our experiments, we use a PC with an Intel Core i7 CPU at 2.8 GHz, a GPU at 1080TI and 16GB of memory. Another algorithms are implemented with python, while our IGNN is implemented by using tensorflow.

In our experiments, we divide the dataset into 2 parts as training data (70% dataset) and test data (30% dataset). In most machine learning experiments, the 70%:30% data split is because testing that data ratio gives high results. The 70%:30% ratio is also used in most of the research models that we refer to, which ensures fairness when comparing our research results with previous studies. Also, in our testing, we use results such as accuracy, recall, precision, F1, which are completely similar to the results of other tests. In addition, we also try to use similar data sets and experimental environments with other experiments to ensure fairness when comparing those results.

2. Comparison with Existing Methods

We compare with the existing methods that do not apply deep learning. Such methods are explained as follows.

- Oyente [4]: is a tool that uses the symbolic execution on control flow graph (CFG) for detecting the security patterns.
- Mythril [11]: uses symbolic execution for SMT solving and taint analysis to detect a variety of security vulnerabilities. It is used as a security analysis tool for EVM bytecode.
- Smartcheck [12]: is an extensible static analysis tool for discovering vulnerabilities, other bug code issues in the smart contracts of Ethereum.
- Securify [13]: is a formal-verification based software for detecting the bugs code of smart contract bugs issue by checking compliance, violation patterns to filter false positives.
- Slither [14]: is a testing-tool that enables developers to find vulnerabilities, enhance the code comprehension, and quickly prototype custom analyses.

Comparison on Reentrancy Vulnerability Detection. In Table II, we have detailed the test results of our IGNN method and five existing methods. Comparing the test results, we see that among the state-of-the-art existing method, SLITHER has the highest result of 77.12%. Our IGNN method gives superior results with an accuracy of 95.78% higher than Slither 18.66%. F1 score of IGNN is higher than Slither's 22.63%. That shows that using of IGNN to vulnerability detection has great potential.

Comparison on Timestamp Dependence Vulnerability Detection. We continue to compare the results of vulnerability detection timestamp dependence. Our comparison results are shown in the middle part of Table II. The state-of-the-art existing methods that achieves the highest accuracy result is 74.20% while IGNN achieves 90.12% which is 15.92% higher than that. This result shows that existing methods using predefined rules can be easily fooled and fail to detect new signs of vulnerability. Moreover, IGNN keeps delivering the best performance in terms of all the four metrics.

Comparison on Infinite Loop Vulnerability Detection. We evaluate our IGNN for infinite loop vulnerability by comparing test results with several methods including:

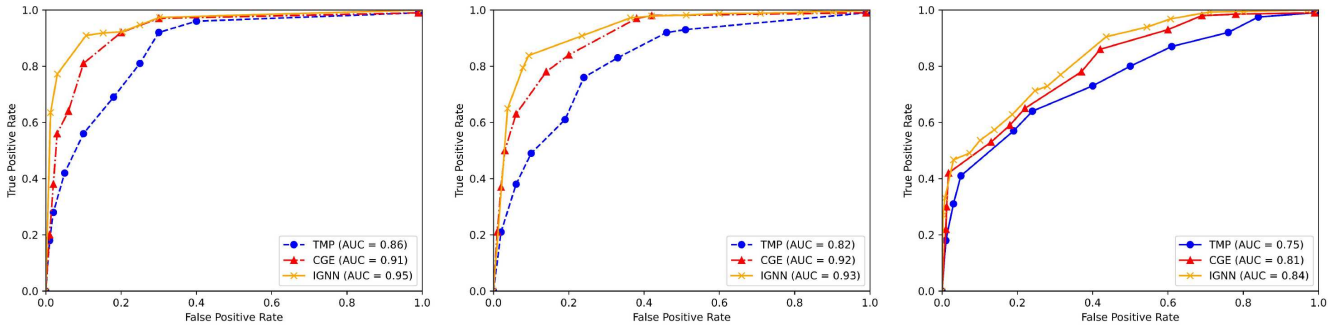
- Jolt [15] checks of 2 consecutive loops, thereby detecting the infinite loop security vulnerability.
- SMT [16] automates detection of infinite loop bugs by relying on satisfiability modulo theories.
- PDA [17] performs program path-based checking for infinite loop detection.
- Looper [18] uses symbolic execution to detect loop bugs.

The last vulnerability we compare is the Infinity Loop. Comparing our method with 4 other methods, the results are shown in the right part of Table II. Looking at Table II, it can be seen that the accuracy of the existing vulnerability detection methods is quite low. One of the reasons is the variety of Infinity vulnerability detection rules, this

[6]Ethereum Smart Contracts
[7]VNT chain Smart Contracts

TABLE II
PERFORMANCE COMPARISON (ACCURACY, RECALL, PRECISION, AND F1 SCORE). A TOTAL OF SEVENTEEN METHODS ARE INVESTIGATED IN THE COMPARISON, INCLUDING STATE-OF-THE ART VULNERABILITY DETECTION METHODS, NEURAL NETWORK-BASED ALTERNATIVES, DR-GCN, TMP, CGE, AND IGNN. "–" DENOTES NOT APPLICABLE

Methods	Reentrancy (ESC dataset)				Timestamp dependence (ESC dataset)				Methods	Infinite Loop (VSC dataset)			
	Acc(%)	Recall(%)	Precision(%)	F1(%)	Acc(%)	Recall(%)	Precision(%)	F1(%)		Acc(%)	Recall(%)	Precision(%)	F1(%)
Smartcheck	52.97	32.08	25.00	28.10	44.32	37.25	39.16	38.18	Jolt	42.88	23.11	38.23	28.81
Oyente	61.62	54.71	38.16	44.96	59.45	38.44	45.16	41.53	PDA	46.44	21.73	42.96	28.26
Mythril	60.54	71.69	39.58	51.02	61.08	41.72	50.00	45.49	SMT	54.04	39.23	55.69	45.98
Securify	71.89	56.60	50.85	53.57	–	–	–	–	Looper	59.56	47.21	62.72	53.87
Slither	77.12	74.28	68.42	71.23	74.20	72.38	67.25	69.72	–	–	–	–	–
Vanilla-RNN	49.64	58.78	49.82	50.71	49.77	44.59	51.91	45.62	Vanilla-RNN	49.57	47.86	42.10	44.79
LSTM	53.68	67.82	51.65	58.64	50.79	59.23	50.32	54.41	LSTM	51.28	57.26	44.07	49.80
GRU	54.54	71.30	53.10	60.87	52.06	59.91	49.41	54.15	GRU	51.70	50.42	45.00	47.55
GCN	77.85	78.79	70.02	74.15	74.21	75.97	68.35	71.96	GCN	64.01	63.04	59.96	61.46
DR-GCN	81.47	80.89	72.36	76.39	78.68	78.91	71.29	74.91	DR-GCN	68.34	67.82	64.89	66.32
TMP	84.48	82.63	74.06	78.11	83.45	83.82	75.05	79.19	TMP	74.61	74.32	**73.89**	74.10
CGE	89.15	87.62	85.24	86.41	89.02	88.10	87.41	87.75	CGE	77.26	56.38	67.51	61.44
IGNN	**95.78**	**94.69**	**93.04**	**93.86**	**90,12**	**89.09**	**87.79**	**89.04**	**IGNN**	**79.75**	**77.47**	73.64	**84.40**



(a) Reentrancy (a) Timestamp dependence (a) Infinite loop

Figure 8. ROC analysis for TMP, CGE, IGNN on the three vulnerability detection tasks. AUC stands for area under the curve.

vulnerability can appear under many different identifiers. For our IGNN, the detection results are much higher with an accuracy of 79.75%, higher than existing methods with the highest accuracy of 20.19%.

3. Comparison Results

We continue to compare our method with other neural networks to discover which method performs the best in detecting smart contract vulnerabilities. Specifically, we compare with some methods such as Vanilla-RNN (Recurrent neural network) [19], LSTM (Long Short-Term Memory networks)[20], GRU (gated recurrent units) [21], GCN (graph convolutional network) [22], DR-GCN [6], TMP (temporal message propagation network) [6], CGE (combining graph feature and expert patterns) [23].

We give the results of comparing methods on Table II and Fig. 8. Looking at Table II, it can be seen that IGNN gives very good results s in terms of all the 4 metrics. The experimental results show that conventional recurrent neural networks Vanilla-RNN, LSTM, and GRU perform no better than State-of-the-art existing methods. However, the GNN-based methods such as GCN, DR-GCN, TMP, CGE and IGNN which are capable of handling graphs, achieve significantly better results than existing methods. Furthermore, it can be seen that converting graph data to images shows better results, it also shows that detecting feature features based on images has better results when done with graphs.

V. CONCLUSION

We have proposed using the Convolutional Neural Network model for automatic detection of smart contract vulnerabilities. Different from the existing methods of using rules defined by experts, leading to low detection accuracy and non-scalable, We combine contract graph with defined security pattern and normalize to grayscale images. We also explore the use of Convolutional Neural Network to learn from normalized images from contract graph and security pattern. Our experiments show superiority over existing vulnerability detection methods and other neural network-based methods. We believe this will be an important step

to further research and develop deep learning techniques in smart contract vulnerability detection. In the future, we will explore this architecture on other vulnerabilities and extend the ability to combine the features of the vulnerabilities instead of training each vulnerability separately.

REFERENCES

[1] S. Wang, Y. Yuan, X. Wang, J. Li, R. Qin, and F.-Y. Wang, "An overview of smart contract: Architecture, applications, and future trends," in *2018 IEEE Intelligent Vehicles Symposium (IV)*, 2018, pp. 108–113.

[2] Y. Liu, J. Xu, and B. Cui, "Smart contract vulnerability detection based on symbolic execution technology," in *Cyber Security*, W. Lu, Y. Zhang, W. Wen, H. Yan, and C. Li, Eds. Singapore: Springer Nature Singapore, 2022, pp. 193–207.

[3] J. Feist, G. Grieco, and A. Groce, "Slither: A static analysis framework for smart contracts," 08 2019.

[4] L. Luu, D.-H. Chu, H. Olickel, P. Saxena, and A. Hobor, "Making smart contracts smarter," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 254–269. [Online]. Available: https://doi.org/10.1145/2976749.2978309

[5] P. Tsankov, A. Dan, D. Drachsler-Cohen, A. Gervais, F. Bünzli, and M. Vechev, "Securify: Practical security analysis of smart contracts," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 67–82. [Online]. Available: https://doi.org/10.1145/3243734.3243780

[6] Y. Zhuang, Z. Liu, P. Qian, Q. Liu, X. Wang, and Q. He, "Smart contract vulnerability detection using graph neural network." in *IJCAI*, 2020, pp. 3283–3290.

[7] M. Allamanis, M. Brockschmidt, and M. Khademi, "Learning to represent programs with graphs," *CoRR*, vol. abs/1711.00740, 2017. [Online]. Available: http://arxiv.org/abs/1711.00740

[8] "everything you need to know about min-max normalization," https://towardsdatascience.com/everything-you-need-to-know-about-min-max-normalization-in-python-b79592732b79, 2020, website.

[9] "Ethereum," https://github.com/ethereum/go-ethereum, 2015, website.

[10] "Vntchain," https://github.com/vntchain/go-vnt, 2018, website.

[11] "A framework for bug hunting on the ethereum blockchain," https://github.com/ConsenSys/mythril, 2017, website.

[12] S. Tikhomirov, E. Voskresenskaya, I. Ivanitskiy, R. Takhaviev, E. Marchenko, and Y. Alexandrov, "Smartcheck: Static analysis of ethereum smart contracts," in *Proceedings of the 1st International Workshop on Emerging Trends in Software Engineering for Blockchain*, 2018, pp. 9–16.

[13] P. Tsankov, A. Dan, D. Drachsler-Cohen, A. Gervais, F. Buenzli, and M. Vechev, "Securify: Practical security analysis of smart contracts," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018, pp. 67–82.

[14] J. Feist, G. Grieco, and A. Groce, "Slither: a static analysis framework for smart contracts," in *2019 IEEE/ACM 2nd International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB)*. IEEE, 2019, pp. 8–15.

[15] M. Carbin, S. Misailovic, M. Kling, and M. C. Rinard, "Detecting and escaping infinite loops with jolt," in *European Conference on Object-Oriented Programming*. Springer, 2011, pp. 609–633.

[16] M. Kling, S. Misailovic, M. Carbin, and M. Rinard, "Bolt: on-demand infinite loop escape in unmodified binaries," *ACM SIGPLAN Notices*, vol. 47, no. 10, pp. 431–450, 2012.

[17] A. Ibing and A. Mai, "A fixed-point algorithm for automated static detection of infinite loops," in *2015 IEEE 16th International Symposium on High Assurance Systems Engineering*. IEEE, 2015, pp. 44–51.

[18] J. Burnim, N. Jalbert, C. Stergiou, and K. Sen, "Looper: Lightweight detection of infinite loops at runtime," in *2009 IEEE/ACM International Conference on Automated Software Engineering*. IEEE, 2009, pp. 161–169.

[19] C. Goller and A. Kuchler, "Learning task-dependent distributed representations by backpropagation through structure," in *Proceedings of International Conference on Neural Networks (ICNN'96)*, vol. 1. IEEE, 1996, pp. 347–352.

[20] H. Sak, A. W. Senior, and F. Beaufays, "Long short-term memory recurrent neural network architectures for large scale acoustic modeling," 2014.

[21] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *arXiv preprint arXiv:1412.3555*, 2014.

[22] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks. 2017," *ArXiv abs/1609.02907*, 2017.

[23] Z. Liu, P. Qian, X. Wang, Y. Zhuang, L. Qiu, and X. Wang, "Combining graph neural networks with expert knowledge for smart contract vulnerability detection," *IEEE Transactions on Knowledge and Data Engineering*, 2021.

Ta Minh Thanh is currently an associate professor and vice dean of Faculty of Information Technology in Le Quy Don Technical University Vietnam. He is also a Postdoctoral Fellow of the Department of Mathematical and Computing Sciences at Tokyo Institute of Technology. He received his B.S. and M.S in Computer Science from National Defense Academy, Japan, in 2005 and 2008 and his Ph.D. from Tokyo Institute of Technology, Japan, in 2015, respectively.He is the member of IPSJ Japan and IEEE. His research interests lie in the area of watermarking, network security, and computer vision.
Email: thanhtm@lqdtu.edu.vn

Pham Trong Linh is currently studying network technology at Le Quy Don University. His field of research is digital watermarking for digital multimedia .
Email: tronglinhmta0611@gmail.com

An Improvemant of the IGEPVO-k Reversible Data Hiding Method

Le Kinh Tai[1], Nguyen Ngoc Hung[2], Dang Tran Duc[3], Can Duc Diep[4], Pham Minh Thai[5]
[1]Training department, Trade Union University, 169 Tay Son, Dong Da, Hanoi, Vietnam
[2]Instutite of Information Technology, Vietnam Academy of Science and Technology, 18 Hoang Quoc Viet, Hanoi, Vietnam
[3]VVECO Joint Stock Compa¬ny, 89 Nguyen Phong Sac, Cau Giay, Hanoi, Vietnam
[4]Electric Power University, 235 Hoang Quoc Viet, Hanoi, Vietnam
[5]Faculty of Information Technology, University of Economics-Technology for Industries 454-456, Minh Khai, Hanoi, Vietnam

Correspondence: Nguyen Ngoc Hung (nnhung@ioit.ac.vn)
Communication: received 31 Jan 2023, revised 10 May 2023, accepted 25 May 2023
Digital Object Identifier: 10.32913/mic-ict-research.v2023.n2.1207

Abstract: PVO-K method is an expansion of PVO method in which all k largest (l smallest pixels) are used as a single unit to embed a bit. In PVOK-method, each pixel of an original image is modified by one at most. GePVO-K method is again a development of the PVOK-method by embedding k bits in k largest pixels and l bits in l smallest pixels. However, this method has two drawbacks. First, each pixel of the original image is changed by two at most, so marked image quality is not good. Second, the location map consisting of two bits for each pixel is so large that in some cases compression code length of the map is greater method's embedding capacity, and then data embedding capacity is zero. Both drawbacks are overcome in the iGePVO-K method. Specifically, in this method, each pixel only is changed by one at most and the two-bit location map as in iGePVO-K method is replaced by a one-bit map and a sequence of binary flags. This paper is a development of the iGePVO-K method by discarding the sequence of binary flags. That means it is not necessary to embed flags with the data in the original image. Therefore, the proposed method not only has a larger data embedding capacity but also has a better image quality than the methods mentioned above.

Keywords: *PVO, PVOK, GePVOK, iGePVOK, reversible data hiding, flag bits.*

I. Introduction

Along with cryptography, data hiding is an important research direction in the field of information security. Data hiding is a technique in which secret data is concealed in a digital image. Traditional data-hiding methods can only extract embedded data but do not restore the original image. This limits their applications. in some sensitive fields, such as medical image processing and military communications. So, new data hiding techniques called reversible (or lossless) data hiding are proposed to overcome this limitation. Reversible data hiding (RDH) techniques not only can extract embedded data but also restore exactly the original image.

The first RDH methods were built based on lossless compression [1–4], in that to have an embedding space we use a lossless compression technique for a certain region of the original image. Since obtained embedding space usually is not large, the embedding capacity (EC) of these methods is generally not high.

After then, Tian first proposed a technique called difference expansion (DE) [5]. The basic idea of DE is to expand the difference between two neighbouring pixels to embed data. Afterwards, the technique of DE got further improvements in the paper [6–8]. Later, the technique of prediction-error expansion (PEE) was proposed by Thodi and Rodriguez [9]. Unlike DE, PEE utilizes more neighbouring pixels and gets more accurate predictions. Then, this method got further investigated [10, 11], and show superior performance. Besides, DE, Ni et al. proposed a histogram-shifting (HS) based method which is of great value [12]. In this method, the peak pixel values of the host image histogram were modified to embed data. When embedding fewer data, Ni et al.'s algorithm has good performance. However, the host image with a flat histogram has a low EC, and the shortcoming had been overcome partly in the paper [13, 14] by introducing the difference-histogram.

The smoothness of pixel blocks is often exploited by many papers [15–17] because using RDH methods on the smooth blocks not only achieves the high embedding capacity but also causes little distortion of pixels.

In [18], Li et al. proposed an excellent predictor based on pixel value ordering (PVO) for embedding data. Specifically, Li et al.'s method divided an original image into non-overlapped blocks of n pixels. Each block is sorted in ascending order, the largest pixel is predicted by the second largest pixel, the second largest pixel and the smallest pixel is predicted by the second smallest pixel. Then one bit is embedded in the largest pixel (smallest) pixel when the prediction error dmax (dmin) is equal to 1 (-1). This method has a high stego image quality, but a low embedding capacity. Peng et al. [19] were given an improvement of the PVO method called the IPVO method in which the information about the positions of the largest pixel and the second-largest pixel in the original pixel block is used to determine dmax. Specifically, dmax is equal to second largest pixel minus largest pixel or vice versa depending on second largest pixel is before or after largest pixel in the original pixel block. The prediction error dmin is computed similarly by using second smallest pixel and smallest pixel. Then, one bit is embedded in the largest (smallest) pixel when dmax (dmin) is equal to zero or one. In general, the IPVO method has an embedding capacity larger than that of the PVO method. It is noted that if a pixel block has $k \geq 2$ pixels being equal to the largest pixel or has $l \geq 2$ pixels being equal to the smallest pixel, then dmax or dmin is equal to zero. When no bit is embedded in the largest pixel or the smallest pixel by the PVO method. To overcome this weakness, Ou et al. [20] have proposed the PVO-K method in which all largest pixels are considered as a unit to embed a bit. In other words, all the largest pixels are kept unchanged or increased by one to embed a bit with a value of 0 or 1. Data embedding in the smallest pixels is carried out similarly. The embedding capacity of this is improved compared with the PVO method, but not very much.

Li et al. [21] have proposed an extension of the PVO-K method, called GePVO-K. In this method, k bits are embedded in the k largest pixels and l bits are embedded in the l smallest pixels. But in the implementation process of this method, each pixel may be modified by 2 at most. So, the marked image quality of Weng et al.'s method is low. Moreover, the location map consists of two bits for each pixel block (two-bit map) and is too large that in some cases the compression code length of the map is greater than the embedding capacity of the method. In these cases, the data embedding capacity (payload) is zero.
Sao et al.'s method [22] is a development of [21], in which the above weaknesses are partially overcome. Specifically, in this method, each pixel is changed by one at most. However, the two-bit map problem has not been completely solved yet. In [22], the two-bit map is replaced by a one-bit map and a sequence of binary flags. Embedding the flag sequence along with the data in the original image reduces the payload and marked image quality of Sao et al.'s method.

In this paper, we propose some improvements in the embedding algorithm of [22] to discard the flag sequence. That is, there is no need to use flag sequences in the embedding and extracting algorithms. So, both the payload and marked image quality of the proposed are better than above mentioned PVO-based methods.
The rest of the paper is organized as follows. The relative words are presented in section 2. The proposed method is introduced in section 3. The experiment results of comparing methods are given in section 4. Finally, some conclusions are provided in section 5.

II. Related Works

In this section, we briefly present methods PVO [18], PVO-K [20], GePVO-K [21] and iGePVO-K [22].

1. PVO method

The method PVO is proposed by Li et al. [18], in which an original gray-scale image of size $R \times C$ is divided into non-overlapped blocks of n pixels. Each block $X = (x_1, ..., x_n)$ is sorted in ascending order to get $X_\sigma = (x_{\sigma(1)}, ..., x_{\sigma(n)})$ such that $x_{\sigma(1)} \leq ... \leq x_{\sigma(n)}$. Moreover if $i < j$ and $x_{\sigma(i)} = x_{\sigma(j)}$ then $\sigma_i < \sigma_j$. One data bit b can be embedded in the largest pixel $x_{\sigma(n)}$ using the formulas:

$$dmax = x_{\sigma(n)} - x_{\sigma(n-1)}. \tag{1}$$

$$x'_{\sigma(n)} = \begin{cases} x_{\sigma(n)} + b, & \text{if } dmax = 1, \\ x_{\sigma(n)} + 1, & \text{no bit is embedded,if } dmax > 1, \\ x_{\sigma(n)}, & \text{no bit is embedded,if } dmax = 0. \end{cases} \tag{2}$$

Similarly, a bit is embedded in the smallest pixel $x_\sigma(1)$ as follows.

$$dmin = x_{\sigma(1)} - x_{\sigma(2)}. \tag{3}$$

$$x'_{\sigma(1)} = \begin{cases} x_{\sigma(1)} - b, & \text{if } dmin = -1, \\ x_{\sigma(1)} - 1, & \text{if } dmin = -1, \\ x_{\sigma(1)}, & \text{if } dmin = 0. \end{cases} \tag{4}$$

It is clear that after embedding, the PVO (pixel value ordering) of X_σ is kept unchanged, and pixels $x_{\sigma(i)}$ with $i = 2, ..., n-1$ are preserved.
At the decoder side, extracting b and restoring $x_{\sigma(n)}$ can be performed as follows:

$$dmax' = x'_{\sigma(n)} - x'_{\sigma(n-1)} \tag{5}$$

$$b = dmax^{'} - 1, \text{ if } dmax^{'} \in \{1, 2\}. \quad (6)$$

$$x_{\sigma(n)} = \begin{cases} x^{'}_{\sigma(n)} - b, \text{ if } dmax^{'} \in \{1, 2\}, \\ x^{'}_{\sigma(n)} - 1, \text{ if } dmax^{'} > 2, \\ x^{'}_{\sigma(n)}, \text{ if } dmax^{'} = 0. \end{cases} \quad (7)$$

Similarly, extracting b and restoring $x_{\sigma}(1)$ can be performed as follows:

$$dmin^{'} = x^{'}_{\sigma(1)} - x^{'}_{\sigma(2)} \quad (8)$$

$$b = abs(dmin^{'}) - 1, \text{ if } dmin^{'} \in \{-1, -2\}. \quad (9)$$

$$x_{\sigma(1)} = \begin{cases} x^{'}_{\sigma(1)} + b, \text{ if } dmin^{'} \in \{-1, -2\}, \\ x^{'}_{\sigma(1)} + 1, \text{ if } dmin^{'} < -2, \\ x^{'}_{\sigma(1)}, \text{ if } dmin^{'} = 0. \end{cases} \quad (10)$$

2. PVO-K method

The method PVO-K is proposed by Ou Li et al. [20] by embedding one bit in the largest pixels, and one bit in the smallest pixels. Suppose sequence X_{σ} has k largest pixels and l smallest pixels, that means:

$$x_{\sigma(1)} = ... = x_{\sigma(l)} < x_{\sigma(l+1)} < ... \leq x_{\sigma(n-k)} < x_{\sigma(n-k+1)} = ... = x_{\sigma(n)}$$

Then similar to (1-2), embedding one bit in k of the largest pixels simultaneously is performed by formulas:

$$dmax = x_{\sigma(n)} - x_{\sigma(n-k)},$$

$$x^{'}_{\sigma(i)} = \begin{cases} x_{\sigma(i)} + b, \text{ if } dmax = 1, \\ x_{\sigma(i)} + 1, \text{ no bit is embedded, if } dmax > 1, \\ x_{\sigma(i)}, \text{ no bit is embedded, if } dmax = 0, \end{cases}$$

with $i = n - k + 1, ..., n$.

Embedding one bit in the l smallest simultaneously is carried similar to (3-4) as follows.

$$dmin = x_{\sigma(1)} - x_{\sigma(l+1)},$$

$$x^{'}_{\sigma(i)} = \begin{cases} x_{\sigma(i)} - b, \text{ if } dmax = -1, \\ x_{\sigma(i)} - 1, \text{ no bit is embedded, if } dmin < -1, \\ x_{\sigma(i)}, \text{ no bit is embedded, if } dmin = 0, \end{cases}$$

with $i = 1, ..., l$.

At the decoder, extracting the embedded bit and restoring k largest pixels is done similarly to (5-7) by formulas:

$$dmax^{'} = x^{'}_{\sigma(n)} - x^{'}_{\sigma(n-k)}$$

$$b = dmax^{'} - 1, \text{ if } dmax^{'} \in \{1, 2\},$$

$$x_{\sigma(i)} = \begin{cases} x^{'}_{\sigma(i)} - b, \text{ if } dmax^{'} \in 1, 2, \\ x^{'}_{\sigma(i)} - 1, \text{if } dmax^{'} > 2, \\ x^{'}_{\sigma(i)}, \text{ if } dmax^{'} = 0, \end{cases}$$

with $i = n - k + 1, ..., n$.

Similar to (8-10), extracting the embedded bit and restoring l smallest pixels is performed as follows.

$$dmin^{'} = x^{'}_{\sigma(1)} - x^{'}_{\sigma(l+1)}$$

$$b = abs(dmin)^{'} - 1, \text{ if } dmin^{'} \in \{-1, -2\},$$

$$x_{\sigma(i)} = \begin{cases} x^{'}_{\sigma(i)} + b, \text{ if } dmin^{'} \in \{-1, -2\}, \\ x^{'}_{\sigma(i)} + 1, \text{ if } dmin^{'} < -2, \\ x^{'}_{\sigma(i)}, \text{ if } dmin^{'} = 0, \end{cases}$$

with $i = 1, ..., l$.

In PVO-K, a one-bit block-based location map that consists of one bit for each block is built as follows. Assign 1 for blocks having at least one pixel with a value equal to 0 or 255 and assign 0 for the remaining blocks. The blocks with a map index of 1 should not be used for embedding as that would cause underflow/overflow.

3. GePVO-K method

The GePVO-K method [21] is a generalization of the PVO-K method by embedding k bits in k largest pixels and l bits in l smallest pixels. In this method, blocks of n pixels are classified into three types:

(a) Blocks with at least one-pixel values of 0, 1, 254 and 255 should not be used for data embedding as this may cause underflow/underflow. Their index in the location map is set by 2 ($LM(X) = 2$)

(b) Blocks with all equal pixels: $x_1 = x_2 = ... = x_n = \alpha, \alpha \in \{2, ..., 243\}$ (flat blocks) will be used to embed $n - 1$ bits. Their index in the location map is set by 1 ($LM(X) = 1$)

(c) The remaining blocks (not-flat blocks) will be used to embed k bits in k largest pixels and l bits in l smallest pixels. Their index in the location map is set by 0 ($LM(X) = 0$)

Since each index in the map is a two-bit binary number, the location map is called a two-bit map that is a binary sequence with a length twice the number of blocks.

The processing of each block depends on its map index as follows.

Case 2.3.1: $LM(X) = 2$, block X is not used to embed the data and it is skipped.

Case 2.3.2: $LM(X) = 1$, keep the first pixel and add $(n-1)$ to-be-embedded bits to the remaining $(n - 1)$ pixels:

$$x^{'}_{i} = \begin{cases} x_i, \text{ if } i = 1, \\ x_i + b_{i-1}, \text{ if } i = 1, ..., n, \end{cases}$$

where b_i are to-be-embedded bits.

Case 2.3.3: $LM(X) = 0$, the block X is sorted in ascending order to get $X_{\sigma} = (x_{\sigma(1)}, ..., x_{\sigma(n)})$. Suppose X_{σ} has k_1

largest pixels and k_2 second largest pixels. Then compute $dmax = x_{\sigma(n)} - x_{\sigma(n-k_1)}$ and embed the data bits based on value of $dmax$ as follows:
Case 2.3.3.1: $dmax > 1$, no bit is embedded, and all largest pixels are increased by 1.
Case 2.3.3.2: $dmax = 1$, k_1 bits $(b_i, i = 1, ..., k_1)$ are embedded in k_1 largest pixels by formulas:

$$x'_{\sigma(i)} = \begin{cases} x_{\sigma(i)} + 1, & i = n - k_1 - k_2 + 1, ..., n - k_1, \\ x_{\sigma(i)} + 1 + b_{i-n+k_1}, & i = n - k_1 + 1, ..., n. \end{cases}$$

For the smallest pixels, suppose X_σ has l_1 smallest pixels and l_2 second smallest pixels. Compute $dmin = x_{\sigma(1)} - x_{\sigma(l_1+1)}$ and data embedding on l_1 smallest pixels is performed in the same way as for k_1 largest pixels.

4. iGePVO-K method

iGePVO-K method [22] is an improvement of GePVO-K in which each pixel is changed by one at most. The location map of this method is a one-bit based-blocked map. Blocks are classified into types:

(a) The block with at least a pixel value equal to 0 or 255 is called underflow/overflow.
(b) The block with all equal pixels is called flat, otherwise is called rough.
(c) The block which contains s different pixel values, with $s > 1$, is called a s-block.

First, a location map is established to distinguish underflow/overflow and non-underflow/overflow blocks by assigning map index 1 for the former $(LM(X) = 1)$ and 0 for the latter $(LM(X) = 0)$.

a) Data embedding in the largest pixels

Data embedding in the largest pixels is performed for the following case:
Case 2.4.1.1: X is rough and $LM(X) = 1$. X is not used to embed data, it is ignored because embedding in this block can cause underflow/overflow.
Case 2.4.1.2: X is a flat block with $x_1 = x_2 = ... = x_n = \alpha$, $0 \le \alpha \le 255$. In this case $(n-1)$ bits are embedded in X as follows:

$$x'_i = \begin{cases} x_i + b_{i-1}, & \text{if } \alpha \le 254, i = 2, ..., n, \\ x_i - b_{i-1}, & \text{if } \alpha = 255, i = 2, ..., n, \\ x_1, & \text{if } i = 1. \end{cases}$$

Case 2.4.1.3: X is a 2-block and $LM(X) = 0$. In this case, X consists of two different pixels values:

$$x_{\sigma(1)} = ... = x_{\sigma(n-k)} < x_{\sigma(n-k+1)} = ... = x_{\sigma(n)}$$

Compute:

$$dmax = x_{\sigma(n)} - x_{\sigma(n-k)}$$

2.4.1.3.1. If $dmax \ge 2$, no bit embedded and k largest pixels are increased by 1:

$$x'_{\sigma(i)} = x_{\sigma(i)} + 1, i = n - k + 1, ..., n,$$

2.4.1.3.2. If $dmax = 1$, k bits are embedded in k largest pixels as follows:

(a) If all k bits are equal to 1, then pixels are modified as in 2.4.1.3.1:

$$x'_{\sigma(i)} = x_{\sigma(i)} + 1, i = n - k + 1, ..., n,$$

(b) If all k bits are equal to 0, then k largest pixels are kept unchanged:

$$x'_{\sigma(i)} = x_{\sigma(i)}, i = n - k + 1, ..., n.$$

(c) If k bits include both 0 and 1: k bits are added to k largest pixels:

$$x'_{\sigma(i)} = x_{\sigma(i)} + b_{i-n+k}, i = n - k + 1, ..., n.$$

Case 2.4.1.4: X is a s-block with $s \ge 3$ and $LM(X) = 0$:

$$x_{\sigma(1)} \le ... = x_{\sigma(n-k_1-k_2)} < x_{\sigma(n-k_1-k_2+1)} = ... = x_{\sigma(n-k_1)} < x_{\sigma(n-k_1+1)} = ... = x_{\sigma(n)}$$

Compute:

$$dmax = x_{\sigma(n)} - x_{\sigma(n-k_1)}$$

2.4.1.4.1 If $dmax \ge 2$, no bit is embedded and then k_1 largest pixels are increased by 1:

$$x'_{\sigma(i)} = x_{\sigma(i)} + 1, i = n - k_1 + 1, ..., n.$$

2.3.4.4.2 If $dmax = 1$, k_1 bits are embedded in k_1 largest pixels as follows:

(a) If all k_1 bits are equal to 1, then k_1 largest pixels are increased by 1:

$$x'_{\sigma(i)} = x_{\sigma(i)} + 1, i = n - k_1 + 1, ..., n.$$

(b) If all k_1 bits are equal to 0, then k_1 largest pixels and k_2 second largest pixels are increased by 1:

$$x'_{\sigma(i)} = x_{\sigma(i)} + 1, i = n - k_1 - k_2 + 1, ..., n.$$

(c) If k_1 bits include both 0 and 1: k_1 bits are added to k_1 largest pixels:

$$x'_{\sigma(i)} = x_{\sigma(i)} + b_{i-n+k_1}, i = n - k_1 + 1, ..., n.$$

b) Data embedding in the smallest pixels

Data embedding in the smallest pixels only is performed in the cases 2.4.1.3 and 2.4.1.4, that is X is a s-block with $s \geq 2$.

For convenience, the pixel block obtained after performing the steps in subsection 2.4.1 is also denoted by X, then X is naturally an s block with $s \geq 2$. Sort X in ascending order to get X_σ. Consider the following two cases:

Case 2.4.2.1: s = 2. Then

$$x_{\sigma(1)} = ... = x_{\sigma(l)} < x_{\sigma(l+1)} = ... = x_{\sigma(n)}$$

Compute:

$$dmin = x_{\sigma(1)} - x_{\sigma(l+1)}$$

2.4.2.1.1. If $dmin \leq -2$, no bit embedded and l smallest pixels are decreased by 1:

$$x^{'}_{\sigma(i)} = x_{\sigma(i)} - 1, i = 1, ..., l$$

2.4.2.1.2. If $dmin = -1$, l bits are embedded in l smallest pixels as follows:

(a) If all l bits are equal to 1, then l smallest pixels are decreased by 1 as in the previous case.

(b) If all l bits are equal to 0, then l smallest pixels are kept unchanged:

$$x^{'}_{\sigma(i)} = x_{\sigma(i)}, i = 1, ..., l.$$

(c) If l bits include both 0 and 1: l bits are subtracted from l smallest pixels:

$$x^{'}_{\sigma(i)} = x_{\sigma(i)} - b_{(i-n+k)}, i = n-k+1, ..., n.$$

Case 2.4.2.2: $s \geq 3$. Then

$$x_{\sigma(1)} = ... = x_{\sigma(l_1)} < x_{\sigma(l_1+1)} = ... = x_{\sigma(l_1+l_2)} < x_{\sigma(l_1+l_2+1)} \geq ... \geq x_{\sigma(n)}$$

Compute

$$dmin = x_{\sigma(1)} - x_{\sigma(l_1+1)}.$$

2.4.2.2.1 If $dmin \leq -2$, no bit is embedded, l_1 smallest pixels are decreased by one.

$$x^{'}_{\sigma(i)} = x_{\sigma(i)} - 1, i = 1, ..., l_1$$

2.4.2.2.2 If $dmin = -1$, embed l_1 bits in l_1 smallest pixels.

(a) If all l_1 to-be-embedded bits are equal to 1: Decrease l_1 smallest pixels by one.

$$x^{'}_{\sigma(i)} = x_{\sigma(i)} - 1, i = 1, ..., l_1$$

(b) If all l_1 to-be-embedded bits are equal to 0: Decrease l_1 smallest pixels and l_2 second smallest pixels by one.

$$x^{'}_{\sigma(i)} = x_{\sigma(i)} - 1, i = 1, ..., l_1 + l_2$$

(c) If l_1 to-be-embedded bits include both 0 and 1: Subtract l_1 bits from l_1 smallest pixels:

$$x^{'}_{\sigma(i)} = x_{\sigma(i)} - b_i, i = 1, ..., l_1$$

After completing the steps in case 2.4.2.2, the marked pixel block $X^{'}$ that contains to-be-embedded bits is obtained.

For ease of presenting the next sections, the following concept is introduced. A block whose pixel value set contains two consecutive values α and β, with $|\alpha - \beta| = 1$ is called a 2-flat block. The block in case 2.4.1.3.2 of subsection 2.4.1 is a 2-flat block.

It is noted that after embedding the data, both flat and 2-flat block types can be converted to 2-flat. So, a sequence of flag bits in which flag 0 for a flat block and 1 for a 2-flat block is established for distinguishing flat and 2-flat blocks in the extracting process.

c) Extracting and restoring in smallest pixels

Given a marked block $X^{'}$, extracting data and restoring pixels on the left are performed as follows.

Case 2.4.3.1: $LM(X) = 1$, X is an underflow/overflow block. No bit is extracted, and $X = X^{'}$.

Case 2.4.3.2: $X^{'}$ is flat or 2-flat, then X is also flat or 2-flat. To determine correctly the type of X, use the sequence of flag bits. If the flag is 0, X is flat, extracting and restoring by formulas:

$$b_{i-1} = \begin{cases} x^{'}_i - x^{'}_1, & \text{if } x^{'}_1 < 255, \\ x^{'}_1 - x^{'}_i, & \text{if } x^{'}_1 = 255, \end{cases} \quad (i = 2, \ldots, n)$$

$x_i = x^{'}_i$, $(i = 1, \ldots, n)$.

Case 2.4.3.3: $X^{'}$ is 2-block:

$$x^{'}_{\sigma(1)} = \ldots = x^{'}_{\sigma(l)} < x^{'}_{\sigma(l+1)} = \ldots = x^{'}_{\sigma(n)}$$

Compute

$$\text{dmin} = x^{'}_{\sigma(1)} - x^{'}_{\sigma(l+1)}.$$

(a) If $dmin \leq -3$, no bit is extracted, l smallest pixels are increased by 1:

$$x_{\sigma(i)} = x^{'}_{\sigma(i)} + 1, \ i = 1, \ldots, l.$$

(b) If $dmin = -2$, all extracted l bits are equal to 1 and restoring l smallest pixels is carried out as in (a).

(c) If $dmin = -1$, all extracted l bits are equal to 0 and restoring l smallest pixels is as:

Case 2.4.3.4: $X^{'}$ is s-block with $s \geq 3$

$$x^{'}_{\sigma(1)} = .. = x^{'}_{\sigma(l_1)} < x^{'}_{\sigma(l_1+1)} = ... = x^{'}_{\sigma(l_1+l_2)} < x^{'}_{\sigma(l_1+l_2+1)} \leq ... \leq x^{'}_{\sigma(n)}$$

Compute

$$\text{dmin} = x'_{\sigma(1)} - x'_{\sigma(l_1+1)},$$

$$\text{dmin2} = x'_{\sigma(l_1+1)} - x'_{\sigma(l_1+l_2+1)}.$$

(a) If $dmin \leq -3$, no bit is extracted, l_1 smallest pixels are increased by 1:

$$x_{\sigma(i)} = x'_{\sigma(i)} + 1,\ i = 1, \ldots, l_1$$

(b) If $dmin = -2$, all extracted l_1 bits are equal to 1 and restoring l_1 smallest pixels is carried out as in (a).

(c) If $dmin = -1$ and $dmin2 \leq -2$, all extracted l_1 bits are equal to zero, restoring l_1 smallest pixels and l_2 second smallest pixels are performed by formulas:

$$x_{\sigma(i)} = x'_{\sigma(i)} + 1,\ i = 1, \ldots, l_1 + l_2.$$

(d) If $dmin = -1$ and $dmin2 = -1$, l_1 bits of 1 and l_2 bits of 0 are extracted. l_1 smallest pixels are increased by 1. Then it is necessary to rearrange the extracted bits as well as (l+m) the received smallest pixels as their original position in block X.

d) Extracting and restoring in largest pixels

The cases $LM(X) = 1$ and X' obtained from flat X have been considered in subsection 2.4.3. In this subsection, consider the next two cases.

Case 2.4.4.1: X' is 2-block:

$$x'_{\sigma(1)} = \ldots = x'_{\sigma(n-k)} < x'_{\sigma(n-k+1)} = \ldots = x'_{\sigma(n)}$$

Compute

$$\text{dmax} = x'_{\sigma(n)} - x'_{\sigma(n-k)}.$$

(a) If $dmax \geq 3$, no bit is extracted, k largest pixels are decreased by 1:

$$x_{\sigma(i)} = x'_{\sigma(i)} - 1,\ i = n - k + 1, \ldots, n.$$

(b) If $dmax = 2$, all extracted k bits are equal to 1 and restoring k largest pixels is carried out as in (a).

(c) If $dmax = 1$, all extracted k bits are equal to 0 and restoring k largest pixels is as:

$$x_{\sigma(i)} = x'_{\sigma(i)},\ i = n - k + 1, \ldots, n.$$

Case 2.4.4.2: X' is s-block with $s >= 3$:

$$x'_{\sigma(1)} \leq \ldots \leq x'_{\sigma(n-k_1-k_2)} < x'_{\sigma(n-k_1-k_2+1)} = \ldots = x'_{\sigma(n-k_1)} < x'_{\sigma(n-k_1+1)} \leq \ldots \leq x'_{\sigma(n)}$$

Compute

$$\text{dmax} = x'_{\sigma(n)} - x'_{\sigma(n-k_1)},$$

$$\text{dmax2} = x'_{\sigma(n-k_1)} - x'_{\sigma(n-k_1-k_2)}.$$

(a) If $dmax \geq 3$, no bit is extracted, k_1 largest pixels are decreased by 1:

$$x_{\sigma(i)} = x'_{\sigma(i)} - 1,\ i = n - k_1 + 1, \ldots, n$$

(b) If $dmax = 2$, all extracted k_1 bits are equal to 1 and restoring k_1 largest pixels is carried out as in (a).

(c) If $dmax = 1$ and $dmax2 \geq 2$, all extracted k bits are equal to zero, restoring k_1 largest pixels and k_2 second largest pixels is performed by formulas:

$$x_{\sigma(i)} = x'_{\sigma(i)} - 1,\ i = n - k_1 - k_2 + 1, \ldots, n$$

(d) If $dmax = 1$ and $dmax2 = 1$, k_1 bits of 1 and k_2 bits of 0 are extracted, k_1 largest pixels are decreased by 1. Then it is necessary to rearrange the extracted bits as well as (k_1+k_2) the received largest pixels as their original position in block X

III. PROPOSED METHOD

1. Comments on the iGePVO-K algorithm

One drawback of the iGePVO-K method is that this method needs to use a sequence of flag bits to distinguish flat and 2-flat blocks in the extracting stage. For ease of understanding, we consider a flat block X_1 and a 2-flat X_2 as follows:

$$X_1 = \begin{bmatrix} 24 & 24 \\ 24 & 24 \end{bmatrix}, X_2 = \begin{bmatrix} 24 & 25 \\ 25 & 24 \end{bmatrix}$$

Suppose we embed bit sequence (1, 1, 0) in X_1, then by subsection 2.4.1.2, we obtain a marked block as follows:

$$X'_1 = \begin{bmatrix} 24 & 25 \\ 25 & 24 \end{bmatrix}$$

Similarly, if embed bit sequence (0, 0, 0, 0) in X_2, then by case 2.4.1.3.2(b) and case 2.4.2.1.2(b), we obtain a marked block:

$$X'_2 = \begin{bmatrix} 24 & 25 \\ 25 & 24 \end{bmatrix}.$$

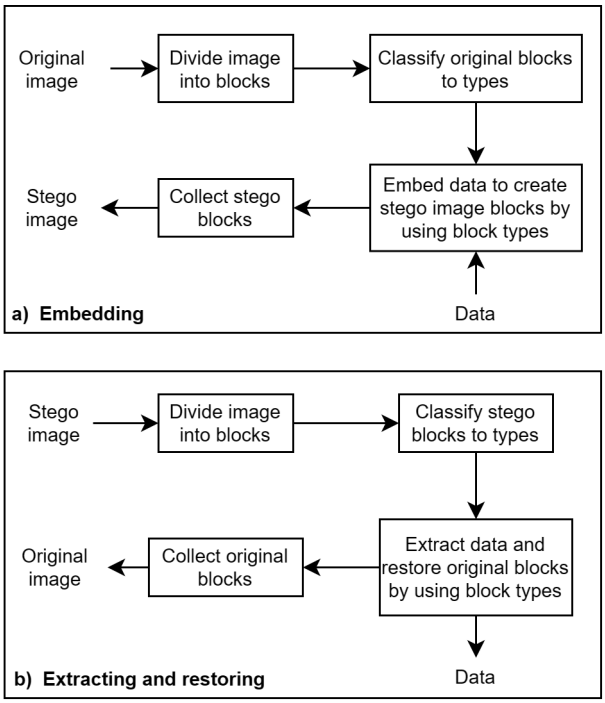


Figure 1. The framework of proposed method

Thus

$$X_1^{'} = X_2^{'} = \begin{bmatrix} 24 & 25 \\ 25 & 24 \end{bmatrix}.$$

Therefore, in the data extracting process, if encounter blocks $X_1^{'}$ and $X_2^{'}$, we will not know if their original blocks are flat or 2-flat. To avoid this ambiguity, we need to use the sequence of flags as follows: the flag of $X_1^{'}$ is equal to 0, then its original block X_1 is flat, while the flag of $X_2^{'}$ is equal to 1, then its original block X_2 is 2-flat.
To be reversible, the flag bits must be embedded with the data in the original image. The number of flag bits is equal to the sum of a flat block number and a flat 2 block number. In many cases, this number is relatively large. It can be in the thousands. Removing the flag bits in the proposed method expands embedding capacity considerably.

2. Embedding algorithm of the proposed method

To remove the flag sequence, it is necessary to modify the embedding formulas in the iGePVO-K method so that only flat blocks can lead to 2-flat marked blocks. To achieve this it is necessary to modify the embedding algorithm as follows.

a) Embedding data in the largest pixels

We will revise formulas in case 2.4.1.3 of the iGePVO-K method by the following new formulas in Case 3.4.1.3 below: Case 3.4.1.3: X is a 2-block and $LM(X) = 0$. In this case, X consists of two different pixel values:

$$x_{\sigma(1)} = ... = x_{\sigma(n-k)} < x_{\sigma(n-k+1)} = ... = x_{\sigma(n)}$$

Compute:

$$dmax = x_{\sigma(n)} - x_{\sigma(n-k)}$$

3.4.1.3.1. If $dmax \geq 2$, no bit embedded, k largest pixels are increased by 1, remaining pixels are decreased by 1:

$$x_{\sigma(i)}^{'} = x_{\sigma(i)} + 1, i = n-k+1, ..., n,$$

$$x_{\sigma(i)}^{'} = x_{\sigma(i)} - 1, i = 1, ..., n-k.$$

3.4.1.3.2. If $dmax = 1$, k bits are embedded in k largest pixels as follows:

1) If all k bits are equal to 1, then pixels are modified as in 2.4.1.3.1:

$$x_{\sigma(i)}^{'} = x_{\sigma(i)} + 1, i = n-k+1, ..., n,$$

$$x_{\sigma(i)}^{'} = x_{\sigma(i)} - 1, i = 1, ..., n-k.$$

2) If all k bits are equal to 0, then k largest pixels are increased by 1:

$$x_{\sigma(i)}^{'} = x_{\sigma(i)} + 1, i = n-k+1, ..., n.$$

3) If k bits include both 0 and 1: k bits are added to k largest pixels:

$$x_{\sigma(i)}^{'} = x_{\sigma(i)} + b_{i-n+k}, i = n-k+1, ..., n.$$

Except for case 2.4.1.3, other cases are not changed.

b) Embedding data in smallest pixels

In this subsection we revise case 2.4.2.1 of the iGePVO-K method by Case 3.4.2.1 below, and do not modify the remaining cases.
Case 3.4.2.1: X is 2-block:

$$x_{\sigma(1)} = ... = x_{\sigma(l)} < x_{\sigma(l+1)} = ... = x_{\sigma(n)}$$

Compute:

$$dmin = x_{\sigma(1)} - x_{\sigma(l+1)}$$

From subsection 3.2.1 it follows that

$$dmin \leq -2$$

So, no formula is performed. In short, ignore this case and do nothing.

TABLE I
MAXIMUM PAYLOAD COMPARISON BETWEEN METHODS

Images	PVO [18]	IPVO [19]	PVO-K [20]	GePVO-K [21]	iGePVO-K [22]	Proposed
Airplane	38454	53055	51505	56305	58092	61566
Baboon	13146	13462	13940	14546	14621	14661
Barbara	29490	48066	40202	41008	47400	51775
Cabeza	54263	71946	73416	79198	78031	81389
Cameraman	43458	71097	63378	69771	72663	96500
Car	31378	47327	41594	44644	51599	56457
Chest	45781	63502	60470	64151	65287	69117
Lena	32108	38713	38951	41417	41895	42709
Phone	40855	58176	52699	53986	58566	62478
Things	33772	49418	44221	43473	51179	55094
Average	36271	51476	48038	50850	53933	59175

TABLE II
PSNR COMPARISON BETWEEN METHODS WITH PAYLOAD = 10000

Images	PVO [18]	IPVO [19]	PVO-K [20]	GePVO-K [21]	iGePVO-K [22]	Proposed
Airplane	63.134	63.463	60.612	55.610	58.252	61.058
Baboon	54.306	54.298	54.428	52.999	54.228	54.290
Barbara	63.893	64.316	60.554	55.040	58.807	62.180
Cabeza	61.942	61.299	60.164	54.296	56.741	59.144
Cameraman	64.043	63.580	60.412	54.593	57.162	61.075
Car	63.257	64.224	60.517	56.280	59.274	62.069
Chest	63.624	63.280	60.630	55.143	57.642	60.517
Lena	60.110	60.056	59.009	55.066	57.726	58.349
Phone	63.798	63.582	60.552	54.926	58.267	61.066
Things	63.071	63.955	60.326	54.475	58.839	61.756
Average	62.118	62.205	59.720	54.843	57.694	60.150

TABLE III
PSNR COMPARISON BETWEEN METHODS WITH PAYLOAD = 20000

Images	PVO [18]	IPVO [19]	PVO-K [20]	GePVO-K [21]	iGePVO-K [22]	Proposed
Airplane	59.018	59.614	57.355	53.324	56.226	57.521
Baboon						
Barbara	59.299	60.835	57.329	52.959	56.671	58.411
Cabeza	58.951	58.179	57.053	52.273	54.886	56.134
Cameraman	60.984	60.883	57.736	53.007	55.787	57.900
Car	58.042	60.707	57.088	53.451	57.248	59.032
Chest	60.068	59.861	57.556	53.128	55.857	57.307
Lena	56.431	56.551	55.651	52.228	54.730	55.010
Phone	59.728	60.217	57.365	52.974	56.199	57.607
Things	58.649	60.076	57.111	52.338	56.485	57.939
Average	59.019	59.658	57.138	52.854	56.010	57.429

TABLE IV
PSNR COMPARISON BETWEEN METHODS WITH PAYLOAD = 30000

Images	PVO [18]	IPVO [19]	PVO-K [20]	GePVO-K [21]	iGePVO-K [22]	Proposed
Airplane	56.21	57.150	55.263	51.563	54.600	55.34
Baboon						
Barbara		58.403	55.041	51.177	54.762	56.027
Cabeza	57.183	56.632	55.265	50.916	53.570	54.321
Cameraman	58.929	59.002	56.017	51.822	54.611	56.204
Car	53.281	57.314	54.535	51.309	54.865	56.526
Chest	57.685	57.790	55.670	51.588	54.360	55.288
Lena	53.441	54.066	53.347	50.327	52.731	52.901
Phone	56.612	57.902	55.316	51.308	54.492	55.473
Things	54.933	57.436	54.820	50.665	54.568	55.516
Average	56.034	57.299	55.030	51.186	54.284	55.288

3. Extracting and restoring algorithm

a) Extracting and restoring in smallest pixels

We only need to revise case 2.4.3.3 by Case 3.4.3.3 as follows

Case 3.4.3.3: $X^{'}$ is 2-block:

$$x^{'}_{\sigma(1)} = ... = x^{'}_{\sigma(l)} < x^{'}_{\sigma(l+1)} = ... = x^{'}_{\sigma(n)}$$

Compute

$$dmin = x^{'}_{\sigma(1)} - x^{'}_{\sigma(l+1)}.$$

(a) If $dmin \leq -4$, no bit is extracted, l smallest pixels are increased by 1, other pixels are decreased by 1

$$x_{\sigma(i)} = x^{'}_{\sigma(i)} + 1, i = 1, ..., l.$$

$$x_{\sigma(i)} = x^{'}_{\sigma(i)} - 1, i = l + 1, ..., n.$$

(b) If $dmin = -3$, all extracted l bits are equal to 1 and restoring n pixels is carried out as in (a).

(c) If $dmin = -2$, all extracted l bits are equal to 0 and $n - l$ largest pixels are decreased by 1:

$$x_{\sigma(i)} = x^{'}_{\sigma(i)} - 1, i = l + 1, ..., n.$$

b) Extracting and restoring in the largest pixels

The extracting and restoring algorithm in the largest pixels of the iGePVO-K method (subsection 2.4.4) consists of two subsections: 2.4.4.1 and 2.4.4.2. In subsection 2.4.4.1, X is a 2-block. For this case, extracting embedded bits and restoring the original block are performed in subsection 3.3.1. So here, nothing to do. We only need to perform steps in 2.4.4.2. After completing the steps in subsections 3.3.1 and 3.3.2, obtain the embedded bits and restore the original.

4. Illustrating examples

a) Example 1

Given a pixel block

$$B = \begin{bmatrix} 24 & 25 \\ 25 & 24 \end{bmatrix},$$

and two bits 1, 1.

3.4.1.1 Embedding process

(a) **Preprocessing**: Scanning row by row, convert B into a sequence$X = (24, 25, 25, 24)$. Sort X in ascending order to get:

$$X_\sigma = (24, 24, 25, 25), \sigma = (1, 4, 2, 3).$$

In this case $n = 4, k = 2$, and $dmax = 1$.

(b) **Embedding in largest pixels**: Since $dmax = 1$, and all bits are equal to 1, we have case 3.4.1.3.2 (a). In this case, largest pixels are increased by 1, other pixels are decreased 1, so:

$$X^{'}_\sigma = (23, 23, 26, 26).$$

(c) **Embedding in smallest pixels**: Compute

$$dmin = 23 - 26 = -3.$$

So by subsection 3.2.2, case 3.4.2.1, nothing to do. In summary, we get

$$X^{'}_\sigma = (23, 23, 26, 26)$$

(d) **Creating a marked block**: Rearrange $X^{'}_\sigma$ according to positions $\sigma = (1, 4, 2, 3)$, we have

$$X^{'} = (23, 26, 26, 23).$$

Converting $X^{'}$ into the block, it follows that the marked block of B is

$$B^{'} = \begin{bmatrix} 23 & 26 \\ 26 & 23 \end{bmatrix}.$$

3.4.1.2 Extracting and restoring process

Given

$$B^{'} = \begin{bmatrix} 23 & 26 \\ 26 & 23 \end{bmatrix}.$$

(a) **Preprocessing**: Similar to 3.4.1(a), from $B^{'}$ we have:

$$X^{'}_\sigma = (23, 23, 26, 26), \sigma = (1, 4, 2, 3).$$

Compute:

$$dmin = 23 - 26 = -3$$

(b) **Extracting in smallest pixels**: Because $dmin = -3$, so by 3.3.1 (b) it follows that two embedded bits are equal to 1. The largest pixels are decreased by 1, other pixels are increased by 1. As a result, we get:

$$X_\sigma = (24, 24, 25, 25)$$

(c) **Extracting in the largest pixels**: According to 3.3.2, nothing to do. Thus we have:

$$X_\sigma = (24, 24, 25, 25)$$

(d) **Restoring the original block**: Rearrange X_σ by positions in σ, we have:

$$X = (24, 25, 25, 24).$$

Convert X into a block. As a result, we get the original block:

$$B = \begin{bmatrix} 24 & 25 \\ 25 & 24 \end{bmatrix}.$$

b) Example 2

Given a pixel block

$$B = \begin{bmatrix} 24 & 25 \\ 25 & 24 \end{bmatrix},$$

and two bits 0, 0.

3.4.2.1 Embedding process

In Example 1, two embedded bits are equal to 1. But in this example, two embedded bits are equal to 0. So in 3.4.1.1 (b), we consider 3.4.1.3.2 (b) instead of 3.4.1.3.2 (a). That

means, only increasing the largest pixels by one. Thus, we obtain:

$$X_{\sigma}^{'} = (24, 24, 26, 26)$$

So,

$$B^{'} = \begin{bmatrix} 24 & 26 \\ 26 & 24 \end{bmatrix}.$$

3.4.2.2 Extracting and restoring process
In Example 1,

$$dmin = -3.$$

But in this example,

$$dmin = 24 - 26 = -2.$$

So by 3.3.1 (c), it follows that two embedded bits are equal to 0, and the largest pixels are decreased by 1. As a result, we get

$$X_{\sigma} = (24, 24, 25, 25)$$

From this, deduce that the original B is equal to

$$B = \begin{bmatrix} 24 & 25 \\ 25 & 24 \end{bmatrix}.$$

IV. Experimental results

To illustrate the results of the theoretical analysis above, we perform experiments on 8 standard grayscale images of size 512×512, they are shown in Fig. 2 for comparing 6 methods: PVO, IPVO, PVO-K, GePVO-K, iGePVO-K, and proposed method. The embedded data is a random binary bit sequence generated by the Matlab function Randi. The programs are written in the Matlab platform and run on the IdeaPad S410p Lenovo computer.

1. Data hiding capacity comparison

Table I presents the maximum payloads of methods in test images. The payload (or data hiding capacity) is equal to the embedding capacity minus the length of the compression code of the location map. Table I shows that the data-hiding capacity of the proposed method is always greater than that of all other relative methods.

2. Stego image quality comparison

The stego image quality comparison of 6 methods in 8 test images is performed on data sets of length 10000 bits (Table II), 20000 bits (Table III) and 30000 bits (TableIV). From these tables, we can conclude that the stego image quality of the proposed always is high. Specifically, the stego image quality is larger 59 dB for 10000 bits, larger 56 dB for 20000 bits, and larger 54 dB for 30000 bits. The proposed has stego image quality lower than PVO and IPVO methods but higher than GePVO-K, iGePVO-K and PVO-K methods.

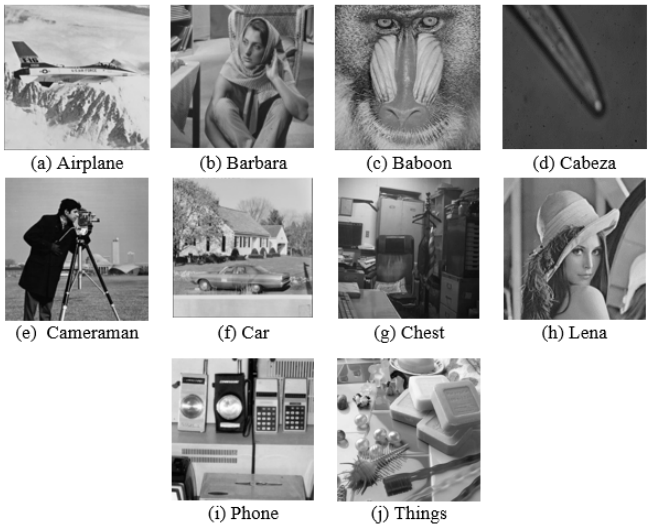


Figure 2. Experimental images

V. Conclusion

The the iGePVO-K method is an extension of PVO-K and GePVO-K methods. It has the embedding capacity larger than PVO-K and GePVO-K methods. However, this method still has a drawback in that it must use a flag bit sequence. Since the flag bit sequence needs to be embedded with the data in the original image, this reduces both its embedding capacity and its image quality. In this paper, we propose some improvements in data embedding formulas of iGePVO-K method to discard the flag bit sequence. So, the proposed method not only has larger embedding capacity but also has better stego image quality than iGePVO-K method. The experimental results have shown that the embedding capacity of the proposed method is higher than the relative existing methods while maintaining good stego image quality.

References

[1] M. Celik, G. Sharma, A. Tekalp, and E. Saber, "Lossless generalized-lsb data embedding," *IEEE Transactions on Image Processing*, vol. 14, no. 2, pp. 253–266, 2005.

[2] M. Celik, G. Sharma, and A. Tekalp, "Lossless watermarking for image authentication: a new framework and an implementation," *IEEE Transactions on Image Processing*, vol. 15, no. 4, pp. 1042–1049, 2006.

[3] J. Fridrich, M. Goljan, and R. Du, "Invertible authentication," *Proceedings of SPIE - The International Society for Optical Engineering*, vol. 4314, pp. 197–208, 08 2001.

[4] J. J. Fridrich, M. Goljan, and R. Du, "Lossless data embedding—new paradigm in digital watermarking," *EURASIP Journal on Advances in Signal Processing*, vol. 2002, pp. 185–196, 2002.

[5] J. Tian, "Reversible data embedding using a difference expansion," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, no. 8, pp. 890–896, 2003.

[6] A. Alattar, "Reversible watermark using the difference expansion of a generalized integer transform," *IEEE Transactions on Image Processing*, vol. 13, no. 8, pp. 1147–1156, 2004.
[7] L. Kamstra and H. Heijmans, "Reversible data embedding into images using wavelet techniques and sorting," *IEEE Transactions on Image Processing*, vol. 14, no. 12, pp. 2082–2090, 2005.
[8] W.-L. Tai, C.-M. Yeh, and C.-C. Chang, "Reversible data hiding based on histogram modification of pixel differences," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 19, no. 6, pp. 906–910, 2009.
[9] D. M. Thodi and J. J. Rodriguez, "Expansion embedding techniques for reversible watermarking," *IEEE Transactions on Image Processing*, vol. 16, no. 3, pp. 721–730, 2007.
[10] Y. Hu, H.-K. Lee, and J. Li, "De-based reversible data hiding with improved overflow location map," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 19, no. 2, pp. 250–260, 2009.
[11] X. Li, B. Yang, and T. Zeng, "Efficient reversible watermarking based on adaptive prediction-error expansion and pixel selection," *IEEE Transactions on Image Processing*, vol. 20, no. 12, pp. 3524–3533, 2011.
[12] Z. Ni, Y.-Q. Shi, N. Ansari, and W. Su, "Reversible data hiding," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 16, no. 3, pp. 354–362, 2006.
[13] X. Chen, S. Xingming, H. Sun, L. Xiang, and Y. bin, "Histogram shifting based reversible data hiding method using directed-prediction scheme," *Multimedia Tools and Applications*, vol. 74, pp. 5747–5765, 02 2015.
[14] S. Kukreja, S. S. Kasana, and G. Kasana, "Histogram based multilevel reversible data hiding scheme using simple and absolute difference images," *Multimedia Tools Appl.*, vol. 78, no. 5, p. 6139–6162, 2019.
[15] X. Ma, Z. Pan, S. Hu, and L. Wang, "High-fidelity reversible data hiding scheme based on multi-predictor sorting and selecting mechanism," *Journal of Visual Communication and Image Representation*, vol. 28, pp. 71–82, 01 2015.
[16] B. Ou, X. Li, W. Li, and Y.-Q. Shi, "Pixel-value-ordering based reversible data hiding with adaptive texture classification and modification," *Digital Forensics and Watermarking*, vol. 11378, pp. 169–179, 2019.
[17] X. Wang, J. Ding, and Q. Pei, "A novel reversible image data hiding scheme based on pixel value ordering and dynamic pixel block partition," *Inf. Sci.*, vol. 310, p. 16–35, 2015.
[18] X. Li, J. Li, B. Li, and B. Yang, "High-fidelity reversible data hiding scheme based on pixel-value-ordering and prediction-error expansion," *Signal Process.*, vol. 93, no. 1, p. 198–205, jan 2013.
[19] F. Peng, X. Li, and B. Yang, "Improved pvo-based reversible data hiding," *Digit. Signal Process.*, vol. 25, no. C, p. 255–265, feb 2014.
[20] B. Ou, X. Li, Y. Zhao, and R. Ni, "Reversible data hiding using invariant pixel-value-ordering and prediction-error expansion," *Image Commun.*, vol. 29, no. 7, p. 760–772, aug 2014.
[21] J.-J. Li, Y.-H. Wu, C.-F. Lee, and C.-C. Chang, "Generalized pvo-k embedding technique for reversible data hiding," *International Journal of Network Security*, vol. 20, pp. 65–77, 01 2018.
[22] N. K. Sao, N. N. Hoa, and P. V. At, "An effective reversible data hiding method based on pixel-value-ordering," *Journal of Computer Science and Cybernetics*, vol. 36, no. 2, p. 139–158, May 2020.

Le Kinh Tai received B.A. degree from Academy of Finance in 2006, degree of master's in human resource management from Trade Union University in 2012, and B.A degree in English from Ha Noi Open University in 2014. He currently works in Trade Union University. His interests are data hiding and information security.
Email: tailk@dhcd.edu.vn

Nguyen Ngoc Hung received his B.E. degree from University of Transport and Communications, in 2009. He achieved M.E. degree from Vietnam National University of Engineering and Technology, in 2014. His interests are data hiding, watermarking, information security.
Email: nnhung@ioit.ac.vn

Dang Tran Duc received his B.E., M.E. degree from University of Transport and Communications, in 2009, 2015 respectively. Currently, he is CEO of VVECO Joint Stock Company. His interests are AI, NLP, and information security.
Email: tranducdang@vveco.vn

Cao Duc Diep received his B.E., M.E. degree from University of Transport and Communications, in 2009, 2015 respectively. Currently, he is a software project management. His interests are software engineering, and information security.
Email: diepcd@epu.edu.vn

Pham Minh Thai received his bachelor's degree in information technology from Nha Trang University in 2008. He received a master's degree in computer science from the Military Technical Academy in 2011. His interests are data hiding, watermarking, information security.
Email: pmthai@uneti.edu.vn

Evaluation of Autoencoder-based Communications with Reconfigurable Intelligent Surfaces

Thi-Nga Dao[1], Chi-Hieu Ta[2]
[1,2]Le Quy Don Technical University, Ha Noi, Vietnam
Correspondence: Thi-Nga Dao (daothinga@mta.edu.vn)
Communication: received 06 Mar 2023, revised 24 May 2023, accepted 03 June 2023
Digital Object Identifier: 10.32913/mic-ict-research.v2023.n2.1204

Abstract: Reconfigurable intelligent surfaces (RIS) have emerged as a promising technique for wireless communication in 5G and beyond. In an environment with lots of physical obstacles, RIS provides an alternative solution to the coverage problem by beam-forming signal towards the desired direction of the receiver. Due to the fixed constellation diagram, traditional modulation schemes cannot be used for the RIS-aided communication system with the receiver movement. In this work, we aim to design and evaluate an autoencoder-based communication model with RIS support, which can adapt to changes in the environment and the receiver's position. Specifically, we use neural networks to present the encoder and decoder of the system and the parameters of these networks are trained to minimize the reconstruction error at the receiver. The simulation setup is based on characteristics of the environment and real RIS. Performance results show the benefits of RIS when a physical obstruction is present between the transmitter and receiver. Moreover, we prove that the selection of RIS codeword plays an important role in system performance.

Keywords: *autoencoder-based communications, reconfigurable intelligent surfaces*

I. Introduction

Reconfigurable intelligent surfaces (RIS) consisting of hundreds to thousands of passive reflecting units have drawn great attention from both academia and industry [1–3]. RIS can be applied to various applications such as wireless communication, wireless power transfer, and wireless security. By intelligently selecting the phase shift of each cell unit, RIS is capable of beam-forming to a desired direction. Thus, the RIS-aided communication system is a potential solution for the coverage problem due to the lack of a light-of-sight path between the transmitter and receiver. In this work, we model and analyze the RIS-based communication system especially when the receiver's location changes over time.

The traditional wireless communication system includes a transmitter, receiver, and a wireless channel between devices. In the transmitter, the symbol is modulated by changing a specific characteristic of the carrier signal such as amplitude, frequency, or phase. While transmitting via a wireless channel, noise with Gaussian distribution is usually added to the signal. Then, the demodulation process is executed at the receiver to determine the information sent by the sender. In the conventional communication model, the modulation block is usually fixed and the constellation diagram is pre-determined. However, a fixed constellation diagram in conventional modulation schemes such as PSK and QAM is not suitable for the case of varying receiver locations. Once the receiver moves to the new position, RIS codeword should be updated for beam-forming toward the new direction and the phase shift of the encoded signal varies accordingly. Therefore, the modulation block should generate complex signals differently so that the constellation diagram of the received signal at the receiver shows maximum separation between adjacent points. To achieve maximum performance at the receiver, we design a RIS-based communication model that adaptively modulates the information based on the receiver's location and the channel state information.

We propose and evaluate an autoencoder-based RIS-aided communication system that consists of five main blocks: transmitter, Tx-RIS channel, RIS, RIS-Rx channel, and receiver. The transmitter is in charge of signal modulation by mapping the information into complex signals with I/Q parts. To speed up training convergence, a normalization layer is added at the end of the transmitter block. While transmitting the signal via the Tx-RIS channel, the modulated signal is attenuated based on channel features including distance from Tx-RIS, antenna gain, and frequency. The amplitude and phase of the modulated signal can be adjusted by controlling the PIN diode attached to each cell

unit. A PIN diode is used as a load of microstrip antenna where the signal is reflected. After being reflected at RIS, the signal goes through the Rx-RIS channel before arriving at the receiver. For training convergence acceleration, another normalization layer is inserted at the beginning of the receiver and the received signal is normalized in the range [-1, 1]. We assume that noise with Gaussian distribution is added to the received signal at the receiver. Then, the actual information is extracted from the received signal using a decoder.

As stated earlier, a flexible modulation scheme that can adapt to the receiver's location, as well as channel state information, is used. To do that, an autoencoder is used to compress the information to a complex signal at the encoder and then the decoder attempts to reconstruct the generated symbol from the compressed vector. Network parameters of AE can be trained to minimize the reconstruction difference between the generated and predicted symbols.

The main contribution of our study is listed below.

- We present the RIS-aided AE-based communication system to overcome the critical coverage problem with the change of receiver's location.
- A more practical RIS-aided communication model is introduced with pre-determined codeword selection based on the location of the receiver and with discrete phase shift by RIS.
- The communication system is evaluated using characteristics of a real RIS prototype and the real environment with different numbers of information bits and ratio of noise power to energy per bit.
- Simulation results show the feasibility of the proposed model and the huge effect of codeword selection on the performance of the communication system.

The rest of the paper is organized as follows: Section II presents basic knowledge of autoencoder and RIS. Then, the communication model is described in Section III. Section IV presents the network setup and analyzes the performance of the proposed model with various network parameters. Finally, we draw the conclusion for our work in Section V.

II. Related Study

In this section, we summarize autoencoder architectures and RIS technique. Then, we compare our work with existing autoencoder-based RIS-supported communication systems to highlight advantages of the proposed model.

Autoencoders are neural networks capable of learning efficient representation of data by minimizing the construction error between the input and reconstructed data. Autoencoders can be leveraged for dimensionality reduction. An autoencoder always consists of two parts: an encoder that maps inputs into the coding vectors with a much smaller size and a decoder for data recovery. In wireless communication systems, data is usually modulated into a radio signal that is then sent in the atmosphere environment. Existing studies [4] show that autoencoders are greatly suitable for data modulation in wireless communications. The coding vectors have two dimensions for I and Q parts. A noise layer with Gaussian distribution is usually added to reflect the natural features of environments. The number of input units and output units is the number of possible symbols.

RIS has recently emerged as a promising technique for coverage and data-rate improvement for wireless communication systems in mmWave bands, which are badly affected by physical obstructions as shown in Figure 2. Wireless power transfer systems become more efficient with the support of RIS, which is proven in a lot of research [5–7]. Moreover, RIS can be leveraged to enhance the security of wireless communication systems by not allowing signal transmission toward eavesdroppers. RIS consisting of a programmable antenna array can be used to beamform the RF signal toward the receiver's direction. Existing works analyzed the performance of these surfaces using both simulations and real devices.

We first present the system model of RIS-aided communication. Assume that the LoS link is blocked by physical obstacles, the received signal at the receiver can be written as:

$$y = h_{TR}^{T} \Psi h_{RR} x + n \tag{1}$$

where $h_{TR} \in \mathbb{R}^{N\times 1}$ denotes the channel response between transmitter and RIS and $h_{RR} \in \mathbb{R}^{N\times 1}$ is for the channel response between RIS and receiver. Meanwhile, $\Psi \in \mathbb{R}^{N\times N}$ is a diagonal matrix of RIS interaction. We define $\psi = \text{diag}(\Psi)$ as the RIS beamforming vector and the optimal value is determined as:

$$\psi^{*} = \text{argmax}(a_{TR} \cdot a_{RR} \psi) \tag{2}$$

where a_{TR} and a_{RR} are the phase shift of TX-RIS and RIS-RX channels, respectively. The optimal RIS phase-shifting configuration is derived as:

$$\psi^{*} = -(a_{TR} + a_{RR}) \tag{3}$$

Because of hardware limitations, there is a huge challenge for the implementation of RIS with continuous phase shift values. In practice, RIS with a limited set of reflection angles becomes popular. For example, authors in [8] evaluated the response of a real-world 160-cell RIS and experimental results show that RIS created a delay of 0^{o} or 180^{o} while beamforming the signal to the receiver's position.

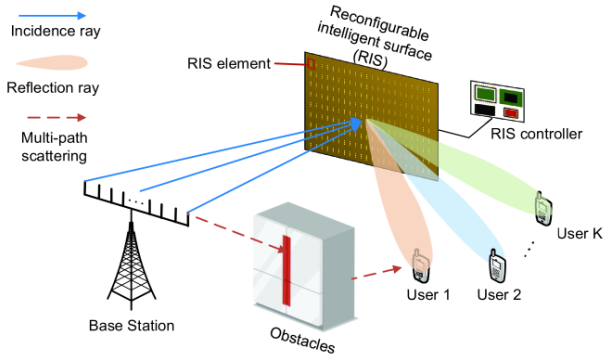


Figure 1: Wireless communications for multi-users with support of RIS [9]

There are some existing works on evaluating the autoencoder-based communication system with RIS support [10, 11]. In [10], they assume that there is a physical obstruction that accounts for a loss between transmitter and receiver. Simulation results show that the symbol error rate of the AE-based RIS-aided system is better than the QPSK-modulated system. In their system, parameters are trained for codeword selection and signal modulation. However, the appropriate codeword of RIS elements should be selected first if the receiver's position is known. Since RIS acts as a directional attenna, reflecting power at the main lobe is much higher than in other directions. Therefore, in our design, we assume that the codeword is determined based on the receiver's position to simplify the system design and training. In [11], the multi-input multi-output RIS-supported AE-based system is considered with a practical channel state information. However, phase shift of RIS is a continuous value, which is not practical in RIS implementation. In our work, RIS with a limited list of codewords is considered based on real experiments in [8].

III. RIS-AIDED AUTOENCODER-BASED COMMUNICATIONS MODEL

When there is no direct path between the transmitter and receiver, RIS can be used to construct an alternative communication channel. The transmitter sends a modulated signal towards RIS and the signal is passively reflected towards the receiver's direction. The RIS controller plays an important role to tune the reflection angle of the signal, which can take values in a discrete set. Assume that RIS consists of N elements and each element is connected to a programming controller that can adjust the element's impedance and the reflected signal direction. In this work, we consider a communication system with a transmitter and a receiver.

There are five main blocks in the AE-based RIS-supported communication system: transmitter, Tx-RIS channel, RIS, RIS-Rx channel, and receiver. We describe these five blocks followed by the training procedure in detail below.

1. Transmitter architecture

A symbol s containing k information bits is fed into the transmitter to generate an I/Q sample. First, one-hot encoding is used to convert s to a vector with the size of 2^k. For example, if $k = 2$, the one-hot vectors are 1000, 0100, 0010, and 0001. Then, a fully-connected neural network (NN) is applied to map the one-hot vector to complex numbers with real and imaginary parts. The NN architecture contains a hidden layer and an output layer with tanh activation. This process can be thought of as a modulation scheme in which the amplitude and phase of I/Q sample are changed based on the input symbol. Using the autoencoder technique, the constellation diagram is learnt automatically. For example, $k = 2$ is equivalent to 4-QAM in the conventional modulation technique.

To limit the maximum transmission power, a normalization layer is used after the NN model:

$$x = \frac{x}{\sqrt{max(\|x\|_2^2)}} \tag{4}$$

There are no trainable parameters in the normalization layer. For the average power constraint, the normalization layer is implemented as:

$$x = \frac{x}{\sqrt{mean(\|x\|_2^2)}} \tag{5}$$

Max or mean operation is computed over mini-batch with batch size of 4.

2. Tx-RIS channel

This channel applies amplitude loss and phase shift to the transmitted signal x. Channel coefficient h_{TR} contains N parameters, which are in the range $[-1, 1]$. If no amplitude loss and phase shift are applied, channel coefficients are set to 1. For ease of implementation, a linear dense layer is used to simulate the Tx-RIS channel. Only amplitude loss is considered by scaling down both real and imaginary parts with the same factor. For example, if the transmitted signal $x = [0.6, 0.8]$ and the element $h_{TR} = 0.5$, the signal received at RIS will be $x_{RIS} = [0.3, 0.4]$. Path loss of Tx-RIS channel α_{TR} is derived according to free-space path loss as below.

$$\alpha_{TR} = \frac{c}{4\pi d_{TR} f}\sqrt{G_{tx} G_{RIS}} \tag{6}$$

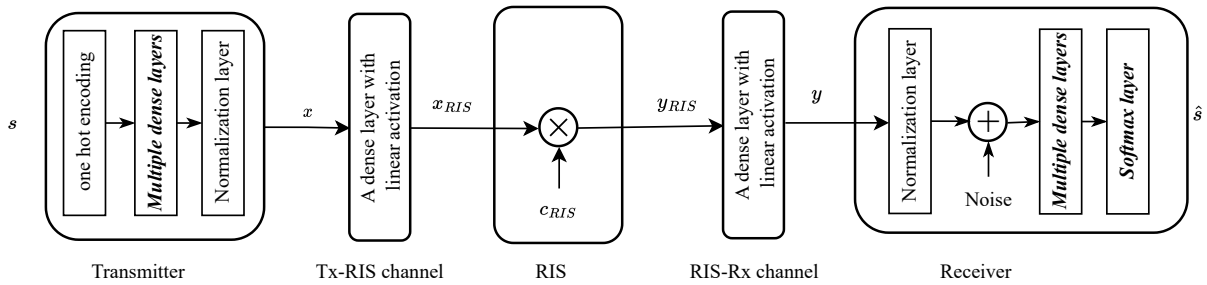


Figure 2: Autoencoder-based communications with RIS (blocks with italic and bold texts contain trainable parameters)

TABLE I: Notations used in the paper

Name	**Description**	**Shape** (if applied)
N	Number of passive elements in RIS	
s	Symbol	k
k	The number of information bits	
x	I/Q samples generated by transmitter	2
h_{TR}	Channel between transmitter and RIS	$(N, 1)$
x_{RIS}	Signal received at RIS	$2N$
c_{RIS}	Codeword of RIS	$2N$
y_{RIS}	Signal output at RIS	$2N$
y	Signal received at receiver	2
h_{RR}	Channel between RIS and receiver	$(N, 1)$
$\hat{s}$	Constructed signal at the receiver	k
β	Standard deviation of noise	
R	Communication rate	
E_b	Energy per bit	
N_0	Power spectrum density	
α_{TR}	Path loss of Tx-RIS channel	
α_{RR}	Path loss of RIS-Rx channel	
c	Transmission speed	
d_{TR}	The distance between transmitter and RIS	
d_{RR}	The distance between RIS and receiver	
G_{tx}	Gain of transmitter antenna	
G_{RIS}	Gain of RIS antenna	
G_{rx}	Gain of transmitter antenna	

where c is the transmission speed of the radio signal in the atmosphere, d_{TR} is the distance between transmitter and RIS, f denotes signal frequency while G_{tx} and G_{RIS} represent antenna gains of transmitter and RIS. Channel coefficient is randomly selected in range $\alpha_{TR} \times [h_{low}, h_{high}]$.

Generally, two parts of x_{RIS} are computed as follows:

$$x_{RIS}[0:N] = x[0] \times h_{TR} \tag{7}$$

$$x_{RIS}[N:2N] = x[1] \times h_{TR}. \tag{8}$$

Then, we combine these $x_{RIS}[0:N]$ and $x_{RIS}[N:2N]$ into x_{RIS}. In our work, parameters in the Tx-RIS channel are constant and could not be updated during training.

3. RIS architecture

In our system, RIS operates as a reflectarray antenna with a reflection angle θ_d, which is determined based on position of the receiver. Codeword c_{RIS} is referred to a practical experiment in [8]. Codeword consists of N rows and two columns. Each row accounts for coefficient of each unit cell. For example, phase shift is quantized using 1 bit and amplitude of signal reflected by RIS is reduced by half. If phase shift is 0^o of a specific cell unit, coefficient will be [0.5, 0.5]. Otherwise, for 180^o phase shift, coefficient will be [-0.5, -0.5]. The codeword vector c_{RIS} is multiplied element-wise by x_{RIS} as below.

$$y_{RIS} = x_{RIS} \odot c_{RIS} \tag{9}$$

4. RIS-Rx channel

Similar to Tx-RIS channel, channel coefficient h_{RR} with N non-trainable parameters represents amplitude attenuation. Path loss of Tx-RIS channel α_{RR} is computed as below.

$$\alpha_{RR} = \frac{c}{4\pi d_{RR} f}\sqrt{G_{rx}G_{RIS}} \tag{10}$$

where c is the transmission speed of the radio signal in the atmosphere, d_{RR} is the distance between RIS and receiver, f denotes signal frequency while G_{rx} and G_{RIS} represent antenna gains of receiver and RIS. We define $y \in \mathbb{R}^2$ as the received signal at the receiver. The components y_I and y_Q account for the I and Q parts and are derived as follows.

$$y_I = y_{RIS}[0:N] \times h_{RR,I} \tag{11}$$

$$y_Q = y_{RIS}[N:2N] \times h_{RR,Q} \tag{12}$$

In this work, we do not consider phase shift and set $h_{RR,I} = h_{RR,Q} = h_{RR}$. If phase shift is taken into account, $h_{RR,I} \neq h_{RR,Q}$.

5. Receiver

The signal y has I and Q components at the receiver. The receiver architecture consists of a normalization layer followed by a noise layer, multiple dense layers, and a softmax layer. The normalization layer ensures that the received signal y has an average amplitude of 1. By doing that, the learning speed is expected to be faster since features of dense layers are in the range [-1,1] at the input of the receiver block. There is no trainable parameter in the normalization layer. Added noise with two dimensions follows Gaussian distribution with the mean of 0 and standard deviation β is:

$$\beta = \sqrt{\frac{1}{2RE_b/N_0}} \tag{13}$$

where communication rate $R = k/n$ with channel use $n = 2$ while E_0 is energy per bit and N_0 denotes noise power spectral density [4]. The noise layer varies for each sample in training and testing sets. The softmax layer includes 2^k units that represent the probability of sending 2^k symbols. The receiver will return the most likely symbol with the highest probability.

6. Training procedure

Layers with trainable parameters are denoted by italic and bold texts in Figure 2. Parameters at encoder and decoder are learnt by minimizing the cross-entropy loss function:

$$L_1 = -\frac{1}{M}\sum_{i=1}^{M}\sum_{j=1}^{2^k} f(s_j)\log p(\hat{s_j}) \tag{14}$$

where M is the mini-batch size, $f(s_j)$ is the one-hot vector of symbol j, $p(\hat{s_j})$ is the probability of the constructed symbol j. L_1 achieves the minimum value when $s_j = \hat{s}_j$ for all j values.

We define θ as trainable parameters at a specific phase. After a training epoch, if using gradient descent, network parameters are updated as:

$$\theta = \theta - \eta\frac{\partial L}{\partial \theta} \tag{15}$$

where η is learning rate and L_1 is a loss function at the considered phase.

TABLE II: Parameters Setup

Parameters	Description
k	{1, 2, 3, 4, 5, 6}
$\frac{E_b}{N_0}$	{2, 4, 6, 8, 10}
N	160
n	2
α_{TR}, α_{RR}	0.065
h_{low}	[0.2, 0.8]
h_{high}	1
Operating frequency f	5.8 GHz
Antenna gain G	10 dBi
Amplitude loss at a RIS unit	3 dB
#Training epochs	50
Learning rate	0.001

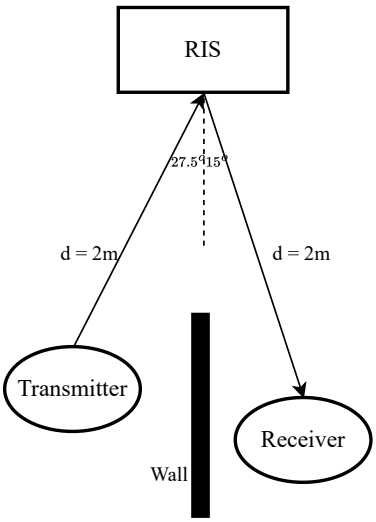


Figure 3: Network topology for performance evaluation

IV. Performance Evaluation

In this section, we first examine the RIS-aided communication system with a transmitter, a receiver, and RIS as shown in Figure 3. Assume that there is a physical obstacle between the transmitter and receiver, which makes direct transmission impossible. The distance between the transmitter/receiver and RIS is set to 2m. The incident angle is set to 27.5^o and the reflection angle is set to 15^o. We refer to RIS characteristics in which the PIN diode is used to control the phase shift of each unit cell in RIS. RIS comprises 160 unit cells (16 x 10) [8]. A nearly 180^o phase shift between ON and OFF states is achieved at frequency band 5.8 GHz. Assume that the reflection amplitude is attenuated around 3 dB in both states. With the above features, we refer to the quantized phase distribution of RIS with the reflection angle of 15^o in Figure 13 of [8]. Various parameters consist of the number of information bits (k), and the ratio of bit energy to noise spectral power ($\frac{E_b}{N_0}$). The antenna gain of the transmitter, receiver, and RIS is set to the same value G of 10 dBi. Then, channel amplitude loss is computed as 0.021. Since the channel is usually random, h_{TR} and h_{RR} follow uniform distribution in range $0.021 \times [h_{low}, h_{high}]$. The default value for h_{low} is 0.2. Table II summarizes the main parameters used in the experiments. A supervised learning method is used to update the network parameters of the proposed system. The number of training and test samples is set to 10^6. When increasing k, we should use more samples for training to better distinguish between constellation points. We use accuracy at the receiver as the performance metric to evaluate the system. Accuracy is defined as the percentage of correct symbols at the receiver. In the encoder block, the transmitter has two fully-connected layers with 64 and 2 tanh units. In the decoder block, there are two fully-connected layers with 64 ReLU hidden neurons and 2^k softmax neurons at the receiver. The number of floating-point operations (FLOPs) including multiplication, addition, and division can be used to show the model complexity of the proposed system. For the setup mentioned above, the number of FLOPs in the transmitter block is: $2 \times 2^k \times 64 + 2 \times 64 \times 2 = 128 \times 2^k + 256$. The number of FLOPs of two-channel blocks is $2 \times 2 \times N = 640$. The RIS block contains $2N = 320$ operations while the number of FLOPs at the receiver is $2 \times 2 \times 64 + 2 \times 64 \times 2^k = 256 + 128 \times 2^k$. Then, the proposed system includes $1,472 + 256 \times 2^k$ operations. The learning rate is set to 0.001. The tanh activation function is used to limit the output layer in the range $[-1, 1]$, which is suitable for power constraint at the transmitter. We provide the full code to help the audience reproduce the results of this paper at https://github.com/daothinga/AE-based-RIS-aided-communication-system.

1. Performance results of RIS-aided communication system

First, we show the constellation generated by autoencoder with I and Q signals in Table 6 during the first training phase when $k = 3$ and $\frac{E_b}{N_0} = 6$ dB. Before training (round 0), the constellation diagram looks random. However, encoded samples are distributed far from each other over training rounds. Finally, in round 4, signals are modulated similarly to 8-PSK in which encoded samples are distributed equally on the circle with radius 1.

When phase shift is continuous, c_{RIS} can take any value in the range $[-1, 1]$. However, phase shift generated by a cell unit is usually discrete in practice. Specifically, c_{RIS} of a cell unit is set to $[0.5, 0.5]$, which corresponds to 0 phase shift and 3 dB amplitude loss, and $[-0.5, -0.5]$ for 180^o phase shift and 3 dB amplitude attenuation.

Figure 5 illustrates the performance of the communication system with k varies from 2 to 6 while $\frac{E_b}{N_0}$ changes from -2 to 10. When $k = 2$, the accuracy at the receiver achieves 100% with all considered values of $\frac{E_b}{N_0}$. When increasing the number of information bits to be sent, the symbol error at the receiver becomes higher. For example, $\frac{E_b}{N_0} = 0$, accuracy at receiver drops 66.3% as k changes from 2 to 6. Undoubtedly, increasing $\frac{E_b}{N_0}$ results in better performance at the receiver.

2. Effects of codeword selection

Assume that a wrong codeword is accidentally fed to the RIS controller. As a result, received power is attenuated significantly. We assume that the receiver moves to a different location and the updated reflection angle is 45^o. However, the previous codeword is still used at RIS, which presents the wrong codeword case. According to [8], the received signal is attenuated 15 dB at the receiver. We compare accuracy in two cases: correct and wrong codeword in Table III. We set k to 2 and 3 while $\frac{E_b}{N_0}$ varies from -2 dB to 10 dB with the step of 2. When using the correct codeword, we can achieve the maximum received signal power. Therefore, accuracy with the correct codeword is much higher than in the case of the wrong codeword. We can infer from Table III that codeword selection plays an important role in the receiver performance. Since the power of the main reflecting lobe is usually much greater than that of other directions, the wrong codeword causes a very low performance at the receiver.

3. Impact of channel state information

We have validated the system performance with different channel responses h_{TR} and h_{RR}. We set h_{low} to 0.2 and

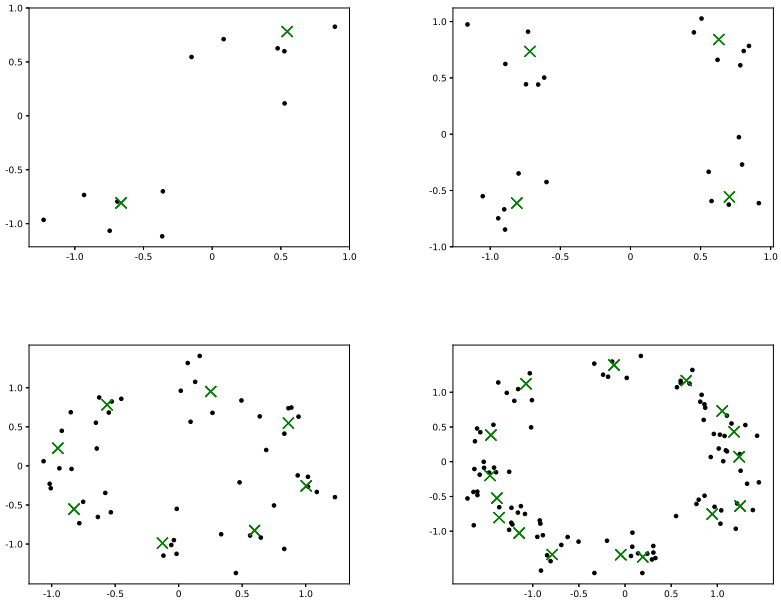


Figure 4: Encoded signals (blue) and received signals (black) with noise when $k = 1, 2, 3, 4$

TABLE III: Effects of codeword selection on Accuracy Performance

$(k, \frac{E_b}{N_0})$	**Accuracy w/ correct codeword**	**Accuracy w/ wrong codeword**
(2, -2)	100%	40%
(2, 0)	100%	60%
(2, 2)	100%	40%
(2, 4)	100%	60%
(2, 6)	100%	65%
(2, 8)	100%	55%
(2, 10)	100%	70%
(3, -2)	92.5%	35%
(3, 0)	100%	25%
(3, 2)	100%	37.5%
(3, 4)	100%	25%
(3, 6)	100%	25%
(3, 8)	100%	32.5%
(3, 10)	100%	60%

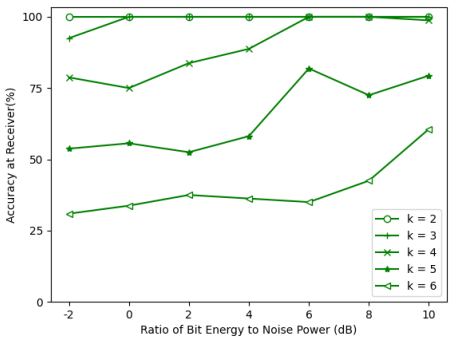
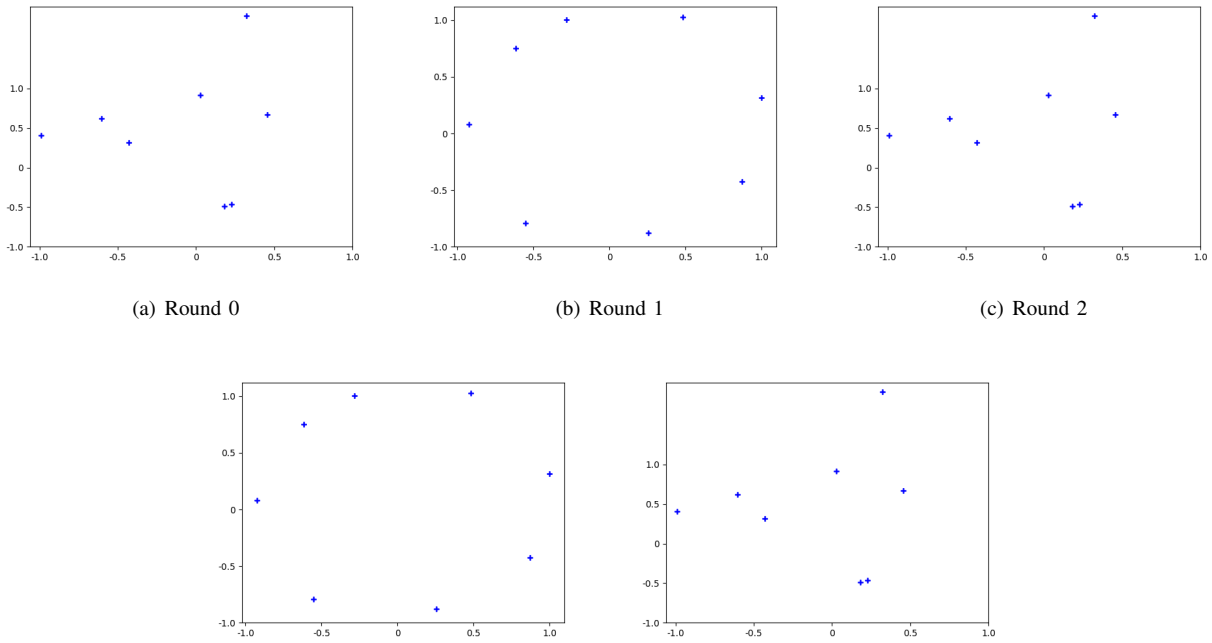


Figure 5: Performance evaluation with different values of k and $\frac{E_b}{N_0}$

0.8 while h_{high} is 1. From the results in Fig 7, we can infer that the AE-based RIS-supported communication system is able to adapt to the change in the channel environment. The performance results in the two cases are quite similar. The proposed system can generate different constellation diagrams based on channel and RIS response characteristics. The modulation scheme based on AE becomes flexible compared to the traditional modulation schemes.

4. Performance comparison between AE-based and M-PSK modulation schemes

In this subsection, we first compare the performance between AE-based and M-PSK modulation schemes with



(a) Round 0 (b) Round 1 (c) Round 2

(d) Round 3 (e) Round 4

Figure 6: Encoded signals during training with $k = 3$ and $\frac{E_b}{N_0} = 6$ dB

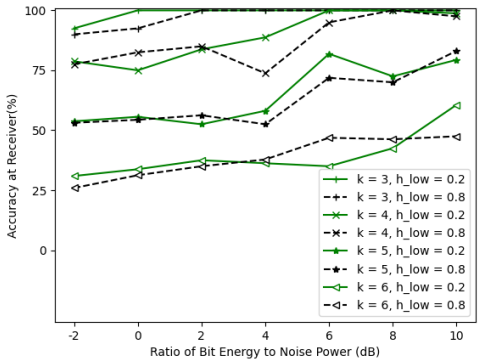


Figure 7: Impacts of channel response on performance of receiver

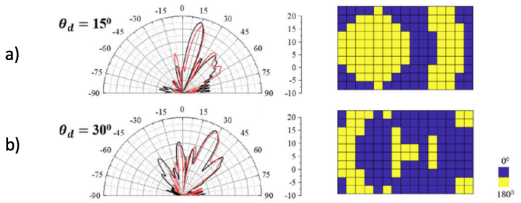


Figure 8: Settings for RIS cells in two cases reffered to [8].

regard to block error rate (BLER) over the Gaussian channel. The experimental scenario is shown in Figure 3. We use the same network parameters as shown in Table II. For example, there are 160 RIS cells and two states (ON and OFF) for each cell. Meanwhile h_{low} = 0.2 and h_{high} =1. However, to achieve a fair comparison result, we consider two different settings for RIS cells as shown in Figure 8. In the first case, 56 out of 160 cells are in ON states with the shift phase 0^o and the rest of them are in OFF states with the shift phase 180^o. Meanwhile, the number of ON and OFF states are 48 and 112 cells, respectively in the second case. We refer to [8] for these two settings. For the AE-based modulation scheme, normalization based on average power constraint is used at the receiver block. We first show the constellation diagrams of two modulation schemes using the first RIS setting in Figure 9. The M-PSK scheme distributes symbols evenly on the circle, which means all symbols have the same power. Meanwhile, the AE-based modulation scheme generates symbols with different amplitudes but still guarantees the average power constraint. As a result, the AE-based modulation scheme achieves much lower BLER than M-PSK as shown in Figure 10 for both cases. More specifically, when there are 8 symbols, AE gains roughly 0.5dB than 8-PSK at BLER lower than 10^{-1} in the first case and gains more than 1dB than 8-PSK at BLER less than 10^{-2} in the second case. Intuitively,

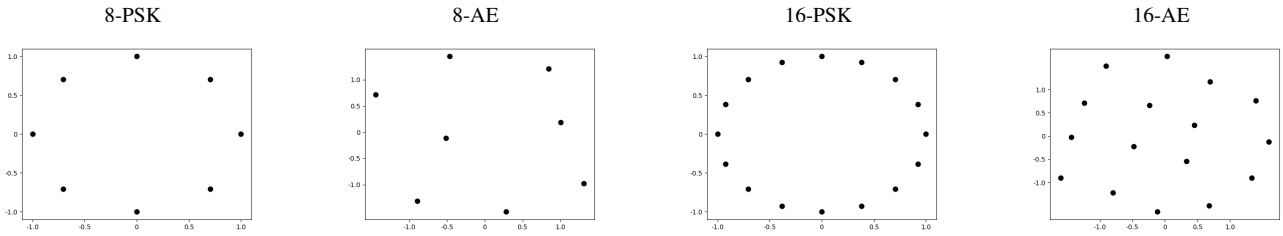
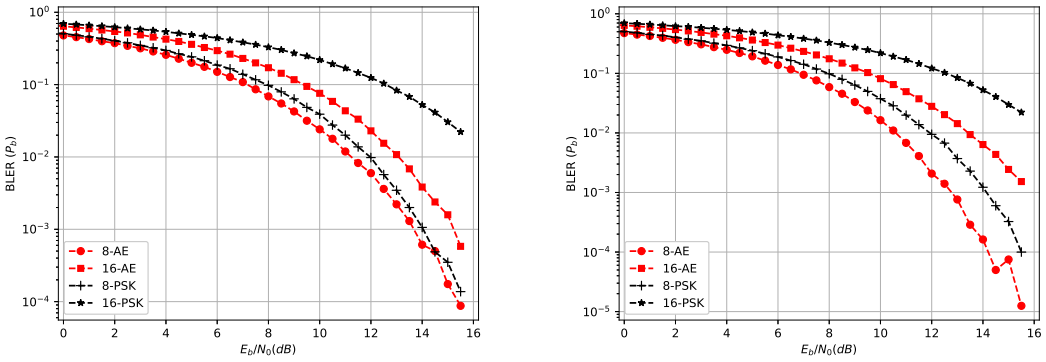


Figure 9: Constellation comparison between M-PSK and AE-based modulation schemes



(a) (b)

Figure 10: BLER Comparison between M-PSK and AE-based modulation schemes: (a) the first setting for RIS cells and (b) the second setting for RIS cells.

regardless of the RIS cell setting, the higher value of k, the higher gain between these two methods. For example, in the first RIS setting, when $\frac{E_b}{N_o}$ = 16 dB and $k = 4$, BLER in the AE-based model is 58 times lower than that in the M-PSK scheme.

5. Discussion

Whenever the accuracy of the communication system is lower than a threshold value, e.g., 99%, we need to check the receiver's location. If the accuracy drop is due to the change in the receiver's position, we need to select a new value for c_{RIS} and then re-train the network parameters of the transmitter and receiver. The training process can be done at a centralized machine and the trained parameters are sent to the transmitter and receiver to update modulation and de-modulation blocks, respectively. Since parameters training usually takes several minutes, the RIS-aided AE-based communication system is more feasible for receivers with low mobility. To support devices with high mobility, several ways can be applied to improve the performance of the system including the increase of transmission power and reduction in the k value.

V. Conclusion

In this work, we evaluate the feasibility of an autoencoder-based RIS-supported communication system when there is no light-of-sight path between transmitter and receiver. When the receiver's location and channel state information change, the proposed communication model is more beneficial than the traditional modulation schemes with the fixed constellation diagram. The simulation performance with various parameters proves the potential of the proposed system. In future works, we plan to consider more realistic situations for performance evaluation such as random channel state and far-field communication.

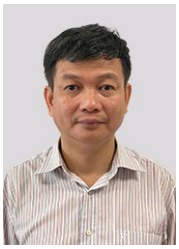
References

[1] J. Rao, Y. Zhang, S. Tang, Z. Li, C.-Y. Chiu, and R. Murch, "An active reconfigurable intelligent surface utilizing phase-reconfigurable reflection amplifiers," *IEEE Transactions on Microwave Theory and Techniques*, 2023.

[2] R. Fara, P. Ratajczak, D.-T. Phan-Huy, A. Ourir, M. Di Renzo, and J. De Rosny, "A prototype of reconfigurable intelligent surface with continuous control of the reflection phase," *IEEE Wireless Communications*, vol. 29, no. 1, pp. 70–77, 2022.

[3] Y. Zhang, W. He, D. He, Y. Xu, Y. Guan, and W. Zhang, "Ris-aided ldm system: A new prototype in broadcasting system," *IEEE Transactions on Broadcasting*, 2022.
[4] T. O'shea and J. Hoydis, "An introduction to deep learning for the physical layer," *IEEE Transactions on Cognitive Communications and Networking*, vol. 3, no. 4, pp. 563–575, 2017.
[5] N. M. Tran, M. M. Amri, J. H. Park, D. I. Kim, and K. W. Choi, "Reconfigurable-intelligent-surface-aided wireless power transfer systems: Analysis and implementation," *IEEE Internet of Things Journal*, vol. 9, no. 21, pp. 21 338–21 356, 2022.
[6] L. Zhao, Z. Wang, and X. Wang, "Wireless power transfer empowered by reconfigurable intelligent surfaces," *IEEE Systems Journal*, vol. 15, no. 2, pp. 2121–2124, 2020.
[7] H. Yang, X. Yuan, J. Fang, and Y.-C. Liang, "Reconfigurable intelligent surface aided constant-envelope wireless power transfer," *IEEE Transactions on Signal Processing*, vol. 69, pp. 1347–1361, 2021.
[8] G. C. Trichopoulos, P. Theofanopoulos, B. Kashyap, A. Shekhawat, A. Modi, T. Osman, S. Kumar, A. Sengar, A. Chang, and A. Alkhateeb, "Design and evaluation of reconfigurable intelligent surfaces in real-world environment," *IEEE Open Journal of the Communications Society*, vol. 3, pp. 462–474, 2022.
[9] B. Di, H. Zhang, L. Song, Y. Li, Z. Han, and H. V. Poor, "Hybrid beamforming for reconfigurable intelligent surface based multi-user communications: Achievable rates with limited discrete phase shifts," *IEEE Journal on Selected Areas in Communications*, vol. 38, no. 8, pp. 1809–1822, 2020.
[10] T. Erpek, Y. E. Sagduyu, A. Alkhateeb, and A. Yener, "Autoencoder-based communications with reconfigurable intelligent surfaces," in *2021 IEEE International Symposium on Dynamic Spectrum Access Networks (DySPAN)*. IEEE, 2021, pp. 242–247.
[11] H. A. Le, T. Van Chien, W. Choi *et al.*, "Ris-assisted mimo communication systems: Model-based versus autoencoder approaches," in *2022 IEEE 33rd Annual International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC)*. IEEE, 2022, pp. 1–6.

Thi Nga Dao received a B.S. degree in Electrical and Communication Engineering from Le Quy Don Technical University, Vietnam in 2013, an M.S. degree in Computer Engineering from University of Ulsan in 2016, and a Ph.D. degree in Computer Engineering from University of Ulsan, South Korea in 2019. She was a postdoctoral fellow at Intelligent Networked Systems Lab (INSLab) of Ewha Womans University from 2020 to 2021. From July 2019, she has been working as a researcher in the Faculty of Radio-Electronic Engineering, Le Quy Don Technical University, Hanoi, Vietnam. Her research interests include machine learning-based applications in communication systems, network security, and network intrusion detection systems.
Email: daothinga@mta.edu.vn



Chi Hieu Ta received B.E degree with honor in Electronic Engineering in Le Quy Don Technical University in 1994, M.E. degree in electronic engineering in National Defense Academy, Japan in 2002 and PhD degree in signal processing in Strathclyde University, UK in 2008. He is currently working at the Faculty of Radio Electronic Engineering, Le Quy Don Technical University. His research interests include precoding an equalisation for MIMO systems, signal processing for communication and microwave engineering.
Email: hieunda@gmail.com

A Method Mining Correlation High Utility Itemsets with Negative Profits

Cao Tung Anh[1], Ngo Quoc Huy[2], Pham Ngoc Diem[3]

[1]Ho Chi Minh City University of Technology
[2]Van Lang University
[3]The Ho Chi Minh City Power Corporation

Correspondence: Cao Tung Anh, ct.anh@hutech.edu.vn
Communication: received 16 jul 2023, revised 26 Aug 2023, accepted 7 Sep 2023
DOI: 10.32913/mic-ict-research.v2023.n2.1228

***Abstract*: High-utility itemset (HUI) mining is an important task in the data mining process. Recently, many algorithms have been proposed to mine sets of HUIs. Most algorithms only work with data sets with positive return values. However, in the real world, the set of business items often includes both positive (+) and negative (-) profit values. To solve this problem, an algorithm is needed to find HUIs with negative returns in the database.**

The issue is how to mine efficiently correlation high utility itemsets with negative profits. Given the transactional database D and the profit table of products, how to find the best correlation high utility itemsets on the database with items having negative profits. To address these issues, the paper proposes a COHUIs_CoHUN method to mine correlation high utility itemsets with negative profits.

Keywords: *Data mining, negative profit, high utility itemsets, correlation itemsets.*

I. OVERVIEW

In today's business, due to the increasingly fierce competition among companies, promotional campaigns with many attractive offers for consumers are becoming more common, with the goal of stimulating purchases. As a result, some promotional products that are attached to regular products are sold at a loss, meaning they create products with negative profit margins. Additionally, companies may also lower the selling price of some products below the purchase price in order to recover capital, thus resulting in negative profit units. High utility itemset mining algorithms currently do not take into account the negative profit margins of these types of products.

Many high utility itemset mining (HUI) algorithms have been developed in recent years. In 2015, the HUP-Miner algorithm [3] was proposed by author S. Krishnamoorthy, which is an improved version of the HUI-Miner algorithm [1] with two new pruning strategies (PU-Prune and LA-Prune) for mining high utility itemsets. In 2016, Lin and colleagues proposed the FHN algorithm [4]; in 2018, KulDeep Singh and colleagues proposed the EHIN algorithm [5] for efficient HUI mining on databases with negative profits. Although these algorithms are effective in mining HUIs, the number of HUIs found often contains many weakly correlated items that do not have practical significance. To address this issue, in 2018, W. Gan and colleagues proposed the CoHUIM algorithm [6] which considers both the correlation and profit relationship between products in transactions. And in 2020, the CoHUI-Miner algorithm [7] was proposed by Bay Vo and colleagues, which uses database projection to reduce the size and introduces a new concept, called the Expected Transaction Utility Prefix, to eliminate pattern sets that do not meet the minimum threshold in the mining process. The algorithm achieved better performance than the CoHUIM [6] algorithm in terms of both runtime and memory usage. In 2019, author W. Gan and colleagues proposed the CoUPM algorithm [8], an improvement upon the CoHUIMiner [6] algorithm that utilizes the Ulility-List storage structure to enhance the performance of mining correlated high utility itemsets.

Based on the aforementioned studies, in this paper, we propose the COHUIs_CoHUN algorithm for mining related high utility itemsets with negative profits. Experimental results show that the proposed algorithm effectively eliminates many redundant HUI sets and outperforms EHIN [5] algorithm in terms of performance.

The rest of this paper is organized as follows: Section 2 presents the theoretical background and some current methods to address the problem of mining related high utility itemsets with negative profits. The experimental results and discussion are presented in Section 3, where we discuss how the proposed algorithm is implemented. Section 4 provides conclusions on the achieved algorithm

and future research directions.

II. METHOD

Among the EHIN-based search algorithms [5], the authors searched for high-utility itemsets for patterns with positive profits first, followed by those with negative profits. This has been shown to reduce the search space by using two separate Primary and Secondary sets of patterns, where the Primary ones are considered more important and the Secondary ones are considered optional. This ensures that a pattern does not have to intersect with all other patterns in order to find the high-utility itemsets, but only needs to intersect with patterns in both Primary and Secondary sets until there are no more itemsets that satisfy the min-util threshold (since the Primary and Secondary sets are sorted by utility). Therefore, improving these search procedures is unnecessary. We propose to use these procedures when performing CoHUN to mine non-redundant high-utility patterns with negative profits.

In the proposed CoHUIs_CoHUN algorithm, we will use an additional minimum correlation threshold minCor (provided by the user). The resulting CoHUIs set (Step 13) will contain all correlation high-utility itemsets with negative profits. Adding the minCor correlation threshold in the EHIN algorithm [5] aims to increase the efficiency of efficiently mining correlated high-utility itemsets on a database of transaction data with redundant negative profits. For the total of unchanging transactions, the resulting correlated high-utility itemsets on the negative-profit database have eliminated redundant patterns. Therefore, the proposed CoHUIs_CoHUN algorithm will have faster computation time than the EHIN algorithm [5], and we have an algorithm capable of mining correlated high-utility itemsets on a negative-profit database.

Algorithm 1: COHUIs_CoHUN

Input D: set of transactions, minUtil: utility threshold, minCor: correlation threshold.

Output CoHUIs: set of non-redundant high-utility itemsets.

Step 1: initialize set $\propto$ to empty

Step 2: create set ρ, which contains elements with positive values in D

Step 3: create set η, which contains elements with negative values in D

Step 4: traverse D, calculate RLU($\propto$,z) $\forall$ z $\in$ ρ and calculate TU

Step 5: find set Secondary(α) = {z|z $\in$ ρ $\wedge$ RLU($\propto$,z) $\geq$ min_util x TU}

Step 6: sort the elements of Secondary($\propto$) in increasing order based on their RTWU($\propto$) values

Step 7: traverse D, remove items z $\in$ Secondary($\propto$) and remove empty transactions T

Step 8: sort the elements in transaction T$\in$D in the order of Secondary($\propto$) and then the negative values

Step 9: assign offsets to each transaction T$\in$D

Step 10: traverse D, calculate RSU($\propto$,z)$\forall$z $\in$ Secondary($\propto$) and Sup(X)$\forall$z $\in$ Secondary($\propto$)

Step 11: find set Primary(α) = {z|z $\in$ Secondary(α)$\wedge$ RSU(α,z)$\geq$ min_util x TU}

Step 12: Search-P(η,α,D,Primary(α),Secondary(α),min_util,min_cor

Step 13: return CoHUIs

Algorithm 2: Procedure Search-P

Input η: set of elements with negative values in D, α: set of elements, α – D: set of transaction dataset, Primary(α): set of Primary for set α, Secondary(α): set of Secondary for set α, min_util is the utility threshold, min_cor: the correlation threshold between elements

Output Set of CoHUIs for set α with positive element

Traverse each element z $\in$ Primary(α)

Step 1: Create set β = α $\cup$ {z}

Step 2: Traverse α – D, calculate U(β), calculate Sup(β), calculate Kulc(β) and create set β - D

Step 3: if Kulc(β) $\geq$ min_cor and U(β) $\geq$ min_util x TU then output β

Step 4: if Kulc(β) > min-cor and U(β) > min_util x TU then Search-N(η, β, β - D, min_util, min_cor)

Step 5: Traverse β - D, calculate RSU(β,z) and RLU(β,z) for element z $\in$ Secondary(α)

Step 6: Primary(α) = z | z $\in$ Secondary(α) | RSU(β,z) $\geq$ min_util x TU

Step 7: Secondary(α) = z | z $\in$ Secondary(α) || RLU(β,z) $\geq$ min_util x TU

Step 8: Search-P(η, β, β - D, Primary(β), Secondary (β), min_util, min_cor)

Algorithm 3: Procedure Search-N

Input η: set of elements with negative values in D, α: set of elements, α – D: set of transaction dataset, min_util is the utility threshold, min_cor: the correlation threshold between elements

Output Set of CoHUIs for set α with negative element

Traverse each element z $\in$ η

Step 1: β = α $\cup$ {z}

Step 2: Traverse α – D, calculate U(β), calculate Sup(β), calculate Kulc(β) and create set β - D

Step 3: if Kulc(β) ≥ min_cor and U(β) ≥ min_util x TU then output β

Step 4: Traverse β - D, calculate RSU(β,z) for element z ∈ η

Step 5: Primary(β) = {z | z ∈ η | RSU(β,z) ≥ min_util x TU}

Step 6: Search-N(Primary(β), β, β - D, min_util, min_cor)

Algorithm illustration: We have a transaction database table with negative profit as follows:

TABLE I
TRANSACTION DATA

T_{id}	a	b	c	d	e
T_1	2	2	-	1	3
T_2	-	1	5	-	1
T_3	-	2	1	3	2
T_4	-	-	2	1	3
T_5	2	-	-	-	-
T_6	2	1	4	2	1
T_7	-	3	2	-	2

TABLE II
PROFIT VALUE TABLE

I	a	b	c	d	e
$EU(X_i)$	2	-3	1	4	1

After calculating the profits of items in transactions and sequentially calculating U(X) according to formula [8] , $TU(T_j)$ according to formula [8], $TWU(T_j)$ (according to formula [8]), we obtain the following tables:

TABLE III
PROFIT BY TRANSACTION

$U(X_i, T_j)$	a	b	c	d	e
T_1	4	-6	-	4	3
T_2	-	-3	5	-	1
T_3	-	-6	1	12	2
T_4	-	-	2	4	3
T_5	4	-	-	-	-
T_6	4	-3	4	8	1
T_7	-	-9	2	-	2

Perform CoHUIs_CoHUN algorithm with the following settings: min_util = 0.2, min_cor = 0.2.

Step 1: Set α = 0

Step 2: Set β as the set of items with U > 0

Step 3: Set η as the set of items with U < 0

Step 4: Calculate RLU(X), T_u

Step 5: Set Secondary = a,c,d,e Secondary = RLU(X) > min_util x T_u

TABLE IV
BENEFITS OF ITEMS IN TRANSACTIONS

	a	b	c	d	e	$TU(T_j)$
T_1	4	-6	-	4	3	5
T_2	-	-3	5	-	1	3
T_3	-	-6	1	12	2	9
T_4	-	-	2	4	3	9
T_5	4	-	-	-	-	4
T_6	4	-3	4	8	1	14
T_7	-	-9	2	-	2	-5
U(X)	12	-27	14	28	12	
TWU(X)	23	26	30	37	35	151

Step 6: Sort Secondary = a < c < d < e

Step 7: Delete items in T that are not in Secondary, set T to empty

Step 8: Merge T's, U(X) = U(Tx_1) + U(Tx_2), TU(T_{new}) = U(Tx_1) + U(Tx_2)

Step 9: Attach Offset to T

Step 10: Traverse D, calculate RSU(X), Sup(X)

Step 11: Set Primary = a,c,d,e

Step 12: Call Search_P(η, α, Primary, Secondary, min_uti, min_cor)

Step 13: Return CoHUIs

TABLE V
TABLE AFTER EXECUTING STEPS 4-10

	a	c	d	e	RTU	b(η)
T_1	4	-	4	3	11	(6)
T_2	-	7	-	3	10	(12)
T_3	-	1	12	2	15	(6)
T_4	-	2	4	3	9	-
T_5	4	-	-	-	4	-
T_6	4	4	8	1	17	(3)
Sup	3	4	4	5		4
RSU	32	47	37	12		

After executing Step 12 with the Search_P procedure, the result of the returned CoHuis set at Step 13 is as follows: **CoHUIs = ({a, c, d}, {a, c, d, e}, {a, c, d, e, b}, {a, d}, {a, d, e}, {a, d, e, b}, {c}, {c, d}, {c, d, b}, {c, d, e}, {c, d, e, b}, {c, e}, {d}, {d, e}, {d, e, b})**

The proposed algorithm has been more effective in exploiting high-utility itemsets in a weighted database with no redundancy by incorporating the user-defined correlation threshold minCor. For example, in the 2018 paper EHIN[5] algorithm discovered 20 CoHUIs, but when applying the proposed algorithm, the number of high-utility itemsets was reduced to 15 sets (eliminating 5 redundant high-utility sets).

III. EXPERIMENTAL RESULTS

The CoHUIsCoHUN algorithm was implemented in Java and run on a Dell Vostro 3500 computer with an Intel

Core i7-1165G7 @ 2.80GHz processor, 16GB RAM, on the Windows 10 operating system. Test databases were downloaded from the SPMF library and consisted of transactional databases with negative profits including Chess, Mushroom, Retail and Accidents. The experimental results of the CoHUN algorithm were compared with the latest algorithm that also mines CoHUIs, which is EHIN [5]. The experimental results were evaluated based on the high-utility itemsets correlation to negative profits.

TABLE VI
EXPERIMENTAL DATA

Dataset	Trans	Items (I)	Avg Len
Chess	3.196	75	(37A)
Mushroom	8.124	119	23
Retail	88.162	16.470	10.3
Accidents	340.183	468	33.8

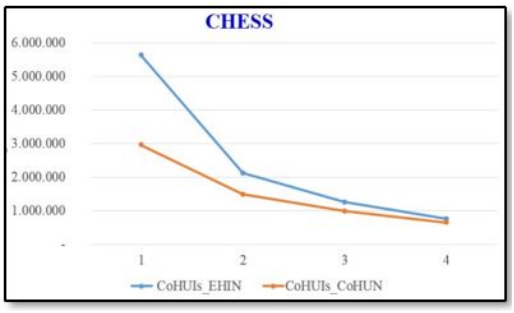


Figure 1. Results on the Chess dataset.

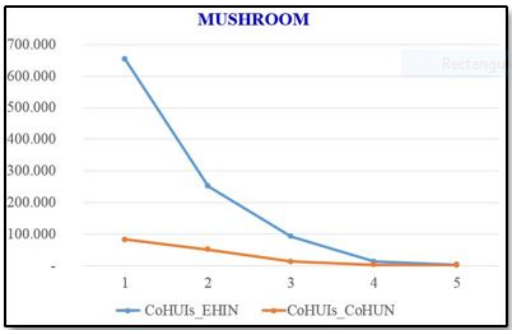


Figure 2. Results on the Mushroom dataset.

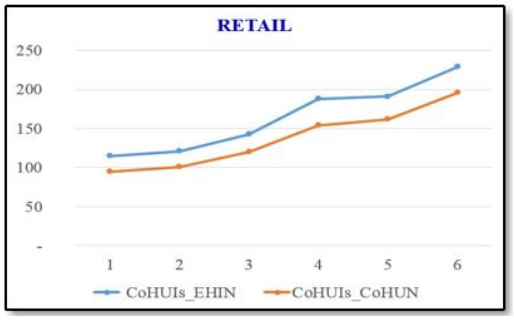


Figure 3. Results on the Retail dataset.

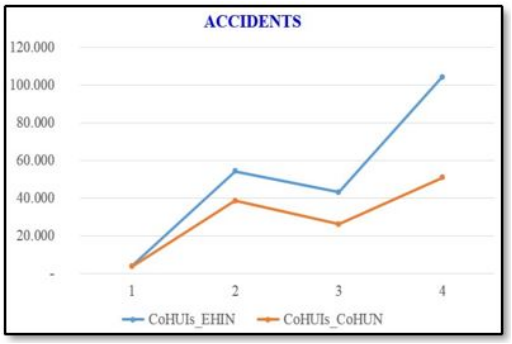


Figure 4. Results on the Accidents dataset.

Figures 1, 2, 3, and 4 show the experimental results comparing the proposed CoHUN algorithm and the EHIN [5] algorithm based on the number of discovered high-utility itemsets (HUIs) at different minimum correlation thresholds minCor. The experimental results demonstrate that the CoHUN algorithm is more effective in discovering HUIs than the EHIN [5] algorithm in all databases, including Chess, Mushroom, Retail, and Accidents. The experimental results demonstrate that the pruning strategies, correlation measures, and data structures used in the proposed CoHUIs_CoHUN algorithm are appropriate for mining correlation CoHUIs on databases with negative profits.

IV. CONCLUSION ANF FUTURE RESEARCH DIRECTIONS

In this paper, we proposed the CoHUIs_CoHUN algorithm for mining correlation high-utility itemsets (CoHUIs) with negative profits. The algorithm employs the PNU-List structure to separate negative and positive profit values for full CoHUIs mining on databases with negative profits. Additionally, the algorithm applies several efficient pruning strategies to reduce search space, including U-Prune, KulcPrune, and LA-Prune. The experimental results demonstrate that the proposed algorithm is more effective in discovering CoHUIs than the EHIN [5] algorithm on all

tested databases with negative profits. The memory usage of the CoHUIs_CoHUN algorithm is also better than that of the EHIN [5] algorithm in utilizing the minCor threshold.

The next proposed development direction is to improve the storage data structure, correlation threshold, and pruning strategy to increase the performance of the algorithm on databases with negative profits, as well as extending the CoHUIs mining to other types of databases such as dynamic databases and growing databases.

REFERENCES

[1] M. Liu, J. Qu, (2012). "Mining high utility itemsets without candidate generation (HUI)". *Proceedings of the 21st ACM International Conference on Information and Knowledge Management*, 55–64.

[2] P. Fournier-Viger, C. W. Wu, S. Zida, V. S. Tseng, (2014). "Faster highutility itemset mining using estimated utility cooccurrence pruning (FHM)", *Proceedings of the International Symposium on Methodologies for Intelligent Systems, Springer*, 83–92.

[3] S. Krishnamoorthy, (2015), "Pruning strategies for mining high utility itemsets (HUP-Miner)", *Expert Systems with Applications* 42(5), 2371-2381.

[4] J. C. W. Lin, P. Fournier-Viger, and W. Gan, (2016)," An efficient algorithm for mining high-utility itemsets with negative unit profits (FHN)", *Knowledge-Based Systems*, 283-298.

[5] KulDeep Singh, Abhimanyu Singh, Harish Kumar Shakya, (2018), "Mining of high-utility itemsets with negative utility (EHIN)", *Wiley, Expert Systems*

[6] W. Gan, J. C. W. Lin, P. Fournier-Viger, H. C. Chao, H. Fujita, (2018), "Extracting non-redundant correlated purchase behaviors by utility measure (CoHUI-Miner)", *Knowledge-Based Systems* 143, 30-41.

[7] B. Vo, L. V. Nguyen, V. V. Vu, M. T. H. Lam, T. T. M. Duong, L. T. Manh, et al., (2020), "Mining Correlated High Utility Itemsets in One Phase (CoHUI-Miner)", vol. 8. *IEEE Access*, 90465-90477.

[8] Ayoub, F. (1982), "Probabilistic completeness of substitution-permutation encryption networks", *IEE Proceedings E (Computers and Digital Techniques)*, 129(5), 195-199.

Cao Tung Anh is currently a Ph.D holder and the head of the Information Systems Department at the Faculty of Information Technology, Ho Chi Minh City University of Technology (HUTECH), Vietnam. His research interests lie in the fields of information systems, data mining, distributed databases, coding theory and information security.
Email: ct.anh@hutech.edu.vn

Ngo Quoc Huy is currently a lecturer in the Faculty of Information Technology at Van Lang University (VLU), Vietnam. His research interests lie in the fields of data mining, distributed databases, and network security.
Email: huy.nq@vlu.edu.vn

Pham Ngoc Diem is currently studying a Master's degree at Ho Chi Minh City University of Technology (HUTECH), Vietnam. Her research interests lie in the fields of data mining, distributed databases
Email: diempn1986@gmail.com