Machine Learning Assisted Differential Distinguishers For Lightweight Ciphers

Anubhab Baksi[1], Jakub Breier[1], Xiaoyang Dong[2], and Chen Yi[2]

[1] Nanyang Technological University, Singapore

[2] Tsinghua University, Beijing, PR China

anubhab001@e.ntu.edu.sg, jbreier@ntu.edu.sg, xiaoyangdong@tsinghua.edu.cn, chenyi19@mails.tsinghua.edu.cn

Abstract. At CRYPTO 2019, Gohr first introduces the deep learning based cryptanalysis on round-reduced SPECK. Using a deep residual network, Gohr trains several neural network based distinguishers on 8-round SPECK-32/64. The analysis follows an 'all-in-one' differential cryptanalysis approach, which considers all the output differences effect under the same input difference.

Usually, the all-in-one differential cryptanalysis is more effective than that only uses one single differential trail. However, when the cipher is non-Markov or its block size is large, it is usually very hard to fully compute. Inspired by Gohr's work, we try to simulate the all-in-one differentials for such non-Markov ciphers through deep learning. As proof of works, we trained several distinguishing attacks following machine learning simulated all-in-one differential approach. We present 8-round differntial distinguishers for Gimli-Hash and Gimli-Cipher, each with trivial complexity. Finally, we explore more on choosing an efficient machine learning model and show a three layer neural network can be used.

Keywords: gimli, distinguisher, machine learning, differential

1 Introduction

Machine Learning (ML) tools have made a great progress in the last decades. At present, such techniques are predominantly used to various fields, such as computer vision [28], machine translation [3,30], autonomous driving [16], to name a few. In the aspect of cryptography however, usage of ML is mainly confined in the context of side channel analysis (SCA), such as [14, 25], to the best of our knowledge.

Therefore the applicability of ML techniques in classical cryptanalysis is not much explored. It seems the community is rather skeptical regarding

The first author would like to thank Raghvendra Rohit (University of Waterloo, Canada) for his kind support. This work is partially supported by TUM CREATE.

such approach. For example, the authors in [1] comment, “Neural networks are generally not meant to be great at cryptography. Famously, the simplest neural networks cannot even compute XOR, which is basic to many cryptographic algorithms”.

Although ML techniques have been used against legacy ciphers, like the Enigma (the German cipher used during second world war) [20], it is not until the work by Gohr [19] at CRYPTO’19 that this line of research finally gets its exposure. The idea here is applied to recover the key attacks on round-reduced `SPECK` using ML.

This work focuses on extending the commonly used model of differential distinguisher by using ML techniques. In the case of differential distinguisher, the attacker XORs a chosen input difference δ to the input of the state of the (reduced round) cipher and watches for a particular output difference $\varDelta$, with randomly chosen inputs. If the $(\delta, \varDelta)$ pair occurs with probability significantly higher for the (reduced round) cipher than what it should be for a random case, the (reduced round) cipher can be distinguished from the random case. This probability of $\delta \to \varDelta$ is modeled by differential branch number [17] or by automated tools like Mixed Integer Linear Programming (MILP) [26]. We extend the modeling for differential distinguisher by incorporating machine learning algorithms.

We argue that the usual modeling with branch number or MILP underestimates attacker’s power. Having been collected the output differences, the attacker can use any technique to distinguish the cipher from the random case. In such a situation, machine learning techniques can reduce the search complexity estimated by existing methods, often to the cube root.

In our machine learning model, we choose t (≥ 2) input differences; say, $\delta_0, \delta_1, \ldots, \delta_{t-1}$. After that, we feed all the output differences to the machine learning model (instead of checking for any fixed output difference).

Detailed discussion of machine learning is out of scope of this work, interested readers may refer to standard textbook such as [21] for the same. We use TensorFlow[1] back-end with Keras[2] API. The optimizer function is based on Adam algorithm [22].

Our Contributions

In this work, we construct machine learning models to simulate the *all-in-one differentials* in [2], to distinguish non-Markov ciphers. We consider

[1]`https://www.tensorflow.org/`
[2]`https://keras.io/`

the classical distinguisher game: Given $\texttt{ORACLE} \overset{\$}{\leftarrow} \{\texttt{CIPHER}, \texttt{RANDOM}\}$, the attacker is to identify whether $\texttt{ORACLE} = \texttt{CIPHER}$ with probability significantly $> 1/2$ and with sufficiently small number of queries.

While more description of our method is given in Section 3, we present a summary here. During the training (offline phase), we take the (possibly round-reduced) CIPHER. With randomly generated key and plaintext, we find generate t (≥ 2) differentials. We feed the output differentials into a machine learning model stating the i^{th} differential belongs to class i. When enough data are generated, we run the model and check for its training accuracy, a. If $a > 1/t$, we proceed to the testing (online phase), otherwise we abort the procedure. During the testing, we randomly generate plaintexts and compute the input differentials (in order). Then, the corresponding output differentials are generated after querying the ORACLE. The ML model predicts the corresponding classes for each of the output differentials. We tally the number of cases where the prediction is successful, i.e., the ML model returns the correct class for the distinguisher. Comparing the relative accuracy for this tally, the attacker is able to determine if $\texttt{ORACLE} = \texttt{CIPHER}$ or $\texttt{ORACLE} = \texttt{RANDOM}$.

Thus we reduce the actual problem of distinguishing the CIPHER from RANDOM into a classification problem. Here we choose the most common ML tool, Multi-Layer Perceptron (MLP).

We apply our strategy to round-reduced GIMLI [10], which is a 2^{nd} round candidate in ongoing the NIST Lightweight Cryptography (LWC) competition. Results are described in Section 4 and can be summarized as follows For GIMLI, we obtain 8-round practical distinguishers on GIMLI-HASH and GIMLI-CIPHER in nonce respecting case in Section 4. The distinguishing complexity is $2^{17.6}$ offline data to train the model offline and $2^{14.3}$ online data to distinguish the cipher. Note that the authors of GIMLI [10] proved that the optimal 8-round differential trail is with probability of 2^{-52}. If we use this trail to distinguish 8-round GIMLI, we need at least 2^{52} online data. Thus, we are able to perform a differential distinguisher in around cube root complexity.

In Section 5, we discuss effects of choosing different neural networks with respect to 8-round GIMLI-CIPHER as the target cipher. We show even a three layer neural network works well for our purpose. We also discover a few differential distinguishers for 8-round GIMLI-CIPHER in the process.

It is to be mentioned that our ML assisted model is generic in nature and can be adopted to any symmetric key setting. All the results presented are practical and can be carried out in around an hour in a modern computer. We would like to emphasize that we use the same model as the differential distinguisher, only the analysis is done by machine learning (instead of branch number or automated tools like MILP). We feed all the output differences to ML (i.e., no output difference is fixed a prior).

2 Background

2.1 Markov Ciphers

At Eurocrypt'91, Lai, Massey and Murphy [23] introduced the concept of Markov Ciphers for iterated ciphers.

Definition 1 (Markov Chain [23]). *Given a sequence of discrete random variables $v_0, v_1, ..., v_r$ is a Markov chain, if for $0 \leq i < r$,*

$$P(v_{i+1} = \beta_{i+1} | v_i = \beta_i, v_{i-1} = \beta_{i-1}, ..., v_0 = \beta_0) = P(v_{i+1} = \beta_{i+1} | v_i = \beta_i). \tag{1}$$

If $P(v_{i+1} = \beta | v_i = \alpha)$ is independent of i for all α and β, the Markov chain is called homogeneous.

Given an r-round iterated cipher, the input of ith round is denoted as Y_i $(0 \leq i \leq r)$ and the corresponding input differences are $\Delta Y_0, \Delta Y_1, ..., \Delta Y_r$. Then, Lai et al. introduced the following definition of Markov cipher etc.

Definition 2 (Markov Cipher [23]). *An iterated cipher with round function $Y = f(X, K)$ is a Markov cipher if there is a group operation $\otimes$ for defining differences such that, for all choices of α $(\alpha \neq 0)$ and β $(\beta \neq 0)$,*

$$P(\Delta Y = \beta | \Delta X = \alpha, X = \gamma)$$

is independent of γ when the subkey K is uniformly random, or equivalently, if

$$P(\Delta Y = \beta | \Delta X = \alpha, X = \gamma) = P(\Delta Y(1) = \beta_1 | \Delta X = \alpha)$$

for all choices of γ when the sub-key K is uniformly random.

Theorem 1. *If an r-round iterated cipher is a Markov cipher and the r round keys are independent and uniformly random, then the sequence of differences $\Delta X = \Delta Y_0, \Delta Y_1, ..., \Delta Y_r$, is a homogeneous Markov chain. Moreover, this Markov chain is stationary if ΔP is uniformly distributed over the non-neutral elements of the group.*

Theorem 1 is adopted from [23]. According to Theorem 1, one can compute the probability of an r-round differential characteristic as,

$$P(\Delta Y_1 = \beta_1, \Delta Y_2 = \beta_2, \ldots, \Delta Y_r = \beta_r | \Delta X = \beta_0) = \prod_{i=1}^{r} P(\Delta Y_1 = \beta_i | \Delta X = \beta_{i-1}). \tag{2}$$

However, the above theory can not be applied to non-Markov ciphers, such as stream ciphers like `SALSA` [8], `TRIVIUM` [15]. One of the most important feature of those primitives is that there are no sub-keys in each iterated round. In this situation, the iterated cipher usually can not be regarded as Markov cipher or a homogeneous Markov chain. This is because, the differences in the former rounds usually have significant effect on the the differences of the latter rounds, where Equation 1 does not hold any more. Hence, we can not use the Equation 2 to compute the probability of a characteristic any more.

As shown in Figure 1, we introduce a toy cipher to explain the dependence of the differences between a 2-round cipher without sub-keys. We use the 4-bit SBox of `GIFT-64` [4] as an example. The differential distribution table (DDT) SBox (`1A4C6F392DB7508E`) is omitted here for the interest of brevity and can be found in [4]. Suppose $\Delta Y_1[0]$ and $\Delta Y_1[1]$ are the input difference of the upper and lower SBox in Figure 1, respectively. We try to compute the probability of a differential characteristic, where $\Delta Y_1 = (\mathtt{2}, \mathtt{3})$, $\Delta W_1 = (\mathtt{5}, \mathtt{8})$, $\Delta Y_2 = (\mathtt{6}, \mathtt{2})$, and $\Delta W_2 = (\mathtt{2}, \mathtt{5})$.

From the DDT of the `GIFT` SBox, it is easy to know the probability of $\Delta Y_1 \to \Delta W_1$ is 2^{-5}. The valid tuples of $(Y_1[0], W_1[0], Y_1'[0], W_1'[0])$ are $(\mathtt{0}, \mathtt{1}, \mathtt{2}, \mathtt{4})$, $(\mathtt{2}, \mathtt{4}, \mathtt{0}, \mathtt{1})$, $(\mathtt{4}, \mathtt{6}, \mathtt{6}, \mathtt{3})$ and $(\mathtt{6}, \mathtt{3}, \mathtt{4}, \mathtt{6})$. The valid tuples of $(Y_1[1], W_1[1], Y_1'[1], W_1'[1])$ is $(\mathtt{d}, \mathtt{0}, \mathtt{e}, \mathtt{8})$ and $(\mathtt{e}, \mathtt{8}, \mathtt{d}, \mathtt{0})$. Only input pairs with $(Y_1[0], Y_1[1]) = (\mathtt{0}, \mathtt{d}), (\mathtt{0}, \mathtt{e})$, $(\mathtt{2}, \mathtt{d})$ and $(\mathtt{2}, \mathtt{e})$ are valid for the differential characteristic. Hence the probability of the characteristic is 2^{-6}, instead of 2^{-9} computed by Equation 2.

2.2 Basic Description of `GIMLI`

`GIMLI` [9] is a cross-platform permutation proposed by Bernstein et al. at CHES'17. Based on `GIMLI` permutation, Bernstein et al. [10] introduced the lightweight hashing and authenticated encryption algorithms, i.e., `GIMLI-HASH` and `GIMLI-CIPHER`, which are included in the 2nd round of NIST Lightweight Cryptography (LWC) competition [27]. `GIMLI` is also used in the open source cryptographic library `LibHydrogen`[3].

[3]https://github.com/jedisct1/libhydrogen

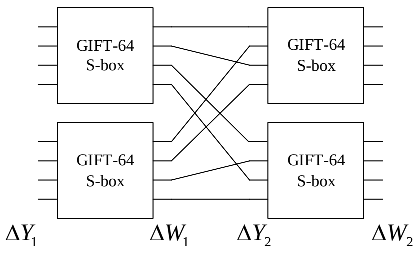

Fig. 1: One-round `GIFT-128` (unkeyed) permutation

`GIMLI` permutation works on a 384-bit state which can be represented as a 3×4 matrix of 32-bit words, namely

$$\mathbf{s} = \begin{bmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \end{bmatrix}$$

where the j^{th} column of $\mathbf{s}$ is $s_{*,j} = s_{0,j}, s_{1,j}, s_{2,j}$, the i^{th} row of s is $s_{i,*} = s_{i,0}, s_{i,1}, s_{i,2}, s_{i,3}$. The details of `GIMLI` permutation is given in Algorithm 1, which is adopted from [9].

Algorithm 1: `GIMLI` permutation

```
Input: $s = (\mathbf{s}_{i,j})$
Output: GIMLI($\mathbf{s}$)
 1: for $r$ from 24 down to 1 inclusive do
 2:     for $j$ from 0 to 3 inclusive do
 3:         $x \leftarrow s_{0,j} \lll 24$                                    ▷ SP-box
 4:         $y \leftarrow s_{1,j} \lll 9$
 5:         $z \leftarrow s_{2,j}$
 6:         $s_{2,j} \leftarrow x \oplus (z \ll 1) \oplus ((y \wedge z) \ll 2)$
 7:         $s_{1,j} \leftarrow y \oplus x \oplus ((x \vee z) \ll 1)$
 8:         $s_{0,j} \leftarrow z \oplus y \oplus ((x \wedge y) \ll 3)$
 9:     if $r \mod 4 = 0$ then                                          ▷ linear layer
10:         $s_{0,0}, s_{0,1}, s_{0,2}, s_{0,3} \leftarrow s_{0,1}, s_{0,0}, s_{0,3}, s_{0,2}$      ▷ Small-Swap
11:     else if $r \mod 4 = 2$ then
12:         $s_{0,0}, s_{0,1}, s_{0,2}, s_{0,3} \leftarrow s_{0,2}, s_{0,3}, s_{0,0}, s_{0,1}$      ▷ Big-Swap
13:     if $r \mod 4 = 0$ then
14:         $s_{0,0} = s_{0,0} \oplus \mathtt{9e377900} \oplus r$                      ▷ Add constant
    return $s_{i,j}$
```

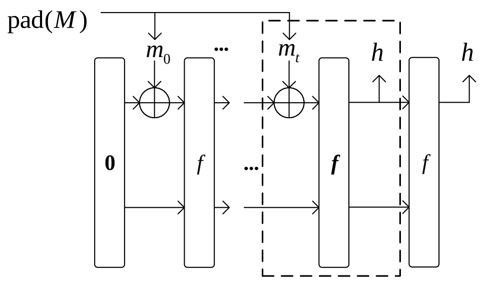

Fig. 2: Construction of `GIMLI-HASH`

`GIMLI-HASH` Taking advantage of `GIMLI` permutation and the Sponge construction [11], `GIMLI-HASH` is defined as shown in Figure 2. The message M is padded and divided into 128-bit blocks, i.e. $m_0, \ldots, m_t$. They are XORed into the state in the *absorb* phase the state and output 256-bit digest in the *squeeze* phase. Our neural distinguisher happens in the processing of the last message block m_t.

`GIMLI-CIPHER` Taking advantage of `GIMLI` permutation and the Monkey-Duplex construction [12], `GIMLI-CIPHER` is defined as shown in Figure 3. Here $\sigma_0, \ldots, \sigma_t$ are the associated data and padded to 128-bit blocks. Here $m_0, \ldots, m_t$ and $c_0, \ldots, c_t$ are the plaintext and ciphertext blocks, respectively. Note that at least one block of associated data is processed. So at least 2 `GIMLI` permutations (48 rounds) are used until the first ciphertext block c_0 generated. In this paper, we target on the case where only one block of associated data is processed.

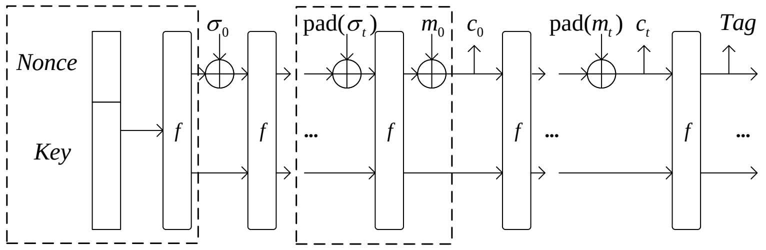

Fig. 3: Construction of `GIMLI-CIPHER`

2.3 Gohr's Work on SPECK (CRYPTO'19)

Using a deep residual neural networks, Gohr in [19] achieves better results than the best classical cryptanalysis on 11-round SPECK [5]. In this work, first train several machine learned distinguishers for round-reduced SPECK based on an all-in-one differential cryptanalysis [2] are trained, where the attacker considers potentially all the output differences of a block cipher for a given input difference and to combine the information derived from them. Under the Markov assumption, Gohr first computed the entire difference distribution table of round-reduced SPECK with a fixed input difference, which can be achieved due to the small block size of SPECK-32/64. Hence, the all-in-one differentials for the 5-/6-/7-/8-round SPECK-32/64 following the Markov assumption are reported. After this, the same distinguishing task via the neural networks is performed.

Concretely, the following steps are used to train the model:

1. Collecting training data by generating uniformly distributed keys and plaintext pairs given a fixed input difference as well as the binary labels Y_i.
2. If the binary label $Y_i = 1$, then the plaintext pairs are encrypted by k-round SPECK to produce the ciphertext pairs. If other wise, the random ciphertext pairs are generated.
3. Pre-process ciphertext pairs to fit the format required by the neural network and start to train them.

Finally, the author gains several machine-learned distinguishers on round-reduced SPECK which are a little more efficient the the distinguishers derived by computing the entire difference distribution table. After that, Gohr used the neural distinguishers to launch several key-recovery attacks on round-reduced SPECK.

Inspired from Gohr's work [19], the following questions come to our mind, and subsequently we employ neural networks:

- As the all-in-one approach considers all the output differences with one input difference, it is more efficient to distinguish the ciphers than that using only one input difference and one output difference. However, computing the all-in-one distinguisher is infeasible when the state of the cipher is large, e.g. 128-bit state. However, as shown by Gohr's attack, the neural networks can match the all-in-one distinguisher well. Hence, we can take advantage of the neural networks to simulate the all-in-one distinguishers for larger state ciphers.
- As shown in Gohr's attack, under Markov assumption, one can compute the all-in-one distinguishers somehow. However, there are many ciphers

which can not be assumed as Markov, such as some permutation-based ciphers or stream ciphers, where there are no sub-key in the iterated rounds (the justification is given in Section 2.1). In these situations, it is hard to compute the all-in-one distinguishers.

3 Machine Learning Based Distinguisher

3.1 Overview

A top-level description of the distinguisher is given in Algorithm 2. As mentioned earlier, here the attacker chooses t (≥ 2) input differences $\delta_0, \delta_1, \ldots, \delta_{t-1}$. In the testing (online) phase, the attacker tries to learn whether `ORACLE` is `CIPHER` or `RANDOM`. In other words, if the accuracy (a) of ML training is higher than what it should be for random data (i.e., $1/t$), the online phase takes place. In the online phase, if the accuracy of predicting the classes (a') is same a, we infer the chosen `ORACLE` = `CIPHER`. On the other hand, if $a' = 1/t$, we conclude `ORACLE` = `RANDOM`. Therefore, no conclusion can be drawn if the training accuracy $a = 1/t$. Also note that, we need at least two differentials. This is in contrast to the classical differential cryptanalysis where only one differential is chosen [29, Chapter 3.4]. Moreover, neither a nor a' can be (significantly) less than $1/t$, as at this bound the ML model is predicts classes uniformly.

Algorithm 2: Differential distinguisher using machine learning

```
1: procedure OFFLINE PHASE (Training)
2:     $TD \leftarrow (\cdot)$                        ▷ Training data
3:     Choose random $P, K$
4:     $C \leftarrow$ CIPHER($P$)
5:     for $i = 0; i \leq t-1; i \leftarrow i+1$ do
6:         $P_i \leftarrow P \oplus \delta_i$
7:         $C_i \leftarrow$ CIPHER($P_i$)
8:         Append $TD$ by $(i, C_i \oplus C)$
                          ▷ $C_i \oplus C$ is from class $i$
9:     Repeat from Step 3 if required
10:    Train ML model with $TD$
11:    ML training reports accuracy $a$
12:    if $a > 1/t$ then
13:        Proceed to Online phase
14:    else                                   ▷ $a = 1/t$
15:        Abort
```

```
1: procedure ONLINE PHASE (Testing)
2:     $TD' \leftarrow (\cdot)$                       ▷ Testing data
3:     Choose random $P$
4:     $C \leftarrow$ ORACLE($P$)
5:     for $i = 0; i \leq t-1; i \leftarrow i+1$ do
6:         $P_i \leftarrow P \oplus \delta_i$
7:         $C_i \leftarrow$ ORACLE($P_i$)
8:         Append $TD'$ by $C_i \oplus C$
9:     Test ML model with $TD'$
10:    for $i = 0; i \leq t-1; i \leftarrow i+1$ do
11:        Model predicts class $j$
12:        $a'$ = proportion of $\sharp\{i = j\}$
13:    if $a' = a$ then
14:        ORACLE = CIPHER
15:    else                                   ▷ $a' = 1/t$
16:        ORACLE = RANDOM
17:    Repeat from Step 3 if required
```

Here we describe the algorithm in terms of the entire state of the permutation for the sake of simplicity. In actual experimentation, we often consider part of the state (such as only the 'rate') part to make the permutation compatible with the primitives that use it.

Discussion on `ORACLE` = `RANDOM`

Given the `ORACLE` is `RANDOM`, if there are i right classifications out of the total t classes, the probability is $\Pr(i) = \frac{\binom{t}{i}(t-1)^{t-i}}{t^t}$. Hence, we can compute the expectation of the number of the right classifications: $E = \sum_{i=0}^{t} i \Pr(i)$.

In the training of the neural networks, the accuracy is the rate of the correct classifications. Namely, for `RANDOM` case, the accuracy is approximately E/t. For example, if $t = 2$, then the expected training accuracy is 0.5; if $t = 32$, the expected training accuracy is 0.03125. So if the training accuracy for a given set of samples is higher than what it would be for random data, we conclude it as `CIPHER`.

3.2 Training and Testing the Model

Our distinguisher works on the (unkeyed) permutation with a suitably chosen number of rounds. For the sake of simplicity, we denote the (unkeyed) permutation with input P as $\texttt{CIPHER}(P)$. The basic work-flow is given next.

Training (Offline)

1. Select t (≥ 2) non-zero input differences $\delta_0, \delta_1, \ldots, \delta_{t-1}$.
2. For each input difference δ_i, generate (an arbitrary number of) input pairs $(P, P_i = P \oplus \delta_i)$. Run the (unkeyed) permutation the input pairs to get the output pairs: $C \leftarrow \texttt{CIPHER}(P)$, $C_i \leftarrow \texttt{CIPHER}(P_i)$ for all i. Then XOR the outputs within a pair to generate the output difference $(C_i \oplus C)$. The output difference together with its label i (i.e., this sample belongs from class i) form a training sample.
3. Check if the training accuracy is $> 1/t$. Otherwise, the procedure is aborted.

Testing (Online)

1. Generate the input pairs in the same way as training. In other words, randomly generate an input P. With the same input differences chosen during training $\delta_0, \delta_1, \ldots, \delta_{t-1}$; generate new inputs $P_i = P \oplus \delta_i$ for all $i = 0(1)t-1$.

2. Collect the outputs C and C_i's by querying ORACLE with input P and P_i's in order, for all $i = 0(1)t - 1$.
3. Generate the testing data as $C \oplus C_i$ for all i and in order.
4. Get the predicted classes from the trained model with the testing data.
5. Find the accuracy of class prediction by the trained ML model.
6. If ORACLE = RANDOM, then the ML model would arbitrarily predict the classes. Therefore the testing accuracy would be (close to) $1/t$. Otherwise, if ORACLE = CIPHER, then the ML model would predict the class for $C \oplus C_i$ as i as accurately as training ($> 1/t$).

3.3 Comparison with Existing Models

All-in-one Differential Cryptanalysis The differences with all-in-one differential cryptanalysis [2] with our ML assisted differential distinguisher are as given:

1. All-in-one differential cryptanalysis takes only one input difference. In our case, we need at least two differentials.
2. All-in-one differential cryptanalysis does not use the cases where the output difference is not same as the pre-determined output difference. We make use of all the cases.

Additionally, our model does not directly lead to a key recovery attack (we leave the problem of key recovery for future research), and also results of multiple rounds in our model cannot be (at least directly) concatenated to infer on higher rounds.

Gohr's Model Our analysis focuses on different direction from that of Gohr's [19]. Gohr tries to prove the ability of ML-based distinguishers, so he selects SPECK [5] with small state size. This way he is able computed the full DDT of round-reduced SPECK to compare with his neural distinguishers. We focus on other directions. More precisely, we consider ciphers with bigger state and non-Markov ciphers. Both of them are hard to compute the all-in-one distinguishers and we successfully to simulate them via machine learning. It can be stated that our algorithm is simpler to understand and implement.

4 Results on Round-Reduced GIMLI

Using a SAT/SMT-based approach, the authors of GIMLI presented the optimal differential trails up to 8 rounds [10], which are given in Table 1.

Note that the differential probability of a trail is DP and the weight of the same trail is $-\log_2(DP)$. If we use the optimal 8-round differential trail to distinguish GIMLI, we need more than 2^{52} plaintext pairs. In this section, we introduce a machine-learning based differential distinguisher on 8-round GIMLI, the data complexity needed to distinguish 8-round GIMLI is only 1000 plaintext pairs.

Table 1: The optimal differential trails for the round-reduced GIMLI

Rounds	1	2	3	4	5	6	7	8
Weight	0	0	2	6	12	22	36	52

Note that we are going to distinguish the output of reduced GIMLI-HASH and GIMLI-CIPHER with a given input difference from random data. We show our analysis for GIMLI-HASH and GIMLI-CIPHER.

GIMLI-HASH For GIMLI-HASH in Figure 2, we focus on the processes of absorbing the last block of message M and squeezing the first 128-bit hash value h as shown in the dash box.

Data Collection. Suppose the full message M is just 127 bytes (denote the i^{th} byte as $M[i]$), then after padded with a zero byte, the 128-bit block is absorbed into the sponge function. Note that the initial 384-bit state is zero except the last byte. We collect the training data by flipping the least significant bit of byte $M[4]$ and $M[12]$. Namely, process the message pairs with difference `1` in the 4^{th} byte or 12^{th} byte and compute the first 128-bit hash values pairs accordingly and collect the difference of hash values Δh as training samples. The training samples are labeled with 0 or 1 according to different message differences. Our neural network is to classify hash differences with the two different message differences into the two labels. Totally, we generate $2^{17.6}$ samples.

GIMLI-CIPHER For GIMLI-CIPHER in Figure 3, we assume there is only one associated data block, so at least 2 GIMLI permutations (48 rounds) are involved until the first ciphertext block c_0 output. We reduce the 48 rounds to 8 rounds to find non-randomness in the ciphertext block c_0 via neural network.

Data Collection. Generate uniformly distributed 256-bit keys K and 128-bit nonce pairs N with difference `1` in the 4^{th} byte or 12^{th} byte of the nonces. Set the associated data and the first message block m_0 to be zero. Compute the ciphertext c_0. Collect the differences of Δc_0 with label 0 or 1 as training samples. Our neural network is to classify the ciphertext differences with the two different nonce differences into the two labels. Totally, we generate $2^{17.6}$ samples.

Training. Training was run for 20 epochs on the data set of size $2^{17.6}$.

Distinguishing `GIMLI` *via Machine-learned Model.* After trained the model in the training phase, we get the machine-learned model for 8-round `GIMLI-CIPHER` stored in ".h5" file. Then we generate $2^{14.3}$ valid samples and also $2^{14.3}$ random data to test the model. It outputs about an accuracy 0.5120 for valid samples and 0.5001 for random data. Namely, we can distinguish 8-round `GIMLI-CIPHER` with $2^{17.6}$ offline data and $2^{14.3}$ online data. We also train 6-/7-/8-round `GIMLI-HASH` and `GIMLI-CIPHER` with the same data complexity, the accuracies are given in Figure 2.

Table 2: Accuracy of neural distinguishers on Round-reduced Initialization of `GIMLI-CIPHER` and `GIMLI-HASH`

Rounds	Accuracy	
	`GIMLI-HASH`	`GIMLI-CIPHER`
6	0.9689	0.9528
7	0.7229	0.6340
8	0.5219	0.5099

5 Choice of Machine Learning Model

For finding a distinguisher, we chose to opt for deep learning techniques that can reveal the hidden structures in the data without explicit feature selection. Our problem is treated as a classification problem where one tries to check whether a particular differential was inserted at the input to the cipher or not. Results in this section where obtained by finding a distinguisher on 8 rounds of `GIMLI` with 2^{17} of training data samples. Number of epochs was set to 5 as for higher numbers the models tend to overfit. When using machine learning algorithms, one has to make a choice on hyperparameters that would perform the best on the given problem. These parameters are normally chosen in an empirical way, by testing various network architectures and following best practices. There are automated techniques to tune the hyperparameters [6, 7], however, these require significant resources. There is also an approach called Neural Architecture Search (NAS) that aims at searching for the best neural network architecture for a given problem [18]. Here we report the outcome from the manual architecture search in Section 5.1. Results in this section were obtained by using Nvidia Quadro RTX 8000 with 48 GB of memory.

5.1 Manual Way to Determine the Architecture

We have tried several different neural network types, including the basic multi-layer perceptron (MLP), Convolutional Neural Network (CNN), and Long Short-Term Memory Network (LSTM). We have varied the width (number of neurons per layer) and the depth (number of layers) of these to find out the best accuracy and speed of learning. We have also tried several different types of activation functions. The input layer size of the model corresponds to bit size of the plaintext for a target cipher, and the output layer size is always two, to indicate whether there a differential can be distinguished or not. Our findings, which contain several distinguishers for `GIMLI-CIPHER`, are stated in Table 3. Architecture denotes number of neurons per layer starting from the input layer. Activation function denotes the function that was used in the hidden layers, as the output layer always used softmax. A summary can be given as follows:

- **CNNs** are not suitable for the purpose of finding a distinguisher. We have tried several architectures, and the accuracy was always 0.5. This is expected result, as CNNs are aimed at recognizing patterns in input data, which helps in image recognition or natural language processing, but does not work for cipher input where the bits are not related in any way.
- **LSTMs** perform better than CNNs, but worse than fine-tuned MLP. The main drawback of LSTMs was the training speed – as they have recurrent layers, these have high memory requirements and more computations are required. Generally, the time required to train LSTM was $\approx 10\times$ more compared to MLP.
- **MLPs** provide the best accuracy, and can be tuned to train in very fast time. Our best result was achieved by MLP with 2 hidden layers and 1024 neurons per hidden layer (MLP III in Table 3), respectively. One can notice that in some cases we used Leaky ReLU as activation function [24], which allows a small, positive gradient when the unit is not active (e.g. input is negative), and therefore is considered more “balanced.” This was an advantage for smaller networks compared to normal ReLU, but for the network with 1.2M parameters, its performance was slightly worse.

Table 3: Benchmarks for manual architecture search with 8-round GIMLI-CIPHER

Network	Architecture	Activation Function	♯ Parameters	Training Time (s)	Accuracy
MLP I*	(128, 296, 258, 207, 112, 160, 2)	ReLU	226,633	330.8	0.5465
MLP II*	(128, 1024, 2)	ReLU	150,658	270.2	0.5462
MLP III*	(128, 1024, 1024, 2)	ReLU	1,200,256	287.4	0.5654
MLP IV*	(128, 256, 128, 64, 2)	LeakyReLU	90,818	307.9	0.5473
MLP V*	(128, 1024, 2)	LeakyReLU	150,658	271.3	0.5470
MLP VI*	(128, 1024, 1024, 2)	LeakyReLU	1,200,256	290.8	0.5476
LSTM I*	(128, 256, 128, 2)	tanh/sigmoid	444,162	2814.6	0.5305
LSTM II*	(128, 200, 100, 128, 2)	tanh/sigmoid	313,170	2727.7	0.5324
CNN I	(128, 128, 128, 100, 2)	ReLU	128,046	475.6	0.5000
CNN II	(128, 1024, 128, 128, 100, 2)	ReLU	604,206	537.3	0.5000

$*$: Distinguisher for 8-round `GIMLI-CIPHER`

6 Conclusion and Open Problems

In this work, we propose a novel idea on finding distinguishers on symmetric key primitives by using machine learning. At the core, we use multiple differentials and convert the problem of distinguisher `CIPHER` from `RANDOM` into a classification problem. Thus, our idea extends from the classical all-in-one differential distinguisher [2, 13]. The classification problem can be efficiently tackled by a machine learning model, thus giving an edge over the all-in-one differential technique. Our method is simple to implement, and also efficient as it can be carried out in around an hour by a modern computer. As a proof of concept, we show our method on `GIMLI` [10]. Although we are unable to find an attack on full round `GIMLI`, we show reduction of search complexity reported by the designers for reduced rounds.

Here we summarize the basic advantages and limitations of our proposed model:

+ The attack model is simple. In terms of neural network, our method works with as simple as a three layer neural network.
+ For round-reduced ciphers, we show that the complexity for differential cryptanalysis can be reduced to around the cube root of the claimed complexity. For example, the designers of `GIMLI` [10] claim the complexity of mounting a differential distinguisher by existing modeling methods would be at least 2^{52} data (see Table 1). However we are able to find the same with complexity of around 2^{17} data.
+ Our work is generic, and can be applied to any symmetric key primitive where the differential cryptanalysis can be applied. Here we target non-Markov ciphers as these are typically more complicated to analyze.

– So far, we are not able to extend our model to cover further rounds.
– For the time being, our model does not have a key recovery functionality.

In a future scope, other non-Markov ciphers and Markov ciphers like `GIFT` [4] can be experimented with. Since the work relies on a classification problem at its core, a Support Vector Machine (SVM) can be used instead of neural network.

References

1. Abadi, M., Andersen, D.G.: Learning to protect communications with adversarial neural cryptography. CoRR **abs/1610.06918** (2016) 2
2. Albrecht, M.R., Leander, G.: An all-in-one approach to differential cryptanalysis for small block ciphers. In: Selected Areas in Cryptography, 19th International Conference, SAC 2012, Windsor, ON, Canada, August 15-16, 2012, Revised Selected Papers. (2012) 1–15 2, 8, 11, 15
3. Bahdanau, D., Cho, K., Bengio, Y.: Neural machine translation by jointly learning to align and translate. arXiv preprint arXiv:1409.0473 (2014) 1
4. Banik, S., Pandey, S.K., Peyrin, T., Sasaki, Y., Sim, S.M., Todo, Y.: GIFT: A small present - towards reaching the limit of lightweight encryption. In: Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings. (2017) 321–345 5, 16
5. Beaulieu, R., Shors, D., Smith, J., Treatman-Clark, S., Weeks, B., Wingers, L.: The simon and speck families of lightweight block ciphers. Cryptology ePrint Archive, Report 2013/404 (2013) `https://eprint.iacr.org/2013/404`. 8, 11
6. Bengio, Y.: Gradient-based optimization of hyperparameters. Neural computation **12**(8) (2000) 1889–1900 13
7. Bergstra, J., Bengio, Y.: Random search for hyper-parameter optimization. Journal of Machine Learning Research **13**(Feb) (2012) 281–305 13
8. Bernstein, D.J.: The salsa20 family of stream ciphers (2007) 5
9. Bernstein, D.J., Kölbl, S., Lucks, S., Massolino, P.M.C., Mendel, F., Nawaz, K., Schneider, T., Schwabe, P., Standaert, F., Todo, Y., Viguier, B.: Gimli : A cross-platform permutation. In: Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings. (2017) 299–320 5, 6
10. Bernstein, D.J., Kölbl, S., Lucks, S., Massolino, P.M.C., Mendel, F., Nawaz, K., Schneider, T., Schwabe, P., Standaert, F., Todo, Y., Viguier, B.: Gimli (2019) 3, 5, 11, 15
11. Bertoni, G., Daemen, J., Peeters, M., Assche, G.V.: Sponge functions. In: Ecrypt Hash Workshop (May 2007). (2007) 7
12. Bertoni, G., Daemen, J., Peeters, M., Assche, G.V.: Duplexing the sponge: Single-pass authenticated encryption and other applications. In: Selected Areas in Cryptography - 18th International Workshop, SAC 2011, Toronto, ON, Canada, August 11-12, 2011, Revised Selected Papers. (2011) 320–337 7
13. Biham, E., Shamir, A.: Differential cryptanalysis of des-like cryptosystems. In: Advances in Cryptology - CRYPTO '90, 10th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1990, Proceedings. (1990) 2–21 15

14. Cagli, E., Dumas, C., Prouff, E.: Convolutional neural networks with data augmentation against jitter-based countermeasures - profiling attacks without pre-processing. In: Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings. (2017) 45–68 1
15. Cannière, C.D., Preneel, B.: Trivium. In Robshaw, M.J.B., Billet, O., eds.: New Stream Cipher Designs - The eSTREAM Finalists. Volume 4986 of Lecture Notes in Computer Science. Springer (2008) 244–266 5
16. Chen, C., Seff, A., Kornhauser, A., Xiao, J.: Deepdriving: Learning affordance for direct perception in autonomous driving. In: Proceedings of the IEEE International Conference on Computer Vision. (2015) 2722–2730 1
17. Daemen, J., Rijmen, V.: The design of rijndael: Aes - the advanced encryption standard. (2002) 2
18. Elsken, T., Metzen, J.H., Hutter, F.: Neural architecture search: A survey. arXiv preprint arXiv:1808.05377 (2018) 14
19. Gohr, A.: Improving attacks on round-reduced speck32/64 using deep learning. In Boldyreva, A., Micciancio, D., eds.: Advances in Cryptology – CRYPTO 2019, Cham, Springer International Publishing (2019) 150–179 2, 8, 11
20. Greydanus, S.: Learning the enigma with recurrent neural networks. arXiv preprint arXiv:1708.07576 (2017) 2
21. Haykin, S.: Neural Networks and Learning Machines (third edition). Pearson (2008) 2
22. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014) 2
23. Lai, X., Massey, J.L., Murphy, S.: Markov ciphers and differential cryptanalysis. In: Advances in Cryptology - EUROCRYPT '91, Workshop on the Theory and Application of of Cryptographic Techniques, Brighton, UK, April 8-11, 1991, Proceedings. (1991) 17–38 4, 5
24. Maas, A.L., Hannun, A.Y., Ng, A.Y.: Rectifier nonlinearities improve neural network acoustic models. In: Proc. icml. Volume 30. (2013) 3 14
25. Maghrebi, H., Portigliatti, T., Prouff, E.: Breaking cryptographic implementations using deep learning techniques. In: Security, Privacy, and Applied Cryptography Engineering - 6th International Conference, SPACE 2016, Hyderabad, India, December 14-18, 2016, Proceedings. (2016) 3–26 1
26. Mouha, N., Wang, Q., Gu, D., Preneel, B.: Differential and linear cryptanalysis using mixed-integer linear programming. In: Information Security and Cryptology - 7th International Conference, Inscrypt 2011, Beijing, China, November 30 - December 3, 2011. Revised Selected Papers. (2011) 57–76 2
27. of Standards, N.I., (NIST), T.: Lightweight cryptography (LWC) standardization process (2019) 5
28. Sonka, M., Hlavac, V., Boyle, R.: Image processing, analysis, and machine vision. Cengage Learning (2014) 1
29. Stinson, D.R.: Cryptography - theory and practice. Discrete mathematics and its applications series. CRC Press (2006) 9
30. Wu, Y., Schuster, M., Chen, Z., Le, Q.V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K., et al.: Google's neural machine translation system: Bridging the gap between human and machine translation. arXiv preprint arXiv:1609.08144 (2016) 1