NNBits: Bit Profiling with a Deep Learning Ensemble Based Distinguisher*

Anna Hambitzer[1 [0000−0002−5357−832X]], David Gerault[1 [0000−0001−8583−0668]], Yun Ju Huang[1 [0000−0002−0820−1005]], Najwa Aaraj[1 [0000−0001−8782−5192]], and Emanuele Bellini[1 [0000−0002−2349−0247]]

Technology Innovation Institute, Cryptography Research Center, 9639 Abu Dhabi, UAE. name.lastname@tii.ae

Keywords: Cryptanalysis · Evaluation tools · Block cipher · Distinguisher · Avalanche dataset · Bit-profiling · Neural networks · Random number generator

Abstract. We introduce a deep learning ensemble (NNBits) as a tool for bit-profiling and evaluation of cryptographic (pseudo) random bit sequences. On the one hand, we show how to use NNBits ensemble to explain parts of the seminal work of Gohr [16]: Gohr's depth-1 neural distinguisher reaches a test accuracy of 78.3% in round 6 for SPECK32/64 [3]. Using the bit-level information provided by NNBits we can partially explain the accuracy obtained by Gohr (78.1% vs. 78.3%). This is achieved by constructing a distinguisher which only uses the information about correct or incorrect predictions on the single bit level and which achieves 78.1% accuracy. We also generalize two heuristic aspects in the construction of Gohr's network: *i)* the particular input structure, which reflects expert knowledge of SPECK32/64, as well as *ii)* the cyclic learning rate. On the other hand, we extend Gohr's work as a statistical test on avalanche datasets of SPECK32/64, SPECK64/128, SPECK96/144, SPECK128/128, and AES-128. In combination with NNBits ensemble we use the extended version of Gohr's neural network to draw a comparison with the NIST Statistical Test Suite (NIST STS) on the previously mentioned avalanche datasets. We compare NNBits in conjunction with Gohr's generalized network to the NIST STS and conclude that the NNBits ensemble performs either as good as the NIST STS or better. Furthermore, we demonstrate cryptanalytic insights that result from bit-level profiling with NNBits, for example, we show how to infer the strong input difference $(0x0040, 0x0000)$ for SPECK32/64 or infer a signature of the multiplication in the Galois field of AES-128.

1 Introduction

The security of cryptographic primitives is often expressed in terms of randomness: Does the primitive behave like a random function or permutation? While

* Published in *Topics in Cryptology CT-RSA 2023: Cryptographers' Track at the RSA Conference 2023, San Francisco, CA, USA, April 24–27, 2023, Proceedings (pp. 493–523)* https://doi.org/10.1007/978-3-031-30872-7_19
Code available under https://github.com/Crypto-TII/nnbits

it is difficult to give a satisfactory answer to this question, there are two main approaches to estimate the answer. The first approach is cryptanalysis: cryptographers scrutinise the primitive, and attempt to break it, through classical attacks, or sometimes, new ones. On the other side, it is also possible to use automated randomness testing tools to obtain an assessment [10,25,31]. Such automated methods are significantly less accurate than cryptanalysis, but they are significantly faster as well (a few hours, vs. continual scrutiny by academics for years or even decades). In this work, we investigate how a machine learning based approach can improve automatic randomness testing, while providing human cryptanalysts with an intuition on where to look to find more advanced attacks.

The choice of machine learning is motivated by its ability to detect complex patterns in many areas, such as image classification, e.g. [23], autonomous vehicle navigation, mastering games, and, recently, time series forecasting [27]. In the game of Go, neural networks in combination with Monte Carlo tree search have achieved superhuman performance without any input of expert knowledge [32]. Deep neural networks are universal in the sense that they can *in principle*[1] represent any function [21].

The idea of applying machine learning techniques to cryptographic tasks has been gaining traction recently. In particular, in his CRYPTO'19 article, Gohr showed for the first time that machine learning algorithms could outperform current state-of-the-art cryptanalysis[2], by exhibiting improved attacks on the block cipher SPECK32/64 using a neural network [16]. Benamira *et al.* [7] further demonstrated that the properties learnt by Gohr's classifiers are not trivial, and it is not fully understood why they perform so well. Understanding what a machine learning algorithm bases its prediction on is a notoriously difficult problem that the *explainable AI* research community focuses on, for example in DARPA's explainable AI program [18]. However, we believe that more explainable techniques are required for machine learning to become part of the standard toolkit of cryptographers.

We tackle the problem of explainability by creating bit profiles which may give relevant information to a cryptanalyst. The bit profiles are created through *ensemble learning*, a widely used technique in machine learning [17,2]. An ensemble consists of a diverse set of predictors, such as neural networks. Neural network ensembles have recently demonstrated impressive results [29] in the prestigious time series forecasting competition *"Makrikadis"*, which was dominated by statistical methods until 2020 [27].

In this paper, we propose NNBits ensemble, a machine learning-based blackbox distinguisher that identifies whether a collection of X bit sequences of length n comes from a random distribution or a function f. In NNBits ensemble, an

[1] Note that this statement has limited practical implications: even if enough data, representational power of the network, as well as sufficient computational resources to train the network are given, the training itself may be an NP-hard problem [26].

[2] This limit has recently been overpassed by human cryptanalysis in [8], giving machine learning a new threshold to overcome.

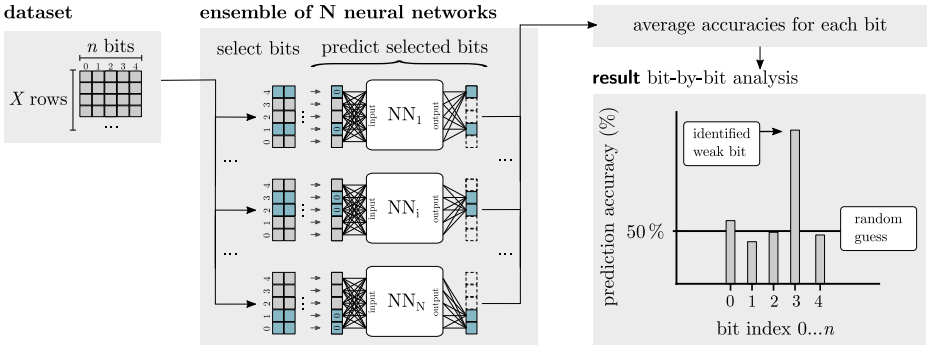

Fig. 1: The architecture of our proposed distinguisher NNBits ensemble.

ensemble of N neural networks is trained to predict a certain subset of the n bits, given the remaining bits as input. If a particular bit can be predicted with a prediction accuracy significantly higher than that of a random guess, the bit is identified as a *weak bit* and the sequences at hand are identified as NOT RANDOM, *i.e.,* coming from the function f. In particular, we focus on the block ciphers SPECK and AES, and our bit sequences are avalanche sequences, built using the difference between ciphertexts corresponding to the encryption of pairs of plaintexts having a single-bit difference. The bit-level granularity in the prediction provides information on which bits are easier to predict, as well as a convenient way for the cryptanalyst to observe dependencies between difference bits. The whole process is rather fast, due to a highly optimised implementation, as well as heavy parallelization of the neural network training across multiple GPUs. This implementation is available under [39], and an overview is given in Figure 1.

Contributions. Our contributions are the following:

1. We present NNBits ensemble, a deep learning ensemble analysis tool for bit-profiling of cryptographic (pseudo) random bit sequences that includes dependencies between different bits (Fig. 1). We provide publicly available source code for NNBits under [39].
2. Using the bit-level granularity of our tool, we provide a possible explanation for the accuracy of Gohr's neural distinguishers for SPECK.
3. We propose and implement a generalization of Gohr's classifiers (GENERALIZED NETWORK) which can be applied to larger datasets, such as the avalanche dataset.
4. We compare NNBits ensemble to NIST STS on the avalanche datasets of SPECK32/64 up to SPECK128/128 and AES-128, and conclude that the NNBits ensemble performs either as good as the NIST STS or better.
5. We demonstrate cryptanalytic insights that result from bit-level profiling with NNBits.

1.1 Organization

The remainder of this paper is structured as follows. In our preliminaries, we introduce the families of symmetric block ciphers SPECK and AES (Section 2.1), statistical tests techniques for cryptographic primitives (Section 2.2) and Gohr's neural distinguisher (Section 2.3). In our methodology, we detail the generation of the avalanche dataset (Section 3.1), the setup of the NIST STS (Section 3.2), and the implementation of NNBits (Section 3.3). Based on the previously presented material we conduct a set of three experiments (Sections 4.1 to 4.3) and conclude our findings in Section 5.

2 Preliminaries

2.1 Block ciphers

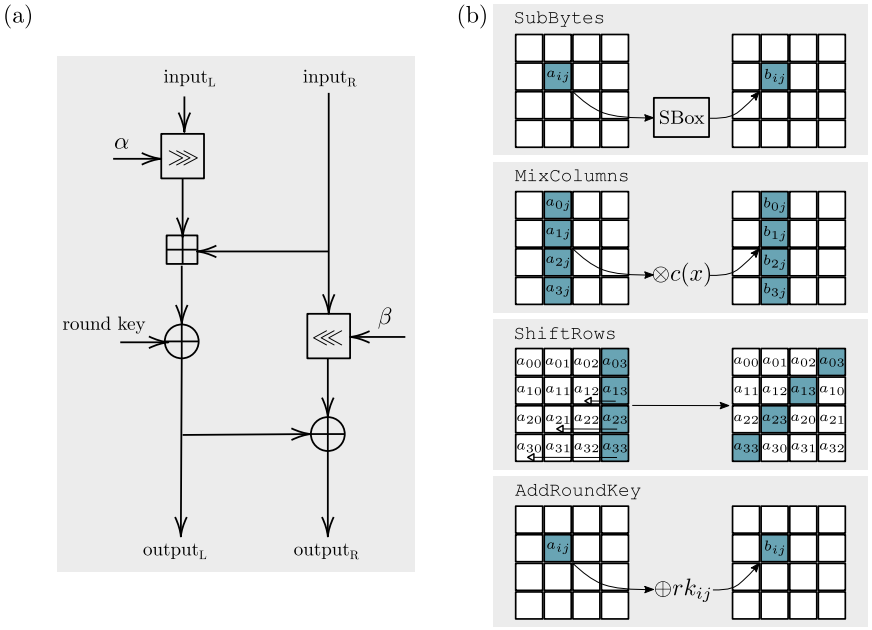

Fig. 2: Illustration of (a) one round of SPECK and (b) the round elements in AES.

SPECK. The block cipher SPECK is an ARX-based design proposed by the National Security Agency (NSA) [5], parametrised by a block size b and a key size k, and denoted by SPECKb/k. In this work, we focus on SPECK32/64, SPECK64/128, SPECK96/144, and SPECK128/128.

One round with the Feistel-like structure of SPECK is illustrated in Fig. 2. In the first round, input_L and input_R would be the first and last half of the plaintext. For other rounds, input_L and input_R come from output_L and output_R in the previous round. The ciphertext is the concatenation of output_L and output_R of the final round. α and β are the rotation parameters stated in the SPECK specifications [5]. The round keys are generated by the key schedule from the input key. This paper focused on the plaintext avalanche dataset, in which the key is fixed to zero. So we do not describe the cipher's key schedule here.

AES-128. The AES [13] (Advanced Encryption Standard) is the most widely used block cipher in the world as of today. It operates on 128 bits blocks, and 128/196/256 bits keys.

The 128 bits input is divided into a 4x4 bytes matrix. The round function is iterated 10, 12 or 14 times (depending on the key size), and is composed of 4 operations: `SubBytes`, `ShiftRows`, `MixColumns`, and `AddRoundKey`, as illustrated in Fig. 2. `SubBytes` substitutes each byte of the state according to a nonlinear `SBox`. `ShiftRows` shifts the second, third, and fourth row by one, two, and three positions to the left. `MixColumns` operates on each column separately, and performs a multiplication with the MixColumns matrix in the the Rijndael Galois field. `AddRoundKey` XORs each byte of the state with a byte of the round key rk_{ij}, derived through the *key schedule* algorithm. This paper focused on AES with 128-bit key size and 10 rounds. We do not introduce the key schedule of AES, since, for the same reason as in SPECK, it is not relevant to our work.

2.2 Statistical analysis of cryptographic primitives

During the years preceding the AES standardization process, statistical tests started being used to measure the security of block ciphers under the black-box approach [19] and to evaluate their quality when used as random number generators [34]. The battery of tests used by NIST [34] had the goal to analyze properties such as the proportion of zeroes and ones within the bitstring being tested (*frequency monobit test*), or within sub-blocks of this bit string (*frequency test within a block*). Such test suites constitute a *distinguisher* testing the *null hypothesis* $\mathcal{H}_0$, which asserts that the bitstring, or a sequence, being tested is random, against the *alternative hypothesis* $\mathcal{H}_a$, that the sequence is not random.

Statistical test results are to be interpreted with their *significance level* α, *i.e.,* the probability of the test rejecting the null hypothesis, given that the null hypothesis is true. For a given test result, the *P-value* is calculated under the assumption of a certain *reference distribution* and corresponds to the probability for test result to be observed if $\mathcal{H}_0$ is true. The null hypothesis $\mathcal{H}_0$ is accepted for a sequence, if the *P-value* is greater than or equal to α.

Our bit-level analysis, shown on Figure 1, studies whether a given bit can be predicted with an accuracy significantly better than a random guess. More specifically, we consider $\alpha = 0.01$; in other words, among 100 random tested sequences, we expect at most one to be (falsely) classified as non-random. Figure 3 illustrates the minimum accuracy p_i needed for our distinguisher to achieve a

significance value of 0.01. Under the assumption $\mathcal{H}_0$ of randomness, the number of successes S (*i.e.,* correct predictions of a bit) over X_{test} independent trials can be studied as a binomial reference distribution, with mean $\mu = p_0 \cdot X_{\text{test}}$ with $p_0 = 0.5$ and standard deviation $\sigma = \sqrt{X_{\text{test}} p_0 (1 - p_0)} = \sqrt{0.25 \cdot X_{\text{test}}}$. Given these parameters, the *P-value* corresponding to the accuracy of a given distinguisher can be derived, for instance using SciPy [40]; intuitively, the higher X_{test} is, the more significant a deviation from a 50% accuracy becomes. This is illustrated in Fig. 3.

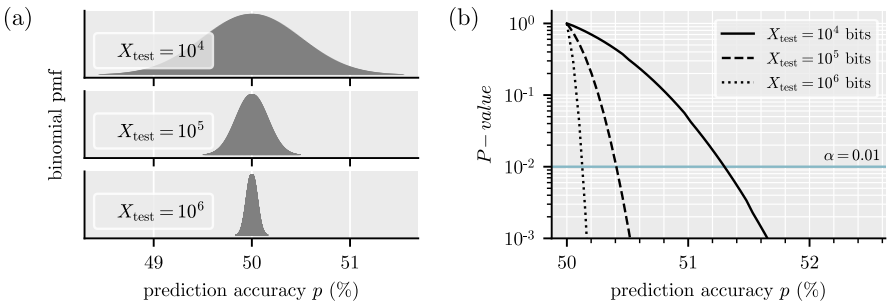

Fig. 3: a) Binomial probability mass function (pmf) for $p_0 = 50\%$ and X_{test}. b) Significance of an observation p in terms of the *P-value* considering a binomial reference distribution centered at $p_0 = 50\%$.

2.3 Gohr's neural distinguisher

In our work, we combine avalanche-based techniques with deep learning. For a thorough introduction to deep learning and neural networks we recommend [17,16]. In the following, we focus in particular on the construction of Gohr's neural distinguisher.

In his seminal paper, published at CRYPTO'19, Aron Gohr [16] proposes to use a deep neural network to distinguish whether pairs of SPECK32/64 ciphertexts correspond to the encryption of pairs of messages with a fixed difference $(0x0040, 0x0000)$, labeled as NOT RANDOM (1), or random messages, labeled as RANDOM (0). The resulting *Neural Distinguisher*, a residual neural network preceded by a size 1 1D-convolution, results in respectively $92.9, 78.8, 61.6$ and 51.4 % accuracy for 5, 6, 7 and 8 rounds of SPECK32/64, and is used to mount practical key recovery attacks on 11 rounds. Subsequent research work focused on explaining what features neural distinguishers can learn and on extending their use to other ciphers, improving on the methodology. In the first category, at Eurocrypt'21, Benamira *et al.* propose an in-depth analysis of the distinguishing properties learned by the Gohr network, both through purely cryptanalytical means and through machine learning techniques [7]. In the second category, Baksi

et al. [4] focus on applying neural distinguishers to GIMLI, ASCON, KNOT and CHASKEY, and propose a different classification task, where the neural distinguisher is asked to predict which of t classes a pair belongs, where a class is determined by an input difference. Yadav *et al.* proposed to extend neural distinguishers for more rounds, by prepending a longer differential trail before the neural distinguisher [42]. In [22], the authors propose to use an SAT solver to look for better input differences and apply their results to various ciphers. At LNNS'21 [6], Bellini and Rossi compare neural and classical distinguishers for the ciphers TEA and RAIDEN. In a closer investigation of Gohr's results, [3] show that an identical neural distinguisher with depth-1, which we will refer to as GOHR DEPTH-1 NETWORK, reaches an almost identical accuracy of 78.3% (vs. 78.8% accuracy for depth-10) in round 6 of SPECK32/64, despite a much reduced parameter space and shorter training times.

```
Input(shape=(num_blocks*word_size*2,))
Reshape((2*num_blocks, word_size))
Permute((2,1))
# Initial convolution
Conv1D(f=32, k=1, 'L2')+BN+ReLU
# Residual blocks
Define 'shortcut'
for i in range(depth):
    Conv1D(f=32, k=3, 'L2')+BN+ReLU
    Conv1D(f=32, k=3, 'L2')+BN+ReLU
    Add 'shortcut'
    Define 'shortcut'
# Prediction head
Flatten()
Dense(64, 'L2')+BN+ReLU
Dense(64, 'L2')+BN+ReLU
Dense(1, 'sigmoid activation', 'L2')
```

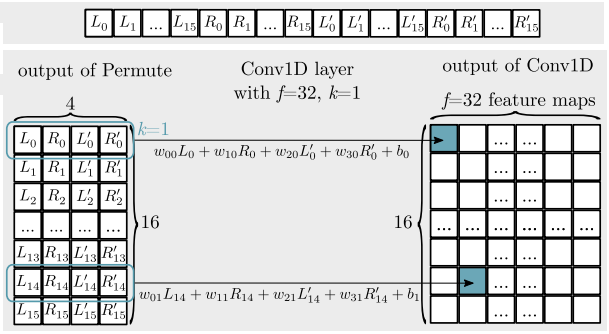

Fig. 4: Pseudo-code for the construction of Gohr's neural distinguisher of a certain `depth` and $f = 32$ filters in each convolutional layer. `Underlined` are design choices which are either heuristic or demonstrate expert knowledge, *i.e.,* dedicated structures of SPECK.

Figure 4 illustrates the construction of Gohr's neural distinguisher. We first discuss the dataset used for neural network training, then provide a discussion of more conventional elements, and finally discuss the particular design choices highlighted `underlined`.

The input to the network has 64-bit length for SPECK32/64 and is given as a ciphertext pair (C, C'), which consists of four words: $(L, R) = Enc_k(P_L, P_R)$ and $(L', R') = Enc_k((P_L, P_R) \oplus (0x0040, 0x0000))$ for the label NOT RANDOM (network output of 1). If the plaintext pair is randomly generated, the sample is labelled as RANDOM (network output of 0). The dataset consists of 10^7 training samples and 10^6 test samples. Approximately half of the samples are RANDOM. The neural network obtains ciphertext pairs (C, C') of the training dataset as input and is trained to predict the label. The output of the neural network is a single neuron `Dense(1)` with a `sigmoid` activation function. The sigmoid's 0 (1) value represents the RANDOM (NOT RANDOM) label prediction. The accuracy of

the distinguisher corresponds to the percentage of correctly predicted labels in the test dataset.
The network itself consists of input transformations, the convolutional blocks themselves, and a prediction head; this structure is reminiscent of the popular image recognition network ResNet [20], in which residual learning was introduced for deep neural networks. In residual learning, information can "skip" (or shortcut) several layers. This is implemented by adding the information of the `shortcut` to the output of a `block`. This enabled for the first time the training of networks with up to 152 layers depth. The residual connections still allow the information to propagate to subsequent layers to be trained, even if an earlier layer has stopped its learning progress.
The combination of a convolutional layer `Conv` with a kernel size of $k = 3$, followed by batch normalization `BN` and a `ReLU` activation function is conventionally used, for example in ResNet [20] or the batchnorm version of VGG networks [33,37]. ResNet and VGG are image recognition networks. In contrast to Gohr's neural distinguisher, they use `Conv2D` layers that move the kernel over the input in two directions `2D` to generate their output, called a *feature map*. In Gohr's network `Conv1D` layers are used, which are often encountered in time-series or text analysis [2]. `Conv1D` only moves the kernel of width k in a single direction (`1D`) over the input to generate one feature map. The number of filters f of the convolutional layer defines how many kernel functions, *i.e.,* weights w_{ij} and biases b_j are learned and how many feature maps $j = 0, \ldots, f-1$ are generated. These kernel functions are linear; nonlinearity is added through the subsequent activation functions, here `ReLU`, which is popular due to its simplicity and fast computation.

ResNet and VGG are winners of the *ImageNet Large Scale Visual Recognition Challenge* (ILSVRC) challenge. We note that Gohr's training dataset is about ten times larger in the total number of presented samples than, for example, ImageNet which has 1.3 million training images. Large training datasets can be necessary to avoid overfitting –learning the dataset by heart– for neural networks, which might also explain the heavy use of `L2` regularization parameters in Gohr's network.
We identify that expert knowledge of SPECK or heuristic choices in the construction of Gohr depth-1 network are reflected in: *i)* the input alterations of `Reshape` and `Permute`, since the *"...choice of the input channels is motivated by a desire to make the word-oriented structure of the cipher known to the network."* [16] and the `Conv1D(...k=1...)` for *"...learning of bit-sliced functions such as bitwise addition..."*[16], *ii)* a particular choice of the `L2` regularization parameter of 10^{-5} used throughout the network and *iii)* a cyclic learning rate for the Adam optimizer which we will discuss in more detail in our experimental section.

3 Methodology

3.1 The plaintext avalanche dataset generation

There are several types of datasets to perform a randomness test on a cipher. We focus on the task of distinguishing a given cipher from a random permutation, in particular, by observing a dataset that incorporates the avalanche effect of the cipher.

The *avalanche effect*, coined by Feistel [14], describes a desirable property of cryptographic encryption algorithms: that a slight change in the input (plaintext or key) creates a significant difference in the resulting ciphertext. The avalanche effect can be studied through avalanche datasets, in which the impact of a minimum input perturbation (flipping a single plaintext bit) on the encrypted output is investigated (e.g. [11,5]). Plaintext avalanche dataset was one of nine datasets to assess the candidates for AES competition [34], as well as the five finalist candidates [35], and their randomness was assessed using statistical tests similar to NIST STS [34]. The effect of small perturbations can also be studied through the Strict Avalanche Criterion (SAC) randomness test [11], which states that flipping a single bit in the input should result in a 50% probability of each output bit to be flipped.

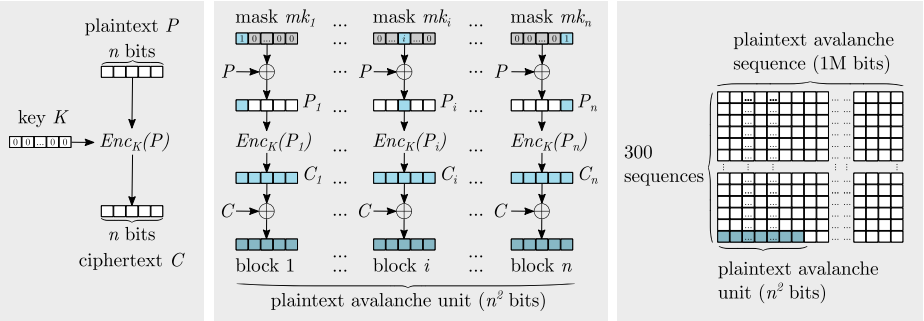

Fig. 5: Illustration of the avalanche dataset generation.

A plaintext avalanche dataset is generated with the following steps as illustrated in Fig. 5. Let Enc_K be an encryption black box with key K: *1)* Let K be the key with all zeros and P a random plaintext. Let C be the ciphertext corresponding to P, that is, $C = Enc_K(P)$. We call one output of the encryption a *block* with n bits. *2)* Define $mask_i$ as the bitstring with 1 at position i and zeros otherwise. Let $P_i = P \oplus mask_i$, and $C_i = Enc_K(P_i)$ be the corresponding ciphertext. *3)* An *avalanche unit* from plaintext P is the concatenated bit string $C \oplus C_1||...||C \oplus C_n$ of a total of n blocks. The total bit length of an avalanche unit is n^2. *4)* An *avalanche sequence* is the concatenation of several avalanche units, one for each mask. The total bit length is ℓ. *5)* In this work, we use avalanche datasets composed 300 avalanche sequences, each sequence being

around 1M bits. This is in line with the parameters of NIST STS. The plaintext avalanche dataset generation was implemented using the NumPy library and a Python implementation of each cipher, which is available in our repository [39].

For example, in SPECK32/64, an avalanche unit contains 1,024 bits, corresponding to 32 blocks, with each block of bit length 32. We concatenate 1000 avalanche units into one avalanche sequence. That is, there are 1,024,000 bits in one avalanche sequence and 307,200,000 bits in this avalanche dataset. The parameters of the avalanche dataset of each cipher are shown in Table 1.

cipher	rounds	block size	key size	avalanche unit size	avalanche units per avalanche sequence	avalanche sequence size
SPECK32/64	22	32	64	1024	1000	1024000
SPECK64/128	27	64	128	4096	250	1024000
SPECK96/144	29	96	144	9216	110	1013750
SPECK128/128	32	128	128	16384	64	1048576
AES-128	10	128	128	16384	64	1048576

Table 1: Avalanche dataset parameters of different ciphers. All sizes are in bits.

3.2 The NIST Statistical Test Suite

We use the current standard NIST SP 800-22 [31], also known as the NIST Statistical Test Suite (NIST STS), to perform the cryptographic randomness tests. In addition to this collection, other similar test suites are available, such as DieHarder [10], TestU01 [25] or ENT [41]. In fact, there are *"an infinite number of possible statistical tests, each assessing the presence or absence of a pattern which, if detected, would indicate that the sequence is nonrandom"* [31]; Such statistical tests can be generated automatically, for example through evolutionary algorithms [36]. The abundance of existing tests leads us to focus, in this work, on the widely accepted standard NIST STS. It is a collection of 15 core statistical tests as listed in Table 2, and with different parameters, a total of 188 statistical tests are conducted. To be noted, although the Lempel-Ziv Compression test is stated in [34], it is not implemented in NIST STS. Referring to the parameters used in [34,35], we generated 300 plaintext avalanche sequences with each ≈1M bits in the dataset, and used an α value of 0.01. The results are discussed in Section 4.3.

3.3 Technical implementation of NNBits

NNBits ensemble is a deep learning ensemble analysis tool for bit-profiling of cryptographic (pseudo) random bit sequences that includes dependencies between different bits (Fig. 1). Here, we first give a quick introduction to ensembling methods in machine learning and then present the technical implementation of NNBits ensemble.

statistical test	test ID	statistical test	test ID	statistical test	test ID
Monobit	1	Rank	7	Approximate Entropy	159
Block Frequency	2	Spectral DFT	8	Random Excursions	160-167
Cusum	3-4	Aperiodic Templates	9-156	Random Excursions Variant	168-185
Runs	5	Periodic Template	157	Serial	186-187
Long Runs of Ones	6	Universal Statistical	158	Linear Complexity	188

Table 2: The 188 statistical tests in the NIST STS

Ensembling or *ensemble learning* is a widely used technique in machine learning [17,2]. An ensemble is a "group of predictors" [2]. The core idea of ensemble methods is that this group of predictors is *diverse*. Diversity is achieved when single predictors make different kinds of errors. There are many methods to create diversity, for example on the *data level*: here, the same algorithm is used for every predictor, however they are trained on different random subsets of the training data. The best-known method is "bagging" (or bootstrap aggregating) by Breimann [9]; the *model level*: here, completely different prediction algorithms are used in combination, e.g. a neural network, a random forest classifier and a logistic regression model in combination; or the *hyperparameter level*: if, for example, different loss functions are chosen during training. This approach is also taken in [29].

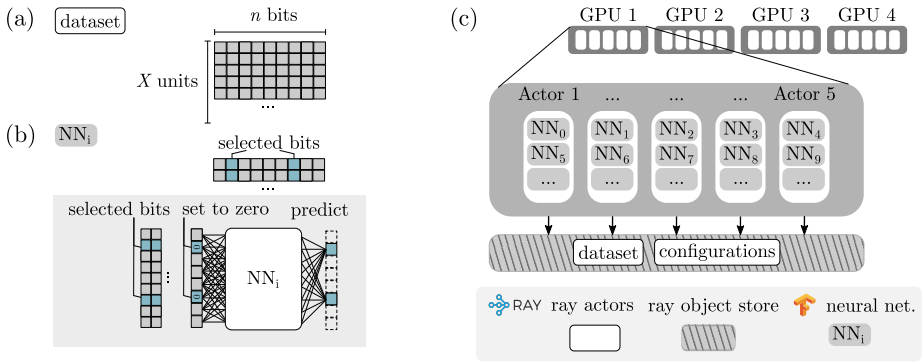

Fig. 6: Technical implementation of the deep learning ensemble distinguisher using the Python packages Ray and TensorFlow on a multi-GPU server.

Figure 6 illustrates the creation of our NNBits ensemble, consisting of a group of N neural networks which are diversified on the data level. Each member NN_i of the group predicts a certain subset i of the n bits in the dataset. At the input side of the network, this subset of bits will be set to zero (Fig. 6b)). The technical implementation uses state-of-the-art parallelization modules that allow for high performance. This allows us to tackle a demanding scenario in Section 4: for example, we identify weak bits in the avalanche dataset of AES-128 for which

a single avalanche unit contains $n = 16384$ bits. The work we present in this manuscript uses Gohr's neural distinguisher (Section 2.3) and extended versions of it. However, NNBits in general allows the user to include any TensorFlow network of their choice.

Figure 6c) shows the technical implementation of our NNBits ensemble. NNBits ensemble uses the Python packages Ray [38,28] and TensorFlow [1] on a multi-GPU server. Ray relies on stateful *actors* to parallelize machine learning tasks. These actors share access to the data, which only needs to be read from the disk once; this is significant, as loading millions of avalanche units for ciphers larger than SPECK32/64 can take several minutes. Since the initialization of neural networks and the manipulation of the data sets are computationally expensive, a reasonable total number N of neural networks in the ensemble is $N \approx 100$.

The source code of NNBits ensemble, as well as a demonstration and instructions to adjust the parameters for different GPU settings, are available in our repository [39].

4 Experimental results and analysis

We have conducted three experiments using the previously introduced methodologies.

	1 Explanation of Gohr's accuracy (Section 4.1)	**2 Generalization of Gohr's distinguisher** (Section 4.2)	**3 Comparison of NNBits and NIST STS** (Section 4.3)
cipher	SPECK32/64	SPECK32/64	SPECK32/64 SPECK64/128 SPECK96/144 SPECK128/128 AES-128
inputs	ciphertext pairs	ciphertext pairs	avalanche units
labels	RANDOM/NOT RANDOM	RANDOM/NOT RANDOM	S1 None S2 RANDOM/NOT RANDOM
samples	10^7 training 10^6 validation	10^7 training 10^6 validation	S1 $\leq 300 \times 10^3$ in total S2 $\leq 3.65 \times 10^6$ in total

Table 3: Overview over our experimental settings.

First, we provide a possible explanation for the accuracy of Gohr depth-1 network using a bit-by-bit analysis with our NNBits ensemble (Section 4.1). Then, we generalize aspects of Gohr depth-1 network to extend the range of applications and obtain the Generalized network (Section 4.2). We can then use Generalized network in combination with NNBits to analyze the

avalanche datasets of SPECK32/64 up to SPECK128/128, as well as AES-128 in Section 4.3 and present bit profiles of SPECK32/64 up to round 7 (of 22) and of AES-128 up to round 2 (of 10) (Section 4.3).
Table 3 summarizes the experimental scenarios and highlights that the datasets of experiment 1 and 2 are similar to Gohr's original dataset (millions of available training inputs with labels), while the setting of the NIST STS comparison in 3-S1 is different and difficult from a machine learning perspective, due to the absence of a labeled dataset and a restricted number of samples.

Our neural network experiments are performed on an Nvidia DGX-A100 server equipped with four A100 Ampere microarchitecture GPUs. Each A100 GPU provides 40536 MiB computational memory.

4.1 Explanation of the accuracy of Gohr's neural distinguisher

In the following experiment, we show that 78.1% of the 78.3% [3, table 3] accuracy obtained by Gohr depth-1 network can be understood in terms of correct predictions of individual bits. We show this by first training one neural network per bit using an NNBits and then constructing a distinguisher from these single bit predictors. Based on the findings, we propose a strategy for future improvements of Gohr depth-1 network.

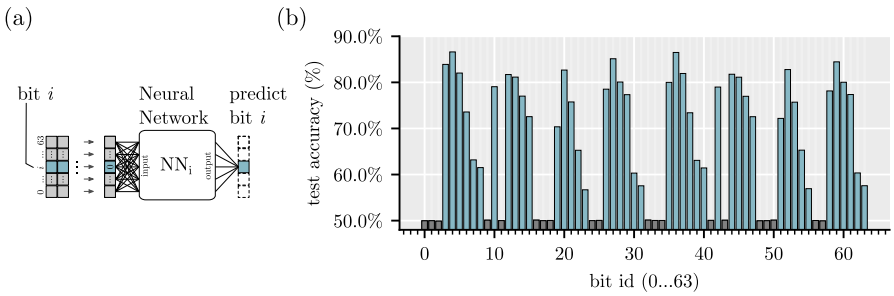

Fig. 7: Bit-by-bit accuracy for ciphertext pairs (C, C') with chosen plaintext difference $(0x0040, 0x0000)$ of SPECK32/64. a) The neural network NN_i is trained to predict bit i of (C, C'). The dataset is generated by setting bit i of (C, C') to zero at the input. b) The test accuracies of $NN_0 \dots NN_{63}$ on 10^6 previously unseen ciphertext pairs (C, C').

Training procedure. We use NNBits to train $N = 64$ neural networks, one to predict each bit of (C, C') of Gohr's original dataset in round 6 of SPECK32/64. Owing to the parallelization provided by NNBits we can train 16 neural networks in parallel on our server, resulting in a total experimental runtime of 4.5 hours. Figure 7a) illustrates that for the training of our neural network NN_i, bit i is set to zero at the input. The task of the particular neural network NN_i is

to predict *bit i*. Therefore, the networks are trained on the NOT RANDOM ciphertext pairs (C, C') subset of Gohr's original dataset (presented in Section 2.3). We use $X_{\text{train}} = 5 \times 10^6$ NOT RANDOM training samples and train each network for 200 epochs. The neural networks NN_i are GOHR DEPTH-1 NETWORK with depth of 1 and 32 filters in each convolutional layer as provided in Gohr's GitHub repository [15].

Single bit results. Figure 7b) shows the test accuracies of each network NN_i. We use $X_{\text{test}} = 5 \times 10^5$ NOT RANDOM samples for testing. A higher test accuracy of NN_i means that Gohr's network is able to predict the value of bit i, given the values of the remaining 63 bits. Accuracies around 50% mean that the network is not better at predicting bit i than a random guess. We observe a pattern of the test accuracies in the first 32 bits which repeats itself over the next 32 bits (small variations in the resulting accuracies are expected in neural network training). Within the first 32 bits, ten bits (0, 1, 2, 9, 11, 16, 17, 18, 24, 25) cannot be predicted by Gohr's network. The highest test accuracy is achieved on bit 4 with 86.6%.

In the following we address two questions: *1)* Can we understand the overall distinguishing accuracy, that is 78.3%, of Gohr's network in terms of such single-bit predictions? In other words: Can we construct a distinguisher using the outcome of these single-bit predictions? *2)* Given we could construct a distinguisher from single-bit predictions, what is a good strategy to improve the accuracy achieved by Gohr even further?

Construct a distinguisher $\boxed{E}$. First, we address question *1)* by constructing a distinguisher from our already trained networks as follows: Our dataset is now identical to Gohr's original dataset, *i.e.,* it contains ciphertext pairs (C, C') with both labels, RANDOM and NOT RANDOM. As in the previous experiment each of our already trained 64 neural networks NN_i predicts one bit. The respective bit is set to zero at the input. Each bit prediction is evaluated in terms of its *correctness*: If the bit was correctly predicted, we save a 1, otherwise a 0. The information about the *correctness of the predictions* is then passed to an MLP –identical to Gohr's neural prediction head. This MLP is then trained to output the label RANDOM or NOT RANDOM for (C, C').

	classical [16, Table 2]	ensemble distinguisher $\boxed{\text{E}}$	Depth-1 [3, Table 2]	Depth-10 [16, Table 2]
accuracy	75.8%	**78.1%**	**78.3%**	78.8%

Table 4: Comparison of the distinguisher accuracies for round 6 for SPECK32/64.

Interpretation. Table 4 shows the resulting accuracies for the ensemble distinguisher E compared to Gohr networks and a classical differential distinguisher. E reaches 78.1% accuracy, only 0.3% below the accuracy of Gohr's original depth-1 network [3]. Note that E does not even make use of the values of the bits, but only of the information about *prediction correctness*. This experiment shows that the largest part of the accuracy of Gohr's network can be understood in terms of *the correctness of single bit predictions* and combining them to a RANDOM/NOT RANDOM decision with an MLP prediction head (identical to the prediction head used in Gohr's distinguisher). This means that one possible way Gohr's network can be understood is as essentially learning the underlying Boolean functions to predict single bits, and then evaluating the number of correct predictions. While we have here discussed the results for round 6 of SPECK32/64 in detail, similar results are obtained when comparing the ensemble distinguisher E with Gohr's depth-1 network in [3, Table 2] for round 5 (92.2% vs 92.7%) and round 7 (60.1% vs 60.8%).

Improvement strategies for Gohr's distinguisher. Now, we address question *2)*. We formulate an improvement strategy for Gohr's distinguisher, given that we can understand the largest part of the accuracy of Gohr's distinguisher in terms of the correctness of single-bit predictions. Using Fig. 7b) we can focus on improving the bits with low accuracy, highlighted in grey.
As noted by [7], differential-linear cryptanalysis [24] seems to be a good explanation for Gohr's classifiers' accuracy. It consists in studying linear relations between bits of the difference δ_t and δ_r, respectively at round t and final round round r. This is more formally expressed in terms of linear masks m_t, m_r: the bias of the bit $b_{m_t,m_r} = \bigoplus_{i=0}^{n-1}(\delta_t \wedge m_t)_i \oplus (\delta_r \wedge m_r)_i$ is studied. With the input difference chosen by Gohr, the difference bits at rounds 1 to 5 are very biased, so it is expected that $\bigoplus_{i=0}^{n-1}(\delta_r \wedge m_r)_i$ would be biased as well for small values of r, allowing better predictions of the bits involved. To improve the accuracies on the bits in grey, two challenges must be overcome: finding potential relevant output masks m_r which are not already used by the classifier, and injecting this additional information into our classifiers.
Note that it is not sufficient to *generally* improve the accuracy of single bit predictions, but the improvements need to be aimed at the particular cases where the distinguisher decides wrongly.

4.2 Generalization of Gohr's neural distinguisher for avalanche datasets

The experiment in the previous section was aimed at gaining more understanding about GOHR DEPTH-1 NETWORK. In the following experiments, we aim to extend the range of application of GOHR DEPTH-1 NETWORK. Here, we eliminate two specific design choices in GOHR DEPTH-1 NETWORK which either relate directly to SPECK or may only work for a specific dataset. The result is a GENERALIZED NETWORK, which we apply to larger datasets in the following sections.

A neural network may perform extremely well in a given problem but completely fail at a seemingly similar one. To generalize a machine learning model it is essential to remove application specific choices. In GOHR DEPTH-1 NETWORK we can identify the following application-specific neural network design choices, as discussed in more detail in Section 2.3: *1) input alterations*, *2) cyclic learning rate*– [16] uses the ADAM optimizer in combination with a cyclic learning rate that varies between 0.002 and 0.0001 over 10 epochs, and *3) kernel regularization*– with a particular L2-regularization parameter.

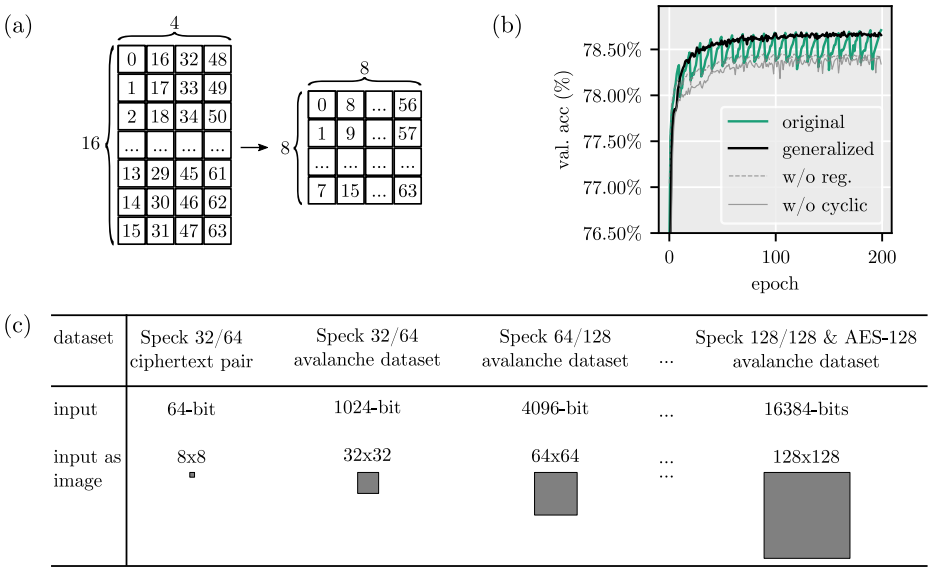

Fig. 8: a) Instead of a reshaping into a 4×16 structure, the generalized network shapes the input-bit sequence into a square shape. b) Training curve comparison of GOHR DEPTH-1 NETWORK (4×16 reshape; cyclic learning rate) and GENERALIZED NETWORK (square-shaped input; AMSGRAD optimizer). c) Representation of the reshaping of different input bit sequences. E.g. the avalanche dataset of SPECK128/128 has $128 \times 128 = 16384$ input bits, which are reshaped into a 128×128-image by the generalized network.

Figure 8a) and b) illustrate the GENERALIZED NETWORK which addresses *1) and 2)* as follows: *1) input reshaping*– We shape the input into a more generic square form, which allows *i)* an easy extension of the distinguisher onto e.g. the avalanche datasets (see Fig. 8c)) and *ii)* a fairer potential comparison with state-of-the-art visual recognition neural networks. The reshaping of the input into a word-like 4x16 bit in GOHR DEPTH-1 NETWORK corresponds to an information gain, so that it can start training with a lower number of possible filters. To learn the same information as GOHR DEPTH-1 NETWORK the number of filters for the convolutional layers is increased by a factor of four in GENERALIZED NETWORK

(32 vs 128). The increased number of filters leads to longer training time per epoch (9 s vs 13 s).
2) AMSGrad algorithm instead of cyclic learning rate– while ADAM is one of the most advanced optimizers, it has been observed that it fails to converge to an optimal solution [30]. This can make it necessary to manually find an optimal learning rate setting to train the neural network. Such a manual choice has a higher likelihood to fail in a new setting. To mitigate the convergence issue, Reddi *et al.* introduce the AMSGRAD algorithm in *"On the Convergence of Adam and Beyond"* [30] at ICLR 2018.

We have trained both, GOHR DEPTH-1 NETWORK and the GENERALIZED NETWORK on Gohr's original dataset (Section 2.3) for round 6 of SPECK32/64 (Fig. 8b)). The cyclic learning rate leads to the "dents" in the *original* graph of GOHR DEPTH-1 NETWORK. If the cyclic learning rate is removed from the training of the GOHR DEPTH-1 NETWORK (*w/o cyclic* in Fig. 8b)) the training results in lower final accuracy. The GENERALIZED NETWORK uses the ADAM optimizer with its standard settings together with AMSGRAD instead of the cyclic learning rate. After 200 epochs both networks converge to the same accuracy. Also in round 5 and round 7 the GENERALIZED NETWORK reaches comparable accuracy to the GOHR DEPTH-1 NETWORK one [3, Table 2] with 92.8% vs 92.7%, respectively 61.0% vs 60.8%.
The shaping into a more generic square form, as well as the removal of the specific cyclic learning rate allow us to easily apply the GENERALIZED NETWORK to avalanche datasets. For example, the avalanche unit of AES-128 contains 16,384 bits, which are now reshaped into a 128×128 "image", as illustrated in Fig. 8c).

4.3 Comparison of NNBITS with the NIST STS

Here, we compare the NNBITS ensemble with the NIST STS. We further show that the NNBITS ensemble analysis can provide additional insights: for example, Gohr's input difference $(0x0040, 0x0000)$ is inferred from the bit-analysis of SPECK32/64, and the round 2 bit-analysis of the AES-128 avalanche dataset is explained by multiplication in the Galois field of AES.
Settings $\boxed{S1}$. The NIST STS operates in a setting which is difficult from a machine learning perspective: We are only given access to a limited number of bits and based on this bit-sequence only, we have to decide if it is generated from an RNG or not. Here, we assume that we may not use any information on the cipher which has potentially generated the sequence, therefore we have to train and test our neural network ensemble on this *limited size* dataset and *without a labelled dataset*. Even for the cipher SPECK32/64 with the smallest avalanche units of 1024 bits each, we only have around 300k avalanche units available for testing and training.
Settings $\boxed{S2}$. Gohr's original training dataset contains millions of training sequences. For completeness, we also train GENERALIZED NETWORK in a setting which is simpler for machine learning, in short: On a labelled dataset and with

a larger amount of data. The details are given in Appendix A.
Here, we first provide a short overview over the results in the different settings, and then provide the detailed results of NIST STS with NNBits, as well as the bit-profiles obtained with NNBits.

	random from round			**time spent per round**			**dataset size**	
cipher	NIST STS	S1	S2	NIST STS	S1	S2	NIST STS & S1	S2
SPECK32/64	6	8	8	≤30 min	≤5 min	≤4 min	≈300 Mbits	≈4 Gbits
SPECK64/128	7	8	8	≤30 min	≤7 min	≤12 min	≈300 Mbits	≈15 Gbits
SPECK96/144	8	8	9	≤30 min	≤10 min	≤24 min	≈300 Mbits	≈34 Gbits
SPECK128/128	9	10	10	≤30 min	≤20 min	≤17 min	≈300 Mbits	≈27 Gbits
AES-128	3	3	3	≤30 min	≤20 min	≤17 min	≈300 Mbits	≈27 Gbits

*: See Table 8 for details. Please note that NIST STS and S1 use a limited, unlabeled dataset, whereas S2 uses an –in comparison– unlimited, labeled dataset. S1 provides bit-profiling while NIST STS and S2 do not. As described in Section 3.3 the NNBits ensemble relies on GPU parallelization on a server and the runtime will depend on the available resources. This runtimes apply to our particular server.

Table 5: Comparison of the NIST STS and our works.

Discussion. Table 5 shows a summary of the randomness tests performed with the NIST STS and our NNBits S1. The runtimes are given as an indication, even though they are not directly comparable, since NNBits and Generalized network use highly parallelized GPU implementations. The comparison *Random from round* shows that the deep learning based tests can gain advantages over the NIST STS in most SPECK-cases for S1 and all SPECK-cases for S2. We conclude that even in a low data setting and without label, S1, the NNBits ensemble can perform well, either as good as the NIST STS or better.
To gain an additional distinguisher comparison to the NIST STS, we have implemented the avalanche tests that were used to analyze Xoodoo [12] for SPECK32/64. The avalanche dependence goes to 32, avalanche weight goes to ≈16, and avalanche entropy goes to ≈32 at round 6, which means all three avalanche criteria are met at round 6 and aligns with our NIST STS results.

In the following sections we will give details on the results obtained with the NIST STS (Section 4.3) and NNBits (Section 4.3). In particular, we will show the bit-profiles generated by the NNBits ensemble and provide a detailed analysis of the same.

Details for the NIST STS experiment To make a fair comparison between NNBits and NIST STS, we use the same plaintext avalanche dataset as introduced in Section 3.1. For the target significance level of $\alpha = 0.01$, at least 292 sequences among all the 300 sequences should successfully pass the examination to pass a particular test. We present a summary of the results of the tests in Table 5. In the table, when we say that an underlying primitive is random at round r, we mean the underlying primitive passes more than 186 of the 188 tests introduced in Section 3.2 and has no more significant variation when increasing

the round number. Figure 12 shows the randomness evaluation in each round by NIST STS tools corresponding to SPECK32/64, SPECK64/128, SPECK96/144, SPECK128/128, and AES-128 respectively.
The total time to execute all tests was approximately three days. All NIST STS experiments were carried out on a server with 112 Intel(R) Xeon(R) Platinum 8280 CPUs, each with 28-cores, 2.70GHz, 1152G RAM.

Details for the NNBits experiment Here, we first describe the experiment to produce the results shown in Table 5. Then we analyze the underlying data of SPECK32/64 and AES-128 in more detail and show that the NNBits ensemble experiments can provide useful cryptanalytic insights.
The NIST STS uses a dataset of 300 Mbits. To make a fair comparison between NNBits and NIST STS, we use here the same plaintext avalanche dataset as introduced in Section 3.1. Therefore, there is only a very limited number of avalanche units for training and testing of the neural network ensemble. Also, we don't assume that we have access to any kind of RANDOM/NOT RANDOM labeled dataset. This results in the settings S1, which are disadvantageous for machine learning.
About half of the avalanche units contained in the 300 Mbits-long dataset are used for training. The detailed settings for the training as well as the detailed test results are shown in the appendix in Table 7. An NNBits ensemble with $N = 100$ neural networks of type Generalized network is constructed as explained in Section 3.3. To cover the whole range of bits in the avalanche units (see Table 7) each neural network predicts around 6% of the bits in an avalanche unit. For example, for SPECK32/64 the avalanche unit contains 1024 bits and a single neural network predicts 63 randomly chosen bits. In the following we present the detailed bit profiles of SPECK32/64, while we discuss the details of AES-128 in the appendix (Appendix D).

Bit profiles of SPECK32/64. Table 5 shows that our NNBits ensemble can distinguish SPECK32/64 avalanche data up to round seven from randomly generated data. In the following we gain more insights from the analysis used for Table 5.

Figure 9 shows the bit-by-bit test accuracy for round 1 to round 7 of the avalanche dataset of SPECK32/64. We observe a region of weak bits around bit 715 through all rounds. This region is related to plaintext differences of $(0x0040, 0x0000)$: in the avalanche sequence, bits $32i \dots 32(i+1)-1$ correspond to the XOR of the original ciphertext with the ciphertext where bit i has been perturbed. Consequently, the perturbation of bit $i = 22$ corresponds to bits $704 \dots 735$ in the avalanche sequence. The perturbation of this bit in terms of a plaintext difference is $(1 \ll 22) = (0x0040, 0x0000)$. Note that $(0x0040, 0x0000)$ is the chosen plaintext input difference used in Gohr [16] for SPECK32/64.

Figure 10 provides a more detailed view for round 7 of SPECK32/64 from Fig. 9. We observe that in round 7 two bits (716 and 732) remain weak, *i.e.,* can be predicted with an accuracy significantly above a random guess.

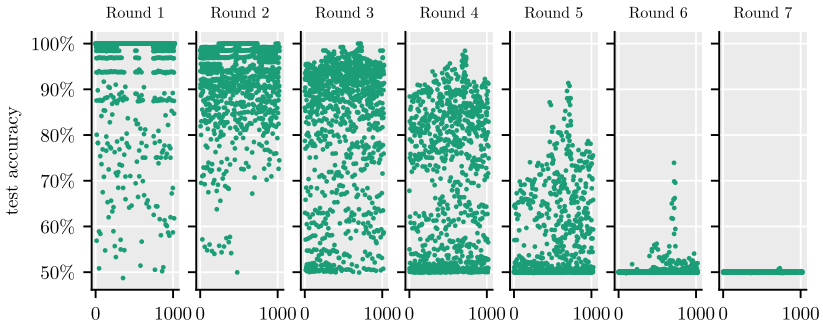

Fig. 9: Mean ensemble prediction accuracy for each bit in SPECK32/64 round 1 to round 7. A zoom into round 7 is provided in Fig. 10.

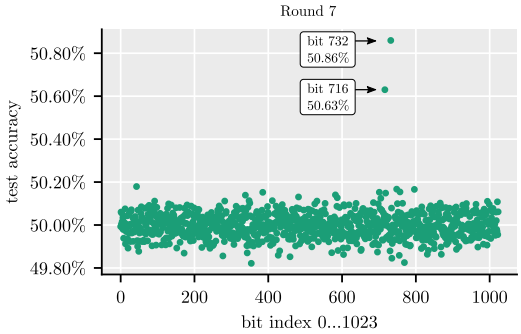

Fig. 10: Detailed view of round 7 of Fig. 9, which demonstrates that bit 716 and 732 are weak and have accuracies significantly above the random guess limit.

To gain further understanding, we used NNBits to perform a targeted analysis on these two bits. In the targeted analysis, we force one or both of these bits to be predicted (instead of randomly choosing the predicted bits among the 1024 avalanche bits). To do so, we use an ensemble of $N = 500$ neural networks, each predicting bit 732; trained on $n_{\text{train}} = 20 \times 1024$ sequences and tested on $n_{\text{test}} = 500 \times 1024$ sequences. Then we analyze the Pearson correlation coefficient of the obtained accuracies with the presence of the remaining bits at the network input. This analysis shows a strong correlation of a high accuracy $A_{732} \gg 50\%$ with the presence of bit 716 at the input of the neural network. Doing the same analysis for bit 716 shows that bit 732 needs to be present at the input to predict bit 732 with an accuracy $A_{732} \gg 50\%$. In conclusion, we find that bit 716 needs to be present at the input to predict bit 732 and vice versa. We can explain this strong correlation as follows.

Bits 704 to 735 correspond to Gohr's input difference $(0x0040, 0x0000)$. With this input difference, we can observe empirically that in round 7, bits 12 and 28 of the output difference (*i.e.,* bits 716 and 732 of the avalanche dataset) are

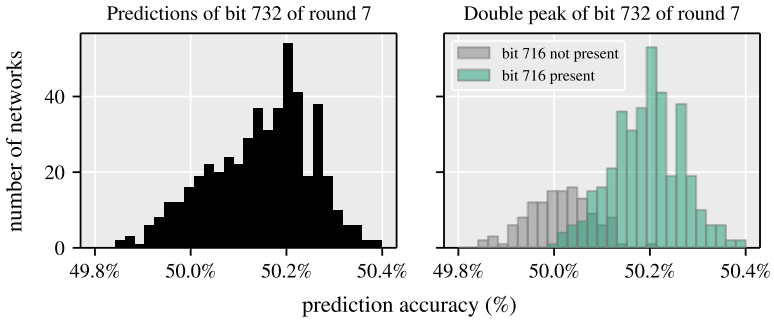

Fig. 11: Correlation analysis of bit 732. Left hand side: Histogram of the prediction accuracies of the single ensemble members. Right hand side: Same underlying data as on the left hand side, however, divided into two histograms. Grey - bit 716 is not present at the input of these neural networks. Green - bit 716 is present at the input of these neural networks.

balanced (*i.e.*, they follow a uniform distribution). On the other hand, at round 6, bit 30 of the output difference (*i.e.,* bit 733 of the 6 rounds avalanche dataset) is biased: it is set to 1 with probability 0.544. By construction, this bit is the XOR of bits 716 and 732 of the 7 rounds avalanche dataset; it follows that the probability for bits 716 and 732 to be different is 0.544. Therefore, losing the information provided by either of these bits harms the ability to predict the other one, in agreement with our finding with the NNBits ensemble.

5 Conclusion

In conclusion, in this work, we introduce a deep learning ensemble (NNBits) for bit-profiling of cryptographic (pseudo) random bit sequences with the following main results.

Neural network explainability (Section 4.1). Improvements of the explainability of neural networks are fundamental not only to understand the additional knowledge which has been learned by the neural networks, but also for their future improvement. We demonstrate how to use NNBits to explain parts of the seminal work of Gohr [16]: Gohr's depth-1 neural distinguisher reaches a test accuracy of 78.3% in round 6 for SPECK32/64 [3]. Using the bit-level information provided by NNBits we can partially explain the accuracy obtained by Gohr (78.1% vs 78.3%). This is achieved by constructing a distinguisher which only uses the information about correct or incorrect predictions on the single bit level.

Deep-learning based statistical test (Sections 4.2 and 4.3). We also generalize two heuristic aspects in the construction of Gohr's network: *i)* the particular input structure, which reflects expert knowledge of SPECK32/64, as well

<table>
<tr><th>Explanation of Gohr's accuracy (Section 4.1)</th><th>Generalization of Gohr's distinguisher (Section 4.2)</th><th>Comparison of NNBits and NIST STS (Section 4.3)</th></tr>
<tr><td rowspan="2">Explained
78.1% vs. 78.3%
accuracy</td><td rowspan="2">Generalized
input reshaping
and
cyclic learning rate</td><td>Random from round for NIST STS and NNBits (S1, S2):
<table><tr><th>cipher</th><th>NIST STS</th><th>S1</th><th>S2</th></tr><tr><td>SPECK32/64</td><td>6</td><td>8</td><td>8</td></tr><tr><td>SPECK64/128</td><td>7</td><td>8</td><td>8</td></tr><tr><td>SPECK96/144</td><td>8</td><td>8</td><td>9</td></tr><tr><td>SPECK128/128</td><td>9</td><td>10</td><td>10</td></tr><tr><td>AES-128</td><td>3</td><td>3</td><td>3</td></tr></table>and provided NNBits bit-profiling of SPECK32/64 and AES-128.</td></tr>
</table>

as *ii)* the cyclic learning rate (Section 4.2). In combination with NNBits the resulting Generalized network can be applied as a statistical test on the plaintext avalanche datasets of SPECK32/64, SPECK64/128, SPECK96/144, SPECK128/128, and AES-128. We conclude that the NNBits ensemble performs as well as the NIST STS or better (Section 4.3).

Bit-by-bit profiling (Section 4.3) . We demonstrate cryptanalytic insights that result from bit-level profiling with NNBits, for example, we show how to infer the strong input difference $(0x0040, 0x0000)$ for SPECK32/64 (Section 4.3) or infer a signature of the multiplication in the Galois field of AES-128 (Appendix D).

NNBits is available under [39] and while we mainly focused on Gohr-like networks, it can be used with any TensorFlow network of interest. NNBits is mainly aimed at a server environment with GPU availability.

A Details for the Generalized network experiment S2

We apply Generalized network as a distinguisher to the avalanche datasets of SPECK32/64, SPECK64/128, SPECK96/144, SPECK128/128 and AES-128 in the settings S2, which are advantageous for machine learning. Table 6 summarizes the experimental settings for each cipher. We generate X bit sequences of the length of avalanche units for the respective cipher. A randomly chosen half of the inputs X have the label $Y = 0$ and contains random data. The other half of the inputs has the label $Y = 1$ and contains avalanche units of a cipher, that is, not random data. A Generalized network, as presented in Section 4.2 is trained on a subset X_{train} to predict the labels Y_{train} for 10 epochs. Subsequently, previously unseen data X_{test} is used to evaluate the accuracy A of the distinguisher.
Table 6 summarizes the avalanche unit bit sizes, the number of avalanche units for training X_{train} and testing X_{test}, as well as the distinguisher's accuracy A for relevant rounds. The accuracy is given as the mean and standard deviation over four runs of the previously described experiment.

cipher	**unit length**	X_{train}	X_{test}	**round**	**accuracy** A
SPECK32/64	1024	3.5×10^6	150×10^3	1/22	(100.00±0.00)%
				...	...
				6/22	(82.70±0.22)%
				7/22	(51.38±0.02)%
				8/22	(50.01±0.16)%
SPECK64/128	4096	3.5×10^6	150×10^3	1/27	(100.00±0.00)%
				...	...
				7/27	(61.27±0.18)%
				8/27	(50.06±0.15)%
SPECK96/144	9216	3.5×10^6	150×10^3	1/29	(100.00±0.00)%
				...	...
				8/29	(55.29±1.25)%
				9/29	(49.99±0.03)%
SPECK128/128	16384	1.5×10^6	150×10^3	1/32	(100.00±0.00)%
				...	...
				9/32	(84.20±0.39)%
				10/32	(50.05±0.09)%
AES-128	16384	1.5×10^6	150×10^3	1/10	(100.00±0.00)%
				2/10	(99.99±0.01)%
				3/10	(49.98±0.07)%

Table 6: Accuracies A for distinguishing avalanche units of the respective cipher from random data. **Bold** is the first round for which the distinguisher offers no advantage over a random-guess.

B Details of NIST results

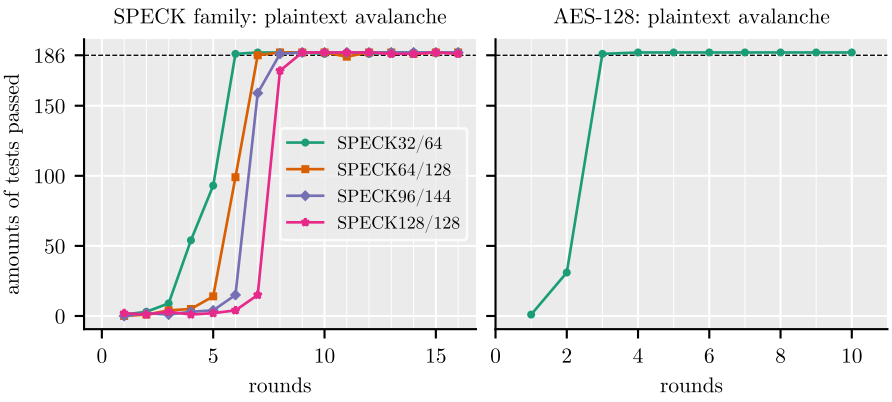

Fig. 12: Randomness evaluation of rounds by NIST STS.

C Details of NNBits results

cipher	single aval. unit (bits)	aval. units in 300 Mbits	aval. units for training	aval. units for testing
SPECK32/64	1024	292968	147456	145512
SPECK64/128	4096	73242	36864	36378
SPECK96/144	9216	32552	16384	16168
SPECK128/128	16384	18310	12288	6022
AES-128	16384	18310	12288	6022

Table 7: Summary of the number of training and testing avalanche units presented to the neural network ensemble for each cipher. The detailed training outcomes for each round are shown in Table 8.

cipher	r	epochs	#(training)	#(testing)	runtime	acc (%)	p value	random
speck32	1	10	147456	145512	1.0	100.0	0	not random
	2	10	147456	145512	1.0	100.0	0	not random
	3	10	147456	145512	1.3	100.0	0	not random
	4	10	147456	145512	1.3	99.15	0	not random
	5	10	147456	145512	1.3	95.01	0	not random
	6	10	147456	145512	1.3	79.17	0	not random
	7	10	147456	145512	1.3	51.35	9.7e-25	not random
	8	10	147456	145512	3.8	50.17	0.19	random
	9	10	147456	145512	4.2	50.23	0.087	random
	22	10	147456	145512	2.7	50.27	0.039	random
speck64	1	40	36864	145512	1.5	100.0	0	not random
	2	40	36864	145512	1.5	100.0	0	not random
	3	40	36864	145512	1.5	100.0	0	not random
	4	40	36864	145512	1.4	99.57	0	not random
	5	40	36864	145512	1.1	95.54	0	not random
	6	40	36864	145512	1.1	77.33	0	not random
	7	40	36864	145512	3.8	50.96	3e-13	not random
	8	40	36864	145512	6.4	50.2	0.12	random
	9	40	36864	145512	6.2	50.18	0.17	random
	10	40	36864	145512	6.2	50.18	0.17	random
	27	40	36864	145512	6.0	50.26	0.051	random
speck96	1	90	16384	145512	1.8	100.0	0	not random
	2	90	16384	145512	1.8	100.0	0	not random
	3	90	16384	145512	1.2	100.0	0	not random
	4	90	16384	145512	1.2	100.0	0	not random
	5	90	16384	145512	1.8	99.62	0	not random
	6	90	16384	145512	1.2	94.82	0	not random
	7	90	16384	145512	1.7	73.47	0	not random
	8	90	16384	145512	3.9	50.71	6e-08	not random
	9	90	16384	145512	9.3	50.22	0.097	random
	10	90	16384	145512	9.7	50.21	0.11	random
	11	90	16384	145512	9.6	50.22	0.092	random
	29	90	16384	145512	9.8	50.2	0.13	random
speck128	1	120	12288	145512	1.3	100.0	0	not random
	2	120	12288	145512	2.1	100.0	0	not random
	3	120	12288	145512	2.2	100.0	0	not random
	4	120	12288	145512	1.2	100.0	0	not random
	5	120	12288	145512	1.5	100.0	0	not random
	6	120	12288	145512	1.3	99.91	0	not random
	7	120	12288	145512	2.4	99.21	0	not random
	8	120	12288	145512	2.2	90.43	0	not random
	9	120	12288	145512	2.2	63.58	0	not random
	10	120	12288	145512	16.3	50.25	0.057	random
	11	120	12288	145512	16.3	50.25	0.055	random
	32	120	12288	145512	16.0	50.28	0.033	random

Table 8: Detailed results for the NNBits analysis presented in Table 5. For each round r the training settings (number of epochs, number of training avalanche sequences, number of testing avalanche sequences, as well as the runtime in minutes), as well as the resulting test accuracy, p-value and randomness result is shown.

D Bit profiles of AES-128.

D.1 AES round 1/10 bit pattern

The previous analysis of SPECK32/64 has shown a particular region of weak bits. In AES-128, however, we find repeating patterns of weak and strong bits in rounds 1 and 2 in the 128 bit sub-blocks of the avalanche unit.

Figure 13 shows details of the patterns observed after one round of AES-128. The complete avalanche unit of AES-128 consists of $128 \times 128 = 16,384$ bits. We analyze the complete avalanche unit in blocks of 128 bits (Fig. 13a)) and can identify four recurring patterns P1...P4 of weak and strong bits that occur throughout the avalanche unit. For example, pattern P1 occurs in the avalanche blocks $s = 0$ to $s = 7$ (Fig. 13b)). Exemplary sections for the distributions of weak and strong bit patterns are shown in Fig. 13c).

After one round of AES-128 96 consecutive bits of the 128 bits in each subblock can be predicted with 100% accuracy. The remaining 32 bits (4 bytes) can be predicted with less than 100% accuracy, which can be understood as follows. The round function of the AES is such that changing one byte in the input results in differences in one column of the output after one round (with the other columns remaining undisturbed). This is a well-known fact about the AES, due in particular to the MDS property of the mixcolumns operation. For the avalanche dataset, this implies that for each subblock of 128 bits (corresponding to one input difference bit), 4 bytes (one column) are nonzero, while the rest of the bytes are all zeroes.

The distribution of patterns of Fig. 13a) and Fig. 13b) is still observable after two rounds of AES (we show the equivalent 2-round patterns in the appendix Fig. 14c)). When encrypting for two rounds, each of the nonzero bytes of round 1 is sent to a different column through the shiftrows operation, and then propagated to a whole column through mixcolumns, so that after two rounds, all the bytes of the dataset are non-zero. Furthermore, there are relations between the bytes of each column: mixcolumns applies a linear transformation to a 4-byte column, and by construction, only one byte is non-zero in each column. Therefore, the resulting values are multiples (in the Galois field of AES) of a single variable, with the coefficients (2,3,1,1), in an order that depends on the position of the 128-bit block in the avalanche dataset. The bytes with coefficient 1 are consistently predicted, whereas only some bits of the bytes with coefficients 2 and 3 are reliably predicted. This explains the peculiar pattern observed in the prediction, where for each group of 4 bytes, there are peaks for 2 bytes, and for some of the bits among the remaining 2 bytes.

D.2 AES round 2/10 bit pattern

Please see Appendix D for the context of the analysis shown in Fig. 14.

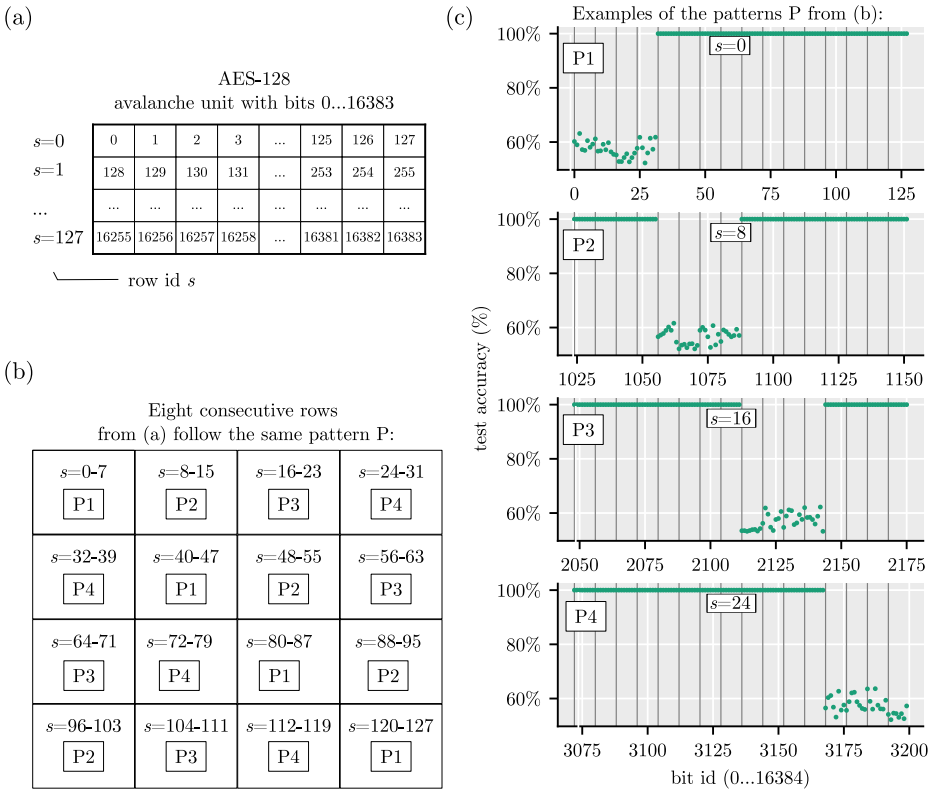

Fig. 13: NNBits ensemble analysis of the avalanche sequence of AES-128. a) The avalanche block s corresponds to bits $128 \times s \ldots 128 \times (s+1)$. b) Four patterns P1 ... P4 occur over the total of $16,384$ bits in one avalanche unit. c) Examples of the recurring byte patterns of weak and strong bits observed in round 1/10.

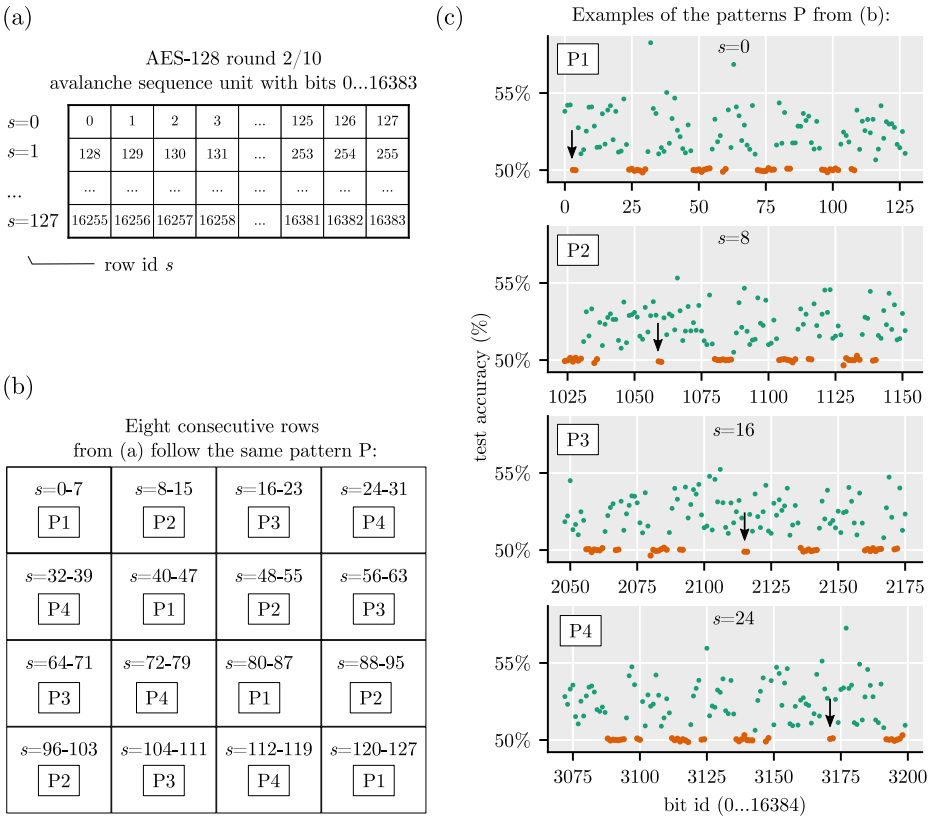

Fig. 14: NNBits ensemble analysis of the avalanche sequence of AES-128. a) The avalanche block s corresponds to bits $128 \times s \ldots 128 \times (s+1)$. b) Four patterns P1 ... P4 occur over the total of $16,384$ bits in one avalanche unit. c) Examples of the recurring patterns of weak (green) and strong (orange) bits. The pattern is actually the same, but shifted with a starting point indicated by the black arrow.

References

1. Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., et al.: Tensorflow: A system for large-scale machine learning. In: 12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16). pp. 265–283 (2016)
2. Aurélien Géron: Hands-on machine learning with Scikit-Learn, Keras and TensorFlow: concepts, tools, and techniques to build intelligent systems. O'Reilly Media (2019), https://www.oreilly.com/library/view/hands-on-machine-learning/9781492032632/
3. Bacuieti, N.N., Batina, L., Picek, S.: Deep neural networks aiding cryptanalysis : A case study of the Speck distinguisher. ePrint pp. 1–24 (2022), https://eprint.iacr.org/2022/341
4. Baksi, A., Breier, J., Chen, Y., Dong, X.: Machine learning assisted differential distinguishers for lightweight ciphers. In: 2021 Design, Automation Test in Europe Conference Exhibition (DATE). pp. 176–181 (2021). https://doi.org/10.23919/DATE51398.2021.9474092
5. Beaulieu, R., Shors, D., Smith, J., Treatman-Clark, S., Weeks, B., Wingers, L.: The SIMON and SPECK Families of Lightweight Block Ciphers. National Security Agency (NSA), 9800 Savage Road, Fort Meade, MD 20755, USA (2013)
6. Bellini, E., Rossi, M.: Performance comparison between deep learning-based and conventional cryptographic distinguishers. In: Intelligent Computing, pp. 681–701. Springer (2021)
7. Benamira, A., Gerault, D., Peyrin, T., Tan, Q.Q.: A deeper look at machine learning-based cryptanalysis. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 805–835. Springer (2021)
8. Biryukov, A., dos Santos, L.C., Teh, J.S., Udovenko, A., Velichkov, V.: Meet-in-the-filter and dynamic counting with applications to speck. Cryptology ePrint Archive (2022)
9. Breiman, L.: Bagging predictors. Machine Learning **24**(2), 123–140 (1996). https://doi.org/10.1023/A:1018054314350
10. Brown, R.G.: DieHarder: A Gnu Public License Random Number Tester. Duke University Physics Department, Durham, NC 27708-0305 (2006), http://www.phy.duke.edu/$\sim$rgb/General/dieharder.php
11. Castro, J.C.H., Sierra, J.M., Seznec, A., Izquierdo, A., Ribagorda, A.: The strict avalanche criterion randomness test. Mathematics and Computers in Simulation **68**(1), 1–7 (2005). https://doi.org/10.1016/j.matcom.2004.09.001
12. Daemen, J., Hoffert, S., Van Assche, G., Van Keer, R.: The design of xoodoo and xoofff. IACR Transactions on Symmetric Cryptology **2018**(4), 1–38 (Dec 2018). https://doi.org/10.13154/tosc.v2018.i4.1-38, https://tosc.iacr.org/index.php/ToSC/article/view/7359
13. Daor, J., Daemen, J., Rijmen, V.: AES proposal: Rijndael (10 1999), available at: https://www.cs.miami.edu/home/burt/learning/Csc688.012/rijndael/rijndael_doc_V2.pdf
14. Feistel, H.: Cryptography and computer privacy. Scientific american **228**(5), 15–23 (1973)
15. Gohr, A.: Deep speck. https://github.com/agohr/deep_speck (2019)
16. Gohr, A.: Improving Attacks on Round-Reduced Speck32/64 Using Deep Learning. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) **11693 LNCS**, 150–179

(2019). https://doi.org/10.1007/978-3-030-26951-7_6, https://doi.org/10.1007/978-3-030-26951-7_6.
17. Goodfellow, I., Bengio, Y., Courville, A.: Deep learning: The MIT Press, vol. 19. The MIT Press (2017), https://mitpress.mit.edu/books/deep-learning
18. Gunning, D., Vorm, E., Wang, J.Y., Turek, M.: Darpa's explainable ai (xai) program: A retrospective. Applied AI Letters **2**, e61 (12 2021). https://doi.org/10.1002/AIL2.61
19. Gustafson, H., Dawson, E., Golić, J.D.: Automated statistical methods for measuring the strength of block ciphers. Statistics and Computing **7**(2), 125–135 (1997)
20. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition. vol. 2016-Decem, pp. 770–778 (2016). https://doi.org/10.1109/CVPR.2016.90, http://image-net.org/challenges/LSVRC/2015/
21. Hornik, K.: Approximation capabilities of multilayer feedforward networks. Neural Networks (1991). https://doi.org/10.1016/0893-6080(91)90009-T
22. Hou, Z., Ren, J., Chen, S., Fu, A.: Improve neural distinguishers of simon and speck. Sec. and Commun. Netw. **2021** (jan 2021). https://doi.org/10.1155/2021/9288229
23. Hu, J., Shen, L., Sun, G.: Squeeze-and-excitation networks. In: 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 7132–7141 (2018). https://doi.org/10.1109/CVPR.2018.00745
24. Langford, S.K., Hellman, M.E.: Differential-linear cryptanalysis. In: Desmedt, Y.G. (ed.) Advances in Cryptology — CRYPTO '94. pp. 17–25. Springer Berlin Heidelberg, Berlin, Heidelberg (1994)
25. L'Ecuyer, P., Simard, R.: TestU01: a Software Library in ANSI C for Empirical Testing of Random Number Generators, Software User's Guide. Département d'Informatique et Recherche opérationnelle, Université de Montréal, Montréal, Québec, Canada (2001), http://www.iro.umontreal.ca/~simardr/TestU01.zip
26. Livni, R., Shalev-Shwartz, S., Shamir, O.: On the computational efficiency of symmetric neural networks. Advances in neural information processing systems **27**, 855—-863 (2014), https://papers.nips.cc/paper/2014/hash/3a0772443a0739141292a5429b952fe6-Abstract.html
27. Makridakis, S., Spiliotis, E., Assimakopoulos, V.: The M4 Competition: 100,000 time series and 61 forecasting methods. International Journal of Forecasting **36**(1), 54–74 (2020). https://doi.org/10.1016/j.ijforecast.2019.04.014
28. Moritz, P., Nishihara, R., Wang, S., Tumanov, A., Liaw, R., Liang, E., Elibol, M., Yang, Z., Paul, W., Jordan, M.I., et al.: Ray: A distributed framework for emerging {AI} applications. In: 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18). pp. 561–577 (2018)
29. Oreshkin, B.N., Carpov, D., Chapados, N., Bengio, Y.: N-BEATS: neural basis expansion analysis for interpretable time series forecasting. In: 8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020. OpenReview.net (2020), https://openreview.net/forum?id=r1ecqn4YwB
30. Reddi, S.J., Kale, S., Kumar, S.: On the convergence of adam and beyond. arXiv preprint arXiv:1904.09237 (2019)
31. Rukhin, A., Soto, J., Nechvatal, J., Smid, M., Barker, E., Leigh, S., Levenson, M., Vangel, M., Heckert, A., Dray, J., Vo, S.: A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications. NIST (2010)

32. Schrittwieser, J., Antonoglou, I., Hubert, T., Simonyan, K., Sifre, L., Schmitt, S., Guez, A., Lockhart, E., Hassabis, D., Graepel, T., Lillicrap, T., Silver, D.: Mastering Atari, Go, chess and shogi by planning with a learned model. Nature **588**(7839), 604–609 (2020). https://doi.org/10.1038/s41586-020-03051-4
33. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. In: 3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings (2015), http://www.robots.ox.ac.uk/
34. Soto, J.: Randomness testing of the advanced encryption standard candidate algorithms. NIST Interagency/Internal Report (NISTIR) (1999), http://www.nist.gov/customcf/get_pdf.cfm?pub_id=151193
35. Soto, J., Bassham, L.: Randomness Testing of the Advanced Encryption Standard Finalist Candidates . NIST Interagency/Internal Report (NISTIR) (2000), https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=151216
36. Švenda, P., Ukrop, M., Matyáš, V.: Determining cryptographic distinguishers for estream and sha-3 candidate functions with evolutionary circuits. Communications in Computer and Information Science **456**, 290–305 (2014). https://doi.org/10.1007/978-3-662-44788-8_17
37. Team, P.: PyTorch ResNet Implementation. https://pytorch.org/hub/pytorch_vision_resnet/ (2022)
38. Team, R.: Ray. https://github.com/ray-project/ray (2022)
39. (TII), T.I.I.: Crypto-tii nnbits. https://github.com/Crypto-TII/nnbits (2022)
40. Virtanen, P.e.a.: SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. Nature Methods **17**, 261–272 (2020). https://doi.org/10.1038/s41592-019-0686-2
41. Walker, J.: ENT: A pseudorandom number sequence test program. Web site (Jan 2008), http://www.fourmilab.ch/random/
42. Yadav, T., Kumar, M.: Differential-ml distinguisher: Machine learning based generic extension for differential cryptanalysis. In: Progress in Cryptology – LATINCRYPT 2021: 7th International Conference on Cryptology and Information Security in Latin America, Bogotá, Colombia, October 6–8, 2021, Proceedings. p. 191–212. Springer-Verlag, Berlin, Heidelberg (2021). https://doi.org/10.1007/978-3-030-88238-9_10