Improving Differential-Neural Cryptanalysis for Large-State SPECK

Tianrong Huang, Yingying Li, Qinggan Fu, Yincen Chen, and Ling Song(✉)

College of Cyber Security, Jinan University, Guangzhou 510632, China
htr_199909@163.com, songling.qs@gmail.com

Abstract. At CRYPTO 2019, Gohr presented a key recovery attack on SPECK32/64 assisted by deep learning. For the shortcoming that this technology cannot be used for large-state block ciphers, Chen *et al.* proposed a deep learning-assisted multi-stage key recovery framework in 2022, based on which key recovery attacks on large-state SPECK variants were successfully mounted. In this paper, we propose a parallelizable multi-stage key recovery framework. This framework uses the neural distinguisher trained by a new strategy to reduce the time required for the attack while maintaining the accuracy of key recovery. We conduct key recovery attacks on round-reduced SPECK64/96 and SPECK96/96. The results indicate that our framework significantly reduces the time complexity. Additionally, we train neural distinguishers over more rounds on partial bits by filtering the input differences, including more ciphertext information in training samples, using a stronger neural network and staged train method. Consequently, we obtain a set of neural distinguishers for 7-round SPECK64 and successfully extend the neural network-based key recovery attacks on SPECK64/96 by one round.

Keywords: Deep learning · Differential cryptanalysis · Key recovery attack · Large-state block cipher · SPECK.

1 Introduction

Symmetric cryptography is a key element in maintaining system security. Analyzing symmetric cryptography helps in better understanding its security. Among symmetric cryptanalysis methods, differential cryptanalysis is one of the most powerful methods, which was proposed by Eli Biham and Adi Shamir in 1990 [6]. Since then, it has become a basic method for modern cryptography research. Differential cryptanalysis studies the propagation of specific input differences in the encryption algorithm to find high-probability differential paths, thereby achieving attacks on the encryption algorithm. It provides a valuable tool for evaluating the security of new encryption algorithms and also provides important guiding principles for the design of new encryption algorithms.

Artificial intelligence and deep learning technology have developed rapidly in the past decade, making breakthroughs in many fields, such as large language

models like ChatGPT [7], AI-based image generation technologies like MidJourney [15], and autonomous driving systems [8]. Deep learning has the characteristics of learning features from a large amount of data, so a natural question is: Can the integration of machine learning with cryptanalysis achieve results that traditional methods cannot? In 2019, Gohr initially combined differential cryptanalysis with neural networks and came up with *neural differential cryptanalysis* [12], which was successfully applied to SPECK32/64 block cipher. This achievement marked a significant advancement in artificial intelligence technology in the field of cryptanalysis.

Since Gohr proposed differential-neural cryptanalysis, the development of this method has primarily focused on improving the neural distinguisher. It mainly involves two directions. The first direction is to improve the neural distinguishers by changing the training data to include more information about the encryption algorithm. Chen *et al.* [11] used multiple ciphertext pairs as a single training sample, which significantly improved the accuracy of the neural distinguishers against SPECK32/64. Liu *et al.* [14] introduced a novel approach using the output features of the penultimate round to generate two-dimensional and non-realistic input data, resulting in distinguishers with extended analysis rounds and higher accuracy for SPECK and SIMON. The other direction is to try different neural networks to train the neural distinguishers. Liu *et al.* [13] used depthwise separable convolutions instead of traditional convolutions, reducing training costs without affecting the accuracy of the distinguishers. Inspired by GoogleNet, Zhang *et al.* [17] added an inception consisting of multiple parallel convolutional layers before the residual networks, significantly enhancing the accuracy of the neural distinguishers for SPECK32/64 and SIMON32/64.

The outstanding performance of differential-neural cryptanalysis has aroused curiosity in the community about the working mechanism of neural distinguishes. In 2021, Adrien *et al.* [4] conducted a detailed analysis of Gohr's neural distinguisher. They found that Gohr's neural distinguisher not only relies on the differential distribution of the ciphertexts but also on the differential distribution of the penultimate and antepenultimate rounds. In 2023, Bao *et al.* [2] discovered the specific form of additional information used by the neural distinguisher, beyond just ciphertext differences. They utilized this information to enhance classical differential distinguishers.

Gohr's key recovery attack method shows excellent performance on small-state block ciphers like SPECK32/64, but it faces limitations with large-state block ciphers where the block size is 64 bits or larger. The reason is that Gohr's attack method requires guessing all the key bits at once. The large key space of large-state block ciphers makes the attack unable to be completed in an acceptable amount of time. At the same time, the neural distinguishers only use the information from part of the ciphertext bits [10]. This results in the key bits associated with the ciphertext bits having little impact on the neural distinguisher not being correctly recovered. To address these issues, Chen *et al.* [9] developed a deep learning-aided multi-stage key recovery framework. This framework employed multiple neural distinguishers, each of them taking ciphertext fragments

as input. Chen *et al.* conducted key recovery attacks on SPECK's large-state variants, demonstrating the framework's effectiveness. Although Chen *et al.*'s work offers a solution for conducting key recovery attacks on large state block ciphers using neural networks within an acceptable time, further exploration is needed to reduce the attack time while maintaining key recovery accuracy. Moreover, compared to the longest-round traditional analysis results [16], the neural network-based approach still has room for further optimization in terms of the number of attacked rounds.

1.1 Our Contributions

In this paper, we improve differential-neural cryptanalysis for large-state SPECK by improving the distinguishers and attack framework.

1. A new neural distinguisher training strategy on partial ciphertext bits is proposed. Neural distinguishers trained using this strategy can perform key recovery attacks on key bits associated with selected ciphertext bits independently, without relying on other key bits.
2. We propose a parallelizable multi-stage key recovery framework. This framework uses two types of neural distinguishers: a set of neural distinguishers on partial ciphertext bits trained using our strategy and a neural distinguisher on full ciphertext. The framework first uses neural distinguishers on partial ciphertext bits to recommend partial keys serially or in parallel. Then the final key is filtered out using a neural distinguisher on the full ciphertext.
3. Under the same device conditions, we conduct 100 key recovery attacks on reduced-round SPECK64/96 and SPECK96/96 using both our framework and Chen *et al.*'s framework. The results indicate that our framework significantly reduces the time required for the attack while maintaining the accuracy of key recovery.
4. To apply differential-neural cryptanalysis to higher-round large-state SPECK, we improve the training method for the neural distinguishers on partial bits. Ultimately, we train a set of distinguishers for 7-round SPECK64/96. Combining with our attack framework, we successfully conduct a key recovery attack on 10-round SPECK64/96.

The results of the attacks and comparison with Chen *et al.*'s attacks are summarized in Table 1. All of our code is available at https://github.com/AI-Cipher-Sicurity/code.

2 Preliminaries

2.1 A Short Description of SPECK

SPECK is a set of lightweight block cipher algorithms designed by the National Security Agency (NSA) of the United States [3]. Recognized by the International

Table 1: Summary of key-recovery attacks on large-state SPECK

Target	Round	Avg. Time [1]	$hw(kg, rk)$ [2]	Configure	Ref.
SPECK64/96	9	66s	1.93	$1 + 1r_{CD} + 6r_{ND}$	[9]
	9	29s	0.80	$1 + 1r_{CD} + 6r_{ND}$	Sect. 3.3
	10	424s	1.89	$1 + 1r_{CD} + 7r_{ND}$	Sect. 4.4
SPECK96/96	10	303s	1	$1 + 1r_{CD} + 7r_{ND}$	[9]
	10	163s	0.44	$1 + 1r_{CD} + 7r_{ND}$	Sect. 3.3

[1] Avg. Time refers to the average time required to perform a single attack on a computer equipped with an NVIDIA 3060 graphics card.
[2] $hw(kg, rk)$ refers to the average Hamming distance between the guessed key and the actual key.

Organization for Standardization (ISO), SPECK is widely adopted in applications ranging from low-power devices to secure data encryption.

Based on the block size and key size, SPECK can be categorized into various versions, such as SPECK32/64, where the number before the slash indicates the block size in bits, and the number after denotes the key size in bits. The round function of SPECK for a block size of $2n$ bits can be described as follows:

$$x_{i+1} = ((x_i \ggg \alpha) \boxplus y_i) \oplus k_i,$$
$$y_{i+1} = (y_i \lll \beta) \oplus x_{i+1},$$

where x_i and y_i represent the n-bit inputs of the round function, k_i is the $n-$bit round key, $\ggg$ and $\lll$ denote right and left bitwise rotations, $\oplus$ represents bitwise XOR, and $\boxplus$ indicates addition modulo 2^n.

The constants α and β are uniquely defined for each version. For example, the SPECK32/64 uses $\alpha = 7$ and $\beta = 2$, while SPECK64/96 and SPECK96/96 employ $\alpha = 8$ and $\beta = 3$.

2.2 Neutral Bits

Neutral bits play a critical role in the field of differential cryptanalysis. It was introduced by Eli Biham *et al.* [5] and extended into simultaneous-neutral bitsets (SNBS) by Bao *et al.* [1]. A neutral bit is a bit that, when flipped, does not affect the propagation of differences through the encryption rounds. Here is the definition of a neutral bit:

Definition 1. *Let $\Delta_{in} \to \Delta_{out}$ be a differential with input difference Δ_{in} and output difference Δ_{out} over a r-round encryption function F. If (P, P') is an input pair and (C, C') is the corresponding output pair where $P \oplus P' = \Delta_{in}$ and $C \oplus C' = \Delta_{out}$, then a bit i is said to be a neutral bit if flipping the i-th bit in both P and P' results in a new output pair that also satisfies the differential.*

In differential-neural cryptanalysis, k neutral bits are usually used to expand a plaintext pair that conforms to the differential path into 2^k plaintext

pairs. During the key recovery attack, neural networks are used to evaluate the composite scores of corresponding ciphertext pairs of these plaintext pairs. This evaluation ensures the accuracy of key recovery.

2.3 Gohr's Neural Distinguisher and Key Recovery Attack for SPECK32/64

The essence of the neural distinguisher is a neural network performing a binary classification task. For a given encryption algorithm S and a specific plaintext difference Δ, the neural network is trained to distinguish the true ciphertext pairs and the random ciphertext pairs. The true ciphertext pairs are obtained by encrypting plaintext pairs with the specific difference Δ using the encryption algorithm S. And the random ciphertext pairs are obtained by encrypting plaintext pairs with a random difference using the same encryption algorithm S. The true ciphertext pairs are labeled with 1, and the random ciphertext pairs with 0. Both of the training and test sets comprise true ciphertext pairs and random ciphertext pairs. The neural network is trained on the training set and its performance is evaluated on the test set. If the accuracy of predicting the ciphertext labels on the test set exceeds 0.5, an effective neural distinguisher is constructed successfully.

Gohr's key recovery attack includes two versions: a basic version and an accelerated version. Here, we briefly introduce the core idea of the basic version of the key recovery attack. For an r-round neural distinguisher with input difference Δ, by prefixing it with an s-round classical differential $\Gamma \to \Delta$, a hybrid distinguisher of $s+r$ rounds is formed. This hybrid distinguisher can be used to recovery the $(s+r+1)$-th round key rk, with the following attack process:

1. Randomly generate a plaintext pair $(P, P \oplus \Gamma)$, expand this plaintext pair into 2^k plaintext pairs using the k neutral bits in the classical differential, and encrypt these 2^k plaintext pairs to the $(s+r+1)$-th round using the encryption algorithm SPECK32/64 to obtain 2^k ciphertext pairs.
2. Traverse all possible values of rk one by one. For each possible value kg:
   (a) Decrypt the 2^k ciphertext pairs by one round using kg and feed the decrypted results into the neural distinguisher for scoring, recording the score as Z_i, where $i \in [1, 2^k]$.
   (b) Calculate the comprehensive score v_{kg} for kg using the following formula: $v_{kg} := \sum_{j=1}^{2^k} \log_2 \left(\frac{Z_j}{1-Z_j}\right)$. If v_{kg} exceeds a threshold c, then consider kg as a possible candidate key.
3. Repeat step 1 until a candidate key emerges.

2.4 Chen *et al.*'s Key Recovery Framework for Large-State SPECK

To apply deep learning-assisted key recovery attacks on large-state SPECK, Chen *et al.* proposed a multi-stage key recovery framework. The schematic of

this framework is illustrated in Figure 1. This framework employs multiple r-round neural distinguishers, denoted as ND_i, where $i \in [1, x]$, each with a plaintext difference Δ_i. These neural distinguishers are trained using fragments of ciphertext pair C_i generated from plaintext pairs that satisfy the difference Δ_i. A classical differential $\Gamma_i \to \Delta_i$, denoted as CD_i, is added before each neural distinguisher ND_i.

The framework divides the complete key from high to low bits into x parts, each represented by B_i, such that the key is composed of all bits in the order $B_x||B_{x-1}|| \ldots ||B_2||B_1$ (where $||$ denotes concatenation). Each stage i recovers the B_i part of the key, starting with the part containing the lowest bits, B_1. The x stages of the attack are executed sequentially, where the $|B_i|$ key bits in stage i (where $|B_i|$ denotes the length of B_i) are recovered based on the key recovery results of the previous $i-1$ stages. That is, in stage i, the attacker has successfully recovered the $B_{i-1}|| \ldots ||B_2||B_1$ part of the key. More details of this framework can be found in [11].

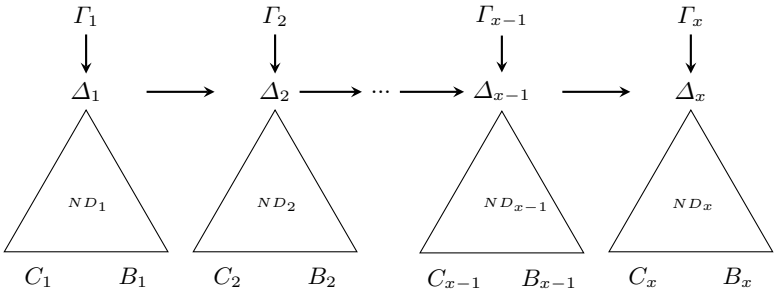

Figure 1: Chen *et al.*'s key recovery framework for large-state SPECK

3 A Parallelizable Key Recovery Framework for Large-State SPECK

The neural distinguishers used in this framework adopt a novel training strategy, allowing the distinguishers to operate independently during key recovery. Compared to Chen *et al.*'s key recovery framework, our method significantly enhances the speed of key recovery while maintaining the accuracy rate. Before delving into the details of our key recovery framework, we will first introduce the training strategy for our neural distinguishers.

3.1 New Training Strategy for Neural Distinguishers

In our parallelizable multi-stage key recovery framework, we use two types of neural distinguishers: a set of neural distinguishers on partial ciphertext bits and

a neural distinguisher on full ciphertext. The training method for the r-round full ciphertext neural distinguisher is the same as that described for Gohr's neural distinguishers in Section 2.3. This involves encrypting plaintext pairs labeled with 0 and 1 to r rounds, and then using the resulting ciphertext pairs to train the neural network. For the r-round partial-bit neural distinguisher, we adopt a training method different from that of Chen *et al.* We do not directly use fragments of r-round SPECK ciphertext to train the neural distinguisher.

For an r-round partial neural distinguisher on r-round ciphertext state bits $[a, b]$, our training method is as follows:

1. For a plaintext pair (P, P') of SPECK with 0 or 1 labels, first encrypt the plaintext pair to $r + 1$ rounds using round keys Ks_1 to Ks_{r+1}, obtaining ciphertext pair $(C_{r+1,0}, C_{r+1,1})$.
2. Keep the value of the bits corresponding to $[a+\alpha, b+\alpha]$ in Ks_{r+1} unchanged, and set the other key bits to zero, thereby obtaining a pseudo-round key PKs_{r+1}. Here, α represents the round constant α in SPECK.
3. Decrypt $(C_{r+1,0}, C_{r+1,1})$ one round using PKs_{r+1}, extract the ciphertext bits in $[a, b]$, and train the neural network with these bits and their corresponding labels.

In the following article, for clarity and ease of discussion, we use *PND* to represent the neural distinguisher on partial ciphertext bits trained using the above method. *FND* is used to represent the neural distinguisher on full ciphertexts.

When using *PND* on ciphertext bit $[a, b]$ for key recovery attacks, we only need to guess the key bits corresponding to *PND*, i.e., bit $[a+\alpha, b+\alpha]$. All other key bits can be set to zero. In contrast, when using a neural distinguisher trained directly with r-round ciphertexts, the partial-bit neural distinguisher relies on other key bits. For example, to recover bits [12, 24] of the $(r + 1)$-round key of SPECK64/96, if using the neural distinguisher trained directly with r-round ciphertexts, due to the carry generated by modular addition, we first need to know the bits [0, 11] of the key. However, with *PND* that trained using our strategy, we can directly guess bits [12, 24] of the key, setting all other bits to zero. Since *PND* does not depend on other key bits, multiple *PND*s can be run parallelly if equipment conditions permit.

3.2 Parallelizable Key Recovery Framework

The parallelizable key recovery attack framework is depicted in Figure 2. The entire key recovery attack process is divided into two stages, *PND*s recommend partial keys, and *FND* filters out the final guessed key. The fundamental idea of the framework is to employ multiple independent *PND*s, and each *PND* performs a key recovery attack on its corresponding key bits. The partial keys recommended by each *PND* are then combined, and the final key is selected by the full ciphertext neural distinguisher *FND*.

For a key recovery attack using our framework that includes x *PND*s and one *FND*, the selection of these x *PND*s and *FND* is based on the following criteria:

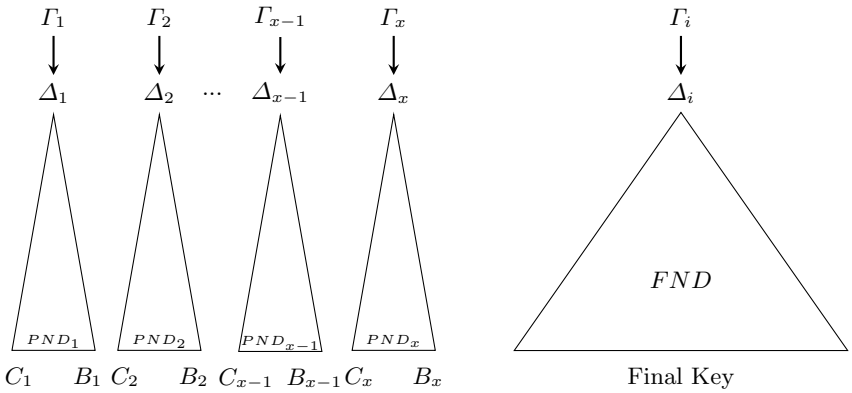

Figure 2: The parallelizable key recovery framework for large-state SPCEK. It utilizes x PND_i and one FND, where each PND_i uses a plaintext difference Δ_i, and the FND uses a plaintext difference Δ. A classical differential CD_i is added in front of each PND and FND, respectively, denoted as $\Gamma_i \to \Delta_i$, where $i \in [1, x]$. Each PND is trained on partial ciphertext bits C_i and is used to recommend partial key bits B_i, where $i \in [1, x]$. The FND is trained on full ciphertext to filter out the final guessed key.

1. The union of the key bits corresponding to the x PNDs should cover as many key positions as possible. For the m key bits not covered in this set, their values should be traversed during FND filters out the final guessed key.
2. The input length for each PND should be moderate to ensure that the workload for each PND is feasible and balanced.
3. During the training of FND, its input difference Δ should be the same as the input difference of certain PND_i. Choosing the same input difference as one of the PNDs allows for the direct use of ciphertext pairs that satisfy the differential path of that PND during FND filters out the final guessed key. At the same time, we should try the input difference of different PNDs as the input difference of FND to find the FND with the highest accuracy. The higher accuracy of FND can improve the accuracy of the final key guessed.

Specifically, supposing the attacker selects x PNDs along with x classical differentials $CD_i := \Gamma_i \xrightarrow{p_i} \Delta_i$, where p_i, $i \in [1, x]$, denotes the probability of the classical differential. And the attacker uses the input difference of the PND_ϵ, where $\epsilon \in [1, x]$, as the input difference of FND. The goal is to recover the round key rk of the last round. During PNDs recommend partial keys, each PND_i recommends N_i value for its corresponding key bits. In each step, at most Q plaintext structures are generated. If no suitable value is recommended after consuming Q plaintext structures, the attack is considered failed. Once all PNDs have completed their recommendations, the recommended keys are combined into $\prod_{i=1}^{x} N_i$ keys. The remaining m key bits not covered by the PNDs are then traversed and combined with these $\prod_{i=1}^{x} N_i$ keys, resulting in a

total of $(2^m) \times \prod_{i=1}^{x} N_i$ recommended full keys. During *FND* filters out the final guessed key, the *FND* selects the most probable key in these $(2^m) \times \prod_{i=1}^{x} N_i$ recommended full keys.

A relatively larger N_i, $i \in [1, x]$ allows *FND* to have more choices when filtering the final key, thereby improving the accuracy of key recovery. It is worth noting that setting N_i, $i \in [1, x]$ to slightly larger values only modestly increases the time needed for *FND* to filter the final key and does not significantly affect the overall speed of the attack.

During *PND*s recommend partial keys, as PND_i only recommends its corresponding key bits, *PND*s are independent of each other. Thus this stage can be executed serially or in parallel if computational resources allow. If this stage is executed in parallel, the speed of the entire attack can be greatly accelerated. Additionally, during *PND*s recommend partial keys, either the basic or accelerated version method of Gohr's key recovery attack can be employed.

The entire attack process is as shown in Algorithm 1.

3.3 Application to SPECK64/96 and SPECK96/96

In this section, we employ the neural distinguisher training strategy and the key recovery framework proposed above to conduct key attacks on SPECK64/96 and SPECK96/96. To directly compare with Chen *et al.*'s framework, we conduct attacks on the cryptographic algorithm using both our framework and that of Chen *et al.* under the same equipment conditions. The experimental setup is a computer equipped with an NVIDIA 3060 graphics card.

Key recovery attack for SPECK64/96 Referencing Chen *et al.*'s 9-round key recovery attack on SPECK64/96, we train three 6-round *PND*s using the same input differences and ciphertext state bits. Our training employs the strategy proposed above: first encrypting the plaintext pairs up to 7 rounds, then zeroing the bits in the 7th round key that are irrelevant to the ciphertext state bits to obtain pseudo-key. The pseudo-key is used to decrypt the ciphertext pairs by one round. Take out the status bits of the decryption result and put them into the neural distinguisher for training. The input difference for the *FND* is the same as the PND_2. Like the traditional method of training neural distinguishers, we encrypt plaintext pairs up to 6 rounds and directly train the neural network with the 6-round full ciphertexts to create the *FND*. Table 2 summarizes the distinguishers we trained.

Similarly, we prepend a one-round classical differential $\Gamma_i \to \Delta_i$ (as shown in Table 3) to each distinguisher listed in Table 2, forming a 7-round hybrid distinguisher HD_i. Since the first nonlinear operation of SPECK does not involve a whitening key, each 7-round hybrid distinguisher can be extended by an additional free round at the top, becoming an 8-round distinguisher. This allows for a 9-round key recovery attack on SPECK64/96.

The entire attack process is executed according to Algorithm 1. During *PND*s recommend partial keys, it is divided into three steps. First, the bits $0 \sim 9$ of the

Algorithm 1 Algorithm for guessing the last round key rk

Input: B_i, C_i, c_i, N_i, $i \in [1, x]$, Q, ϵ.
Output: the final key guessed rk.

Initialize a variable $S \leftarrow \varnothing$.
for $i \in [1, x]$ **do**
  for $j \in [1, Q]$ **do**
    Generate plaintext pairs that satisfy the difference Γ_i.
    Utilize the k neutral bits that exist in classical differential $\Gamma_i \rightarrow \Delta_i$ to expand the plaintext pair into the plaintext structure.
    Obtain the corresponding ciphertext structure by the encryption algorithm.
    Initialize a list $L_i \leftarrow \emptyset$.
    for each of the $2^{|B_i|}$ possible values kg_i correspond to B_i **do**
      Partially decrypt the 2^k ciphertext pairs in the ciphertext structure by one round using kg_i.
      Extract the status bit in C_i.
      Feed the status bit into PND_i, obtain 2^k scores Z_j for $j \in [1, 2^k]$.
      Combine the scores using the formula: $\nu_{kg_i} = \sum_{j=1}^{2^k} \log_2 \left(\frac{Z_j}{1-Z_j} \right)$.
      if $\nu_{kg_i} > c_i$ **then**
        store (kg_i, ν_{kg_i}) in L_i.
      end if
    end for
    if $L_i = \emptyset$ **then**
      if $j = Q$ **then**
        return Attack Fail.
      end if
    end if
    if $L_i \neq \emptyset$ **then**
      Sort L_i according to the scores of the guessed keys.
      Take the top N_i guessed keys as the recommended keys for step i.
      if $i = \epsilon$ **then**
        Set $S \leftarrow$ The ciphertext structure.
      end if
      Break.
    end if
  end for
end for
Combine the partial keys guessed from different steps and obtain $\prod_{i=1}^{x} N_i$ complete guessed keys.
Initialize a list $L \leftarrow \emptyset$.
for each of the $\prod_{i=1}^{x} N_i$ complete guessed keys fkg_i **do**
  Decrypt the ciphertext structure in S by one round using fkg_i.
  Feed the decryption result into FND, obtain 2^k scores Z_j for $j \in [1, 2^k]$.
  Combine the scores using the following formula: $\nu_{fkg_i} = \sum_{j=1}^{2^k} \log_2 \left(\frac{Z_j}{1-Z_j} \right)$, store (fkg_i, ν_{fkg_i}) in L.
end for
Sort L according to the scores of complete guessed keys, take the complete guessed key with max score as the final key guess rk.
Return the final key guess rk.

last round subkey are recommended, followed by the bits $10 \sim 21$, and finally, the bits $22 \sim 31$.

In the attack, the constant Q is set to 32, meaning that at most 32 ciphertext structures are generated during *PND*s recommend partial keys. For the three steps during *PND*s recommend partial keys, the threshold for filtering incorrect keys is set to $c_1 = c_2 = c_3 = 10$. The value of ϵ is 2, implying that when PND_2 recommends some suitable value, the ciphertext structure is saved for *FND* to filter out the final guessed key. In each *PND* step, 10 recommended keys are saved, that is, $N_1 = N_2 = N_3 = 10$.

Under these experimental settings, 100 trials are conducted. In these trials, 100 final key guesses are returned. The average time consumption is 29 seconds. The Hamming distance between the final key guess kg and the correct round key rk, denoted as $hw(kg, rk)$, averaged 0.80 over 100 trials.

For a direct comparison with Chen *et al.*'s experiment, we replicate their experiment on the same equipment. Likewise, generate at most 32 ciphertext structures in each stage and use the same batch size for neural network scoring. In 100 trials, 100 final key guesses are returned, with an average time consumption of 66 seconds. The average Hamming distance is 1.93.

Details on $hw(kg, rk)$ for both trials are summarized in Table 4, where trial 1 and trial 2 represent our experiments and the replicated experiments of Chen *et al.*, respectively.

The experiments demonstrate that our framework can accelerate the attack while ensuring the accuracy of key recovery. This is significant for attacks with more rounds, where longer classical differentials with lower probabilities can greatly increase the time required. Our framework can significantly reduce the time needed. Additionally, if the equipment allows, our multiple *PND*s can be executed in parallel, further accelerating the attack speed.

Table 2: Neural distinguishers against 6-round SPECK64

Distinguisher	Δ_i[1]	C_i[2]	Acc.[3]
PND_1	$\Delta_{[42]}$	$\{17 \sim 8\}$	0.613
PND_2	$\Delta_{[47]}$	$\{29 \sim 18\}$	0.662
PND_3	$\Delta_{[33]}$	$\{31, 30, 7 \sim 0\}$	0.620
FND	$\Delta_{[47]}$	$\{31 \sim 0\}$	0.754

[1] Δ_i: The input difference. $\Delta_{[i]}$ represents a single-bit difference whose i-th bit is the only active bit.

[2] C_i: The index of bits of x_r that are fed into distinguishers. It also includes the same index of bits of y_r, where x_r (resp. y_r) is the left (resp. right) n-bit word of a full r-round output state.

[3] Acc.: The accuracy of the PND_i or *FND*.

Key recovery attack for SPECK96/96 To perform key recovery attacks on 10-round SPECK96/96, we train four 7-round SPECK96/96 *PND*s and a *FND*.

Table 3: One-round classical differentials to be prepended to the distinguishers in Table 2

$\Gamma_i \to \Delta_i$	NB's	p_i
$\Delta_{[50,47,7]} \to \Delta_{[42]}$	$\{39 \sim 30\}$	2^{-2}
$\Delta_{[55,52,12]} \to \Delta_{[47]}$	$\{39 \sim 30\}$	2^{-2}
$\Delta_{[41,38,30]} \to \Delta_{[33]}$	$\{29 \sim 20\}$	2^{-2}

Table 4: Statistics on $hw(kg, rk)$ over 100 trials of the 9-round attack on SPECK64

$hw(kg, rk)$	0	1	2	3	4	5	6	7
trial 1	58	16	19	4	1	2	0	0
trial 2	28	42	22	5	2	1	0	1

Similarly, we employ the training strategy described above to train the *PND*s. This involves encrypting plaintext pairs up to 8 rounds, then decrypting them to 7 rounds using a pseudo-key, and finally inputting the state bits into the neural network. The input difference for the *FND* is the same as PND_1. It is trained by directly inputting 7-round full ciphertexts into the neural network. The details of these distinguishers are summarized in Table 5.

We prepend a one-round classical differential $\Gamma_i \to \Delta_i$ (as shown in Table 6) to each distinguisher in Table 5, forming an 8-round hybrid distinguisher HD_i. By extending one free round, the formed 9-round hybrid distinguisher can attack 10-round SPECK96/96.

During *PND*s recommend partial keys, each step recommends 12 bits of the key: specifically, bits $0 \sim 11$, $12 \sim 23$, $24 \sim 35$, and $36 \sim 47$. *FND* then filters out the final key.

In the attack, the constant value Q is set to 32. For the four steps during *PND*s recommend partial keys, the threshold for filtering incorrect keys is $c_1 = c_2 = c_3 = c_4 = 15$. The value of ϵ is 1, meaning that when PND_1 makes a key recommendation, the used ciphertext structure is saved for *FND*'s final key selection. For the four *PND* steps, $N_1 = 3$ and $N_2 = N_3 = N_4 = 17$. N_1 is set to 3 because the top three keys recommended by PND_1 almost always include the correct key for that part. N_2, N_3, and N_4 are set to 17 to increase the likelihood that the recommended keys include the correct key.

Under these experimental settings, we conduct 100 experiments with an average time consumption of 163 seconds. In these experiments, 100 final key guesses are returned. The average Hamming distance between the final key guess kg and the correct round key rk is 0.44. Details on $hw(kg, rk)$ are summarized in Table 7, where trial 3 represents our experiments.

We replicate Chen *et al.*'s experiment on the same equipment. Similarly, at most 32 ciphertext structures are generated in each stage, and the same batch size is used for neural network scoring. In 100 trials, 100 key recommendations

are returned, with an average time consumption of 303 seconds. The average Hamming distance is 1. Details on $hw(kg, rk)$ are summarized in Table 7 for trial 4.

Experimental results also show that using our key recovery framework can improve attack speed while ensuring key accuracy.

Table 5: Neural distinguishers against 7-round SPECK96

Distinguisher	Δ_i	C_i	Acc.
PND_1	$\Delta_{[55]}$	$\{19 \sim 8\}$	0.681
PND_2	$\Delta_{[67]}$	$\{31 \sim 20\}$	0.608
PND_3	$\Delta_{[77]}$	$\{43 \sim 32\}$	0.595
PND_4	$\Delta_{[89]}$	$\{47 \sim 44, 7 \sim 0\}$	0.601
FND	$\Delta_{[55]}$	$\{47 \sim 0\}$	0.832

Table 6: One-round classical differentials to be prepended to the distinguishers in Table 5

$F_i \rightarrow \Delta_i$	NB's	p_i
$\Delta_{[63,60,4]} \rightarrow \Delta_{[55]}$	$\{30 \sim 21\}$	2^{-2}
$\Delta_{[75,72,16]} \rightarrow \Delta_{[67]}$	$\{50 \sim 41\}$	2^{-2}
$\Delta_{[85,82,26]} \rightarrow \Delta_{[77]}$	$\{49 \sim 40\}$	2^{-2}
$\Delta_{[94,49,38]} \rightarrow \Delta_{[89]}$	$\{61 \sim 52\}$	2^{-2}

Table 7: Statistics on $hw(kg, rk)$ over 100 trials of the 10-round attack on SPECK96

$hw(kg, rk)$	0	1	2	3	4	5	6
trial 3	76	14	4	4	1	0	1
trial 4	34	49	9	3	3	0	2

Interpretation of Results We have summarized the experimental results and compared them with those of Chen *et al.* in Table 1. As can be seen, the average attack time for the same number of rounds is lower than that of Chen *et al.*. If multiple GPUs are used to separately perform the key recovery for each *PND*, our key recovery attack will be further accelerated. Additionally, the average

Hamming distance of the recovered keys is lower than that of Chen *et al.*. Compared to the attack framework of Chen *et al.*, our approach does not require the *PND* to depend on the lower bits of the key when recommending partial key bits. This can prevent errors in certain bits of the key recommended by the previous *PND* from further affecting the operation of the next *PND*, thereby reducing the number of erroneous key bits. This is one of the reasons why our accuracy is higher. Furthermore, when filtering the final keys, we used the *FND* with the highest accuracy, ensuring the accuracy of the final keys.

4 Improved Partial Neural Distinguisher for Ciphertext

In this section, we try to use neural networks to train a set of 7-round distinguishers for SPECK64/96, and conduct key recovery attacks on 10-round SPECK64/96.

4.1 Selection of Input Differences

To train a set of distinguishers for SPECK 64/96 with longer rounds, we filter the input differences of the neural network before training. Gohr adopted ($0x0040$, 0) as the input difference of the neural distinguisher for the attack on SPECK32/64. Due to the constraints at the highest bit, the carry effect produced by the first round of modular addition is eliminated, so that the difference ($0x0040$, 0) propagates to ($0x8000$, 0) with a probability of 1. Therefore, making the difference distribution more concentrated is beneficial to the neural network in learning ciphertext knowledge. Similarly, for attacks on SPECK64/96, we choose ($0x80$, 0), ($0x8000$, 0), and ($0x800000$, 0) as the input differences for the neural network. These differences are likely to eliminate the carry effect of modular addition in the initial rounds, leading to a more concentrated distribution of differences.

4.2 Network Architecture

In [17], Zhang *et al.* proposed a new network architecture and significantly improved the neural distinguishers for SPECK32/64 and SIMON32/64. We employ the neural network constructed by Zhang *et al.* to train our *PND*s. The neural network architecture consists of four main components: an input layer, an initial convolutional layer, a residual tower, and a prediction head. The structure of the neural network is shown in Figure 3, and more detailed information about this neural network can be found in [17].

4.3 Input Representation

To train more powerful distinguishers, the training samples should contain more information. Therefore, when training the *PND*s, we put fragments of multiple ciphertext pairs into a training sample, and the training sample also includes information from the penultimate round.

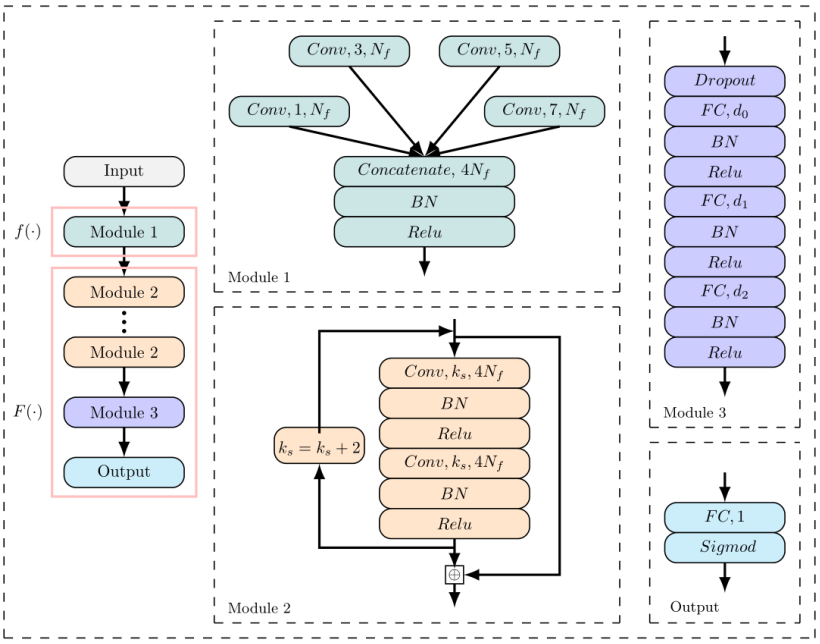

Figure 3: The network architecture

To train a *PND* on state bits $[a, b]$ for 7-round SPECK64/96, our training samples are generated as follows:

1. Initialize a variable $S \leftarrow \varnothing$
2. For a plaintext pair (P, P') of SPECK with label 0 or 1, expand (P, P') into 16 pairs using 4 neutral bits.
3. For each of these 16 plaintext pairs (P_i, P'_i), $i \in [1, 16]$:
   (a) Encrypt the plaintext pair (P_i, P'_i) to 8 round using round keys Ks_1 to Ks_8, obtaining ciphertext pair $(C_{i,8}, C'_{i,8})$.
   (b) Keep the value of the bits corresponding to $[a + \alpha, b + \alpha]$ in Ks_8 unchanged, and set the other key bits to zero, thereby obtaining a pseudo-round key PKs for 8 rounds.
   (c) Decrypt $(C_{i,8}, C'_{i,8})$ one round using PKs to obtain $(C_{i,7}, C'_{i,7})$.
   (d) If $(C_{i,7}, C'_{i,7}) = (x_7||y_7, x'_7||y'_7)$, directly compute (y_6, y'_6) according to the SPECK round function. $S = S||(y_6[a-3, b-3], y'_6[a-3, b-3], x_7[a, b], y_7[a, b], x'_7[a, b], y'_7[a, b])$.
4. Return S as a training sample.

4.4 Training *PND* Using Staged Train Method

To facilitate the neural network to learn ciphertext knowledge, we use the staged train method to train *PND* on state bits $[a, b]$ with an input difference Δ for

7 rounds. Firstly, we train a *PND* on state bits $[a-8, b-8]$ for 6 rounds with an input difference Δ, the constant 8 here refers to the value of the round constant α of SPECK64/96. Then we use our 6-round *PND* to recognize 4-round SPECK64/96 ciphertexts with the input difference Δ' (the most likely difference to appear three rounds after the input difference Δ). The training is conducted on 10^7 samples for 20 epochs with a learning rate of 10^{-4}. Then, we train the distinguisher to recognize 7-round SPECK64/96 with the input difference Δ by processing 10^7 freshly generated samples for 10 epochs with a learning rate of 10^{-4}. Finally, the learning rate is reduced to 10^{-5} after processing another 10^7 new instances for 10 epochs. Finally, we train a set of *PND*s for 7-round SPECK64/96.

We use Gohr's network structure to train the distinguisher on full ciphertext. The input difference of this neural distinguisher is the same as the input difference of PND_2. Table 8 summarizes the details of these distinguishers.

Table 8: Neural distinguishers against 7-round SPECK64

Distinguisher	Δ_i	C_i	Acc.
PND_1	$\Delta_{[47]}$	$\{21 \sim 10\}$	0.603
PND_2	$\Delta_{[39]}$	$\{10 \sim 0\}$	0.641
PND_3	$\Delta_{[56]}$	$\{31 \sim 21\}$	0.565
FND	$\Delta_{[39]}$	$\{31 \sim 0\}$	0.623

Key Recovery Attack for 10-Round SPECK64/96 Similarly, we prepend a one-round classical differential $\Gamma_i \to \Delta_i$ (as shown in Table 8) to each distinguisher listed in Table 9, forming an 8-round hybrid distinguisher HD_i. With the addition of the free round, we can conduct a key recovery attack on 10-round SPECK64/96.

Table 9: One-round classical differentials to be prepended to the distinguishers in Table 7

$F_i \to \Delta_i$	NB's	p_i
$\Delta_{[55,52,12]} \to \Delta_{[47]}$	$\{35 \sim 30\}$	2^{-2}
$\Delta_{[47,44,4]} \to \Delta_{[39]}$	$\{25 \sim 20\}$	2^{-2}
$\Delta_{[63,60,20]} \to \Delta_{[56]}$	$\{9 \sim 4\}$	2^{-2}

The entire attack process is executed according to Algorithm 1. During *PND*s recommend partial keys, it is divided into three steps. First, the bits $2 \sim 13$ of the last round subkey are recommended, followed by the bits $0 \sim 2$ and $24 \sim 31$,

and finally, the bits $13 \sim 23$. It should be noted that the 2nd and 13th bits of the key are recommended repeatedly, we only need to select one of the values recommended by the *PND*s.

In the attack, the constant value Q is set to 32. During *PND*s recommend partial keys, the threshold for filtering incorrect keys is $c_1 = c_2 = c_3 = 15$. The value of ϵ is 2, meaning that when PND_2 makes a key recommendation, the used ciphertext structure is saved for *FND*'s final key selection. During *PND*s recommend partial keys, $N_1 = N_2 = N_3 = 15$.

A total of 100 experiments are performed under these experimental settings. The average time consumption of these attacks is 424 seconds. In these 100 experiments, 65 return a final key guess kg. The average Hamming distance is 1.89. Details on $hw(kg, rk)$ are summarized in Table 10, where trial 5 represents these experiments.

Table 10: Statistics on $hw(kg, rk)$ over 100 trials of the 10-round attack on SPECK64/96

$hw(kg, rk)$	0	1	2	3	4	5	6	7	8	9
trial 5	9	22	18	10	2	2	0	0	1	1

5 Conclusion

In this paper, we propose a parallelizable multi-stage key recovery framework based on neural distinguishers, accelerating the key recovery attack on large-state block ciphers. To demonstrate the effectiveness of our proposed framework, we conduct comparative experiments on SPECK64/96 and SPECK96/96. The results show that our proposed framework can effectively speed up the attack process. At the same time, we enhance the ability of the distinguisher in the multi-stage recovery framework to enable it to identify partial bits of higher-round ciphertexts, thereby achieving a 10-round attack on SPECK64/96.

Acknowledgements We would like to thank the anonymous reviewers for their helpful comments and suggestions. This research was funded by National Natural Science Foundation of China under Grant Nos. 62132008, 62372213.

References

1. Bao, Z., Guo, J., Liu, M., Ma, L., Tu, Y.: Enhancing differential-neural cryptanalysis. In: Agrawal, S., Lin, D. (eds.) Advances in Cryptology – ASIACRYPT 2022. vol. 13791, pp. 318–347. Springer Nature Switzerland, Cham (2022). https://doi.org/10.1007/978-3-031-22963-3_11

2. Bao, Z., Lu, J., Yao, Y., Zhang, L.: More insight on deep learning-aided cryptanalysis. In: Guo, J., Steinfeld, R. (eds.) Advances in Cryptology – ASIACRYPT 2023. vol. 14440, pp. 436–467. Springer Nature Singapore, Singapore (2023). https://doi.org/10.1007/978-981-99-8727-6_15
3. Beaulieu, R., Shors, D., Smith, J., Treatman-Clark, S., Weeks, B., Wingers, L.: The SIMON and SPECK lightweight block ciphers. In: Proceedings of the 52nd Annual Design Automation Conference. pp. 1–6. Association for Computing Machinery (2015). https://doi.org/10.1145/2744769.2747946
4. Benamira, A., Gerault, D., Peyrin, T., Tan, Q.Q.: A deeper look at machine learning-based cryptanalysis. In: Canteaut, A., Standaert, F.X. (eds.) Advances in Cryptology – EUROCRYPT 2021. vol. 12696, pp. 805–835. Springer International Publishing, Cham (2021). https://doi.org/10.1007/978-3-030-77870-5_28
5. Biham, E., Chen, R.: Near-Collisions of SHA-0. In: Franklin, M. (ed.) Advances in Cryptology – CRYPTO 2004. vol. 3152, pp. 290–305. Springer Berlin Heidelberg, Berlin, Heidelberg (2004). https://doi.org/10.1007/978-3-540-28628-8_18
6. Biham, E., Shamir, A.: Differential cryptanalysis of DES-like cryptosystems. Journal of CRYPTOLOGY **4**, 3–72 (1991)
7. Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J.D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., Amodei, D.: Language models are few-shot learners. In: Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., Lin, H. (eds.) Advances in Neural Information Processing Systems. vol. 33, pp. 1877–1901. Curran Associates, Inc. (2020)
8. 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. pp. 2722–2730 (2015)
9. Chen, Y., Bao, Z., Shen, Y., Yu, H.: A deep learning aided key recovery framework for large-state block ciphers. Cryptology ePrint Archive, Paper 2022/1659 (2022), `https://eprint.iacr.org/2022/1659`
10. Chen, Y., Shen, Y., Yu, H.: Neural-aided statistical attack for cryptanalysis. The Computer Journal **66**(10), 2480–2498 (2022)
11. Chen, Y., Shen, Y., Yu, H., Yuan, S.: A new neural distinguisher considering features derived from multiple ciphertext pairs. The Computer Journal **66**(6), 1419–1433 (2023)
12. Gohr, A.: Improving attacks on round-reduced Speck32/64 using deep learning. In: Boldyreva, A., Micciancio, D. (eds.) Advances in Cryptology – CRYPTO 2019. vol. 11693, pp. 150–179. Springer International Publishing, Cham (2019). https://doi.org/10.1007/978-3-030-26951-7_6
13. Liu, J., Ren, J., Chen, S.: A deep learning aided differential distinguisher improvement framework with more lightweight and universality. Cybersecurity **6**(1), 47 (2023)
14. Liu, J., Ren, J., Chen, S., Li, M.: Improved neural distinguishers with multi-round and multi-splicing construction. Journal of Information Security and Applications **74**, 103461 (2023)
15. Rombach, R., Blattmann, A., Lorenz, D., Esser, P., Ommer, B.: High-resolution image synthesis with latent diffusion models. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. pp. 10684–10695 (2022)

16. Song, L., Huang, Z., Yang, Q.: Automatic differential analysis of ARX block ciphers with application to SPECK and LEA. In: Liu, J.K., Steinfeld, R. (eds.) Information Security and Privacy. vol. 9723, pp. 379–394. Springer International Publishing, Cham (2016). https://doi.org/10.1007/978-3-319-40367-0_24
17. Zhang, L., Wang, Z., wang, B.: Improving differential-neural cryptanalysis. Cryptology ePrint Archive, Paper 2022/183 (2022), https://eprint.iacr.org/2022/183