Lightweight Neural SBox evaluation

B. Alony[1] and T. Levi[1]

[1]Bar-Ilan University, Ramat Gan, Israel

20 Sept, 2025

Abstract

Substitution Boxes (SBoxes) are fundamental components of modern symmetric cryptography, and their design remains an active area of research. Exhaustively exploring possible mappings is infeasible, as the search space grows exponentially and the calculating evaluation metrics is resource-intensive. In this work, we investigate the use of Binary Neural Networks (BNNs) to accelerate these evaluations. Our initial experiments show promising results on simpler problem instances, highlighting the potential of this approach as a novel research direction.

1 Introduction

Substitution Boxes (S-Boxes) play a central role in symmetric cryptography by introducing the confusion element in block cipher constructions, such as the Substitution–Permutation Network (SPN), the Feistel network, and others. Since the properties of an S-Box directly influence both security and efficiency, the search for "good" S-Boxes — where the definition of "good" depends on the application and the tradeoffs considered — has been an active research topic for decades. Stronger S-Boxes can reduce the number of cipher rounds and lower implementation costs, making their design both practically significant and mathematically challenging.

Many approaches in this field rely on exhaustively searching a subspace believed to have high potential for good solutions. While these methods differ in detail, they all require evaluating every S-Box within the chosen subspace. These computations are time-intensive, so accelerating them would benefit any search method that follows this strategy. Given that S-Box quality metrics are largely linear in nature, we hypothesize that a neural network could efficiently approximate these calculations.

Specifically, we employed Binary Neural Networks (BNNs) to address this challenge, as they are lightweight, simple, and fast to train. We explored various architectures and optimization strategies, achieving a very low false negative rate — a property that is particularly valuable when the goal is to reliably and efficiently identify good solutions.

2 Background

In this section, we provide the preliminaries relevant to this work. We assume the reader is of mathematical or scientific background, but we do not assume any previous knowledge in cryptography or NNs.

2.1 Background: NNs & BNNs

Traditional Neural Networks (Fig 1) consist of layers of interconnected units that compute weighted sums followed by nonlinear activations and are trained end-to-end with backpropagation to minimize a task-specific loss. Depth enables hierarchical feature learning, which has driven strong performance across modalities (vision, audio, text). However, this accuracy typically relies on high-precision (e.g., 32-bit floating-point) parameters and activations, incurring substantial compute, memory bandwidth, and energy costs that favor datacenter-class accelerators. To deploy models under tight resource constraints, engineers commonly apply quantization (e.g., 8-bit integer inference), pruning, and other compression methods, accepting a controlled accuracy–efficiency trade-off. This context motivates more aggressive precision reduction strategies such as binarization.

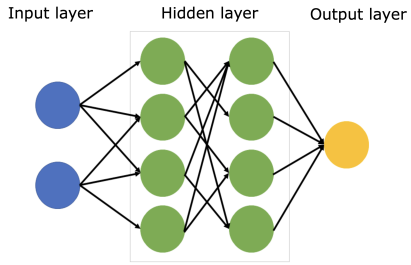

Figure 1: Traditional Neural Network diagram

Binarized Neural Networks (BNNs) constrain weights—and often activations—to a single bit (e.g., $[-1, +1]$), replacing floating-point multiply–accumulates with bit-wise operations (XNOR plus population count). This yields dramatic memory savings (up to $\sim 32\times$ per parameter vs. FP32), high arithmetic intensity, and efficient implementations on FPGAs, ASICs, and modern CPUs that exploit bit-level parallelism. Architecturally, BNNs mirror conventional feed-forward or convolutional networks but incorporate components (e.g., batch normalization, learnable scaling factors, and selective retention of higher precision in the first/last layers) to stabilize training and preserve signal magnitude.

The main challenges are the information loss from 1-bit quantization and the non-differentiability of the binarization function. In practice, training uses surrogate gradients such as the straight-through estimator and regularization/architecture tweaks to mitigate accuracy degradation, which is typically modest on simple datasets but more pronounced on large-scale or fine-grained tasks. Despite these limitations, BNNs occupy a valuable point in the design space: when memory, latency, or energy are primary constraints—such as embedded or edge deployments—BNNs can deliver substantial efficiency gains while maintaining acceptable accuracy, provided their training is carefully engineered and mixed-precision compromises are used where needed

2.2 Background: Block ciphers, SPNs & SBoxes

Theoretically, a **block cipher** is a family of keyed permutations that, with the key unknown, should be indistinguishable from a random permutation. Constructing such families directly is infeasible, so practical ciphers use heuristic structures, the most common being the **Substitution–Permutation Network (SPN)** (Fig 2).

An SPN applies the confusion–diffusion principle: a nonlinear **substitution box (S-Box)** introduces confusion on small input blocks, while a linear permutation layer (P-box) diffuses this effect across the state. Round keys are added each iteration, and the process is repeated for multiple rounds.

Since the S-Box is the only source of nonlinearity, the cipher's security depends critically on its design. Stronger S-Boxes reduce the number of rounds required, directly impacting efficiency.

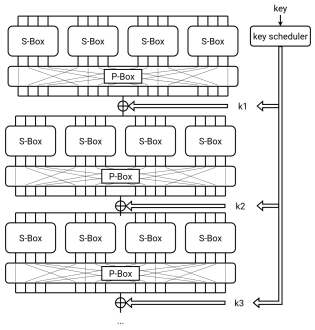

Figure 2: Substitution–Permutation Network (SPN)

2.3 Background: SBox metrics

To evaluate the confusion ability of an S-Box, we have mathematical expressions - metrics - we use to quantify different aspect of its immunity to attacks. These metrics are defined for any Boolean (vectoral) function, $F : \mathbb{F}_2^n \to \mathbb{F}_2^m$. As we are only dealing with Boolean functions, we will abuse the notations and denote such function as such: $F : n \to m$.

In this section and going forward, $\oplus$ represents the exclusive or (XOR) operator, and $a \cdot b$ is the inner-product of two boolean vectors, modulo 2. Lastly, $x[i]$ is the i'th bit of the boolean vector x.

Note: There exist many different SBox metrics, some of which (such as Differential uniformity) we haven't mentions as they are currently irrelevant to our work.

Linear Approximation Immunity

- The *Linearity* of a function, denoted as $\mathcal{L}(F)$, is the following:

$$\mathcal{L}(F) = \max_{u \in \mathbb{F}_2^n, v \in \mathbb{F}_2^m / \{0\}} |\sum_{x \in \mathbb{F}_2^n} (-1)^{v \cdot F(x) \oplus u \cdot x}|$$

  Remark: Many works use a parallel metric called *Nonlinearity*:

$$\mathcal{NL}(F) = 2^{n-1} - \mathcal{L}(F)/2$$

  While it predictably carries the same connotation (in an antonym), the way it is used is different: rather than evaluate the n-bit function as a whole, they evaluate each coordinate function's nonlinearity value, and show the min/max/avg of those values.

- writing a table of the expression inside the maximum for all u, v would result in what is known as the *Linear Approximation Table* (LAT) of the function, the entries of which are sometimes denoted $\widehat{F}(u, v)$.

- The ***Linear Probability*** of a function corresponds to its immunity to Linear approximations. It is denoted as $LP(F)$, and defined as follows:

$$LP(F) = \left(\frac{\mathcal{L}(F)}{2^n}\right)^2$$

For out purposes, we will use LP as our main metric for linear approximation immunity.

The Strict Avalanche Criterion

- The *Strict Avalanche criterion* (SAC) was introduced by A.F. Webster and S.E. Tavares [1], and they word it as follows: "If a cryptographic function is to satisfy the strict avalanche criterion, then each output bit should change with a probability of one half whenever a single input bit is complemented".

  More formally, for every $0 \leq i, j \leq n-1$:

$$\Pr_{x \in \mathbb{F}_2^n}[y[j] = 1 | y = F(x) \oplus F(x_i)]$$

  When x_i is x with the i'th bit flipped.

Algebraic Degree

For an $n \to 1$ Boolean function, the algebraic degree (AD) is defined as the degree of the function's algebraic normal form (ANF) polynomial representation.

It corresponds to the highest degree of any monomial appearing in this polynomial.

In addition to the AD being indicative to the non-linearity of the SBox, it also correlates to the implementation cost.

3 Proposed approach

As mentioned earlier, we use Binary Neural Networks (BNNs) to estimate S-Box metrics. Specifically, we train BNNs on a dataset where the inputs are coordinate ($n \to 1$ bit boolean functions) output vectors and the outputs are binary vectors representing various metric values. The following subsections describe our approach to designing and generating the dataset, as well as the design and training of the BNN model.

3.1 Dataset Design & Generation

Dataset Design

The outputs in our dataset are 9-bit vectors representing three S-Box metrics: Linear Probability (LP), Algebraic Degree (AD), and the Strict Avalanche Criterion (SAC), as described in the Background section.

For LP, we defined four thresholds: 0.0625, 0.140625, 0.25, and 0.5, and for AD, four thresholds: 2, 3, 4, and 5. In the first version of the dataset, the first four bits of the output vector indicate whether the LP of the input is below each corresponding threshold. For example, if the LP value is 0.1, the first four bits are `0111`, since it exceeds the first threshold but is below the remaining three. Similarly, the next four bits encode the same information for AD. The last bit indicates whether the input satisfies the SAC.

In a second version of the dataset, we modified the encoding for the first eight bits to represent exclusive intervals rather than thresholds. In this scheme, a bit is set to 1 if the corresponding metric value falls between the thresholds. Using the previous example with LP = 0.1, the first four bits would now be `0100` instead of `0111`, reflecting that 0.1 lies in the second interval.

Dataset Generation

Pseudo-code for the generation algorithm can be found in Alg 1

To evaluate the quality of the dataset, we examined the distribution of individual bits. As shown in Fig. 3, bit 4 ($AD \leq 2$) appears very infrequently, which causes certain classes to be underrepresented. To address this imbalance, we experimented with removing bit 4 from the dataset (i.e., modifying the labels of each sample accordingly). This reduced the number of underrepresented classes. The resulting class distributions, both with and without bit 4, are presented in Fig. 4.

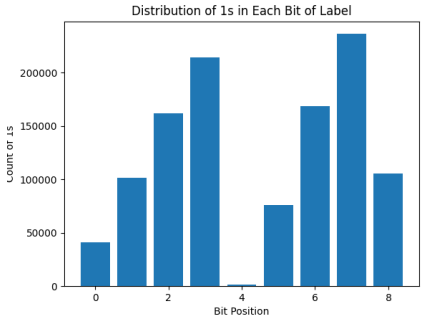

Figure 3: Bit distribution: count of 1's per output bit

Note: Since the dataset was constructed around SBox metrics, and SBoxes with stronger metrics are rare, the resulting dataset exhibits an inherent class imbalance. In particular, classes corresponding to high-quality SBoxes are underrepresented compared to those with weaker metrics. We attempted to mitigate this imbalance by increasing the overall dataset size in order to obtain more samples from the minor-

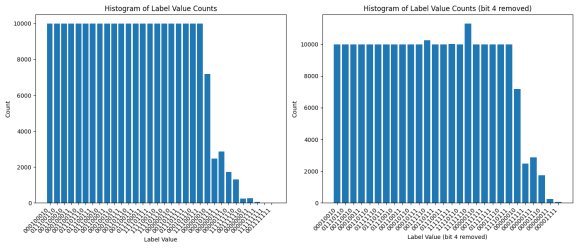

Figure 4: Class distribution: number of samples in each class, with and without bit 4

ity classes. However, this did not lead to a noticeable improvement in performance (as will be discussed in section 4). While the imbalance cannot be fully resolved due to the underlying distribution of SBoxes, we believe that the model can still achieve meaningful performance despite this limitation.

3.2 Model, Training & Evaluation.

Model Architecture

For our task, we chose a relatively simple model architecture to serve as a baseline.

We experimented with fully connected (linear) layers, testing configurations with one or two hidden layers, each containing 256 neurons. Each hidden layer was followed by a ReLU activation function to introduce non-linearity into the network.

In addition, we also explored convolutional architectures with kernel size 3. Specifically, we tested (i) a network with a single convolutional layer with 256 output channels, and (ii) a network with two convolutional layers of 32 and 16 output channels, respectively. These convolutional layers were intended to capture local patterns in the input data before passing the representation into the fully connected layers.

Training Procedure

We trained the Binary Neural Network using a standard technique for BNNs. During training, the model's intermediate activations are kept as real-valued (i.e., not binarized) to ensure that the loss function remains differentiable. This allows gradient descent to effectively update the model's weights. Only during evaluation are the activations binarized to -1 or $+1$, which enables efficient inference while preserving the behavior of a true binary network.

For the training setup, we used a Logistic loss function the Adam optimizer and with a learning rate of 10^{-3}. The dataset was randomly divided into training and test sets, with 80% of the samples used for training and the remaining 20% reserved for testing. We also experimented with different weight decay parameters $(10^{-4}, 10^{-5})$, as well as with no weight decay at all.

Evaluation Criteria

When evaluating the model, we have several metrics to measure its performance:

- **False negative and true negative rates (%):** The proportions of false negatives (**FN**) and true negatives (**TN**) among all samples. These are important for two reasons: (1) FN correspond to good SBoxes that the model failed to identify, representing a loss we want to minimize, and (2) TN correspond to bad SBoxes that are correctly filtered out, which reduces the need for recalculating their metrics and thus provides a speedup over manual evaluation.
- **Test accuracy (%):** The percentage of output bits the model predicted correctly on the test set.
- **Recall:** Given by $\frac{TP}{TP+FN}$, it measures the proportion of positive cases correctly identified. This is especially important for us, as it indicates

the rate of good SBoxes we correctly detect (and miss).

- **Precision:** Given by $\frac{TP}{TP+FP}$, it measures the proportion of samples labeled as positive that are actually positive.
- **F1 score:** Given by $\frac{2 \cdot \text{Recall} \cdot \text{Precision}}{\text{Recall}+\text{Precision}}$, a standard metric that balances precision and recall.

As our goal is to identify good SBoxes and rule out bad ones, the most important metric for us is **Recall**. We also report Test accuracy, as it indicates the model's improvement over training epochs.

4 Results

4.1 Experiment results

From the experimental results, we draw the following conclusions:

- The strongest models were (i) the single linear-layer network trained with no weight decay for 100 epochs, (ii) the two-layer convolutional architecture trained on the same 10-million iteration dataset, and (iii) the two-layer convolutional architecture trained without bits 4–7.
- There appears to be no clear correlation between overall loss/accuracy and the bit 0 metrics. This is likely because loss and accuracy are influenced by all output bits, while our practical use case is primarily concerned with bit 0. This observation suggests that designing a loss function more aligned with our target metric could further improve performance.
- Somewhat counter-intuitively, removing label bits tended to reduce performance rather than improve it. This may indicate that different SBox metrics are correlated in ways that the model is able to exploit.
- The second version of the dataset, which effectively re-encodes the same SBox metrics into more exclusive classes, did not improve results. In fact, it slightly worsened performance.
- Beyond accuracy considerations, convolutional architectures are more computationally expensive and therefore slower to train compared to linear models.

4.2 In Depth Analysis On Best Models

In this section, we will detail more results and deeper analysis on the best performing models: (i) the single linear-layer network trained with no weight decay for 100 epochs, (ii) the two-layer convolutional architecture trained on the same 10-million iteration dataset, and (iii) the two-layer convolutional architecture trained without bits 4–7. For convenience, we have summarized the relevant Models and scores in Table 1

Table 1: results summary of best models.

Model	TN	FN	Recall
(i) 1-layer linear	36.62	1.89	86.23
(ii) 2-layer conv.	8.76	0.48	96.40
(iii) 2-layer conv., without bits 4-7	38.40	1.17	91.43

Firstly, even though model (ii) has great Recall & FN scores, its very low TN means it rules out a very small set of bad options, which means it will not work well for time reduction. Thus, we cannot use it as an acceleration platform. This effect seems to have been caused by a bias on bit 0 (a tendency to set it to 1) - a bias avoided when training on less label bits on model (iii).

Comparing models (i) and (iii) is less straightforward, as they were trained on different dataset variants. Nevertheless, for the specific task of identifying SBoxes with optimal LP (i.e., those with bit 0 set), both models can be applied despite the differences in label structure. The least meaningful comparisons

between them are in terms of loss and accuracy, but this is not a major concern, as we have already established that these metrics are of limited relevance to our goals.

When comparing the two, model (iii) appears to outperform model (i) across all reported metrics and may initially seem like the stronger candidate for an acceleration platform. However, this advantage comes at a cost: convolutional models are considerably slower to evaluate, which can offset the intended time-saving benefits. In addition, they are substantially larger and more complex, making it significantly more expensive to implement in hardware.

5 Conclusions And further Research

In this work, we compared several predictive models for estimating the linearity property (LP) of Boolean functions and evaluated their potential as components of an acceleration platform. Our results show that convolutional models can capture the relevant structure more effectively than simpler fully connected approaches, though they come at the cost of increased computational complexity and hardware requirements.

Overall, the findings are encouraging: if similar results can be obtained for more complex tasks—such as larger input sizes or multiple coordinate functions—this approach has the potential to yield significant time acceleration.

Looking ahead, we envision integrating the model into an acceleration pipeline: first ruling out many bad options automatically, and then applying manual filtering on the reduced candidate set. Exploring this challenge in practice, quantifying the actual time savings, and embedding such accelerators into full-fledged search strategies represent promising directions for future research.

Acknowledgments

This work was carried out as a final project in the Advanced Computer Architectures course at Bar-Ilan University. The authors would like to thank Dr. L. Yavitz for leading the course, and Mr. Z. Jahshan for his guidance and assistance with this project.

Full python code and resources can be found in `https://github.com/Tommyl404/Lightweight-Neural-SBox-evaluation/tree/main`

References

[1] A. F. Webster and S. E. Tavares, "On the design of s-boxes," in *Advances in Cryptology — CRYPTO '85 Proceedings*, H. C. Williams, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1986, pp. 523–534.

Table 2: Results summary across ALL dataset sizes, architectures, and variants.

				Bit 0				
Method	**Train acc (%)**	**Test acc (%)**	**Loss**	TN %	FN %	Recall %	Precision %	F1
Dataset variation								
10k (50 epochs)	95.17	63.35	0.1142	51.10	0.00	100.00	1.33	0.0262
10k (100 epochs)	95.69	57.15	0.1007	49.15	0.00	100.00	0.98	0.0195
10M (50 epochs)	89.79	60.33	0.1980	39.63	3.52	74.23	17.82	0.2874
10M (100 epochs)	90.01	61.22	0.1931	**36.62**	**1.89**	**86.23**	19.22	0.3143
100M (50 epochs)	87.92	60.53	0.2494	38.71	4.43	73.13	21.20	0.3287
100M (100 epochs)	88.20	58.54	0.2457	39.09	4.45	73.28	21.64	0.3341
New_10M (50 epochs)	89.85	62.40	0.1968	39.85	3.06	77.56	18.54	0.2993
New_10M (100 epochs)	90.00	61.26	0.1920	38.54	2.19	84.16	19.61	0.3180
Network Structure and Weight Decay (10M dataset, 100 epochs)								
Single layer — no weight decay	90.01	61.22	0.1931	36.62	1.89	86.23	19.22	0.3143
2 layers — no weight decay	90.15	61.57	0.1920	41.77	4.45	67.58	17.29	0.2750
Single layer, weight decay $1e^{-4}$	87.92	59.17	0.2284	67.41	9.12	32.56	18.75	0.2380
Single layer, weight decay $1e^{-5}$	89.80	60.20	0.1960	40.45	3.19	76.93	18.90	0.3034
2 layers, weight decay $1e^{-4}$	89.82	60.39	0.1952	40.64	3.82	71.85	17.58	0.2825
2 layers, weight decay $1e^{-5}$	94.83	63.85	0.1169	28.47	1.63	88.03	17.20	0.2877
1 layer conv, weight decay $1e^{-5}$	86.56	43.10	0.2528	7.57	0.00	100.00	14.95	0.2601
2 layer conv, weight decay $1e^{-5}$	87.05	62.18	0.2447	**8.76**	**0.48**	**96.40**	14.22	0.2478
Variants (10M, 100 epochs, $1e^{-5}$ **WD)**								
Without bit 4	88.59	62.56	0.2198	42.15	3.98	71.05	18.15	0.2891
Without bits 4–7	82.23	57.39	0.3442	39.57	3.63	73.49	17.73	0.2857
2 layer conv., weight decay $1e^{-5}$**, without bits 4-7**	76.69	43.13	0.4391	**38.40**	**1.17**	**91.43**	20.63	0.3366

Algorithm 1 Dataset Generation

Input: n: S-Box input size, N: number of iterations
Output: Dataset of $(f, label)$ pairs, with max 10k per class

Define $LPThresholds = [0.0625, 0.140625, 0.25, 0.5]$
Define $ADThresholds = [2, 3, 4, 5]$
Dataset $\leftarrow \emptyset$
for $k = 1$ to N **do**
  Generate random S-Box f of size $n \to 1$
  Compute $LP(f)$, $AD(f)$, $SAC(f)$
  Initialize $label = []$
                                                    // Encode LP thresholds
  for each threshold t in $LPThresholds$ **do**
    if $LP(f) < t^*$ **then**
      append 1 to $label$
    else
      append 0 to $label$
    end if
  end for
                                                    // Encode AD thresholds
  for each threshold t in $ADThresholds$ **do**
    if $AD(f) < t^*$ **then**
      append 1 to $label$
    else
      append 0 to $label$
    end if
  end for
                                                    // Encode SAC
  if f satisfies SAC **then**
    append 1 to $label$
  else
    append 0 to $label$
  end if
                                                    // Add to dataset only if label count $\leq$ 10k
  if number of entries in Dataset with this $label < 20{,}000$ **then**
    Add $(f, label)$ to Dataset
  end if
end for
return Dataset

*In the second version, bits are set if the value is between thresholds, rather than bellow them.