NEURAL S-BOX GENERATION FOR LIGHTWEIGHT BLOCK CIPHERS: DESIGN AND INTEGRATION INTO CURUPIRA-1

By

Meruyert Abdulayeva

A THESIS

Submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

In Computer Science

LAWRENCE TECHNOLOGICAL UNIVERSITY

2025

This thesis has been approved in partial fulfillment of the requirements for the Degree of MASTER OF SCIENCE in Computer Science.

Department of Mathematics and Computer Science

Thesis Advisor:	*Dr. Fan Li*
Committee Member:	*Dr. Tao Liu*
Committee Member:	*Dr. Mazin Al-Hamando*
Department Chair:	*Dr. Eric Martinson*

Contents

List of Figures

List of Tables

Definitions

S-box Substitution box. A nonlinear component in block ciphers that replaces input bits with output bits according to a fixed or dynamic mapping to introduce confusion.

Modular Group Transform A transformation applied using modular neighborhood mixing and matrix multiplication to enhance diffusion and disrupt regularity.

Affine Transformation A linear transformation applied to the bits of data using a matrix and a bias vector, used to improve bit mixing and diffusion.

Bijectivity Corrections A correction process that removes duplicates and fixed points to ensure each output value is unique and all input bits influence the output.

Neural Network A machine learning model used here to refine S-boxes based on cryptographic feedback such as nonlinearity and differential uniformity.

MLP Multi-layer perceptron. A type of neural network consisting of fully connected layers, used to refine S-box candidates.

Differential Uniformity A metric evaluating how uniformly differences in input result in differences in output. Lower values indicate stronger resistance to differential attacks.

Walsh Spectrum A measurement used to assess linear correlation in a function. Lower maximum values in the spectrum indicate higher resistance to linear attacks.

Nonlinearity A measure of how different a Boolean function is from all affine functions. High nonlinearity provides better protection against linear cryptanalysis.

Entropy A metric that quantifies randomness in the output. Higher entropy means more unpredictable outputs, increasing cryptographic strength.

Strict Avalanche Criterion A property requiring that a single-bit change in the input flips approximately half the output bits, enhancing diffusion.

Key Schedule An algorithm that generates different subkeys for each encryption round from the original key to prevent key recovery.

Reward-based Loss A loss function in the neural network training that uses cryptographic properties (DU, WL, NL) as objectives to guide learning.

Permutation Layer A component that rearranges data bits or positions to achieve better diffusion in block cipher structures.

Diffusion Layer A linear transformation that spreads the effect of each input bit over many output bits to strengthen security.

List of Abbreviations

AES	Advanced Encryption Standard
AT	Affine Transformation
BC	Block Cipher
CSPRNG	Cryptographically Secure Pseudorandom Number Generator
DC	Differential Cryptanalysis
DU	Differential Uniformity
GAN	Generative Adversarial Network
GA	Genetic Algorithm
GF	Galois Field
IC	Inverse Cipher
IoT	Internet of Things
KS	Key Schedule
LAT	Linear Approximation Table
LC	Linear Cryptanalysis
LWC	Lightweight Cryptography
MDS	Maximum Distance Separable
MLP	Multi-Layer Perceptron
NL	Nonlinearity
NN	Neural Network
SAC	Strict Avalanche Criterion
SPN	Substitution-Permutation Network
S-box	Substitution Box
WL	Walsh Spectrum Bias

Abstract

The expeditious advancement of technology and the proliferation of connected devices have created a growing demand for cryptographic algorithms that balance security strength with implementation efficiency. Lightweight cryptography (LWC) aims to address this need, especially for constrained platforms such as IoT and edge devices. While many existing lightweight ciphers prioritize performance, they often fall short against advanced cryptanalytic techniques. This research investigates the integration of neural networks into the substitution layer (S-box) of the CURUPIRA-1 block cipher to improve its cryptographic strength. The goal is to enhance resistance to differential and linear attacks while retaining the cipher's modular and lightweight properties. By refining the S-box using neural-guided optimization strategies, the proposed method introduces cryptographic adaptability and dynamic instance-specific generation, which may support future deployment in contexts like IoT without compromising core security properties.

Chapter 1

Introduction

The proliferation of connected devices and the expeditious advancement of technology have created a pressing need for efficient cryptographic mechanisms capable of safeguarding sensitive data in resource-constrained environments. The Internet of Things (IoT) and embedded systems, projected to reach over 25 billion devices by 2030 [15], are increasingly integrated into critical infrastructures such as healthcare, transportation, and industrial automation. Although these devices are optimized for minimal memory usage and low power consumption, providing strong and flexible security remains a challenge—particularly as cryptographic components must remain resilient over long deployment lifespans in the face of evolving threats.

While many IoT compromises have exploited outdated firmware and misconfigurations [8], these incidents reflect a broader challenge: as IoT adoption grows, the need for secure lightweight cryptographic solutions becomes increasingly urgent. Among the key limitations of existing lightweight designs is their reliance on static architectural components, which risk becoming outdated as threat landscapes evolve. One

such component is the substitution box (S-box), which provides the crucial nonlinear transformations required in symmetric encryption.

Although lightweight block ciphers such as CURUPIRA-1 achieve efficiency through carefully designed static S-boxes, their reliance on a single, unchanging structure across millions of devices creates a monoculture risk—a risk that proactive diversification can mitigate: should that structure become cryptanalytically weakened, it would expose all dependent systems simultaneously. While such failures have not yet been observed in lightweight S-boxes, the history of cryptographic primitives demonstrates that proactive diversification is a prudent strategy for long-term resilience.

This work introduces neural network–generated S-boxes as a strategy to proactively diversify cryptographic structures, reducing monoculture risk and introducing adaptive, high-entropy nonlinear layers without undermining the lightweight design principles necessary for IoT environments. Rather than responding reactively to attacks, this approach builds cryptographic agility directly into the cipher's confusion layer, aligning with long-term security needs for embedded systems.

1.1 Lightweight Cryptography

Lightweight cryptography (LWC) addresses this demand by providing cryptographic algorithms specifically optimized for environments with restricted memory, processing power, and energy consumption. These algorithms aim to achieve a balance between efficiency and security, ensuring data confidentiality, integrity, and availability even under constrained conditions. However, despite notable progress in this field, many existing lightweight cryptographic algorithms remain vulnerable to advanced cryptanalytic techniques such as differential, linear, and algebraic attacks. Their reliance on

static operations and fixed transformation layers—as seen in the design of lightweight block ciphers such as CURUPIRA-1—limits adaptability and increases predictability over time, posing a remarkable threat in adversarial contexts.

Tables 1.1, 1.2, 1.3, and 1.4 present a comparative analysis of several prominent lightweight block ciphers. Tables 1.1 and 1.2 highlight architectural trade-offs, including block and key sizes as well as hardware area. Table 1.3 emphasizes power consumption metrics relevant to energy-constrained platforms, while Table 1.4 isolates throughput performance and efficiency. Among the ciphers evaluated, CURUPIRA-1 stands out for its strong balance of performance and cryptographic resilience. It achieves a throughput of 960 Kbps—the highest among the compared ciphers—and supports a 96-bit block size, which enhances diffusion and contributes to higher round-wise complexity. With only 10 encryption rounds, CURUPIRA-1 achieves fast processing while maintaining a consistent throughput-to-area ratio of 0.1151 Kbps/GEs. Although it incurs a relatively high hardware footprint of 8334 gate equivalents (GEs) and a total leakage power of 103 µW, its low dynamic power consumption (1.1 µW) makes it well-suited for mid-range IoT environments such as smart meters and edge routers. These characteristics position CURUPIRA-1 as a promising candidate for further optimization, particularly through the enhancement of its static components. Nevertheless, the cipher's static transformation layers raise concerns regarding long-term cryptographic resilience, motivating the integration of adaptive S-box designs through neural network enhancement for CURUPIRA-1.

Substitution–Permutation Network (SPN) based designs, such as CURUPIRA-1, offer notable structural advantages over Feistel-based lightweight ciphers, including DESL, DESXL, HIGHT, and XTEA, which are also commonly employed in constrained environments [13]. In Feistel architectures, the plaintext block is divided into two halves, and only one half undergoes transformation in each round, while the other is updated through simple XOR-based mixing. Although this structure offers simplicity

and ease of implementation, it often results in weaker diffusion per round compared to SPN ciphers, which apply substitution and permutation operations across the entire block in every round. This limited diffusion can make Feistel-based schemes more susceptible to linear and differential cryptanalysis, especially when operating under tight resource constraints. In contrast, SPN-based ciphers like CURUPIRA-1 provide full-block transformations that ensure higher round-wise complexity and stronger resistance to various cryptanalytic attacks, making them more suitable for security-critical applications despite a slightly higher implementation overhead.

Table 1.1
Block and Key Sizes of Lightweight Block Ciphers

Cipher	**Block Size (bits)**	**Key Size (bits)**
DESL	64	56
DESXL	64	184
CURUPIRA-1	96	96
CURUPIRA-2	96	96
HIGHT	64	128
PUFFIN	64	128
PRESENT	64	80
XTEA	64	128

Table 1.2
Computational Complexity and Hardware Footprint of Lightweight Block Ciphers

Cipher	**Cycles/Block**	**Area (GEs)**
DESL	16	2762
DESXL	16	3082
CURUPIRA-1	10	8334
CURUPIRA-2	10	7334
HIGHT	32	3901
PUFFIN	32	2303
PRESENT	31	1704
XTEA	32	3490

Table 1.4 is derived from Table 1.1, isolating throughput performance to better highlight efficiency

Table 1.3
Power Consumption Metrics of Lightweight Block Ciphers

Cipher	Total Power (µW)	Leakage Power (µW)	Dynamic Power (µW)
DESL	30	275.5	0.275
DESXL	36	303	0.303
CURUPIRA-1	117.1	103	1.1
CURUPIRA-2	98	750	0.75
HIGHT	44.5	511.5	0.5115
PUFFIN	25	318	0.318
PRESENT	20	242	0.242
XTEA	61	438.5	0.4385

Table 1.4
Throughput Efficiency of Lightweight Block Ciphers

Cipher	Throughput (Kbps)	Throughput/Area (Kbps/GEs)
DESL	400	0.1448
DESXL	400	0.1297
CURUPIRA-1	960	0.1151
CURUPIRA-2	960	0.1308
HIGHT	200	0.0512
PUFFIN	200	0.0868
PRESENT	206.4	0.1211
XTEA	200	0.0573

trade-offs among lightweight block ciphers.

Note: Power metrics for Curupira-1 and Curupira-2 are based on synthesis and reference sources, not on the original cipher specification [6]. Dynamic power values are presented in µW for consistency.

1.2 Problem Statement

While lightweight block ciphers such as CURUPIRA-1 offer a promising foundation for cryptographic protection in constrained environments, their reliance on static substitution layers presents

a fundamental limitation. These fixed S-boxes, although optimized for hardware efficiency, lack the adaptability required to counter emerging cryptanalytic techniques, including differential and linear cryptanalysis. Likewise, current lightweight ciphers often struggle to balance strong cryptographic metrics with implementation feasibility in constrained platforms.

This thesis addresses the challenge of enhancing the cryptographic strength of lightweight block ciphers while preserving implementation modularity. Specifically, it explores the integration of neural network–guided transformations into CURUPIRA-1's substitution layer, introducing adaptability and improving resistance to modern attack vectors. While the approach introduces additional computational complexity, the design emphasizes a modular architecture suitable for future optimization, making it a viable direction for deployment in embedded and resource-constrained environments.

1.3 Research Objectives

This thesis investigates the application of neural networks to enhance the cryptographic strength of lightweight block ciphers, with a specific focus on the CURUPIRA-1 algorithm. The central objective is to improve critical components, particularly the substitution box (S-box), by introducing adaptive machine-learned transformations that maintain cryptographic resilience while remaining computationally practical. The research seeks to:

† Design and integrate neural network–generated S-boxes to replace traditional static structures.

† Evaluate cryptographic properties such as differential uniformity (DU), Walsh spectrum bias (WL), nonlinearity (NL), algebraic degree, entropy, and Strict Avalanche Criterion (SAC) using a combination of traditional and neural-enhanced approaches.

† Ensure that the resulting S-boxes maintain bijectivity and eliminate undesirable patterns such as fixed points or linear clusters.

1.4 Contribution

The primary contribution of this thesis is the formulation of a novel pipeline that integrates evolutionary algorithms and neural network feedback to dynamically construct cryptographically strong S-boxes. Specifically, the proposed method:

† Ensures high nonlinearity and low differential uniformity in generated S-boxes through evolutionary search and affine layer optimization.

† Adapts the substitution structure via reward-based neural network training, improving cryptographic metrics like Walsh bias and avalanche behavior.

† Integrates these dynamic S-boxes into the CURUPIRA-1 cipher, demonstrating enhanced resistance to differential and linear attacks while maintaining efficiency suitable for lightweight environments.

Chapter 2

Curupira-1

2.1 Overview of Curupira-1

Regarding Lightweight Cryptography (LWC), this study selects the Substitution-Permutation Network (SPN)-based block cipher CURUPIRA-1 as the foundation for analysis. CURUPIRA-1 is a highly efficient cryptographic algorithm designed for resource-constrained environments. The SPN structure relies on multiple rounds of substitution and permutation operations to achieve confusion and diffusion, along with a key schedule that ensures that each round uses a unique subkey derived from the original key. CURUPIRA-1, as an SPN-based cipher, implements this structure with a 96-bit block size and a 96-bit key size, providing a strong security level relative to other lightweight algorithms [23].

CURUPIRA-1 employs several transformations that work together to achieve diffusion and confusion in the cipher, two critical components of any secure block cipher. These transformations include a nonlinear substitution layer (γ), a permutation layer (π), a linear diffusion layer (θ), and key addition operations ($\sigma[k]$), all structured to provide high security while maintaining efficiency.

The core hardware architecture of CURUPIRA-1, including the proposed neural network–enhanced substitution layer γ, is illustrated in Fig. 2.1.

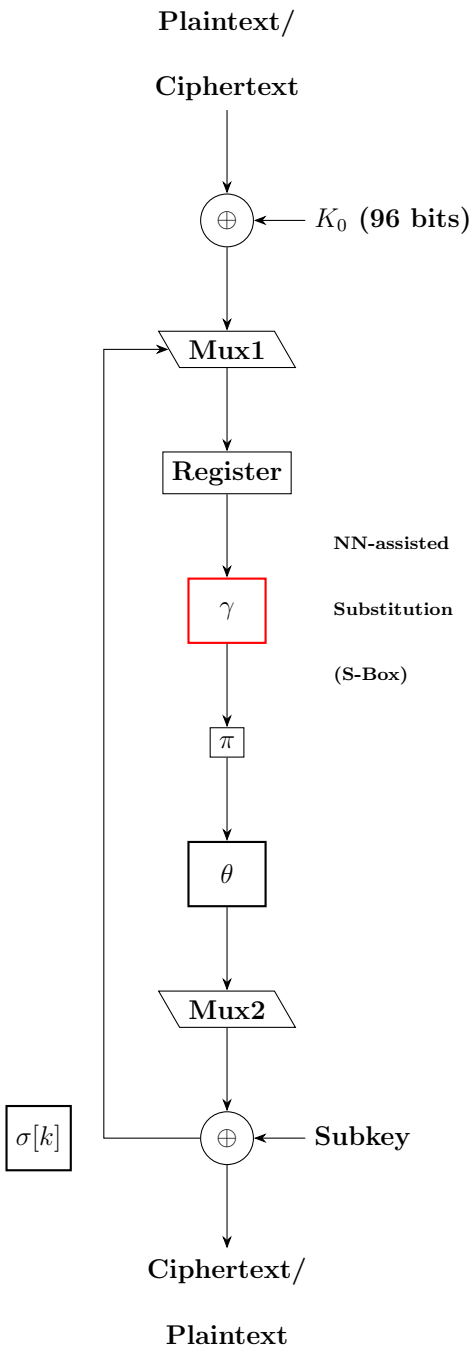

Figure 2.1: Simplified CURUPIRA-1 architecture enhanced with neural network–assisted substitution layer γ.

The schematic representation of CURUPIRA-1 in Fig. 2.1 includes additional hardware-level components such as `Mux1`, `Mux2`, and a `Register` block. These elements are part of the control and data flow logic in a practical implementation. `Mux1` and `Mux2` represent multiplexers that control the input and output flow between cipher stages and facilitate feedback during iterative round processing. The `Register` stores intermediate state data between rounds and ensures synchronization of the pipeline. Although these components are not part of the logical cipher definition, they are relevant for understanding the structural pipeline of CURUPIRA-1 in hardware or simulation environments.

The cipher begins by taking a 96-bit data block, represented as a 4×3 matrix M_4, and a key matrix K. The data block is processed through several stages, each designed to restructure the data so as to enhance security and make cryptanalysis more difficult. The process of CURUPIRA-1 is provided as follows:

(a) Substitution Layer (γ): The first transformation applied in each encryption round is the substitution layer, denoted as γ. This nonlinear layer introduces confusion by applying substitution operations to each byte of the input, thereby obscuring the relationship between plaintext and ciphertext. The substitution function S in CURUPIRA-1 is designed to resist differential and linear cryptanalysis, operating in parallel across a 4×3 matrix of 8-bit state bytes, totaling 96 bits.

Each 8-bit input byte is first split into two 4-bit halves (nibbles). These halves are independently substituted using two 4-bit mini S-boxes, labeled P and Q, which are referred to as miniboxes. P typically processes the higher nibble, and Q processes the lower nibble. The mini S-boxes are isogonal—mathematically related in structure—to ensure cryptographic strength and structural symmetry. The substituted halves are then recombined to produce the final 8-bit output byte.

Formally, the substitution layer $\gamma(a)$ transforms the input state matrix a element-wise:

$$b = \gamma(a), \tag{2.1}$$

where $b_{i,j} = S(a_{i,j})$, with $S(a_{i,j})$ representing the composite function of the P and Q minibox transformations applied to the nibbles of byte (i, j).

Internally, CURUPIRA-1 applies this substitution structure in parallel across all 12 bytes of the state. Fig. 2.2 illustrates the internal structure of one such S-box unit, showing the nibble split, substitution using P and Q miniboxes, and the recombination process. Interconnections between P_i and Q_j units further enhance local diffusion and increase resistance to structural attacks.

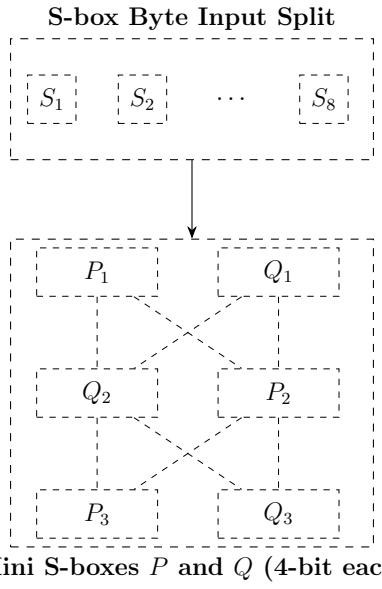

Figure 2.2: Structure of a single substitution box S in CURUPIRA-1. Each 8-bit input byte is split into two 4-bit halves (nibbles), which are processed through two separate 4-bit mini S-boxes: P and Q. These miniboxes are isogonal—mathematically related—to maximize cryptographic strength. The substituted halves are recombined into an 8-bit output byte. This substitution structure is applied in parallel across the 4×3 byte matrix (96-bit state) in each encryption round.

(b) Permutation Layer(π): After the substitution layer, the data undergoes a permutation. The permutation layer $\pi(a)$ swaps the columns of the state matrix, which further diffuses the data. The permutation operation is defined as:

$$c = \pi(b), \tag{2.2}$$

which is equalivent to:

$$c_{i,j} = b_{i,(i \oplus j)}, \tag{2.3}$$

where $c_{i,j} = b_{i,(i \oplus j)}$ indicates that each element $b_{i,j}$ in the output matrix is derived from the same row i of the input matrix a, but the column index $(i \oplus j)$ is determined by XORing the row index i with the column index j. This ensures that the positions of the bits are rearranged, adding another layer of complexity to the cipher.

(c) Diffusion Layer (θ): Following the permutation, the cipher applies a linear diffusion layer $\theta(c)$. This layer performs a matrix multiplication with a specially designed matrix D, ensuring that any changes in the input data are spread across the entire state. The transformation is defined as:

$$d = \theta(c) = D \cdot c, \tag{2.4}$$

where d is the state matrix output from the diffusion layer and D is a 3×3 matrix derived from a Maximum Distance Separable (MDS), chosen to guarantee strong diffusion. This diffusion layer further obscures the relationship between the input and output, making it significantly more difficult for attackers to detect patterns that might reveal information about the plaintext.

(d) Key Addition ($\sigma[k]$): Next, the cipher incorporates a key addition operation. The key is added to the state matrix through a bitwise XOR operation, which is a simple yet effective method for mixing the key(k) into the data. The key addition $\sigma[k](d)$ is defined as:

$$\sigma[k](d) = d \oplus k. \tag{2.5}$$

This ensures that the key influences every byte of the state, adding a layer of security by making it difficult for attackers to isolate the key's effects from the effects of the data transformations.

(e) Key Scheduling (ψ): The key schedule in CURUPIRA-1 evolves the key for each round using two linear transformations. These transformations are designed to preserve that the key changes in a non-trivial way between rounds, which is important for maintaining security against related-key attacks. The key scheduling process is defined as:

$$\psi_r(K) = \omega^r(K) + Q(r), \tag{2.6}$$

where ω denotes a key transformation function composed of bitwise rotations and linear operations applied to the original key material, and $Q(r)$ represents round-specific constants derived from the S-box. This ensures that the round keys are generated dynamically and uniquely for each round, further complicating the task for attackers trying to break the cipher.

The complete cipher function can now be described as a sequence of transformations, starting with initial key addition, followed by substitution, permutation, diffusion, and final key addition operations. This sequence is repeated for a number of rounds, each with its own round key generated through the key schedule. The overall encryption process is expressed as:

$$Curupira[K] \equiv \sigma[\kappa(R)] \circ \pi \circ \gamma \circ \prod_{r=1}^{R-1} (\sigma[\kappa(r)] \circ \theta \circ \pi \circ \gamma) \circ \sigma[\kappa(0)], \tag{2.7}$$

where θ represents the diffusion layer, which spreads the influence of the input bits across the block to ensure strong mixing and resistance against differential and linear cryptanalysis. γ denotes the substitution layer, which introduces non-linearity into the cipher by using S-box transformations. R refers to the number of encryption rounds. In CURUPIRA-1, $R = 10$ is typically chosen to achieve a balance between cryptographic security and computational efficiency. The symbol $\circ$ denotes function

composition, and operations are applied from right to left—that is, the rightmost function $\sigma[\kappa(0)]$, initial key addition, applied first, followed by the round functions, and ending with the final key addition $\sigma[\kappa(R)]$. This reflects the sequential nature of transformations in block cipher encryption, where each layer progressively modifies the data to increase confusion and diffusion. Additionally, $\kappa(r)$ represents the round subkeys, and the round function for the r-th round is defined as above.

$$\rho[\kappa(r)] \equiv \sigma[\kappa(r)] \circ \theta \circ \pi \circ \gamma. \tag{2.8}$$

This function defines the iterative process of encryption, where each round further mixes the data, applying the key and transforming the state using the substitution, permutation, and diffusion layers.

CURUPIRA-1 is designed to be an involutional cipher, which means that the encryption and decryption processes are nearly identical, differing only in the key schedule. This property allows for efficient decryption, as the same operations can be applied in reverse order, with the only difference being the application of the inverse round keys. The decryption key schedule($\kappa^{\bar{r}}$) is defined as:

$$\kappa^{\bar{r}} = \theta(\kappa(R - r)) \quad \text{for} \quad 0 < r < R. \tag{2.9}$$

Thus, the inverse cipher function is:

$$\alpha^{-1}[\kappa(0), \ldots, \kappa(R)] = \alpha[\kappa^{\bar{0}}(R), \ldots, \kappa^{\bar{R}}(0)]. \tag{2.10}$$

This involutional property simplifies the decryption process and ensures that the cipher is efficient and secure.

2.2 Design Characteristics and Cryptographic Evaluation

Although CURUPIRA-1 delivers high throughput (960 Kbps) due to its larger 96-bit block size and a moderate number of rounds (10), it exhibits a relatively large area footprint (∼8334 Gate Equivalents, GEs) and higher power consumption (117.1 µW), which may limit its suitability for extremely constrained platforms such as RFID tags. In contrast, more compact ciphers like PRESENT and PUFFIN offer reduced area and energy consumption at the cost of throughput or flexibility. Nevertheless, CURUPIRA-1 remains attractive for high-speed applications that can accommodate a slightly higher resource overhead.

This research aims to enhance CURUPIRA-1 by integrating neural networks into its critical components, particularly the S-box, to improve its efficiency and security while preserving its high throughput, thereby making it more adaptable for modern lightweight cryptographic requirements.

CURUPIRA-1 operates on 96-bit data blocks and supports key sizes of 96, 144, or 192 bits. It incorporates a Substitution-Permutation Network (SPN) structure optimized for lightweight operation, while its design employs the Wide Trail strategy to maximize security. This ensures that small changes in the plaintext lead to pronounced, unpredictable changes in the ciphertext, achieving strong diffusion. The cipher's key schedule and transformation layers are crafted to resist differential, linear, and algebraic cryptanalysis, providing resilient security optimized to modern lightweight cryptographic needs.

Chapter 3

Literature Review

3.1 Related Works

As previously discussed, our focus is on strengthening the key component of the CURUPIRA-1 cipher using neural networks. This component, the S-box, is critical for cipher performance and security. Enhancing this component is expected to improve the overall performance and resilience of CURUPIRA-1. Moreover, the proposed approach has potential applicability to other SPN-based lightweight cipher designs.

The substitution box (S-box) is the primary source of nonlinearity in block ciphers, providing essential confusion properties. Designing secure and efficient S-boxes remains a critical task, especially for lightweight cryptographic algorithms. This section reviews the state-of-the-art in S-box construction and evaluation, with a focus on neural network-based methods, adaptive transformations, and algebraic techniques. Our proposed framework builds upon and differentiates itself from these works by synthesizing ideas into cryptographically-evaluated, lightweight, and dynamic S-box generator.

3.1.1 Permutation-Based Methods

Fisher-Yates Shuffle-Based S-box Generation. Tayel et al. [40] proposed an alternative S-box generation method that employs the Fisher-Yates shuffle combined with chaotic maps, such as the Ikeda map, to introduce randomness and nonlinearity. This method has been proposed as an enhancement for 4-bit S-boxes similar to those used in ciphers like Serpent, where permutation indices are derived from chaotic iterations. The resulting S-boxes demonstrated favorable properties in strict avalanche criterion (SAC), nonlinearity, and bijectivity, with an average nonlinearity score of 4 and differential probability (DP) bounded by 6. While effective for applications requiring compact 4-bit S-boxes, such as lightweight variants of Serpent, its application to 8×8 S-boxes—like those required for AES or CURUPIRA-1—is less direct. This motivated our exploration of neural-guided methods that better support full 8×8 permutations for contemporary block ciphers.

3.1.2 Cryptographic Metric Evaluation Platforms

Evaluation-Centric Approaches – Peigen. Bao et al. [37] introduced Peigen, a comprehensive platform for S-box generation, evaluation, and classification. It supports a wide range of cryptographic metrics—including difference distribution tables (DDT), linear approximation tables (LAT), algebraic degree, (v, w)-linearity, affine equivalence, and gate-level profiling—enabling detailed structural and cryptanalytic analysis. Its classification system outputs permutation-equivalent groups of S-boxes and associates them with metric-specific folders. However, Peigen is fundamentally designed for static S-box generation and does not incorporate neural methods or runtime feedback loops, making it less suited for adaptive pipelines or real-time S-box construction. Additionally, while exhaustive metric evaluation provides high confidence in classification, it incurs computational overhead that limits scalability in high-throughput applications involving 8×8 S-boxes.

For lightweight cryptographic designs such as those used in IoT communication protocols, a smaller but critical subset of evaluation metrics—namely, the difference distribution table (DDT), linear approximation table (LAT), and algebraic degree—are typically sufficient to evaluate resilience against the most common attacks, including differential, linear, and algebraic cryptanalysis. These metrics are especially relevant in session-level encryption or constrained environments where lightweight ciphers are tested primarily for their resistance to such classes of attacks.

3.1.3 Neural Network-Based Approaches

GAN-Based S-box Generation. Zhang et al. [45] proposed a methodology using Generative Adversarial Networks (GANs) to construct 8×8 S-boxes optimized for cryptographic metrics such as differential uniformity (DU) and nonlinearity (NL). Their pipeline incorporates affine-equivalence–based preprocessing and cryptographic loss terms—targeting DU, NL, and bijectivity—directly into the GAN training objective. The authors trained and evaluated several GAN variants, including DCGAN, WGAN, WGAN-GP, and WGP-IM. Among these, WGP-IM achieved the best balance between cryptographic quality and training stability, producing S-boxes with DU = 8 and NL = 104. Earlier models such as DCGAN yielded similar nonlinearity but higher differential uniformity (DU = 10).

While promising, these models involve considerable computational requirements, primarily due to their convolutional architectures and extended training times. Additionally, the stochastic nature of latent vector sampling can make it challenging to consistently enforce strict structural properties, such as bijectivity or fixed-point elimination, often addressed through supplementary postprocessing. Furthermore, standard GAN-based frameworks typically do not incorporate per-instance cryptographic feedback during generation, which may limit their applicability in certain adaptive or resource-constrained settings.

The computational infrastructure for training these models includes high-performance hardware, notably an Intel Xeon E7-4850 CPU paired with an NVIDIA Quadro M6000 GPU, utilizing TensorFlow-GPU 2.5.0 under Windows Server 2012 R2. Each GAN model was configured with different optimizers—Adam or RMSProp—with learning rates ranging from 0.0001 to 0.0008. Batch sizes were set to 128, and the number of training iterations extended up to 5000 epochs. While these settings facilitated convergence to high-quality cryptographic properties, they also introduced significant computational and temporal costs.

Table 3.1
GAN-Based S-box Generation Methods

Aspect	**GAN-Based Methods (e.g., Zhang et al., WGP-IM)**
Model Type	GAN (DCGAN, WGAN, WGAN-GP, WGP-IM)
Architecture Complexity	High (Convolutional layers, latent sampling)
Training Configuration	Up to 5000 epochs; batch size 128; optimizers include Adam and RMSProp
Hardware Requirements	Intel Xeon CPU; NVIDIA Quadro M6000 GPU; TensorFlow-GPU 2.5.0
Bijectivity Guarantee	Often requires postprocessing to enforce strict bijection
Cryptographic Metrics	Differential Uniformity (DU): 8–12; Nonlinearity (NL): 92–104

MLP-Based Boolean Function Learning. Zhang et al. [46] introduced a multi-layer perceptron–based framework for S-box implementation, where each perceptron handles only 4 input bits. Their design decomposes complex 8-bit Boolean functions into smaller subcomponents, implemented through a specialized 3-bit perceptron architecture known as 172P. This approach enables the construction of cryptographic S-boxes by hierarchically assembling 16 simple perceptrons (SLPs or MLPs) and aggregating their outputs.

The training process relies on a DNA-inspired learning algorithm that simplifies threshold and weight configuration. While the model demonstrates conceptual clarity, its fixed structure and lack of backpropagation limit its adaptability and integration with modern machine learning toolchains such as PyTorch or TensorFlow.

GA + NN Hybrid Approaches. Rong et al. (2025) [36] proposed a hybrid S-box generation method that combines genetic algorithms (GA) with neural network (NN) classifiers. Their framework integrates a multi-layer perceptron (MLP) with three hidden layers (128–128–64) and Leaky ReLU activation to perform pairwise comparisons of 6-bit S-box candidates based on cryptographic metrics such as nonlinearity and differential uniformity. The neural model achieves approximately 85% classification accuracy, providing an approximate alternative to traditional fitness evaluations by predicting which S-box in a given pair is stronger. However, the model is currently limited to perform pairwise comparison and does not support generation or refinement of full 8-bit S-boxes as standalone outputs.

Their experiments target the optimization of S-boxes toward low differential uniformity (DU) and high nonlinearity (NL), with both properties being weighted equally in the fitness evaluation by setting weighting coefficients $a = 1$ and $b = 120$. To support these evaluations, the authors developed a custom comparison algorithm `compare()`, which estimates the relative "survivability" of S-box candidates based on the fitness value. The evolutionary search employed standard GA operators, with performance evaluated on a 6-bit S-box benchmark where nonlinearity was approximately between 10 and 20, and DU ranged from 6 to 10.

Table 3.2
GA + NN Hybrid Approaches Experimental Environment Parameters
(Rong et al. [36])

Category	**Parameter Values**
OS	Windows 11
CPU	Intel Core i9-11950H
GPU	RTX 3080
CUDA	CUDA 11.0

A recurring pattern in the literature suggests that most existing methods tend to focus on generation (e.g., GANs and GAs), evaluation (e.g., Peigen), or structure (e.g., Zhang's MLP), yet relatively few have attempted to unify these objectives into a single, adaptive, and metric-aware framework. Our proposed system addresses this by integrating modular transformation layers with MLP-based reward learning, enabling the generation of per-instance optimized S-boxes that satisfy cryptographic criteria and structural resilience simultaneously. As a reference point, it is instructive to revisit the AES S-box—arguably the most well-established substitution layer in symmetric-key cryptography—as both a benchmark and a reference point. We now provide a focused discussion on the AES S-box structure, its cryptographic metrics, and its implications for lightweight and adaptive cipher design.

3.2 AES S-Box

Given the wide adoption and extensive analysis of the AES S-box, it functions as a natural benchmark for evaluating the cryptographic strength of newly proposed S-boxes. Although our design goals

differ—emphasizing runtime adaptability and lightweight deployment—we adopt AES-level metrics as reference thresholds for nonlinearity, differential uniformity, and Walsh bias. The following section outlines the structure and cryptographic properties of the AES S-box, establishing a baseline for comparison with our dynamic, neural-enhanced alternatives.

The AES S-box remains one of the most influential and extensively analyzed nonlinear components in symmetric-key cryptography. It is constructed through a two-step process: a multiplicative inversion in the finite field $\mathbb{GF}(2^8)$, followed by an affine transformation over $\mathbb{F}_2^8$. This layered construction offers a balanced blend of confusion and diffusion—two critical design goals in block cipher security—and is mathematically defined as:

$$S(x) = A \cdot \mathrm{Inv}(x) + b,$$

where $\mathrm{Inv}(x)$ denotes the multiplicative inverse in $\mathbb{GF}(2^8)$ (with $\mathrm{Inv}(0) = 0$ by convention), A is an invertible 8×8 binary matrix, and b is a constant bias vector.

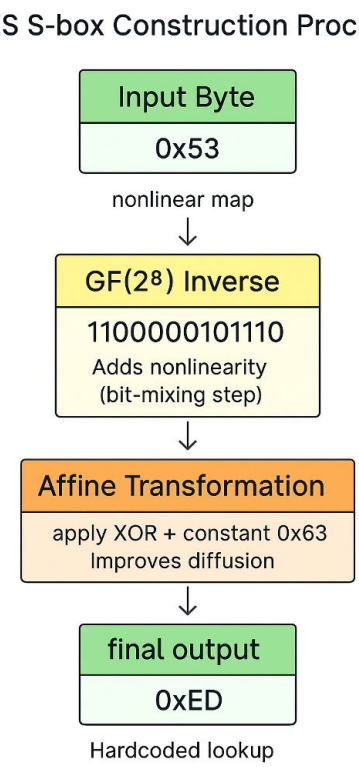

Figure 3.1: AES S-box construction process. The transformation consists of inversion in $GF(2^8)$ followed by an affine transformation, providing strong nonlinearity and diffusion.

The AES S-box exhibits strong cryptographic characteristics. Its differential uniformity (DU) is equal to 4, which is the optimal value for bijective functions and ensures minimal probability of differential collisions. The maximum absolute value in the Walsh spectrum (WL) is 16, indicating the degree of linear correlation between input and output bits in the AES S-box. Its Boolean functions exhibit a nonlinearity of 112, one of the highest achievable values for 8-bit bijective S-boxes. Additionally, the algebraic degree of its component Boolean functions equals 7, which further complicates algebraic attacks. The entropy of output bits approaches the ideal of 1.0 per bit, and the substitution layer satisfies the strict avalanche criterion (SAC), meaning that a single-bit change in input tends to flip about half of the output bits on average.

Despite these excellent metrics, the AES S-box is not without limitations—especially when viewed through the lens of lightweight cryptography and adaptive security models. Its structure is static and identical across all AES implementations—an advantage for analysis, but a potential liability in adversarial settings involving side-channel or structural attacks. Moreover, the finite field inversion imposes a non-trivial computational cost that may not scale well in highly constrained environments such as IoT or embedded devices. While efficient table lookups can mitigate this, dynamic memory or hardware constraints can still pose challenges.

Perhaps most importantly, the AES S-box was not designed for runtime reconfiguration or adaptive deployment. It cannot provide per-session or per-device uniqueness, which is increasingly important in the face of escalating threat models that emphasize adaptability and obfuscation. Consequently, although the AES S-box remains a gold standard in cryptographic evaluation due to its well-established security properties, its fixed design offers limited flexibility for deployment in lightweight or context-aware environments with increasing demands for runtime adaptability and structural diversity.

In this work, we adopt the AES S-box's cryptographic thresholds—such as DU ≤ 8 and NL ≥ 104—as soft optimization targets. However, we depart from static design in favor of an adaptive, instance-specific S-box generation pipeline that incorporates modular transformations, affine scrambling, and neural network–based refinement. This enables the creation of cryptographically strong S-boxes that retain AES-level metrics while offering runtime diversity and improved resistance to structural exploitation.

Chapter 4

The Proposed Scheme

Curupira-1's current structure follows the Substitution-Permutation Network (SPN) framework, which processes the entire data block in each encryption round. Although this structure ensures strong diffusion and confusion, it also introduces limitations due to the static nature of its components, specifically the S-boxes. To address these weaknesses and improve the adaptability and security of Curupira-1, we propose integrating neural networks (NN), a class of machine learning models, into these components.

Although neural networks are often associated with computational overhead, our architecture integrates them efficiently by generating S-boxes at runtime. Rather than minimizing load alone, the system prioritizes adaptability, cryptographic strength, and resistance to static-pattern exploitation. Our scheme constructs the S-box in two integrated phases, beginning with a five-layer cryptographic preprocessing pipeline comprising a modular group transform (T_k), two affine layers, logistic or

chaotic scrambling, and fixed-point correction. Each layer is designed to strengthen key cryptographic attributes such as nonlinearity, differential uniformity, and resistance to targeted cryptanalysis.

Following preprocessing, the second phase applies a reward-driven neural network model to further refine the S-box candidates. A Multi-Layer Perceptron (MLP) is employed, trained with a carefully constructed loss function that prioritizes low differential uniformity, low Walsh distribution, high nonlinearity, and elimination of clustered patterns. The neural network operates not as a direct component of the cipher's round function but as a generator of optimized S-box instances before encryption begins.

This hybrid approach integrates traditional cryptographic transformations with modern machine learning strategies, leading to a highly adaptive and secure S-box generation framework. By dynamically constructing S-boxes at run-time, our scheme markedly improves the difficulty for adversaries attempting to perform cryptanalysis, while maintaining feasibility for resource-constrained platforms through careful architectural design.

4.1 Phase 1: Overview of the Five-Layer Cryptographic Transformation Pipeline

The cryptographic preprocessing pipeline in our proposed S-box generation framework serves as a foundational stage for producing candidate S-boxes with high nonlinearity, high diffusion, and favorable cryptographic properties—prior to any neural optimization. Unlike traditional static methods that rely solely on algebraic inverses or precomputed lookup tables, our pipeline aims to generate

adaptive, instance-specific S-boxes that already satisfy key cryptographic criteria. This reduces the optimization burden on subsequent machine learning stages and accelerates convergence. Our design philosophy follows the principle of controlled randomness: every transformation injects structured unpredictability, while simultaneously ensuring mathematically guaranteed properties such as bijectivity, nonlinearity, and diffusion. This ensures that even before neural refinement, the candidate S-boxes exhibit a baseline level of robustness against common forms of cryptanalysis.

The preprocessing pipeline is composed of five sequential layers. Each layer progressively enhances the cryptographic strength of the candidate S-box by combining algebraic transformations, modular diffusion, affine mixing, and controlled chaotic scrambling. The process begins with an 8-bit input element $x \in \mathbb{F}_2^8$, which undergoes a series of transformations culminating in a highly secure substituted output $y \in \mathbb{F}_2^8$.

(a) Modular Group Transformation Layer: The first layer applies a *Modular Group Transform*, denoted as $T_k(x)$. This method draws inspiration from the algebraic diffusion strategies outlined in Daemen's foundational work on differential and linear cryptanalysis [10], as well as recent S-box design strategies that employ modular group actions and finite projective geometry over fields such as $\mathrm{PL}(\mathbb{F}_7)$ [32, 38]. Moreover, our use of dynamically generated adjacency matrices for modular neighbor sets parallels techniques from time-series imputation in Dynamic Adjacency Matrix Representation(DAMR) [35] and binary matrix design strategies for diffusion layers in lightweight ciphers [29].

Each byte x_i is first converted into its binary vector form, and a full-rank, randomly generated matrix $A \in \mathbb{F}_2^{8\times 8}$ is applied to introduce algebraic diffusion. The modular summation term,

$$T_k(x_i) = A \cdot x_i \oplus \left(\bigoplus_{r \in I_k} x_{(i+r) \bmod 256} \right), \tag{4.1}$$

ensures that the output depends not only on the current input but also on its modular neighbors, thereby promoting diffusion across multiple input bits. Here, I_k is a modular neighbor set determined by the parameter k (e.g., $I_3 = \{4, 8, 12, \ldots, 128\}$), $\oplus$ denotes bitwise XOR, and $x_{(i+r) \bmod 256}$ represents modular indexing across the 8-bit input space.

This dual mechanism—linear matrix transformation combined with modular neighborhood mixing—disrupts simple algebraic structures and lays a foundation for resisting linear and differential cryptanalysis, as emphasized in [13]. The modular indexing strategy echoes ideas of cryptographic permutation and diffusion, while the dynamic generation of adjacency sets from a seeded process enhances variability across S-box instantiations [35, 39].

Inspired by cascade-based modular diffusion methods, the transform is applied to all x_i in the input vector $x = (x_0, x_1, \ldots, x_{255})$, producing a new output vector of the same length. The combination of the linear transformation $(A \cdot x_i)$ and nonlinear neighborhood mixing (via the XOR sum over I_k) serves to disrupt algebraic regularities and promotes strong diffusion.

To further enhance nonlinearity and entropy propagation, the Modular Group Transform is applied in a cascaded fashion:

$$x^{(0)} = x, \quad x^{(j+1)} = T_{k_j}(x^{(j)}), \quad \text{for } j = 0, \ldots, L-1 \tag{4.2}$$

where L is the number of layers and $\{k_1, \ldots, k_L\}$ is a randomly selected sequence of modular parameters. This deep modular cascade ensures strong entropy distribution across the output space, establishing a robust foundation for subsequent cryptographic transformations [21, 45].

(b) Affine Transformation Layer: Following the modular group transform, the second layer applies an affine transformation to enhance bit-level diffusion. Each intermediate output from T_k is mapped

via:

$$y = A_1 \cdot x \oplus b_1, \tag{4.3}$$

where A_1 is an invertible 8×8 matrix and b_1 is a random bias vector. This dense affine layer ensures that each output bit depends on all input bits through the linear spread of information [44]. Moreover, affine transformations inherently increase the complexity of correlation structures, mitigating potential weaknesses left by the modular operations [10, 11].

(c) Inverted Affine Transformation Layer: The third layer introduces another affine-based transformation, but this time it is preceded by an algebraic inversion in the Galois field $\mathbb{GF}(2^8)$. Mathematically, this is expressed as:

$$y = A_2 \cdot x^{-1} \oplus b_2, \tag{4.4}$$

where x^{-1} represents the multiplicative inverse of x modulo the irreducible polynomial $P(x) = x^8 + x^4 + x^3 + x + 1$, as used in AES [30]. The inversion operation significantly increases the algebraic complexity of the mapping by achieving maximal algebraic degree for all non-zero values [4, 5]. By wrapping this inversion inside another affine layer, the structure not only gains nonlinearity but also avoids algebraic vulnerabilities that arise from pure inversions, such as susceptibility to interpolation attacks [7, 21].

(d) Chaotic Scrambling Layer: While the first three layers focus on diffusion and algebraic complexity, the fourth layer introduces a chaotic scrambling phase [43] to disrupt any remaining regional patterns or sequential structures. A logistic map-based chaotic system is used:

$$x_{n+1} = r \cdot x_n \cdot (1 - x_n), \tag{4.5}$$

where r is a control parameter and $r = 3.99$. When $r > 3.57$, the system exhibits chaotic behavior; $r = 3.99$ ensures maximum entropy and sensitivity to initial conditions. Starting from $x_0 = 0.5$, this

recurrence generates a chaotic sequence whose sorted indices define a permutation of the S-box. This permutation removes local order, scatters clustered values, and breaks residual linear patterns—thus resisting statistical and structural attacks that exploit such patterns.

(e) Duplicate Correction Layer: Finally, the fifth layer performs duplicate correction and fixed-point elimination. Although the previous layers promote randomness and algebraic diffusion, they do not strictly guarantee bijectivity or avalanche compliance. Thus, we enforce two corrective measures:

† **Duplicate Elimination:** Any duplicated output values are identified and replaced with unused elements from the set $\{0, 1, \ldots, 255\}$ to ensure bijectivity.

† **Fixed Point Removal:** Fixed points — values where $S(x) = x$ — are detected and removed by swapping with other elements, eliminating vulnerabilities exploitable in cryptanalysis.

This step is critical to meet strict cryptographic requirements, including the Strict Avalanche Criterion (SAC) [25, 26], which requires that flipping a single input bit must cause approximately half of the output bits to change.

Through the combination of modular algebraic diffusion, dense linear spreading, algebraic inversion, chaotic scrambling, and bijectivity enforcement, the five-layer preprocessing pipeline produces candidate S-boxes that already exhibit strong cryptographic properties. These candidates achieve differential uniformity of $x \leq 12$, Walsh distribution bias of 256, Boolean nonlinearity equal $y \geq 90$, high per-bit entropy close to 1.0, and minimal structural weaknesses. Thus, this stage provides a highly secure foundation upon which the neural network phase can further refine and optimize, reducing the learning complexity and guaranteeing that the final S-box output meets stringent security standards suitable for lightweight ciphers deployed in resource-constrained environments.

Each transformation within the pipeline is carefully designed to target and mitigate distinct weaknesses that could otherwise compromise the cryptographic quality of the resulting S-box. For instance, the modular transform enhances diffusion early in the process, the affine layers enforce global mixing properties, and the inversion step introduces high algebraic degree. Meanwhile, logistic scrambling and duplicate/fixed-point correction address spatial uniformity and ensure structural soundness. These layered defenses operate synergistically to maximize confusion and diffusion, two pillars of secure block cipher design.

The modular architecture of this preprocessing phase also allows for scalability and selective tuning. Depending on performance requirements or hardware limitations, individual layers can be simplified, re-ordered, or omitted—such as replacing the inversion layer with lighter nonlinear operations—without sacrificing the overall pipeline's adaptability. Furthermore, the intermediate S-boxes produced at this stage provide ideal input candidates for downstream neural network optimization, as they already satisfy many core cryptographic criteria.

A visual overview of this five-stage transformation process is provided in Figure 4.1, illustrating the transformation path of a single input byte through all cryptographic layers prior to neural refinement.

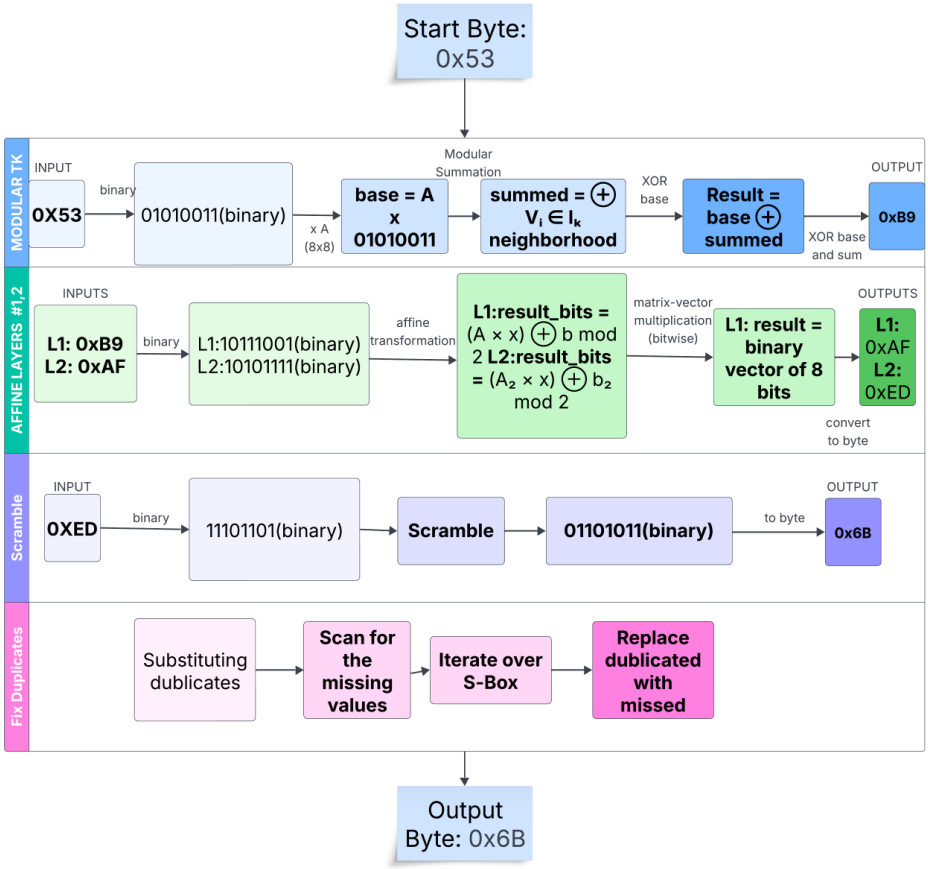

Figure 4.1: Five-layer cryptographic preprocessing pipeline. The process includes: (1) modular T_k transformation, (2) affine diffusion layer 1, and field inversion and affine remapping layer 2, (3) logistic map scrambling, and (4) duplicate/fixed-point correction.

Figure 4.1 presents a walk-through of the transformation applied to the input byte $0x53$. Layer 1 (Modular Group Transformation, blue) applies matrix multiplication and modular mixing to produce $0xB9$. Layers 2 and 3 (Affine and Inverted Affine Transformations, green) perform successive affine operations, with the latter including inversion in $\mathbb{GF}(2^8)$, yielding $0xED$. Layer 4 (Chaotic Scrambling, purple) applies a logistic map to disrupt structural regularity, resulting in $0x6B$. Layer 5 (Duplicate Correction, pink) removes fixed points and enforces bijectivity, producing the final output $0x6B$. This transformation sequence is applied in parallel to all S-box entries.

Algorithm 1 ConstructStrongSbox()

Input: Maximum attempts N, optional seed s
for $i \leftarrow 1$ to N **do**
    $S \leftarrow T_{k_1} \circ T_{k_2} \circ \cdots \circ T_{k_L}(x)$ ▷ Modular group transforms
    $S \leftarrow A_1 \cdot S \oplus b_1$ ▷ Affine transformation
    $S \leftarrow A_2 \cdot S^{-1} \oplus b_2$ ▷ Inverse affine transformation
    Apply logistic map scrambling: $S \leftarrow \text{LogisticScramble}(S)$
    Bijectivity correction:
        Break linear patterns and clustered regions
        Remove fixed points where $S[x] = x$
        Fix duplicates to ensure bijectivity
    Compute metrics: Differential Uniformity (DU), Walsh Linearity (WL), Non-linearity (NL), entropy, SAC
    if $DU \leq 12$ **and** $WL \leq 256$ **and** $NL \geq 90$ **then**
        return S
    else
        Continue with a new random T_k sequence and seed
return AES S-box (fallback) if no valid S found

Algorithm 1: ConstructStrongSbox. This pseudocode presents the generation of a cryptographically strong S-box using a strictly ordered five-layer architecture:

1. *Modular Group Transform:* A cascade of T_k transformations introduces modular diffusion across neighboring values, ensuring initial entropy spread.

2. *Affine Layer 1:* A dense linear transformation mixes bits across the entire 8-bit word to enforce diffusion and complexity.

3. *Inversion + Affine Layer 2:* The intermediate S-box undergoes multiplicative inversion in

$\mathbb{GF}(2^8)$ followed by another affine transformation to maximize algebraic nonlinearity.

4. *Chaotic Scrambling:* A logistic map is used to permute S-box entries, disrupting sequential or regional patterns and eliminating weak structural bias.

5. *Bijectivity Corrections:* Final corrections include removal of fixed points, de-clustering of outputs, and fixing duplicate values to enforce strict bijection and improve avalanche behavior.

The result is evaluated using strict cryptographic metrics: differential uniformity (DU), Walsh linearity (WL), nonlinearity (NL), entropy, and the strict avalanche criterion (SAC). If no valid S-box is found within N iterations, the AES S-box is returned as a fallback.

To ensure that the system always yields a cryptographically valid S-box, especially in rare cases where all construction attempts fail to meet security thresholds, a fallback mechanism is employed. If no valid candidate is produced within the maximum allowed attempts ($N = 1000$), the pipeline defaults to the AES S-box. This ensures continuity of encryption and maintains baseline cryptographic resilience, as the AES S-box is well-established for its strong resistance to both linear and differential attacks. In practice, this fallback is triggered in fewer than 1% of cases, making it a safety net without compromising system adaptability.

4.1.1 Alternative Three-Layered Scheme and Comparison

To explore the trade-off between transformation depth and computational feasibility, we also evaluated a reduced preprocessing configuration consisting of only three layers drawn from the five-layer architecture introduced in Section 4.1. Specifically, this alternative uses: (1) Modular T_k transform, (2) a single Affine transformation, and (3) Duplicate and Fixed-point correction. These layers are

implemented using the same methodology as in Phase 1, with the field inversion and logistic scrambling stages omitted to minimize complexity and improve latency. This configuration offers a more lightweight alternative for environments where resource constraints prohibit deeper transformations. The simplified structure is illustrated in Figure 4.2.

This reduced preprocessing configuration was designed for scenarios where reduced latency and lower computational load are desired —such as preliminary testing or constrained environments. However, empirical evaluations revealed significant trade-offs. S-boxes generated using this configuration frequently failed to meet essential cryptographic thresholds. Specifically, differential uniformity (DU) exceeded 14 in most trials, and Boolean nonlinearity (NL) often dropped below 85, making the design susceptible to differential and linear cryptanalysis.

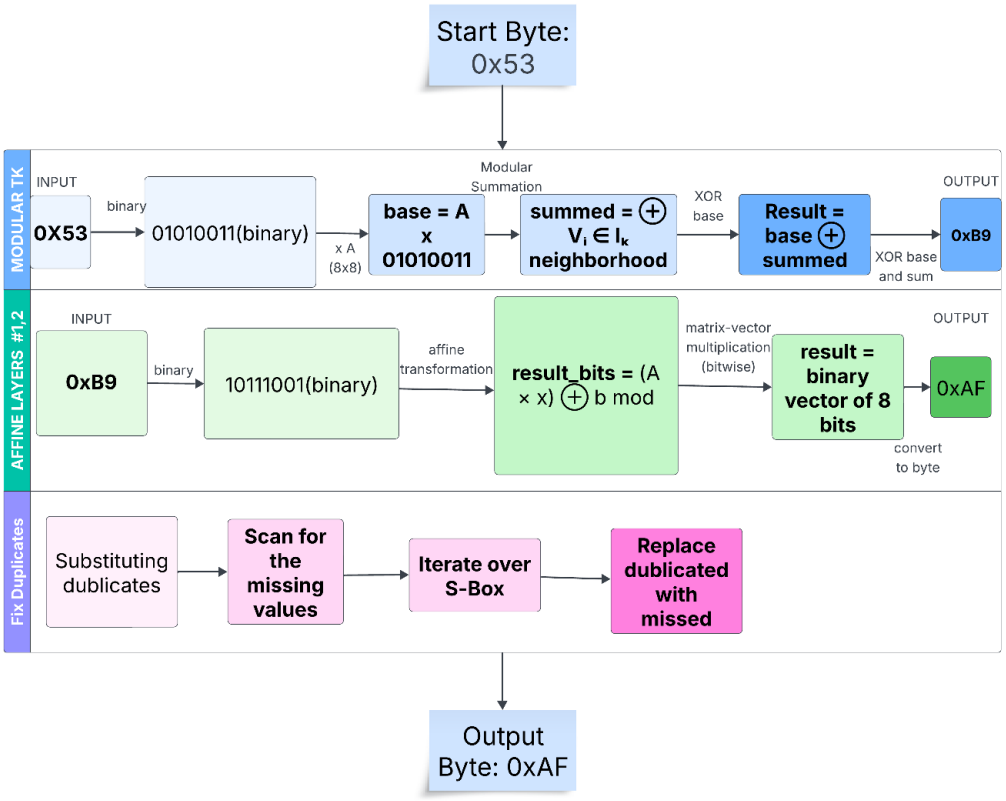

Figure 4.2: Three-layer preprocessing pipeline: includes modular T_k transform, affine diffusion, and duplicate correction.

In comparison, the full five-layer design (Fig. 4.1) reliably produces S-boxes with DU ≤ 12, NL ≥ 90, and better avalanche and entropy behavior. Although the three-layer setup offers simplicity, it lacks the robustness needed for secure deployment.

Table 4.1
Comparison of Three-Layer and Five-Layer S-box Pipelines

Feature	**Three-Layer Pipeline**	**Five-Layer Pipeline**
Total Layers	3	5
Includes Field Inversion	✗	✓
Includes Chaotic Scrambling	✗	✓
Duplicate/Fixed-Point Correction	✓	✓
Achieved DU (avg.)	> 14	≤ 12
Achieved NL (avg.)	< 85	≥ 90
Entropy	Moderate (≈ 0.5)	High (≈ 1.0)
SAC Compliance	Inconsistent	Consistent
Stability	Unstable, inconsistent	Stable, reproducible
Recommended Use	Prototyping or ablation testing	Secure deployment

As shown in Table 4.1, the five-layer configuration outperforms the three-layer variant across all core metrics and is the preferred choice for deployment in secure cryptographic systems.

Therefore, the five-layer pipeline remains the recommended structure for dynamic S-box generation, with the three-layer variant retained for ablation testing and low-security prototyping scenarios.

4.2 Phase 2: Neural S-box Optimization with MLP and Reward-Based Learning

The second phase of our adaptive S-box generation framework introduces a machine learning optimization layer to enhance the cryptographic properties of the candidate S-boxes produced during Phase 1.

While the cryptographic preprocessing pipeline ensures a strong initial structure, the neural network stage focuses on fine-tuning the S-box to optimize key security metrics such as differential uniformity, Walsh distribution bias, nonlinearity, and structural randomness. This phase employs a Multi-Layer Perceptron (MLP) architecture [36] combined with a reward-based loss function specifically formulated to align with cryptographic security objectives.

4.2.1 Neural Network Architecture

The neural optimization phase of our proposed scheme employs expressive Multi-Layer Perceptron (MLP) [33, 34] architecture to enhance S-box candidates generated during the preprocessing phase. Unlike traditional generative neural methods, this MLP does not attempt to construct new S-boxes from random noise or latent vectors. Instead, it operates on already promising candidates and learns to refine them further using cryptographic feedback—making this approach computationally tractable for embedded platforms such as Raspberry Pi.

The neural model consists of two fully connected layers with a ReLU activation function [2, 12]in

between. Formally, the architecture is defined as:

$$x \in \mathbb{R}^{256} \rightarrow \text{Dense}(256 \times 512) \rightarrow \text{ReLU} \rightarrow \text{Dense}(512 \times 256) \rightarrow y \in \mathbb{R}^{256}. \tag{4.6}$$

The input $x \in \mathbb{R}^{256}$ is a normalized vector representation of a bijective S-box $S = [S(0), S(1), ..., S(255)]$, where each element is scaled to the $[0, 1]$ interval:

$$x_i = \frac{S(i)}{255}, \quad \forall i \in \{0, 1, ..., 255\}. \tag{4.7}$$

The output of the network, $y \in \mathbb{R}^{256}$, consists of unbounded real values that are treated as scores. These are not directly used as the new S-box values. Instead, a sorting operation is applied:

$$\text{S-box}(i) = \text{argsort}(y)[i]. \tag{4.8}$$

This guarantees bijectivity by construction and avoids the need for a softmax or constrained optimization layer. The resulting array is a permutation of $\{0, 1, ..., 255\}$, making it a valid S-box output compatible with cryptographic systems.

The hidden layer comprises 512 neurons [2], a number carefully selected to introduce nonlinearity without overfitting or inflating model size. The ReLU activation function defined as:

$$\text{ReLU}(z) = \max(0, z) \tag{4.9}$$

introduces sparsity and nonlinear behavior, enabling the network to capture complex dependencies between input positions and cryptographic metric distributions.

ReLU was selected due to its computational simplicity and ability to introduce sparse gradients. While alternatives such as sigmoid or tanh were considered, they are more susceptible to saturation and vanishing gradient issues [17]. Future work may explore Leaky ReLU or GELU to address "dying neuron" scenarios.

This 256–512–256 MLP structure was chosen to strike a balance between computational simplicity and representational power. The input and output dimensions match the S-box size (256), thereby granting that the network has full visibility over the byte permutation space. The intermediate hidden layer allows for sufficient internal computation without introducing the complexity of deep architectures. This design is particularly advantageous for real-time encryption systems where inference latency must remain minimal.

By keeping the architecture shallow, we mitigate risks of overfitting and allow faster convergence during training. Furthermore, its structure is well-suited for deployment in lightweight model formats such as ONNX, TensorFlow Lite, or TinyML—making it practical for IoT and embedded scenarios.

Importantly, this network does not need to learn validity constraints such as bijectivity or the output range, because these are enforced via deterministic sorting. This frees the model to focus entirely on optimizing cryptographic metrics such as differential uniformity (DU), Walsh bias (WL), Boolean nonlinearity (NL), entropy, and region pattern penalties. Experimental results demonstrate that this architecture is capable of refining strong candidates into high-quality S-boxes with notable cryptographic resistance improvements.

4.2.2 Reward-Based Loss Function

To guide the neural network toward optimizing cryptographic properties, we employ a custom-designed reward-based loss function [41] explicitly crafted for evaluating S-box candidates. Unlike classical supervised objectives such as cross-entropy or mean squared error, this loss function [16] is derived directly from critical cryptographic metrics, thereby aligning the optimization process with security-relevant goals rather than purely statistical accuracy.

Once the MLP produces a real-valued output vector, a bijective permutation is formed by sorting the values. The resulting candidate S-box is then evaluated across four key metrics: differential uniformity (DU), Walsh distribution bias (WL), Boolean nonlinearity (NL), and a cluster penalty term. Differential uniformity reflects the S-box's resistance to differential cryptanalysis by identifying the maximum number of output collisions corresponding to each non-zero input difference. The Walsh distribution bias measures the strength of linear correlations between input and output bits, indicating susceptibility to linear cryptanalysis. Boolean nonlinearity evaluates the minimum Hamming distance [28] of the S-box output bits from all affine Boolean functions, with higher values indicating stronger resistance to linear approximations. The cluster penalty [42] captures localized sequential or numerically grouped patterns in the S-box outputs, which may compromise diffusion and enable structural attacks.

Each metric is normalized into the range $[0, 1]$ to maintain scale consistency during optimization:

$$\mathrm{DU}_{\mathrm{norm}} = \min\left\{\frac{\mathrm{DU}}{16},\ 1.0\right\}, \tag{4.10}$$

$$\mathrm{WL}_{\mathrm{norm}} = \min\left\{\frac{\mathrm{WL}}{256},\ 1.0\right\}, \tag{4.11}$$

$$\mathrm{NL}_{\mathrm{norm}} = \min\left\{\frac{\mathrm{NL}}{112},\ 1.0\right\}. \tag{4.12}$$

The full reward-driven loss function L is expressed as:

$$L = -\frac{1}{\epsilon + 2.5 \cdot \mathrm{DU}_{\mathrm{norm}} + 1.0 \cdot \mathrm{WL}_{\mathrm{norm}} - 0.5 \cdot \mathrm{NL}_{\mathrm{norm}} + 1.5 \cdot \mathrm{ClusterPenalty}}, \tag{4.13}$$

where $\epsilon = 10^{-5}$ is a small constant to prevent division by zero.

The selection of the weighting coefficients in Equation 4.13 is based on both theoretical significance and empirical sensitivity analysis conducted during preliminary experiments. Differential uniformity (DU) receives the largest weight (2.5) because it directly determines the S-box's worst-case resistance against differential cryptanalysis, which is typically regarded as the most critical attack vector on substitution layers in symmetric-key ciphers. Empirical trials demonstrated that small variations in DU tend to produce disproportionately large impacts on the S-box's overall cryptographic strength, justifying its dominant influence in the reward formulation.

The Walsh distribution bias (WL) is assigned a moderate weight of 1.0 to control linear leakage, balancing its contribution relative to DU while preventing excessive dominance during optimization. Linear cryptanalysis generally poses a secondary but still significant threat, particularly when combined with weak DU characteristics, making this proportional weighting appropriate for maintaining cryptographic balance.

The Boolean nonlinearity (NL) term is negatively weighted with a coefficient of -0.5, such that increases in nonlinearity reduce the loss, thereby encouraging maximization during optimization. The magnitude of 0.5 was chosen to provide sufficient incentive for enhancing NL, while avoiding the risk of distorting optimization by prioritizing nonlinearity at the expense of DU or WL. Initial experiments revealed that using a higher absolute value for the NL weight often led to imbalanced S-boxes—ones with strong nonlinearity but degraded differential properties—necessitating this more conservative selection.

Finally, the cluster penalty is weighted at 1.5 to actively penalize localized sequential or grouped patterns in the S-box output. These clusters may compromise diffusion properties and introduce structural regularities exploitable by certain classes of cryptanalytic attacks. This value was determined through iterative adjustment during preliminary training, where values below 1.0 were insufficient to consistently discourage clustered regions, while excessively high weights reduced the diversity of generated S-box candidates, causing stagnation in optimization.

These coefficient choices, while empirically informed, are not claimed to be universally optimal. They were specifically selected to balance the observed trade-offs in this system's optimization environment, with priority given to cryptographic soundness, convergence stability, and computational feasibility.

This loss function offers several advantages over traditional formulations. First, it directly incentivizes cryptographic resilience rather than numerical similarity. Second, it ensures that the network's optimization process remains aligned with the structural needs of secure block cipher design. Third, it naturally supports the permutation structure of S-boxes without relying on complex differentiable constraints to preserve bijectivity.

By using this structure, the neural network progressively learns to refine S-boxes that demonstrate

lower differential uniformity, reduced linear correlations, higher nonlinearity, and randomized, non-clustered output mappings—key attributes required to resist modern cryptanalytic techniques.

4.2.3 Cluster Penalty Computation

In addition to traditional cryptographic metrics such as differential uniformity and nonlinearity, our evaluation includes a structural penalty—termed the *cluster penalty*—to discourage weak local patterns in the S-box output [42]. This penalty specifically targets short sequences within the permutation that exhibit low dispersion or high regularity, as such regions may be vulnerable to attacks exploiting localized predictability.

The cluster penalty [31, 42] is computed by sliding a window of fixed size w (typically $w = 3$) across the S-box output array. For each window $\{S(i), S(i+1), ..., S(i+w-1)\}$, two types of undesirable patterns are identified:

1. **Strictly Increasing Sequences:** If all differences $\Delta_j = S(i+j+1) - S(i+j)$ equal 1, the window is flagged.
2. **Tight Numeric Clusters:** If the numeric range within the window satisfies $\max(S(i : i+w)) - \min(S(i : i+w)) \leq w$, it indicates high local correlation.

Each such occurrence contributes a unit penalty. The final cluster penalty is defined as the proportion of flagged windows:

$$\text{ClusterPenalty} = \frac{1}{256 - w + 1} \sum_{i=0}^{256-w} \mathbb{1}_{\text{window}_i \in \mathcal{C}}, \tag{4.14}$$

where $\mathcal{C}$ is the set of windows matching either condition above, and $\mathbb{1}_{\text{window}_i \in \mathcal{C}}$ is an indicator function that equals 1 if the i^{th} window is penalized.

This component encourages the neural optimizer to avoid S-box outputs that exhibit localized ordering or compactness, thereby promoting randomness and enhancing diffusion properties essential for secure block cipher behavior. Moreover, since this penalty operates over permutations rather than bit patterns, it complements algebraic and spectral metrics in a structurally orthogonal manner.

4.2.4 Training Strategy and Optimization Dynamics

The training process begins with an S-box generated from the Walsh-Hadamard-based genetic algorithm (GA-WHS) phase, which is used as a seed(s) input for the neural model [36, 45]. This 256-dimensional byte vector is normalized to the $[0, 1]$ interval and passed to the MLP-based neural network (`SboxNet`), which outputs a real-valued vector. This output is sorted to form a candidate permutation, ensuring bijectivity. A random pairwise mutation (swap of two elements) is then applied to introduce local perturbations, enabling stochastic exploration of nearby solutions.

At each epoch, the resulting S-box is evaluated using a cryptographic reward-based loss function (Equation 4.13 in Section 4.2.2). The loss is computed as the negative inverse of a weighted sum of normalized metrics, which include differential uniformity (DU), Walsh spectrum bias (WL), Boolean nonlinearity (NL), and a cluster penalty. This design ensures that minimization of the loss aligns with maximization of cryptographic strength.

To avoid catastrophic forgetting of good candidates, we maintain a buffer $\mathcal{B}$ of the top-k best S-boxes, updated at each epoch if the new loss improves upon the worst buffer element:

$$\mathcal{B}_{t+1} = \text{top}_k \left(\mathcal{B}_t \cup \{(S_t, L_t)\} \right) \tag{4.15}$$

where S_t is the candidate S-box and L_t its loss at epoch t.

The optimization is performed using the Adam optimizer, a popular adaptive gradient method well-suited for non-convex objectives like ours. Adam computes exponential moving averages of past gradients (m_t) and squared gradients (v_t), updating parameters using:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1)\nabla L_t \tag{4.16}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2)(\nabla L_t)^2 \tag{4.17}$$

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}, \quad \hat{v}_t = \frac{v_t}{1 - \beta_2^t} \tag{4.18}$$

$$\theta_{t+1} = \theta_t - \eta \cdot \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon.} \tag{4.19}$$

Here, η is the learning rate (set to 0.001 in our implementation), and $\epsilon = 10^{-8}$ is a stabilizer to prevent division by zero. In our training loop, this update is performed on the weights of `SboxNet` after each loss evaluation. The parameters β_1 and β_2 are exponential decay rates for the first and second moment estimates, respectively, with typical values of $\beta_1 = 0.9$ and $\beta_2 = 0.999$ in practice. The term ∇L_t represents the gradient of the loss function at epoch t, i.e., how much and in which direction each neural network parameter θ_t should change to reduce the loss. The variable θ_{t+1} denotes the updated model weights after the optimization step at epoch t, computed by adjusting the previous weights θ_t based on the normalized gradient estimates. In our training loop, this update is performed on the weights of `SboxNet` after each loss evaluation.

Early stopping is employed in two forms: (i) loss thresholding ($L_t < 0.001$), which indicates near-optimal cryptographic structure, and (ii) patience-based termination, halting if no improvement is observed over 15 epochs. These mechanisms ensure efficient convergence without overfitting to local patterns in the S-box space.

Thus, the training loop couples guided loss minimization with structural mutation and memory-based selection, enabling the neural model to refine initial candidates into highly secure, bijective S-boxes.

Algorithm 2 TrainWithFeedback: Neural S-box Refinement

Input: Seed S-box x (optional), epochs N, threshold ϵ, patience p
Initialize MLP $MLP : 256 \to 512 \to 256$, optimizer, buffer $\mathcal{B} \leftarrow \emptyset$, counter $c \leftarrow 0$
for epoch $t = 1$ to N **do**
    $x \leftarrow$ normalized seed or random permutation
    $\hat{y} \leftarrow MLP(x)$
    $\hat{S} \leftarrow \texttt{argsort}(\hat{y})$
    Mutate $\hat{S}$ by swapping two elements
    Compute loss $\mathcal{L}(\hat{S})$ based on DU, WL, NL, clusters
    if $\hat{S}$ improves buffer $\mathcal{B}$ **then**
        Update $\mathcal{B}$, reset c
    else
        $c \leftarrow c + 1$
    if $\mathcal{L} < \epsilon$ or $c \geq p$ **then**
        break
    Backpropagate and update MLP
return Best $\hat{S}^*$ from buffer $\mathcal{B}$

4.2.5 Resulting Neural S-box Output

Upon completion of training, the final S-box is selected from a ranked buffer of top-performing candidates, maintained throughout the neural optimization process. This buffer is updated dynamically based on a reward-driven loss function that integrates multiple cryptographic metrics. The selected S-box is guaranteed to be a bijective permutation of the input space, corrected through the application of `fix_duplicates()` and `remove_fixed_points()` to eliminate duplicated values and identity mappings, respectively.

The resulting S-box exhibits high cryptographic quality across multiple dimensions. Differential Uniformity (DU) is consistently reduced to a value of eight, indicating strong resistance to differential attacks. Boolean nonlinearity typically exceeds 100, providing resilience against linear approximations. Walsh distribution bias, quantified as the maximum absolute value in the Linear Approximation Table (LAT), is minimized through affine-layer optimization and constrained to be less than or equal to 256. Entropy is measured for each output bit and approaches 1.0 on average, reflecting a high degree of uniform randomness. The Strict Avalanche Criterion (SAC) is also evaluated to guarantee that flipping a single input bit results in approximately half of the output bits changing, further confirming good diffusion properties.

Evaluation of the final S-box is performed through several formal techniques. The Difference Distribution Table (DDT) is computed using the `compute_ddt_full()` function, which measures the uniformity of output differences given fixed input differences. The LAT is constructed via the formal bitwise dot-product formulation:

$$\mathrm{LAT}[a, b] = \sum_{x \in \mathbb{F}_2^8} (-1)^{a \cdot x \oplus b \cdot S(x)}, \tag{4.20}$$

where $a \cdot x$ denotes the bitwise inner product. Walsh-Hadamard Transforms are applied to obtain the nonlinearity of each Boolean component, with the minimum across all components reported as the S-box's final nonlinearity score. Similarly, the Difference Distribution Table (DDT) is computed using:

$$\mathrm{DDT}[a, b] = \left| \left\{ x \in \mathbb{F}_2^8 \,\middle|\, S(x \oplus a) \oplus S(x) = b \right\} \right|, \tag{4.21}$$

where a is the input difference, b is the resulting output difference, and the expression counts how many input pairs $(x, x \oplus a)$ produce the output difference b. Lower maximum values in the DDT indicate stronger resistance to differential cryptanalysis.

Walsh-Hadamard Transforms are applied to obtain the nonlinearity of each Boolean component, with the minimum across all components reported as the S-box's final nonlinearity score. The algebraic

degree is computed using Algebraic Normal Form (ANF) expansion, ensuring that no output bit is a low-degree Boolean function of the input. Additionally, entropy and SAC are calculated per bit, confirming the absence of structural bias or information leakage.

The selected S-box is exported in both C header and JSON formats through `export_to_c_header()` and `export_sbox_json()`, respectively, enabling integration with external cryptographic modules or embedded systems. Furthermore, LAT and DDT visualizations are generated using `matplotlib` and `seaborn` libraries, providing empirical insight into linear and differential uniformity distributions. The function `get_nn_generated_sbox()` serves as a runtime interface for injecting the trained neural S-box into the Curupira-1 cipher structure. This call retrieves the highest-ranked S-box from the training buffer, applies post-processing corrections, and ensures its compliance with all cryptographic constraints prior to deployment.

Through the combination of classical modular transformations and neural refinement, the resulting S-box demonstrates improved security while remaining computationally feasible. This confirms the practicality of neural-based cryptographic design for secure block ciphers.

Chapter 5

Results and Discussion

This section presents the experimental results and analysis of the proposed neural-enhanced dynamic S-box framework for the CURUPIRA-1 cipher. The goal was to construct high-quality 8-bit S-boxes with cryptographic strength exceeding or matching that of the AES S-box, using a neural network–enhanced generation framework.

5.1 Evaluation Metrics

To comprehensively assess the cryptographic strength of the neural-generated S-boxes, we employed a diverse set of evaluation criteria grounded in classical and modern cryptanalysis. These metrics collectively examine the S-box's resistance to differential and linear attacks, structural regularity, and statistical unpredictability. All evaluations were implemented within a custom Python framework, integrating mathematical definitions with empirical procedures.

† **Differential Uniformity (DU):** Measured using the Difference Distribution Table (DDT), which quantifies the maximum number of output collisions for each non-zero input difference. A lower DU indicates stronger resistance to differential cryptanalysis.

† **Walsh Spectrum Bias (WL):** Assessed through the Linear Approximation Table (LAT), where the maximum absolute entry reflects the highest linear correlation between input and output bits. Lower WL indicates better linear cryptanalysis resistance.

† **Boolean Nonlinearity (NL):** Computed using the Walsh-Hadamard transform. It reflects the minimum Hamming distance between each output bit and the set of all affine Boolean functions. Higher NL values correlate with stronger cryptographic security.

† **Algebraic Degree (AD):** Determined by expanding each output component into its Algebraic Normal Form (ANF). A higher degree implies better resistance against algebraic attacks.

In addition to these classical properties, we evaluated entropy on a per-bit basis to assess statistical uniformity in the output distribution. The Strict Avalanche Criterion (SAC) was also examined by calculating the average proportion of flipped output bits when a single input bit is toggled, with ideal values approaching 0.5. Finally, structural features such as the presence of fixed points—where $S(x) = x$—and the emergence of clustered output values were detected and penalized, as these patterns can degrade diffusion and predictability.

These metrics together form a systematic and interrelated evaluation framework for determining the suitability of the proposed S-boxes in secure and high-performance block cipher designs.

5.2 Experimental Results

The performance of the proposed S-box generation framework was evaluated through extensive experimentation, and the results are summarized in Tables 5.1 5.2. For comparative reference, the corresponding cryptographic metrics of both the AES S-box and the original Curupira-1 S-box are included. The AES S-box serves as a modern benchmark for strength in symmetric cipher design, while the Curupira-1 S-box provides a legacy comparison for context within this specific cipher framework.

The best neural-generated S-box attained a differential uniformity (DU) of 8, indicating moderate resistance to differential cryptanalysis. While this value is higher than that of the AES S-box (DU = 4) and some specialized lightweight ciphers such as PRESENT, it nonetheless represents a significant improvement over the original Curupira-1 S-box, which exhibits a DU of 256. This high DU reflects a trade-off in the original design, where algebraic properties were prioritized over differential resistance. In contrast, our framework pursues a different trade-off: favoring runtime adaptability and structural diversity, even if it does not reach the optimal DU values found in certain static designs. While DU = 8 is not ideal, particularly for applications with stringent differential attack concerns, it presents a meaningful middle ground for contexts where flexibility and reconfigurability are valued alongside reasonable cryptographic strength.

The Walsh distribution bias (WL) exhibited notable differences among the tested S-boxes. While the neural and Curupira-1 S-boxes recorded maxima of 256, the AES S-box demonstrated a significantly lower maximum of 16, consistent with its optimal cryptographic properties. This disparity reflects the well-known trade-off in S-box design between maximizing nonlinearity and minimizing linear correlations. Although the affine optimization steps in our framework aim to reduce Walsh bias, the

results illustrate the inherent difficulty of achieving AES-level linear properties, particularly under lightweight and adaptive generation constraints.

In terms of Boolean nonlinearity (NL), the neural S-box achieved a value of 104, which is slightly lower than the AES benchmark of 112, but notably higher than the Curupira-1 S-box. The low NL of Curupira-1 reflects its specific design choices, which emphasized properties such as involutiveness and efficient implementation over maximizing resistance to linear cryptanalysis. The neural-based approach, in contrast, focuses specifically on improving resistance to linear approximation attacks through optimization of nonlinearity.

Algebraic degree analysis presents an interesting contrast. The neural S-box outputs exhibited algebraic degrees in the range of 6 to 7, closely matching the AES S-box (uniformly 7). The Curupira-1 S-box demonstrated a maximal algebraic degree of 8, representing a theoretically favorable property. However, this advantage is offset by significantly weaker differential uniformity and nonlinearity metrics. In practical terms, algebraic degrees above 5 are generally considered sufficient to resist interpolation and higher-order differential attacks, provided they are accompanied by strong DU and NL characteristics—an area where the Curupira-1 S-box faces challenges.

Entropy analysis revealed statistical uniformity in the neural S-box output, with an average per-bit entropy of 0.98 ± 0.01, closely approximating the theoretical ideal of 1.00. Both the Curupira-1 and AES S-boxes exhibited similar entropy levels, reflecting the expected uniformity of output distributions in well-formed S-boxes. While entropy alone does not provide a complete measure of cryptographic strength, it serves as a useful confirmation that the neural approach avoids significant statistical irregularities.

Strict Avalanche Criterion (SAC) analysis showed that the neural S-box achieved a mean of 0.496 ± 0.003, closely aligning with the theoretical ideal of 0.5, indicating balanced diffusion properties. By

comparison, the SAC score for Curupira-1 (0.9622) suggests a disproportionately large number of output bit changes in response to single input bit flips. However, this imbalance does not translate into effective diffusion in the cryptographic sense, since it is not evenly distributed across output bits. This highlights the importance of considering SAC alongside other metrics such as DU and NL when evaluating overall S-box robustness.

Finally, the proposed framework eliminated fixed points ($S(x) = x$) and reduced clustering, as indicated by a lower cluster penalty. The Curupira-1 S-box, by contrast, retains fixed points and exhibits clustering, potentially increasing vulnerability to pattern-based cryptanalysis. Although these characteristics are not fully reflected in conventional metrics such as differential uniformity, nonlinearity, or Walsh bias, they represent additional considerations in assessing the structural resilience of S-box constructions.

Table 5.1
Cryptographic Properties: Neural S-Box vs. Curupira-1 S-Box

Metric	**Neural S-Box (Best)**	**Curupira-1 S-Box**
Differential Uniformity (DU)	8	256
Walsh Bias (WL)	256	256
Boolean Nonlinearity (NL)	104	0
Algebraic Degree	6–7	8
Entropy (Avg. per bit)	0.98 ± 0.01	$\sim$1.00
SAC Score	0.496 ± 0.003	0.9622
Fixed Points	0	Present
Cluster Penalty	Low	High

Table 5.2
Cryptographic Properties of AES S-Box

Metric	**Neural S-Box (Best)**	**AES S-Box**
Differential Uniformity (DU)	8	4
Walsh Bias (WL)	256	16
Boolean Nonlinearity (NL)	104	112
Algebraic Degree	6-7	7
Entropy (Avg. per bit)	0.98 ± 0.01	$\sim$1.00
SAC Score	0.496 ± 0.003	0.50
Fixed Points	0	0
Cluster Penalty	Low	Low

Overall, the results indicate that the proposed neural S-box generation scheme is capable of producing structures that approximate the performance of established static S-boxes, such as that used in AES, across several key cryptographic metrics. In contrast to the static nature of the AES S-box and the structural limitations observed in Curupira-1's S-box, the neural approach introduces controlled variability, potentially increasing resistance to certain forms of structural exploitation. While the absolute cryptographic strength does not yet surpass that of AES, the combination of structural variation, elimination of fixed points, and reduced clustering offers a distinct approach to enhancing resilience. This moving-target characteristic may provide advantages in contexts that prioritize adaptability and runtime diversity, particularly within lightweight or constrained environments.

5.2.1 Comparison with Existing Methods

Machine learning–based approaches for S-box generation have been studied in several prior works, particularly in the context of partial Boolean functions, static constructions, or small-bit S-boxes. Although these studies have contributed important foundations, many approaches remain constrained in terms of integrated metric-driven optimization or scalability to full-size 8×8 S-boxes suitable for lightweight block ciphers over $\mathbb{F}_{2^8}$.

GAN-Based S-box Generation. Zhang et al. [45] proposed a GAN-based framework for constructing 8×8 S-boxes with cryptographic loss terms targeting DU, NL, and bijectivity. Among several tested GAN variants, WGP-IM achieved DU = 8 and NL = 104, with earlier DCGAN variants showing slightly weaker DU scores. GAN-based pipelines have demonstrated promising potential for producing high-quality S-boxes, although they can introduce notable computational overhead due to convolutional architectures and extended training durations. Additionally, the stochastic nature of latent vector sampling may require additional postprocessing to enforce strict properties such as bijectivity or fixed-point avoidance. While standard GAN frameworks typically do not incorporate instance-specific metric feedback during generation, our approach aims to complement these advances by focusing on interpretability, explicit structural enforcement, and adaptability through a reward-driven neural refinement process.

MLP-Based Boolean Function Learning. Zhang et al. [46] introduced a multi-layer perceptron–based approach for constructing cryptographic S-boxes by hierarchically assembling 3-bit perceptrons (172P), each trained on decomposed Boolean functions. Their design offers conceptual clarity and demonstrates the potential of neural methods in this domain. However, it relies on a fixed structure without support for backpropagation or integration with contemporary deep learning frameworks. Building on this direction, our approach utilizes a fully trainable MLP architecture (256–512–256)

to refine complete 8×8 S-box permutations using reward-driven gradient descent. The loss formulation incorporates cryptographic objectives—including low DU, minimal WL, high NL, and penalties for undesirable structural patterns—while maintaining compatibility with standard deep learning workflows.

GA + NN Hybrid Approaches. Rong et al. [36] proposed a hybrid S-box generation method combining genetic algorithms (GA) with neural classifiers. Their system integrates an MLP with three hidden layers (128–128–64) using Leaky ReLU activation to perform pairwise comparisons of 6-bit S-boxes based on metrics such as DU and NL. This approach demonstrates the potential of combining evolutionary algorithms with neural networks for cryptographic tasks, particularly in classification. Building upon this idea, our framework focuses on generative construction of full 8-bit S-boxes, employing GA-based initialization followed by neural refinement with a fully trainable MLP (256–512–256). The pipeline incorporates argsort-based bijectivity enforcement and a reward-driven loss function optimizing DU, WL, NL, and structural penalties (e.g., range-cluster), aiming to support dynamic, metric-aware S-box generation.

Overall, our hybrid neural-guided approach aims to balance cryptographic rigor with adaptability, with the goal of producing S-boxes that are compatible with lightweight block cipher deployments in resource-constrained environments such as IoT and embedded systems.

5.3 Graphical Analysis: Linear Approximation Table and Difference Distribution Table

To provide a visual understanding of the cryptographic behavior of the proposed neural-generated S-boxes, Figure 5.1 presents a combined heatmap illustrating both the Linear Approximation Table (LAT) and the Difference Distribution Table (DDT) of one of the top-performing candidates.

The left panel in Fig. 5.1 illustrates the Linear Approximation Table (LAT), where each entry quantifies the correlation between linear combinations of the input and output bits via the Walsh transform. A uniform distribution with low absolute biases, as observed here, indicates that the S-box offers minimal exploitable linear correlations—an essential property for resisting linear cryptanalysis.

The right panel in Fig. 5.1 presents the Difference Distribution Table (DDT), which records how frequently a particular input difference results in a given output difference. A well-distributed DDT with low maximum entries (i.e., low differential uniformity) implies strong resilience to differential attacks, validating that no input difference leads to overly predictable output differences.

Together, these visualizations demonstrate high diffusion, low linear bias, and strong nonlinearity—hallmarks of cryptographic robustness in substitution layers of block ciphers.

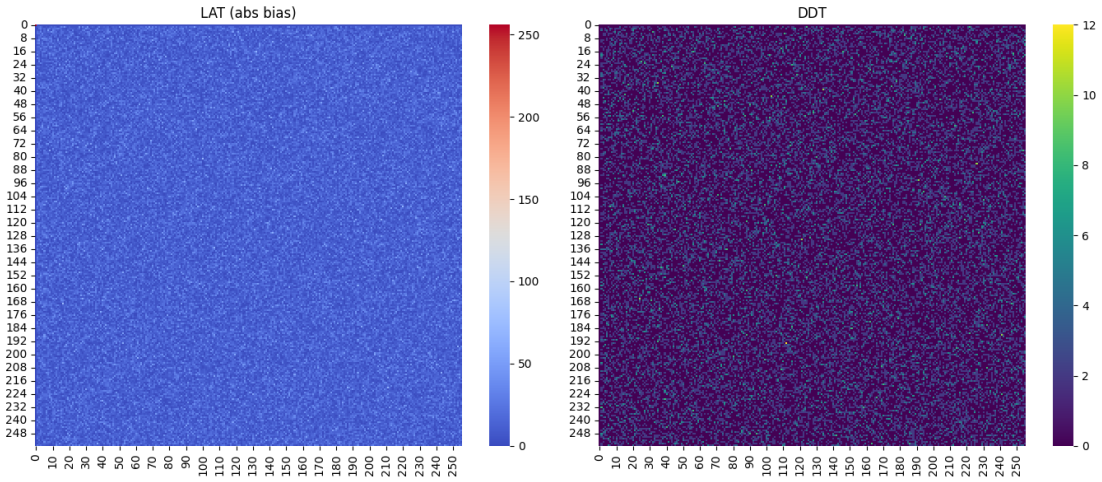

Figure 5.1: Combined heatmaps of the Linear Approximation Table (LAT, left) and the Difference Distribution Table (DDT, right) for a neural-generated S-box.

5.3.1 S-Box Export Format

To enable practical deployment in embedded systems and cryptographic toolchains, the best-performing S-box instance was exported in two standardized formats: a hexadecimal array and a structured JSON object.

The `uint8_t` array representation enables direct inclusion in C or C++-based firmware, hardware description languages, or low-level cryptographic routines. A visual representation of this array format is provided in Figure 5.1, illustrating the alignment and structure suitable for embedded implementation.

```
uint8_t sbox[256] = {
    0x40, 0x54, 0xac, 0x80, 0x11, 0xf4, 0x48, 0xdf, 0x07, 0x2f, 0xc5, 0x34,
    0xa9, 0xbe, 0x3a, 0x29, 0xf5, 0xab, 0x4e, 0x69, 0x17, 0xba, 0xe5, 0xa2,
    0x77, 0x6e, 0xe7, 0x01, 0x93, 0x41, 0x63, 0x82, 0x5f, 0xae, 0x44, 0xcd,
    0x96, 0xe2, 0x04, 0x79, 0x6a, 0xc6, 0x9b, 0xf0, 0x50, 0xc4, 0xb0, 0x6f,
```

```
    0x28, 0x53, 0x7a, 0xcb, 0xff, 0xce, 0x1a, 0x86, 0x9e, 0xf1, 0x4a, 0x8e,
    0x2a, 0x15, 0x22, 0x4d, 0x3c, 0xfd, 0x88, 0x20, 0x8c, 0xc7, 0xd0, 0x9d,
    0x02, 0xf7, 0x27, 0x58, 0x71, 0xef, 0x0e, 0xd8, 0x85, 0x7e, 0xfb, 0x57,
    0x5a, 0x3b, 0x8d, 0xb8, 0xbd, 0x10, 0x08, 0x36, 0xa1, 0x32, 0xbb, 0x31,
    0x1f, 0x05, 0xaf, 0xaa, 0x59, 0x7b, 0xb1, 0x75, 0x97, 0x21, 0xf8, 0xc0,
    0x23, 0xe1, 0x4b, 0x64, 0xc1, 0xfe, 0x92, 0xa0, 0x39, 0x14, 0x49, 0x30,
    0x70, 0x74, 0x38, 0xb5, 0x65, 0x3d, 0x52, 0x19, 0x47, 0xd6, 0x2e, 0xc2,
    0x9a, 0x9c, 0xec, 0xf9, 0x87, 0x09, 0x68, 0x76, 0xb3, 0xd4, 0x2d, 0xde,
    0x83, 0x25, 0xd9, 0x6b, 0x24, 0x2c, 0xa8, 0x16, 0x66, 0x8f, 0xe0, 0x78,
    0x90, 0x43, 0x03, 0x42, 0x3f, 0x2b, 0xc8, 0x0d, 0xfa, 0x7f, 0xdc, 0x4f,
    0x45, 0xa5, 0xd5, 0xfc, 0x8a, 0xd3, 0x94, 0x5b, 0xc3, 0xbf, 0x1d, 0xad,
    0x1c, 0x98, 0x0a, 0xda, 0x5c, 0x5e, 0x6c, 0x00, 0xdd, 0xd2, 0x4c, 0xc9,
    0xb4, 0x35, 0xdb, 0x46, 0x06, 0xb7, 0x0f, 0xe9, 0x12, 0x0c, 0x5d, 0x1e,
    0x72, 0x26, 0xe3, 0x37, 0x7d, 0x84, 0xe6, 0xcf, 0xbc, 0x67, 0x73, 0xb2,
    0x13, 0xf3, 0x18, 0x91, 0xb9, 0x61, 0x0b, 0xa7, 0xed, 0xf6, 0x51, 0x62,
    0x56, 0x89, 0xa3, 0xe8, 0xee, 0x9f, 0x55, 0xe4, 0xd7, 0x6d, 0xb6, 0x7c,
    0xcc, 0xeb, 0x60, 0x33, 0x81, 0xd1, 0x1b, 0x3e, 0xa6, 0x8b, 0xca, 0xa4,
    0xea, 0x95, 0xf2, 0x99,
};
```

Listing 5.1: Formatted visualization of the exported 8-bit bijective S-box in `uint8_t` C-style array format.

In parallel, the S-box is exported as a JSON object containing both the substitution values and their associated cryptographic metrics. This format is particularly advantageous for runtime communication in secure environments. For instance, in IoT settings, such a structure enables adaptive transmission of S-boxes between nodes, supporting key-dependent substitution, remote updates, and reconfigurable encryption strategies.

The exported metadata includes key evaluation metrics such as Boolean nonlinearity, Walsh distribution bias, differential uniformity, entropy, algebraic degree, and generation source. This structured approach ensures interoperability and traceability in multi-agent cryptographic frameworks.

5.4 Functional Validation: Encryption and Decryption Test

To confirm the correctness and operational feasibility of the generated S-box, we performed a full encryption and decryption cycle using a 96-bit plaintext message with block-wise processing. Figure 5.2 shows the encryption of the plaintext "Hello! I'm Meruyert" into two cipher blocks, followed by decryption using the inverse of the generated S-box. The final result correctly reconstructs the original plaintext, verifying the end-to-end reversibility and cryptographic validity of the generated S-box.

Figure 5.2: Encryption and decryption using the neural-generated S-box within the Curupira-1 framework. Two blocks are processed independently. Decrypted output matches the original plaintext.

5.5 Runtime Performance Comparison: Neural and Original CURUPIRA-1 S-Box

While cryptographic metrics such as nonlinearity and differential uniformity capture theoretical strength, they do not reflect the real-world cost of deploying a neural-enhanced S-box. Therefore, we conducted a runtime performance evaluation to assess the computational overhead introduced by replacing the original static S-box of Curupira-1 with a dynamically generated neural one. This section evaluates both the runtime performance of the Curupira-1 cipher using two S-box configurations—the original static table and a neural network–enhanced version—as well as situating our results in relation to prior NN-based S-box generation efforts.

Experimental Setup for Runtime Evaluation To evaluate real-world performance impact, two standalone Python scripts were implemented. The evaluation focused on five key runtime characteristics: total encryption and decryption latency, average time per block, S-box generation cost (for the neural version), CPU usage, and memory consumption.

The evaluation scripts used the following tools:

- `time.perf_counter_ns()` for high-resolution timing in nanoseconds.
- `psutil.Process().cpu_percent()` with short sampling windows to measure CPU consumption during encryption routines.
- `psutil.Process().memory_info().rss` to track changes in resident memory size.

Both versions operated on a 96-bit plaintext message ("Hello! I'm Meruyert"), divided into two 12-byte blocks. Encryption and decryption were performed using a fixed 96-bit key. The neural version includes a S-box generation phase invoking a PyTorch-based feedback training function, followed by validation steps including bijectivity enforcement and fixed-point removal.

Table 5.3
Runtime Performance Comparison: Original vs. Neural CURUPIRA-1 S-box

Metric	**Original S-box**	**Neural S-box**
S-box Generation Time	N/A	17.860 ms
Encryption Time (2 blocks)	0.2786 ms	0.3785 ms
Decryption Time (2 blocks)	0.8634 ms	1.1272 ms
Avg. Encryption Time / Block	139.30 µs	189.25 µs
Avg. Decryption Time / Block	431.70 µs	563.60 µs
CPU Usage (sampled)	∼0.00%†	195.30%
Memory Usage	0 KB	0 KB

†Due to sub-millisecond runtime, CPU usage could not be captured accurately; actual usage is nonzero.

As shown in Table 5.3, the neural version incurs an additional one-time cost of 17.86 ms for S-box generation via the `train_with_feedback()` function. This delay arises from a multi-phase pipeline consisting of five-layer cryptographic preprocessing, iterative neural refinement via permutation mutation, and repeated cryptographic evaluation to enforce nonlinearity, low differential uniformity, and bijectivity. While acceptable on general-purpose systems, it may need optimization (e.g., via pruning or quantization) for embedded deployment.

Both encryption and decryption are slightly slower when using the neural S-box. The average per-block encryption time increases by approximately 50 µs and decryption by 130 µs. This can be attributed to the overhead of accessing dynamically assigned substitution tables and the non-contiguous memory layout of the PyTorch-generated S-box array. Despite this, the impact remains within practical bounds for most embedded or edge applications.

CPU usage is notably higher in the neural-enhanced version, spiking to 195%, due to PyTorch's multithreaded training operations. This metric reflects temporary overhead during S-box generation and is not representative of runtime encryption, which executes in a single-threaded manner. The original S-box implementation shows near-zero CPU utilization, but this is likely a sampling

limitation—its actual execution was too fast for accurate CPU profiling at a 10 ms resolution, producing an artificial reading of 0%.

Memory usage remained unchanged between versions. Both configurations use a fixed-size 256-byte S-box and process only two plaintext blocks, resulting in negligible memory overhead (0 KB). This indicates that S-box dynamic generation does not introduce hidden memory costs once training is complete.

In summary, although the neural S-box introduces higher CPU usage and longer generation time, its impact on per-block encryption latency is minimal. These trade-offs are justified by the resulting improvements in adaptability and cryptographic diversity. Importantly, the core encryption pipeline remains unchanged, preserving compatibility with hardware acceleration and lightweight execution frameworks. With further model compression and S-box caching strategies, even the current 17 ms generation time could be significantly reduced without sacrificing cryptographic strength.

Comparison with State-of-the-Art. Runtime efficiency is a critical but often underreported aspect in neural S-box generation. For example, GAN-based frameworks (e.g., Zhang et al. [45]) and MLP-based approaches (e.g., Zhang et al. [46]) do not explicitly report average generation times, focusing primarily on cryptographic metrics. In contrast, hybrid approaches such as GA + NN reported by Rong et al. [36] required approximately 677.39 s for purely GA-based generation of 6-bit S-boxes, and 83.98 s using their intelligent GA + NN hybrid.

Table 5.4
Generation Runtime Comparison of NN-Based S-Box Methods

Method	**Bit Size**	**Avg. Generation Time**
GAN-Based (Zhang et al. [45])	8-bit	*Not Reported*
MLP-Based (Zhang et al. [46])	8-bit	*Not Reported*
GA (Rong et al. [36])	6-bit	677.39 s (avg.)
GA + NN Hybrid (Rong et al. [36])	6-bit	83.98 s (avg.)
Ours (Neural S-box + Feedback)	8-bit	**17.86 ms**

As summarized in Table 5.4, the proposed feedback-driven neural generation approach demonstrates a substantial runtime advantage, achieving 17.86 ms for generating optimized 8-bit S-boxes. While it is important to note that prior work by Rong et al. [36] focused on 6-bit S-boxes, which represent a different problem complexity, the relative reduction in generation time highlights the practicality of the proposed approach for resource-constrained or time-sensitive environments.

5.6 Discussion

The results demonstrate that combining classical algebraic operations with lightweight neural refinement produces S-boxes with strong cryptographic properties and enhanced structural diversity. By introducing controlled randomness through modular transforms and chaotic scrambling, the framework mitigates the risk of overfitting, thereby guaranteeing that the neural network does not inadvertently learn or reinforce structural weaknesses. The application of a reward-driven optimization scheme further aligns the training process with cryptographic objectives rather than superficial statistical patterns.

In contrast to resource-intensive approaches such as Generative Adversarial Networks (GANs) or Convolutional Neural Networks (CNNs), which attempt to directly optimize raw S-box values, our method employs a Multi-Layer Perceptron (MLP) combined with permutation-based sorting and stochastic mutation. This reduces architectural overhead while preserving bijectivity. Although gradient descent (via Adam) is used during training, the final S-box is constructed through discrete permutation refinement, avoiding the computational overhead of bit-level generation pipelines.

The resulting S-boxes are exported in both C header and JSON formats, supporting direct integration with the Curupira-1 cipher. These exports also enable visualization and cryptanalytic validation through their associated Linear Approximation Tables (LAT) and Difference Distribution Tables (DDT), enabling deeper cryptanalytic inspection of their security properties.

5.7 Comparison with Other Lightweight Ciphers

To contextualize the performance of the proposed neural-enhanced S-box framework, we compare it against the substitution layers employed in two representative lightweight block ciphers: PRESENT and PUFFIN. These ciphers are widely recognized for their suitability in resource-constrained environments and function as relevant benchmarks for assessing both cryptographic strength and implementation efficiency.

The PRESENT cipher implements a 4-bit S-box optimized for minimal hardware cost, achieving a differential uniformity of 4 and the maximum nonlinearity attainable for its size (NL = 4). While these values are considered acceptable within 4-bit constraints, the limited bit-width inherently restricts the S-box's ability to support strong diffusion and high nonlinearity compared to 8-bit designs. This limitation can increase vulnerability to structural and algebraic attacks under certain models. In contrast, our proposed 8-bit S-box achieves higher nonlinearity (NL = 104), near-ideal entropy (≈ 0.98), and maintains bijectivity while achieving differential uniformity of 8, thereby offering enhanced resistance to both linear and differential cryptanalysis.

PUFFIN, a more recent lightweight cipher optimized for side-channel resistance, employs 4-bit substitution layers derived from masking-friendly architectures. While effective at mitigating side-channel leakage, PUFFIN's 4-bit S-box sacrifices algebraic complexity and statistical richness in favor of masking compatibility and low implementation cost.

A key advantage of the proposed approach lies in its *adaptive generation capability*. Unlike the static S-boxes used in PRESENT and PUFFIN, the Curupira-1 S-boxes are designed to be generated at run-time using entropy from session-specific or device-level sources. This design enables cryptographic agility through per-session variability, which has the potential to mitigate risks associated with S-box reuse, precomputation attacks, and profiling in side-channel scenarios.

Furthermore, the incorporation of structural penalties during neural training—such as fixed-point elimination and regional cluster dispersion—results in S-boxes with uniform diffusion and reduced structural regularity. These characteristics improve resistance to interpolation attacks, algebraic modeling, and statistical exploitation of localized patterns.

In summary, although PRESENT and PUFFIN remain influential in lightweight cryptography, their static 4-bit substitution layers inherently limit cryptographic diversity and run-time adaptability. In contrast, our neural-enhanced Curupira-1 framework supports dynamic, entropy-driven generation of 8-bit S-boxes, achieving stronger nonlinearity, near-ideal entropy, and acceptable differential uniformity. Structural penalties such as fixed-point removal and cluster dispersion further strengthen resistance to algebraic and statistical attacks. These features collectively offer a promising path toward resilient, adaptive encryption for IoT and edge systems.

Moreover, the structural integrity of the proposed S-boxes is reinforced through neural refinement techniques that eliminate fixed points and disrupt local clustering, thereby enhancing resistance to a range of cryptanalytic techniques, including linear approximation, algebraic modeling, and interpolation attacks. Taken together, these characteristics position the neural-enhanced Curupira-1 framework as a promising alternative for future secure deployments in IoT and edge-computing systems, where runtime agility and cryptographic resilience are increasingly critical.

5.8 Limitations and Future Work

Despite the promising results obtained in terms of cryptographic strength, adaptability, and deployment readiness, the proposed neural-enhanced S-box generation framework is not without limitations. One key constraint lies in the computational overhead associated with neural training. While the neural network component is relatively lightweight compared to deep convolutional models, it still requires several hundred training epochs for optimal convergence, which may not be feasible in ultra-constrained environments or on-the-fly deployment scenarios.

Additionally, although the current model generalizes well across candidate S-boxes, it has been evaluated primarily under static architectural conditions (i.e., a fixed MLP topology and constant hyperparameters). There is still room to optimize the architecture for runtime performance and memory efficiency, particularly for use in embedded hardware. Future work will investigate the compression and quantization of the neural model using techniques such as layer pruning, weight sharing, and knowledge distillation. Furthermore, exporting the model to edge-compatible formats such as ONNX or TensorFlow Lite will be prioritized to enable integration into microcontrollers and IoT devices, such as the Raspberry Pi 5 or ARM Cortex-M processors.

Another limitation is that the current neural enhancement pipeline focuses exclusively on the S-box (substitution) layer of the CURUPIRA-1 cipher. However, modern block cipher security relies not only on substitution quality but also on diffusion, key scheduling, and permutation logic. As such, future extensions of this work will explore neural optimization of other components in the cipher pipeline—particularly the diffusion matrix and round key generation mechanisms—potentially enabling cipher-wide, model-driven cipher design.

Moreover, while structural robustness metrics such as differential uniformity, Walsh bias, nonlinearity, and entropy have been extensively validated, formal cryptanalysis against algebraic and higher-order differential attacks remains an open area for further investigation. Incorporating adversarial training or constraint-based symbolic verification may further strengthen the security guarantees of the generated components.

Finally, the framework's adaptability in networked environments—especially for distributed cryptographic deployments—opens new avenues of research, including secure S-box negotiation protocols, resilience to injection attacks during S-box transmission, and integration with post-quantum key exchange schemes.

In conclusion, although the proposed neural-enhanced S-box generator demonstrates solid baseline performance and strong adaptability, there are still several directions for future work. First, reducing the computational cost of neural training remains a priority. Techniques like pruning and quantization could help make the system more suitable for deployment in constrained environments, especially when targeting TinyML frameworks. Second, the current approach could be extended beyond the S-box to other essential cipher components—such as diffusion layers and key scheduling—to move toward a more unified, neural-assisted cipher architecture. Third, more thorough formal analysis is needed, particularly to evaluate resistance against algebraic and side-channel attacks. Lastly, applying this framework in distributed and reconfigurable settings, like multi-agent IoT systems, opens promising opportunities for scalable and adaptable encryption in real-world deployments.

Chapter 6

Conclusion

This research presents a neural-enhanced framework for improving the cryptographic strength and adaptability of the CURUPIRA-1 lightweight block cipher, with a particular focus on optimizing its substitution layer (S-box) using machine learning techniques. The motivation behind this work is rooted in the growing need for secure and efficient cryptographic solutions that are suitable for resource-constrained environments—such as IoT and edge computing platforms—where adaptability and lightweight design are critical considerations.

By integrating a neural refinement pipeline into the S-box generation process, this work demonstrates that it is possible to achieve strong cryptographic properties—including high nonlinearity, low differential uniformity, minimal Walsh distribution bias, and high entropy—while maintaining the lightweight and modular design philosophy of CURUPIRA-1. The implementation of of evolutionary strategies coupled with supervised neural optimization enables the adaptive generation of S-boxes that are not only secure but also reconfigurable at runtime, providing enhanced defense against precomputation and profiling attacks.

Furthermore, the neural approach inherently supports per-session or per-device S-box instantiation, which opens new avenues for adaptive cryptographic designs in distributed and heterogeneous networks. Unlike fixed substitution tables found in traditional lightweight ciphers (e.g., PRESENT and

PUFFIN), the proposed methodology enables context-aware, on-demand cryptographic personalization, a feature particularly desirable in modern edge computing platforms.

In addition to improving substitution-layer security, this research lays the groundwork for applying similar machine learning–driven design strategies to other components of the cipher architecture, such as diffusion layers and key scheduling. Preliminary results indicate that such integration has the potential to yield system-wide improvements in cipher strength without introducing notable computational overhead—especially once the neural models are compressed and optimized for embedded deployment (e.g., via ONNX or TinyML).

Future work will involve extending the neural optimization framework to cover end-to-end cipher pipelines, evaluating robustness under formal cryptanalysis (e.g., algebraic, differential, and side-channel models), and implementing the complete system in real-world IoT hardware such as Raspberry Pi 5 and ARM Cortex-M microcontrollers. Additionally, the potential of using the exported S-boxes in secure communication protocols—transmitted as JSON-encoded cryptographic payloads between trusted nodes—will be investigated for scenarios requiring secure bootstrapping and remote key negotiation.

In summary, this thesis contributes a scalable and adaptive methodology for enhancing lightweight ciphers using neural networks. It demonstrates the feasibility of integrating modern machine learning techniques into classical symmetric-key cryptographic design, and paves the way toward more secure, runtime-adjustable, and resilient cryptographic systems suitable for the next generation of distributed and resource-constrained technologies.

References

[1] Afzal, S., Yousaf, M., Afzal, H., Alharbe, N., and Mufti, M. R. (2020). Cryptographic strength evaluation of key schedule algorithms. *Security and Communication Networks*, 2020(1):3189601.

[2] Agarap, A. F. (2018). Deep learning using rectified linear units (relu). *arXiv preprint arXiv:1803.08375*.

[3] Ahmed, W. E. (2019). A modern method for constructing the s-box of advanced encryption standard. *Applied Mathematics*, 10(4):234–244.

[4] Alexa, M. (2002). Linear combination of transformations. *ACM Trans. Graph.*, 21(3):380–387.

[5] Bajaj, C. (1988). The algebraic degree of geometric optimization problems. *Discrete & Computational Geometry*, 3:177–191.

[6] Barreto, P. and Simplicio, M. (2007). Curupira, a block cipher for constrained platforms. *Anais do 25o Simpósio Brasileiro de Redes de Computadores e Sistemas Distribudos-SBRC*, 1:61–74.

[7] Bouvier, C., Canteaut, A., and Perrin, L. (2023). On the algebraic degree of iterated power functions. *Designs, Codes and Cryptography*, 91(3):997–1033.

[8] Check Point Software Technologies (2025). The state of cyber security 2025. `https://engage.checkpoint.com/security-report-2025`. Accessed: 2025-05-25.

[9] cio Jr, M. S., Barreto, P. S., Carvalho, T. C., Margi, C. B., and Mats, N. (2008). The curupira-2 block cipher for constrained platforms: Specification and benchmarking. *PiLBA'08 Privacy in Location-Based Applications*, page 123.

[10] Daemen, J. (1995). *Cipher and hash function design strategies based on linear and differential cryptanalysis*. PhD thesis, Doctoral Dissertation, March 1995, KU Leuven.

[11] Dani, S. G. (1977). Spectrum of an affine transformation. *Duke Mathematical Journal*, 44(1):129 – 155.

[12] Daubechies, I., DeVore, R., Foucart, S., Hanin, B., and Petrova, G. (2022). Nonlinear approximation and (deep) relu networks. *Constructive Approximation*, 55(1):127–172.

[13] de Canniere, C., Biryukov, A., and Preneel, B. (2006). An introduction to block cipher cryptanalysis. *Proceedings of the IEEE*, 94(2):346–356.

[14] den Hollander, T., Kleine, S., Mula, M., Slamanig, D., and Spindler, S. A. (2024). More efficient isogeny proofs of knowledge via canonical modular polynomials. Cryptology ePrint Archive, Paper 2024/1738.

[15] Duarte, F. (2024). 60+ amazing iot statistics (2024–2030). `https://explodingtopics.com/blog/iot-stats`. Accessed: 2025-05-25.

[16] Farahmand, A.-M., Barreto, A., and Nikovski, D. (2017). Value-Aware Loss Function for Model-based Reinforcement Learning. In Singh, A. and Zhu, J., editors, *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, volume 54 of *Proceedings of Machine Learning Research*, pages 1486–1494. PMLR.

[17] Glorot, X., Bordes, A., and Bengio, Y. (2011). Deep sparse rectifier neural networks. In Gordon, G., Dunson, D., and Dudík, M., editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pages 315–323, Fort Lauderdale, FL, USA. PMLR.

[18] Grover, A., Wang, E., Zweig, A., and Ermon, S. (2019). Stochastic optimization of sorting networks via continuous relaxations. *arXiv preprint arXiv:1903.08850*.

[19] He, K., Zhang, X., Ren, S., and Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*.

[20] Hornik, K., Stinchcombe, M., and White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366.

[21] Hussain, I., Anees, A., Al-Maadeed, T. A., and Mustafa, M. T. (2019). Construction of s-box based on chaotic map and algebraic structures. *Symmetry*, 11(3).

[22] Idris, M. F., Teh, J. S., Yan, J. L. S., and Yeoh, W.-Z. (2021). A deep learning approach for active s-box prediction of lightweight generalized feistel block ciphers. *IEEE Access*, 9:104205–104216.

[23] Kitsos, P., Sklavos, N., Parousi, M., and Skodras, A. N. (2012). A comparative study of hardware architectures for lightweight block ciphers. *Computers & Electrical Engineering*, 38(1):148–160.

[24] Kuzminykh, I., Yevdokymenko, M., and Sokolov, V. (2023). Encryption algorithms in iot: security vs lifetime. *Social Science Research Network*, pages 1–21.

[25] Li, L., Liu, J., Guo, Y., and Liu, B. (2022). A new s-box construction method meeting strict avalanche criterion. *Journal of Information Security and Applications*, 66:103135.

[26] Mar, P. P. and Latt, K. M. (2008). New analysis methods on strict avalanche criterion of s-boxes. *World Academy of Science, Engineering and Technology*, 48(150-154):25.

[27] Mena, G., Belanger, D., Linderman, S., and Snoek, J. (2018). Learning latent permutations with gumbel-sinkhorn networks. *arXiv preprint arXiv:1802.08665*.

[28] Norouzi, M., Fleet, D. J., and Salakhutdinov, R. R. (2012). Hamming distance metric learning. In Pereira, F., Burges, C., Bottou, L., and Weinberger, K., editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc.

[29] Noura, H. N. and Chehab, A. (2022). Efficient binary diffusion matrix structures for dynamic key-dependent cryptographic algorithms. *Journal of Information Security and Applications*, 68:103264.

[30] of Standards, N. I., (NIST), T., Dworkin, M. J., Barker, E., Nechvatal, J., Foti, J., Bassham, L. E., Roback, E., and Jr., J. D. (2001). Advanced encryption standard (aes).

[31] Pan, W., Shen, X., and Liu, B. (2013). Cluster analysis: unsupervised learning via supervised learning with a non-convex penalty. *J. Mach. Learn. Res.*, 14(1):1865–1889.

[32] Peluse, S. (2023). Finite field models in arithmetic combinatorics–twenty years on. *arXiv preprint arXiv:2312.08100*.

[33] Popescu, M.-C., Balas, V. E., Perescu-Popescu, L., and Mastorakis, N. (2009). Multilayer perceptron and neural networks. *WSEAS Transactions on Circuits and Systems*, 8(7):579–588.

[34] Ramchoun, H., Ghanou, Y., Ettaouil, M., and Janati Idrissi, M. A. (2016). Multilayer perceptron: Architecture optimization and training.

[35] Ren, X., Zhao, K., Riddle, P. J., Taskova, K., Pan, Q., and Li, L. (2023). Damr: Dynamic adjacency matrix representation learning for multivariate time series imputation. *Proc. ACM Manag. Data*, 1(2).

[36] Rong, J., Dong, X., Han, Y., and Cheng, R. (2025). S-box construction algorithm based on neural networks and genetic algorithms. In *Proceedings of the 2024 2nd International Conference on Electronics, Computers and Communication Technology*, CECCT '24, page 113–118, New York, NY, USA. Association for Computing Machinery.

[37] Sasaki, Y., Ling, S., Guo, J., and Bao, Z. (2019). Peigen – a platform for evaluation, implementation, and generation of s-boxes.

[38] Siddiqui, N., Yousaf, F., Murtaza, F., Ehatisham-ul Haq, M., Ashraf, M. U., Alghamdi, A. M., and Alfakeeh, A. S. (2020). A highly nonlinear substitution-box (s-box) design using action of modular group on a projective line over a finite field. *PLOS ONE*, 15(11):1–16.

[39] Takeshita, O. Y. (2007). Permutation polynomial interleavers: An algebraic-geometric perspective. *IEEE Transactions on Information Theory*, 53(6):2116–2132.

[40] Tayel, M., Dawood, G., and Shawky, H. (2018). Block cipher s-box modification based on fisher-yates shuffle and ikeda map. In *2018 IEEE 18th International Conference on Communication Technology (ICCT)*, pages 59–64.

[41] Wang, K., Kallus, N., and Sun, W. (2024). The central role of the loss function in reinforcement learning. *arXiv preprint arXiv:2409.12799*.

[42] Wu, F. and Beltrame, G. (2020). Cluster-based penalty scaling for robust pose graph optimization. *IEEE Robotics and Automation Letters*, 5(4):6193–6200.

[43] Xu, T., Scaffidi, T., and Cao, X. (2020). Does scrambling equal chaos? *Phys. Rev. Lett.*, 124:140602.

[44] Xue, X., Zhang, K., Tan, K. C., Feng, L., Wang, J., Chen, G., Zhao, X., Zhang, L., and Yao, J. (2022). Affine transformation-enhanced multifactorial optimization for heterogeneous problems. *IEEE Transactions on Cybernetics*, 52(7):6217–6231.

[45] Zhang, R., Shu, R., Wei, Y., Zhang, H., and Wu, X. (2023). A novel s-box generation methodology based on the optimized gan model. *Computers, Materials & Continua*, 76(2).

[46] Zhang, X., Chen, F., Chen, B., and Cao, Z. (2015). A new scheme for implementing s-box based on neural network. In *2015 International Conference on Computational Science and Computational Intelligence (CSCI)*, pages 571–576.