

A Fully Trained Neural Network Surrogate of the AES128 Encryption

Mohamed El Hassani *, Mahdi Madani, and El-Bay Bourennane

ImViA Laboratory, Université Bourgogne Europe, IMVIA UR 7535 Dijon, France

Email: Mohamed.El-Hassani@ube.fr (M.E.H.); Mahdi.Madani@ube.fr (M.M.); ebourenn@ube.fr (E.-B.B.)

*Corresponding author

Manuscript received March 31, 2026; revised June 24, 2026; accepted August 27, 2026; published October 19, 2026

Abstract—This paper presents a fully learned neural network surrogate of the Advanced Encryption Standard (AES) encryption algorithm under any oracle-provided key. Each round is split into three learned modules—ShiftRows-SubBytes, MixColumns, and AddRoundKey—with the round key given as input so one model handles arbitrary keys. Trained on fewer than 50,000 samples, the surrogate empirically attains perfect bit accuracy with zero observed errors over millions of unseen evaluations, without overfitting. Inspecting the models reveals behaviour atypical of deep learning: accuracy drops with depth, hinges on a sharp hidden-dimension threshold, and rests on near-binary sparse weights that collapse under pruning. As an exact, differentiable model of AES, the surrogate provides an empirical baseline for gradient-based and surrogate-assisted analysis of cryptographic primitives.

Keywords—deep learning, Advanced Encryption Standard (AES), surrogate modeling, gradient-based learning, block cipher emulation

I. INTRODUCTION

The Advanced Encryption Standard (AES) algorithm [1] is a symmetric cipher operating on 128-bit blocks of data. It is an internationally standardized algorithm that represents the prevailing reference for symmetric key encryption [2]. The AES is widely integrated in many fields, such as social messaging (Whatsapp-Messenger), and hardware components (such as processors, Systems on Chip (SoC), Field-Programmable Gate Arrays (FPGA)) [3, 4].

The AES algorithm is still resisting cryptanalysis attacks trying to break the whole 10-round system. For example, the most efficient attack in this scope is the biclique attack [5]. This attack successfully reduced the key space from 2^{128} to $2^{126.1}$, but it is still resistant against cryptanalysis. The side channel attacks analysing the residual power traces, also fail to find practical vulnerabilities to attack the algorithm [6]. Therefore, despite the different techniques used to crack the AES algorithm (brute force attacks, side channel attacks, plaintext-ciphertext attack, etc.), actually, there is no known system that succeeds.

With the advent of Artificial Intelligence (AI) in recent years, several studies have attempted to model the AES encryption using Deep Learning (DL) models. However, the complexity and non-linearity of its internal functions have highlighted the difficulty of training the models. Furthermore, all the models presented in the literature suffered from either overfitting or insufficient approximation of the AES behavior [7–10]. Consequently, even the best models could not achieve 100% accuracy. This paper presents a significant model trained to learn the AES algorithm. The principle adopted consists of modeling the fundamental operations of the AES algorithm using classical DL models. The result of

our study is a neural network surrogate learned end to end of the 10 rounds of the AES128 algorithm for encryption. Unlike other studies [7–10], our models are not a result of overfitting, and they achieve empirically an accuracy of 100%. That will be confirmed through extensive testing. Additionally, the resulting neural network will be inspected and compared to a similar architecture trained on a task inherently more suitable for gradient-based optimization, to highlight the unique underlying characteristics of our result.

The remainder of this paper is structured as follows. In Section II, previous works in this area are discussed. In Section III, the processing steps of each of the AES internal functions are presented. In Section IV, the proposed model, the training and testing environment are explained step by step. An inference latency test of the model is also conducted, and the potential and limitations for deep-learning assisted cryptanalysis are presented. In Section V, the model is compared to a similar architectural design in order to understand the internal model functioning. The model's efficiency is also compared to the previous works. In Section VI, the irregularity of the AES emulation task and its unsuitability for deep learning are highlighted as the work is concluded, followed by some directions for future work.

II. RELATED WORK

Ameen and Mahdi [7] used a small Multilayer Perceptron (MLP) [11] (4-16-16-1) trained on a constrained input space, with the key embedded within the fully connected layer. This work thereby laid a preliminary foundation for this line of research. Jeong *et al.* [8] used 2 models for the encryption emulation of a few cipher blocks including AES with rounds ranging from 1 round to the full 10 rounds. The models used are a 3-layer MLP (hidden layers of sizes 512, 1024, 512), as well as a 3-layer Bidirectional Long Short-Term Memory (LSTM) (256 units per layer) with a fully connected layer. To train these models, they used $2^{22} = 4,194,304$ training samples. The reported bit accuracy was 50% for 1 round of AES128, which is equivalent to random guessing. They then concluded that AES128 could not be emulated due to its large key size (128 bits) compared to the other block ciphers they analyzed. Hu and Zhao [9] deployed a 4-hidden-layer-MLP to run a plaintext restoration attack on AES. This can be viewed as an inverse encryption emulation, making the two approaches equivalent in terms of complexity. This is because from a machine learning standpoint both involve learning an arbitrary but fixed permutation on 128-bit space. However, Baek and Kim [12] and references therein critiqued the reproducibility of their work as an overfitting result. Alimanov *et al.* [10] further support this by showcasing the

difficulty of learning the AES encryption with deep learning-based techniques. They created four separate models each simulating one internal function of the AES algorithm, as follows:

- SubBytes modeled per-byte by training a 1-hidden-layer MLP with 1000 neurons to learn the byte substitution (one network for each of the 16-byte positions).
- ShiftRows modeled with a single-hidden-layer MLP (128 hidden neurons).
- MixColumns modeled using four separate Convolutional Neural Networks (CNNs) [13] (one per column) each with two convolutional layers and one fully-connected layer.
- AddRoundKey modeled with the same 128-neuron MLP architecture as ShiftRows.

Using 65,000 random samples for training, the combined system of 4 sub-models achieved about 99.97% accuracy on the full AES128 encryption. In their work, they also discussed using backpropagated gradients crossing multiple networks to train a target network. They concluded that the gradients for AES training are not very meaningful and also heavily attenuated and modified after crossing a foreign network.

In this work, improvements on prior efforts are made by:

- Achieving (empirically) 100% accuracy for full AES128 encryption.
- Reducing the number of models deployed.
- Using more compact architectures.
- Including the AES key as part of the model's input (so that one learned model can handle encryption with arbitrary keys, rather than one model per key).
- Studying the inner structure of the resulting models.

Our proposed architectures are able to generalize with 100% accuracy from <50,000 samples to essentially the entire input space of 128-bit plaintexts (and 128-bit keys) with no observed errors. This indicates that the said architectures have effectively learned the underlying encryption function, rather than simply overfitting or mimicking outputs on the training data.

III. AES ALGORITHM OVERVIEW

The AES is a block cipher operating under the control of Cipher Key (CK) and processes four internal functions during R rounds. Three main versions are found in the literature, different only by the size of the used CK and the number of rounds. AES 128 uses a CK of 128-bits and operates on 10 rounds, AES 192 uses a CK of 192-bits and operates on 12 rounds, and AES 256 uses a CK of 256-bits and operates on 14 rounds. As in the literature studies, during the rest of this work, the AES 128 version will be used to conduct the experiments. The internal architecture of the AES algorithm is based on four main functions, SubBytes, ShiftRows, MixColumns, and AddRoundKey. AES128 functions in rounds, so for each encryption operation it completes 10 sequential rounds on the plaintext. A 128-bit input block is viewed as a 4×4 matrix of bytes (16 bytes total). The first operation of the round is SubBytes. It replaces each individual byte with another byte following a predetermined AES substitution table [1]. The rows of the resulting block will be shifted i positions to the left, where i is the number of the row in the block. This is the ShiftRows function.

The columns of the resulting block will be multiplied with a fixed matrix in the Galois Field (GF) (2^8) domain where each byte is viewed as a polynomial of its coefficients, and

uses the irreducible polynomial as defined in Eq. (1).

$$m(x) = x^8 + x^4 + x^3 + x + 1 \quad (1)$$

The last operation of the round is the AddRoundKey operation. It corresponds to a simple Exclusive-OR (XOR) between the 128-bit state and the 128-bit round key. It should be noted that the initial 128-bit key is expanded to 11 128-bit keys as per the key schedule. That means before starting a new round, an intermediate round key will be calculated. Each round uses one of these generated keys (Fig. 1). The key schedule was excluded from this work since the round transformations (SubBytes, ShiftRows, MixColumns, AddRoundKey) are structurally independent of it, which only derives the subkeys consumed by AddRoundKey, which already accepts an arbitrary key. Our focus is thus the round transformations themselves.

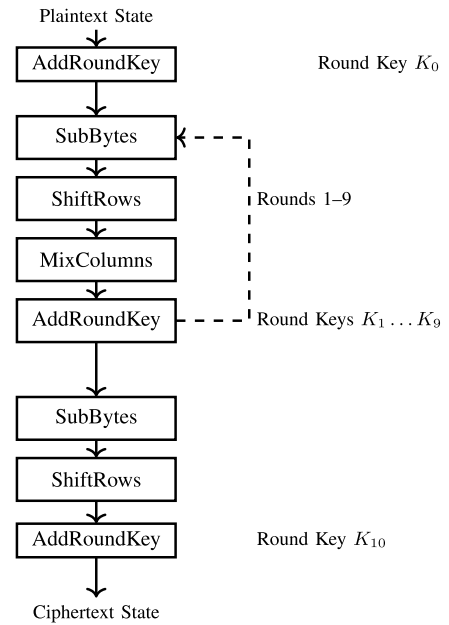


Fig. 1. Structure of the AES128 encryption algorithm.

IV. PROPOSED ENCRYPTION PIPELINE

The simulation of a full round with a single model fails using various architectures such as MLPs, CNNs, and even Transformers [14]. After a deep investigation and many experiments, the MixColumns operation was found to be causing the failure of the learning process. Consequently, a transformer could be trained to 100% accuracy applying ShiftRows $\circ$ SubBytes $\circ$ AddRoundKey, but the same architecture couldn't be trained to apply MixColumns all by itself (where $\circ$ is the function composition symbol). Therefore, it was chosen to implement (with respect to the AES round operations' order) ShiftRows $\circ$ SubBytes as a single model, MixColumns as a single model, and AddRoundKey as a single model (to keep the round order).

A. ShiftRows $\circ$ SubBytes

The proposed architecture for this task is an MLP. It was noticed that training the ShiftRows $\circ$ SubBytes model with 128 bits input and output resulted in little to no learning. However, representing the input and output blocks (originally 128-bits) as 16 bytes (values from 0 to 255), the same model

easily learned the process. To counter this challenge while keeping a uniform 128-bit input and output for all models, a byte encoding layer is added before the MLP. This layer corresponds to a single fully connected from 8 input bits to a 384 hidden dimension (Fig. 2). This makes it easier for the model to understand the byte structure that governs this task.

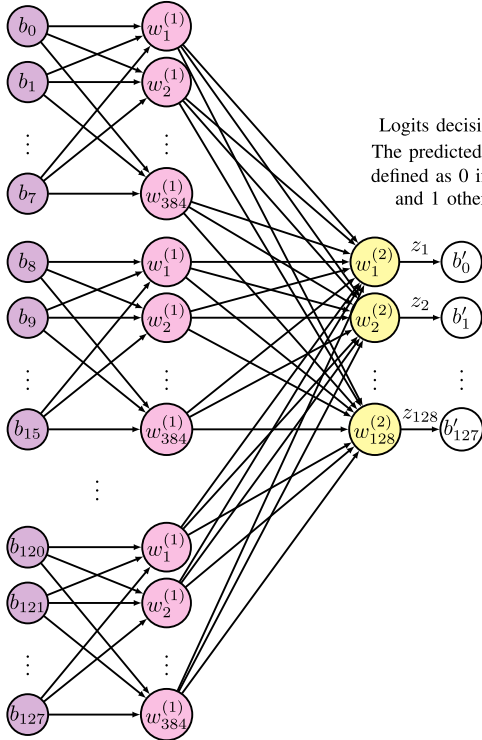


Fig. 2. Architecture of the proposed MLP emulating the ShiftRows $\circ$ SubBytes functions.

The dataset size used is 60,000 (128-bit-plaintext–128-bit-ciphertext pairs), 50,000 for training, and 10,000 for validation. Training parameters are:

- Learning rate of 0.001.
- Batch size of 512.
- Adam optimizer.
- 60 training epochs.
- Binary cross-entropy as the loss function.
- Simple bit-accuracy as evaluation metric.

The training converges to 100% validation accuracy. Furthermore, to validate this block, inference was run on 500,000 ($10\times$ the training set size) previously unseen 128-bit Plaintext samples. The model produced the correct ShiftRows $\circ$ SubBytes output for every one of them. This demonstrates that the model truly learned the task, using a very compact architecture.

B. MixColumns

To model the MixColumns operation, several MLP architectures were experimented (treating the input as separate bytes, as individual columns, or as a flat 128-bit vector). As a result, the best achieved accuracy was 98% which means that the probability of the model inferring the entire 128-bit ciphertext correctly is merely 7% (Probability to infer 128 bits correctly is equal to the probability to infer one bit correctly raised to the power 128: $0.98^{128} = 0.075 = 7.5\%$). Instead, a column CNN was deployed represented in Fig. 3. This architecture acts on columns, therefore it was instantiated 4 times but trained simultaneously (same loss—

same optimization). It achieved 100% binary accuracy, and passed our inference test on 500,000 new samples without error. The training parameters were identical to those of the previous model.

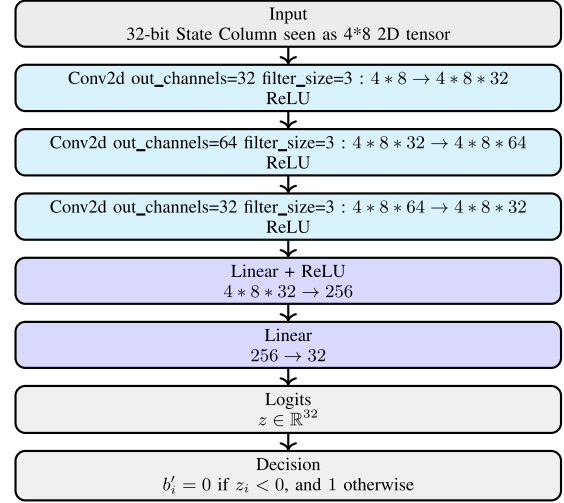


Fig. 3. Architecture of the CNN used for the MixColumns experiment (1 column).

C. AddRoundKey

To model the AddRoundKey (ARK) operation, an MLP model was used. To allow our surrogate to handle arbitrary keys (rather than a single fixed key), the key was included as part of the input vector. Specifically, the ARK model takes a 256-bit input (the plaintext concatenated with its 128-bit key) and outputs the 128-bit ciphertext produced by XORing the plaintext with the key. The proposed model is the MLP shown in Fig. 4. 50,000 training pairs were used for the AddRoundKey model, similar in size to earlier sets. This was done by taking 500 random plaintexts and for each plaintext pairing it with 100 random keys (yielding $500 \times 100 = 50,000$ unique plaintext-key pairs). The training followed the same parameters as the previous ones. Consequently, the desired binary accuracy of 100% was achieved, and the test on 500,000 samples was validated without error.

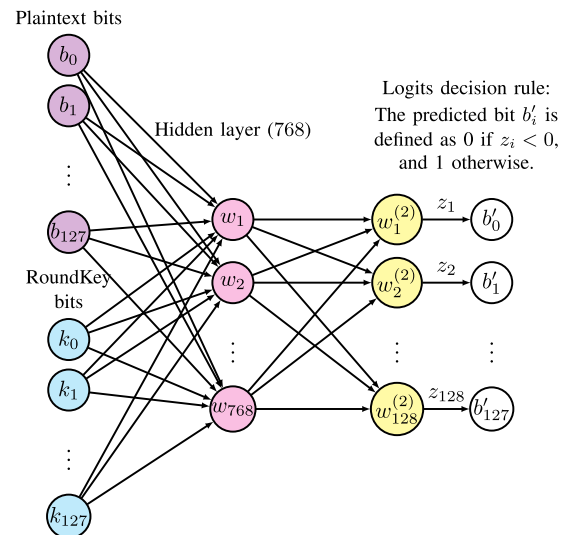


Fig. 4. Architecture of the MLP used for the ARK experiment.

D. Full AES128 Setup

Finally, the learned models were cascaded to perform full AES128 encryption. The entire learned encryption pipeline was then tested on 500,000 new random plaintexts (with random keys). The cascaded architecture produced zero errors. In total, the AddRoundKey module was invoked 5,500,000 times (11 per encryption) with no errors, the ShiftRows $\circ$ SubBytes module 5,000,000 times (10 per encryption) with no errors, and the MixColumns module 4,500,000 times (9 per encryption) with no errors. In summary, a neural surrogate for the AES128 encryption that produces exact outputs was successfully built.

E. Inference Latency

A comparison of the latency and throughput between the neural network implementation and a classical CPU implementation of the AES128 encryption algorithm was conducted (Table 1). All measurements were performed on an NVIDIA Tesla V100-PCIE GPU (32 GB) and an Intel Xeon Gold 6248 CPU (2.50 GHz), running PyTorch 2.3.0 (CUDA 12.1) and NumPy 2.0.2 with TensorFlow-32 enabled. All timings were taken at a batch size of 262,144 blocks (one block = one 16-byte AES state), the throughput-saturated regime in which the GPU's per-call overhead is fully amortized. Neural-model timings are end-to-end, they include byte-to-bit packing, the forward pass, and bit-to-byte unpacking, since the models operate natively on bit vectors whereas the CPU reference operates on bytes.

Table 1. Inference throughput and per-block latency at batch size = 262144

Primitive	Impl.	Throughput (blk/s)	Latency (μ s)
MixColumns	Neural	8.3×10^5	1.205
	NumPy	8.38×10^6	0.119
ShiftRows $\circ$ SubBytes	Neural	3.41×10^6	0.293
	NumPy	1.73×10^7	0.058
AddRoundKey	Neural	1.30×10^7	0.077
	NumPy	5.56×10^8	0.0018

The throughput and latency comparison are reported here mainly for completeness. As expected, the NumPy/CPU reference is faster than the neural surrogate across all three primitives, with the gap widening as the underlying operation becomes cheaper to compute directly—AddRoundKey being a plain per-byte XOR.

F. Potential and Limitations in Deep-learning-assisted Cryptanalysis

Although no attack is mounted, the surrogate is relevant to gradient-based cryptanalysis in one specific sense: it is a continuous, differentiable relaxation of AES that agrees with the cipher on all binary inputs. This is the same enabling step used against learned hash functions, where a discrete map is made differentiable so that an input can be optimized by gradient descent toward a target output [15, 16]. Given such a model, one could in principle relax the key to continuous values and descend on it to match a known plaintext-ciphertext pair—a gradient-based key search. Such input-optimizing methods are well established for images, where adversarial-example techniques perturb an input along its gradient to steer the out-put [17, 18]. They succeed there because the input gradient is a reliable local guide: stepping along it steadily moves the output toward the target. AES is

designed to break exactly this property—by diffusion, a single-bit change scrambles the output—so the surrogate's input gradients, while defined and non-zero (the model trains by backpropagation), may offer only a weak signal toward the correct key [10]. Whether they can be made to drive a practical key search is an open question for future work. Our contribution is the relaxation itself—an exact, differentiable model of AES—which is the prerequisite such methods require.

V. AN EXAMINATION OF INTERNAL MODEL REPRESENTATIONS AND COMPARISON

A. Internal Model Representations

In this section, the reasons why these models are different will be explained. The focus is on the AddRoundKey (ARK) model, selected for its simplicity, which will be compared to a reference model with a similar architecture. This model is an MLP with one hidden layer using the same hidden dimension as our ARK model. This MLP is trained on the MNIST handwritten digit dataset [13] (60,000 training samples, which closely matches the 50,000 samples used to train our models). This learning was chosen because it is a canonical benchmark in deep learning. This model will be referred to as the MNIST-MLP moving forward.

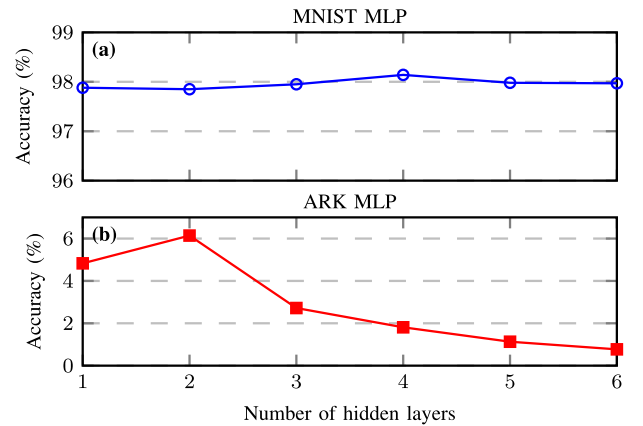


Fig. 5. Accuracy vs. model depth (number of hidden layers) for the (a) MNIST MLP and (b) ARK MLP.

Firstly, the relationship between the number of hidden layers, and the final accuracy was inspected. An average hidden layer dimension of 128 was chosen, and the number of hidden layers was varied from 1 to 6, with each configuration trained for 50 epochs. To evaluate the MNIST MLP, accuracy was measured by dividing the number of correct predictions by the total number of samples. For the ARK MLP, byte-level accuracy was adopted, that is, the number of correct bytes divided by the total number of bytes. For reference a random guess with the ARK MLP yields a success probability of 0.0039 (0.39%). A random guess for the MNIST MLP yields 10% (10 output classes total). Byte-level accuracy was chosen for the ARK model, because block-level accuracy (all 128 bits correct or not, i.e., percentage of correct 128-bit blocks) stayed at zero in these experiments—inexpressive of the learning progress. The result is shown in Fig. 5. From Fig. 5(a), it can be concluded that, for the MNIST MLP, accuracy exhibits a slight positive correlation with the number of hidden layers. The ARK MLP,

on the other hand, was strongly impacted in a counterintuitive manner. Accuracy decreases with the number of hidden layers, as illustrated by Fig. 5(b). That is because the gradient signals produced by these models for the backpropagation do not travel well [10]. Effectively, the deeper the network, the harder it was to train on the AES mapping.

Secondly, the relationship between the dimension of the hidden layer and the final accuracy was inspected. The number of hidden layers was fixed to 1, and the hidden dimension was varied for the 2 models. The same metric of accuracy was used for the MNIST MLP, and block accuracy was used for the ARK MLP. Both models' respective capacities to produce correct results in their ground truth space are now assessed. The ARK model is certainly at a natural disadvantage since its random guess probability in the ground truth space is $1/2128$ (near zero), while the random guess probability for the MNIST MLP is 10%. However, the interest lies in the evolution of the values taken by the accuracy, not the values themselves. That being said, Fig. 6 displays the final accuracy as a function of the hidden layer's dimension.

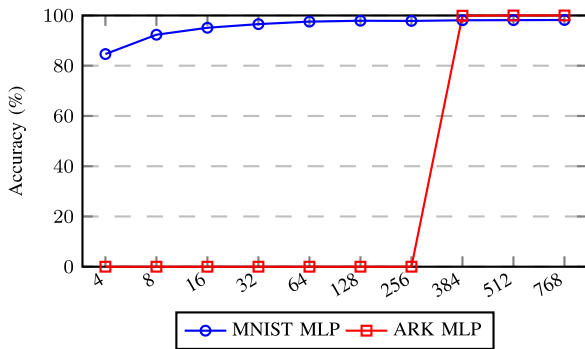


Fig. 6. Accuracy as a function of the hidden layer dimension for the MNIST MLP and the ARK MLP.

MNIST accuracy increases gradually as the hidden dimension grows, whereas ARK's accuracy leaps only when the model is large enough. It goes from not learning at all, to learning almost perfectly. This implies the existence of a class of architectures that are unable to uncover the underlying pattern (Hidden layer dimension < 384), and a class of architectures that can do so with ease (Hidden layer dimension ≥ 384). The counterintuitive phenomenon here is that the difference between these classes is very light. That is because the model does not assimilate the task like a true combinatory circuit, it approaches AES behaviour progressively. By the time the model learns to infer 128-bits correctly simultaneously (choose the right element out of 2128 possible elements), the learning is already almost complete.

To examine the internal workings of the model, the distribution of the weight values for both models was visualized as a histogram (Fig. 7(a) and (b)).

The ARK model is very sparse, with many weights near zero compared to the MNIST MLP. That is due to it trying to approximate a XOR function, therefore bringing weights closer to a two-state representation. Whereas the MNIST MLP's weight values follow a more uniform distribution. However, the maximum ARK weight value is 2.87, while it

is 0.28 for the MNIST MLP, and the weight magnitudes are observably larger for the ARK MLP. This showcases how weights are distributed in exact boolean transformation versus statistical feature separation.

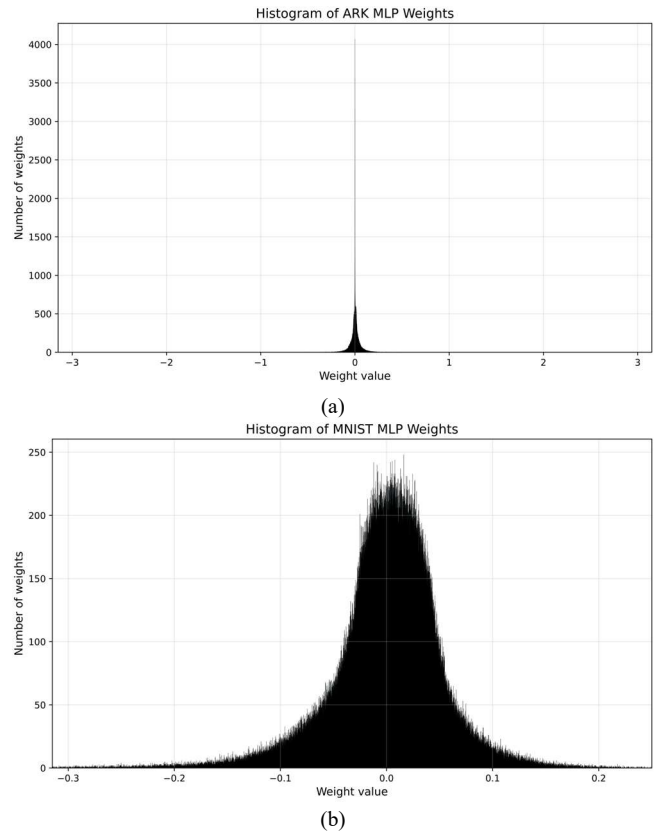
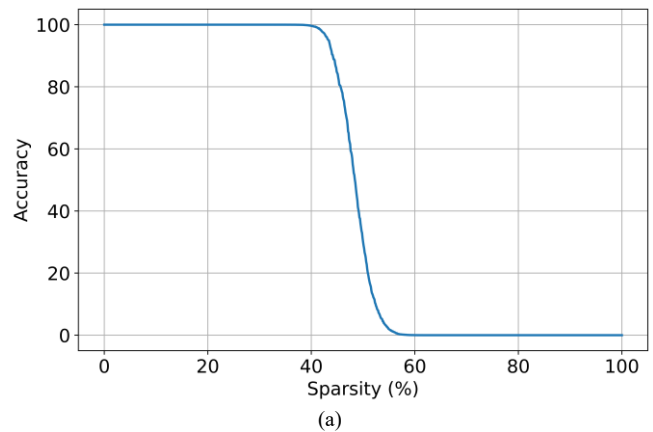


Fig. 7. Comparison of weight and bias value distributions for the two architectures: (a) ARK MLP weight distribution; (b) MNIST MLP weight distribution.

Given that the ARK model is sparse, pruning (unstructured) is proposed to test how it affects the accuracy of this model, compared to the reference MNIST MLP. This means that a percentage of the smaller value weights will be deactivated, and the model's reaction will be observed. The comparison will be between the 768 hidden dimension versions of the ARK and MNIST MLP, trained to reach respective validation accuracies of 100% and 98.2% without any pruning. Final convergence accuracy will be plotted as a function of the percentage of weights rendered inactive called sparsity (Fig. 8).



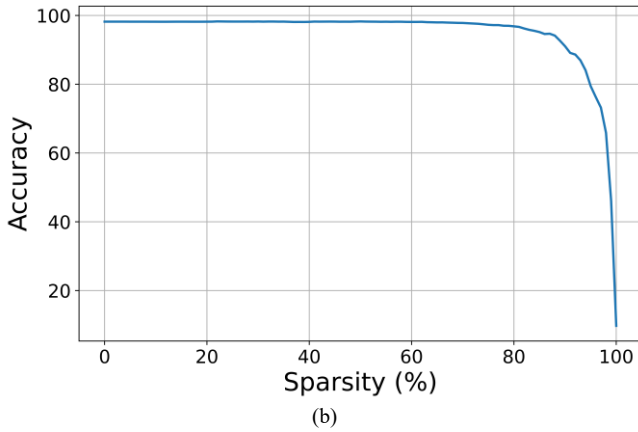


Fig. 8. Accuracy versus the percentage of inactive weights (sparsity) for both architectures. (a) ARK MLP; (b) MNIST MLP.

From the plot, the existence of a critical sparsity for the ARK model can be concluded, after which accuracy plummets to 0%. This suggests that the ARK model’s weights can be divided into two groups—essential vs. redundant. Once pruning starts affecting even a few of the essential weights, the model’s performance collapses. It can also be attributed to the size of the output ground truth domain. While for the MNIST MLP, the descent of accuracy is somewhat smoother and does not have a sharp cutoff like ARK. Meaning the accuracy value decreased progressively as more pruning was applied. For reference, the MNIST model’s accuracy drops from 90% to 20% while the sparsity percentage changes by 8.5%. As for the ARK model, accuracy drops from 90% to 20% while the sparsity percentage changes only by 4.39%. The MNIST model is also far more resistant to pruning than the ARK MLP, another manifestation of the studied difference between boolean transformation (ARK), and statistical separation (MNIST).

The models described in the previous section (represented by the ARK model) present many differences compared to more typical deep learning models (represented by the MNIST model):

- They achieved perfect accuracy throughout extensive testing without overfitting. That is not so typical of deep

learning models that often accomplish a complex task indeed, but remain prone to error.

- Putting more layers in the model did not mean better accuracy, actually it was quite the opposite (Fig. 5). Adding layers sharply penalized the final accuracy.
- The models are extremely architecture-dependent. Meaning there was no convergence for a given architecture, but near 100% accuracy after light changes to that architecture (Fig. 6).
- The models are sparse in terms of weights (Fig. 7). Additionally, the models’ parameters (weights) are either essential to the model’s functioning, or somewhat irrelevant, as observed after progressively pruning the ARK model (Fig. 8).
- While data representation impacts final accuracy and convergence in deep learning, it usually does so gradually. However, in our case, data representation can be the only difference between a model guessing randomly, and reaching perfect accuracy. An example of this is the ShiftRows ◦ SubBytes task (section IV).

B. Comparison to Previous Works

Table 2 compares our approach to the prior efforts discussed in Section II. Beyond the accuracy difference, two structural differences stand out. First, all prior approaches fix the key throughout the training data, so each trained system is tied to a single key; ours receives the round key as an explicit input and therefore emulates the cipher under arbitrary keys with a single set of weights. Second, relative to the closest baseline [10], the per-round ensemble was reduced from 22 networks to 3 while using fewer training samples. The prior baseline used 22 networks: 16 for SubBytes (one per byte), one each for ShiftRows and AddRoundKey, and 4 for MixColumns (one per column). Our pipeline instead uses a single network for ShiftRows ◦ SubBytes, one for MixColumns (counted as a single model since training and optimization were joint, with a shared loss over the full 128-bit output), and one for AddRoundKey. It should be noted, however, that for Ref. [8], a bit accuracy of 50% is equivalent to random guessing, and that Ref. [7] was trained on a constrained input space and does not generalize to the full AES.

Table 2. Comparison with prior deep-learning AES emulation approaches

Work	Architecture	#Nets	Training samples	Key as input	AES128 accuracy
Ameen and Mahdi [7]	Small MLP	1	Constrained	No	Partial ^a
Jeong <i>et al.</i> [8]	MLP or BiLSTM	1	2 ²²	No	50% ^b
Hu and Zhao [9]	4-layer MLP	1	—	No	Overfit [12]
Alimanov <i>et al.</i> [10]	MLPs + CNNs	22 ^c per round	~65,000	No	99.97%
Our work	MLPs + CNN	3 ^d per round	<50,000	Yes	100%

VI. CONCLUSION AND FUTURE WORK

This paper presented an optimized, minimal, non-overfitting neural surrogate of the AES128 encryption algorithm under arbitrary keys. Each round was decomposed into three learned modules, with the round key supplied as an input rather than fixed in the weights. The surrogate empirically reached perfect bit accuracy with no observed errors across millions of unseen evaluations, while reducing the per-round ensemble from 22 networks to 3. Inspecting a learned module further revealed behaviour atypical of deep learning: accuracy that falls as depth grows, a sharp width threshold separating no learning from near-perfect accuracy,

and near-binary, sparse weights that collapse beyond a critical pruning level—unlike a comparable MNIST classifier.

DL and classical model architectures are not very efficient at uncovering the complex binary relationships and patterns produced by the AES algorithm. This is due to the diffusion capability of the AES algorithm that does not bode well for the “continuity” aspect inherent to DL. As an example, if a single pixel value is changed in a picture of a cat, that picture would still be indifferent for the human eye as well as a cat detection model. This illustrates the continuity in natural images—small changes don’t usually flip the label. Deep learning thrives on such continuity. In stark contrast, for AES a change in one input bit completely scrambles the output

(due to the algorithm's diffusion property). There is no continuity or gradual change—one-bit difference gives an essentially independent output. Traditional deep learning models struggle in this setting. Therefore, achieving more advanced learning on this task may require a new class of specialized models or fundamentally different learning mechanisms. This investigation is left for future work, where the hope would be learning a full AES round with a single network.

CONFLICT OF INTEREST

The authors declare no conflict of interest.

AUTHOR CONTRIBUTIONS

M.E.H. conceived and designed the study, developed and implemented the proposed design, performed the experiments and the analysis, and wrote the original draft; M.M. supervised the work and contributed to the discussion, review, and correction of the manuscript; E.B.B. supervised the work and contributed to the discussion, review, and validation of the results; all authors have read and approved the final version of the manuscript.

FUNDING

This work was supported by a doctoral contract (contrat doctoral) funded by the French Ministry of Higher Education, Research and Innovation (MESRI).

ACKNOWLEDGMENT

Calculations were performed using HPC resources from DNUM CCUB (Centre de Calcul de l'Université Bourgogne Europe).

REFERENCES

- [1] J. Daemen and V. Rijmen. (1999). AES proposal: Rijndael. *Submission to the NIST*. [Online]. Available: <https://csrc.nist.gov/csrc/media/projects/cryptographic-standards-and-guidelines/documents/aes-development/rijndael-ammended.pdf>
- [2] R. Ganesh, B. U. I. Khan, A. R. Khan *et al.*, "A panoramic survey of the advanced encryption standard: From architecture to security analysis, key management, real-world applications, and post-quantum challenges," *International Journal of Information Security*, vol. 24, no. 5, 216, 2025.
- [3] J. Kala, J. Panda, and L. Tanwar, "FPGA implementation of a high throughput low power advanced encryption standard (AES-128)

- cipher," in *Proc. 2023 10th International Conference on Signal Processing and Integrated Networks (SPIN)*, 2023, pp. 367–372.
- [4] B. Sarkar, S. Avik, D. Debanjan *et al.*, "A survey on the advanced encryption standard (AES): A pillar of modern cryptography," *International Journal of Computer Science and Mobile Computing*, vol. 13, no. 4, pp. 68–87, 2024.
- [5] A. Bogdanov, D. Khovratovich, and C. Rechberger, "Biclique cryptanalysis of the full AES," in *Proc. International Conference on the Theory and Application of Cryptology and Information Security*, 2011, pp. 344–371.
- [6] S. Naserelden, N. Alias, A. Altigani *et al.*, "Advance attacks on AES: A comprehensive review of side channel, fault injection, machine learning and quantum techniques," *Edelweiss Applied Science and Technology*, vol. 9, no. 4, pp. 2471–2486, 2025.
- [7] S. Y. Ameen and A. H. Mahdi, "AES cryptosystem development using neural networks," *International Journal of Computer and Electrical Engineering*, vol. 3, no. 2, 2011.
- [8] O. Jeong, E. Ahmadzadeh, and I. Moon, "Comprehensive neural cryptanalysis on block ciphers using different encryption methods," *Mathematics*, vol. 12, no. 13, 1936, 2024.
- [9] X. Hu and Y. Zhao, "Research on plaintext restoration of AES based on neural network," *Security and Communication Networks*, vol. 2018, no. 1, 2018.
- [10] K. Alimanov, M. Lukac, and S. Nagayama, "Empirical hardness of AES cipher," *Bulletin of Networking, Computing, Systems, and Software*, vol. 13, no. 1, pp. 29–37, 2024.
- [11] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, pp. 533–536, 1986.
- [12] S. Baek and K. Kim, "Recent advances of neural attacks against block ciphers," in *Proc. Symposium on Cryptography and Information Security (SCIS)*, 2020.
- [13] Y. LeCun, L. Bottou, Y. Bengio *et al.*, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.
- [14] A. Vaswani, N. Shazeer, N. Parmar *et al.*, "Attention is all you need," *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 30, 2017.
- [15] F. Cakir, K. He, S. A. Bargal *et al.*, "Hashing with mutual information," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 41, no. 10, pp. 2424–2437, 2019.
- [16] L. Struppek, D. Hintersdorf, D. Neider *et al.*, "Learning to break deep perceptual hashing: The use case NeuralHash," in *Proc. the 2022 ACM Conference on Fairness, Accountability, and Transparency (FAccT)*, 2022, pp. 58–69.
- [17] C. Szegedy, W. Zaremba, I. Sutskever *et al.*, "Intriguing properties of neural networks," in *Proc. International Conference on Learning Representations (ICLR)*, 2014.
- [18] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," in *Proc. International Conference on Learning Representations (ICLR)*, 2015, pp. 1–11.

Copyright © 2026 by the authors. This is an open access article distributed under the Creative Commons Attribution License which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited ([CC BY 4.0](https://creativecommons.org/licenses/by/4.0/)).