Universiteit Leiden

Master Computer Science

HyperDrive: Hyperband with Gaussian Process for Efficient SCA Model Tuning

Name:	Ruilin Ma
Student ID:	3500926
Date:	03/06/2024
Specialisation:	CS: Artificial Intelligence
1st supervisor:	Guilherme Perin
2nd supervisor:	Nele Mentens

Master's Thesis in Computer Science

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

HyperDrive: Hyperband with Gaussian Process for Efficient SCA Model Tuning

Ruilin Ma

Leiden Institute of Advanced Computer Science, Leiden University
r.ma@liacs.leidenuniv.nl

Abstract. Side channel analysis is one of the top topics studied in cryptography. By analyzing the side signals emitted during the encryption and decryption processes, such as power traces, electromagnetic waves, heat signatures, etc, one could exploit these to attack and acquire the sensitive secret processed on a processing unit. With the advance of deep learning techniques, one could utilize the power of deep neural networks to learn the information within side channel patterns and perform an attack afterward swiftly. However, deep neural networks are hard to optimize because of their complexity and the profound number of different configurations of hyperparameter settings. Without prior knowledge, the enormous search space makes it difficult to evaluate all possible model configurations progressively, and the cost of an exhaustive search is overwhelming, making it practically infeasible. This work aims to address these issues from two perspectives. This work first provides a clear view and explanation of the most important factors for a good model to be trained, with coverage of architectural and learning hyperparameters. This is achieved by comprehensively studying the relationship and influence of the hyperparameters for competent SCA models. This study then proposes and evaluates HyperDrive, an enhanced Hyperband with Gaussian Process algorithm, to substantially speed up tuning velocity, reduce the overall cost, and obtain on-par or even better performance in finding the closest-to-optima model configuration for SCA tasks. Based on the comprehensive experimentations and analysis, this work aims to enable more powerful and efficient deep-learning-based SCA attacks.

Keywords: Side Channel Analysis · Hyperparameter Optimization · Deep Learning

Contents

1 Introduction

Side-channel analysis (SCA) is a powerful tool for obtaining secret information by manipulating the side-channel information that is widely used in the cryptography field. From the attack side, this technique can be utilized to guess the secret key of the encryption algorithms and obtain the plain text. From the defense side, the side-channel information is also a valuable asset for developers to optimize the implementation of the encryption algorithm in hardware and software to reduce the likelihood of being compromised. Side-channel information refers to the information that is unintentionally leaked in the encryption and decryption process, including but not limited to power consumption [KJJ99b], electromagnetic waves [QS01], thermal emissions [HS13], acoustic signals [GST14], etc. This information is the byproduct of the encryption and decryption process, as the processing unit inevitably generates redundant heat and electromagnetic wave emissions that contain the pattern of how the algorithm is being executed. Such information, if intentionally and properly collected, can be used to reverse-engineer to guess the key used in the process, and eventually lead to the revealing of the original plain text that should, in most cases, be kept secret.

The success of side-channel analysis is based on the accurate extraction and analysis of the side-channel information that contains leakage patterns, which requires great expertise and a huge amount of accumulated experience from the researcher to proceed. This is, however, sometimes struggling for the analyzer, as this information, such as power consumption traces, may contain extensive noise that will interfere with the interpretation of the information. Also, cryptography operations can be executed on various hardware platforms, such as ARM, X86/64, RISC-V, etc., and the corresponding information and noise pattern may differ significantly, posing additional obstacles to overcome to retrieve and analyze useful information. Therefore, traditional SCA often requires the analyzer to possess abundant knowledge and experience with hardware and software, which is unfortunately not always accessible.

The development of computational resources and algorithms opens up new opportunities and possibilities for SCA. The rapidly evolving AI techniques, such as machine learning (ML) and the accessibility of high-performance computation systems, offer a powerful approach to dealing with the complexity and difficulties of side-channel analysis [PPM+22]. The idea is to develop machine learning algorithms and let the machine automatically detect the pattern within the side-channel information collected. This can bypass the limitations of human experts and dig out the hidden pattern with high efficiency. Also, the capacity of the machine learning model to learn new patterns makes it flexible for performing analysis in various hardware platforms. The capability of fast-evolving is also an advantage of such an approach, as the model can quickly evolve and gain a performance boost when fed with additional data.

Applications of deep learning (DL) to SCA, first proposed in [MPP16], is a particularly interesting methodology utilized in performing SCA with AI. Deep learning is a subcategory of machine learning, referring to utilizing deep neural networks (DNN) to learn a task. Deep neural networks are a kind of Artificial Neural Networks (ANN) that mimic how the human brain works. Each ANN consists of multiple layers, with multiple nodes. The nodes mimic actual human neurons and act as the processing and storing units, and nodes in different layers are connected with each other in some defined manner. The connections between nodes contain weights that get updated constantly across the training phase, which is regarded as the learning process. When the weight updates converge at a certain point, or the training budget is depleted, the model is considered trained and can be used to perform tasks like classification and regression, etc. DNN is a special case of ANN that contains multiple hidden layers, literally meant by the name 'Deep'. DNN is a power tool that performs various ML tasks and is considered a potential SCA approach.

However, the deep learning method demands high volume computational resources,

and it requires much time and devotion to find a model that can predict the key correctly, which is a normal obstacle for machine learning tasks. The field of deep-learning-based SCA has been suffering from these problems for quite a while, as researchers do not have a complete understanding of the various deep-learning models that are on the market, and people usually spend tremendous time searching for promising models with naive search methods such as grid search or random search [WPP20, PWP21]. Although the application of random search to find efficient deep neural network hyperparameters tends to result in efficient models for public datasets, advanced search strategies would be necessary against real-world and noisy scenarios. The main limitation is that more advanced search methods also demand higher computational complexity (e.g., genetic algorithms or reinforcement learning). To speed up the search process while maintaining the performance of the searched model, this work finds the correlation between different hyperparameters inside a DNN model. We also present a Hyperband and its variant to increase model search efficiency.

This paper is organized into the following sections. Section 1 is the general introduction of the work; section 2 demonstrates the background knowledge of deep-learning-based SCA; section 3 discusses some of the related work of this study; section 4 identifies the drawbacks and undiscovered territories of the current relevant researches, and proposes the objectives of the work; section 5 focuses on the methodologies involved in presenting this paper; section 6 comprehensively explains the experiment setup and corresponding test results in four sets of experiments; section 7 discusses the findings obtained and highlights recommendations for the domain based on the results, and also mentions the limitation of the work critically; and eventually section 8 concludes this work and motivates outlooks for future work.

2 Background

The basis of side-channel analysis is the side-channel information, which requires adequate extraction and careful analysis. This process relies on hardware and software, combined with prerequisite knowledge of the encryption algorithm. When trying to fulfill the task with the help of machine learning techniques, the background and mechanism of relevant approaches are also extremely informative and helpful. This section introduces and discusses detailed methodologies of the full pipeline of side-channel analysis.

2.1 Encryption algorithms

Encryption is a process of transforming the unprotected original information (referred to as the *plain text*) to protected information (referred to as the *encrypted text* or *ciphered text*) that people except the intended recipient can not understand, even if they manage to acquire access of such information. Figure 1 depicts a simple logic flow chart of a symmetric encryption algorithm. As a defense of secure information exchange, encryption technologies have been rapidly developing and widely applied to almost every aspect of the cyber world. From secure payment systems that verify each transaction made to end-to-end encryption systems in numerous online chatting software, encryption algorithms have been deployed in almost every online system to protect people from being attacked by unwanted intruders.

Many encryption algorithms have been developed over the long history of cyber attacks and defenses. Some of the most famous ones are the Data Encryption Standard (DES), Rivest–Shamir—Adleman(RSA), and Advanced Encryption Standard (AES). Many variants of each algorithm were later proposed to enhance security.

DES was originally developed by IBM in the early 1970s to protect sensitive government data. Officially adopted in 1977, DES became widely used in various commercial and

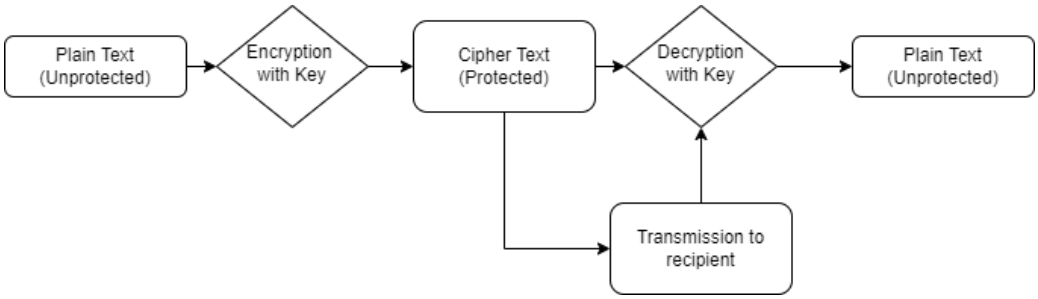

Figure 1: Logic and operation flow of a simple encryption algorithm (symmetric).

financial systems due to its efficiency in hardware implementations [NAHAB23]. DES is a symmetric encryption algorithm, meaning the same key is used in both encryption and decryption processes. The algorithm operates on 64-bit blocks using a 56-bit key, which, despite initial adequacy, was later critiqued for its vulnerability to brute-force attacks as computational power grew. By the late 1990s, with the advent of distributed computing, DES was proven to be insecure, which led to the development of its successors.

The RSA algorithm, named after its developers Ronald Rivest, Adi Shamir, and Leonard Adleman, and published in 1977, marked a significant advancement in cryptographic technology [RSA78][Bon99]. Unlike symmetric key algorithms like the DES, RSA is an asymmetric system that utilizes a pair of keys—a public key for encryption and a private key for decryption. This separation addresses the challenge of secure key distribution, one of the primary limitations of symmetric systems. The security of RSA is rooted in the computational difficulty associated with the factorization of large integers, an area in which no efficient solving technique has yet been discovered. As such, RSA remains a cornerstone in digital security, particularly in applications requiring secure key exchanges.

The AES algorithm, established by the National Institute of Standards and Technology (NIST) in 2001, is a substitution-permutation network that significantly enhances the security previously offered by DES. AES was selected through an open competition involving numerous international candidates and is based on the Rijndael cipher by Vincent Rijmen and Joan Daemen [DDR99][Her09]. The standard specifies three key sizes—128, 192, and 256 bits—and operates on blocks of 128 bits, irrespective of key size. Unlike its predecessor, AES is designed to withstand all known forms of cryptanalytic attack, making it robust enough for both government and private sector applications worldwide. Its adoption reflects a shift towards securing systems against increasingly sophisticated attacks, affirming its role in modern cryptographic practices. Figure 2 shows a visualization of the encryption and decryption phase of the AES algorithm.

In this paper, the AES algorithm is chosen as the attack subject to evaluate the performance and effectiveness of SCA analysis and attacks, as it is by far one of the most advanced, secure, and widely used encryption algorithms.

2.2 Side-channel information

Side-channel attacks exploit information gained from the physical implementation of a cryptographic system rather than weaknesses in the algorithms themselves. Regardless of what encryption model, it is always executed with a code implementation on a certain hardware platform. Like every other program that is executed, the algorithm's running will force the processing unit to fetch and store data and perform certain computations. This is a phase where the law of physics dominates, and the computation performed on the processing unit will consume energy and emit heat and electromagnetic waves. Power

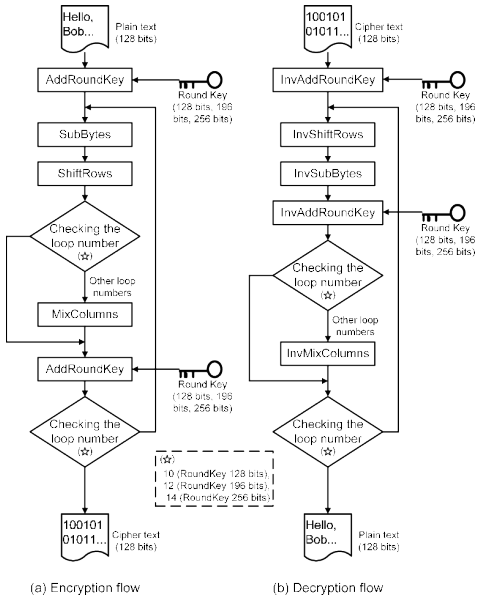

Figure 2: Illustration of the encryption and decryption processes of the AES algorithm. AES-128,192 and 256 share mostly the same logic. [KFNO13]

consumption patterns, heat emissions, and electromagnetic emissions are all considered to be side information, as they are the byproduct of the execution of the encryption algorithm and are not intended to be produced.

Power consumption has a high correlation with the task as well as the code that is executed. For example, when a line of code produces a result of 1 and stores it, it will result in a slightly higher power consumption than when a 0 is produced. This is because all modern hardware consists of transistors that are stacked together. Different operations create different voltage levels and current values for the integrated circuit that deals with the computation. When a probe is attached to a hardware platform where encryption is executed, the power consumption signatures can be acquired and displayed as waveforms of Voltage-Time, Current-Time, or Power-Time, via equipment like an oscilloscope. Based on this information, one can guess the secret, like the key inside cryptographic operations, and gain access to the plain text that should not be exposed. Figure 3 illustrates a classical setup for performing power-based SCA.

Heat emission, like power consumption, is also a byproduct of the computation on processing units. According to Joules Law, electric circuits create heat when the current flows through a resistant medium, which is the case for all current processing units on the market, regardless of the processor's instruction set architecture (ISA) or the hardware packages. Heat can be measured and recorded independently by a heat sensor placed near or attached to the processing unit, and heat emission could have a similar trend and a high correlation with power consumption, given the same thermal condition. For example, when the processing unit runs at full power, it will consume more power from the power supply and produce more excessive heat. Therefore, it is sometimes combined with power consumption measurement to better reflect on the nature of the task that is running and infer the secret from it with relevant prerequisite knowledge like the ISA and the encryption algorithm.

Electromagnetic waves are also a form of side-channel information associated with cryptography operations [AARR02]. The oscillator in the processing unit controls the speed of the opening and closing of transistor gates, sometimes referred to as the clock speed or the frequency of the processor. A tougher task will push the processing unit to

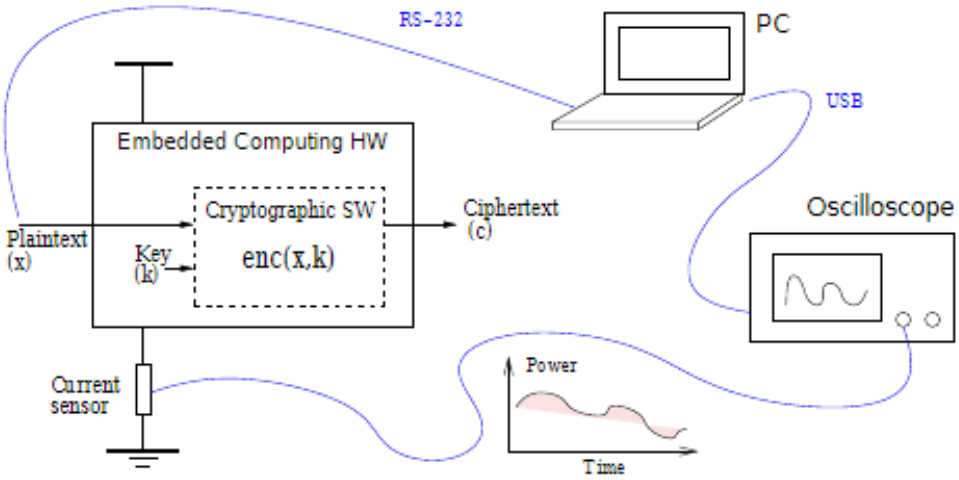

Figure 3: A classical setup for performing power-based SCA. [EWTS14]

run at a higher clock speed, and the oscillator will run at a higher frequency. Running at different clock speeds, the processor emits electromagnetic waves with distinct signatures, which can be probed and analyzed. When performing encryption operations, variations in the electromagnetic field caused by different instructions or data being processed can potentially reveal information about the cryptographic operations being performed and the secret in the process.

In this study, the power consumption is chosen. The reason is that power consumption directly correlates with the computational activities happening within the device, and measuring the power consumption of a device can be relatively straightforward and requires less specialized equipment than some other side-channel information.

2.3 Learning models

The traditional approach to processing the side-channel information acquired and performing side-channel analysis is by manual statistical techniques such as Simple Power Analysis (SPA) [Koc96] and Differential Power Analysis (DPA) [KJJ99a]. Though proven feasible and reliable, such approaches are highly dependent on the skills and expertise of the professional analyzer, as the pattern inside a side-channel trace is submerged beneath heavy noise. It usually takes a long time for the analyzer to carefully look into every detail of the side information at hand to determine the possible key or other secrets of the encryption process, and characteristics and properties of different hardware platforms and encryption algorithms differ significantly, making it hard for a single researcher to carry out the same task in different situations.

To address these limitations, machine learning techniques have been proposed to replace the human expertise in the loop and automatically detect hidden patterns inside side-channel traces. Machine learning is an approach that allows the machine to learn from samples and decide by itself, which greatly enhances efficiency and enables the capacity to transfer the learned knowledge to other platforms. One of the widely used methods employed is deep learning, a popular technique in machine learning. Performing such analysis with deep learning is called deep-learning-based SCA, a promising way to perform such tasks. Deep learning methods, particularly Multi-Layer Perceptrons (MLP) and Convolutional Neural Networks (CNN) [ZBHV19], have been increasingly applied in this

domain due to their powerful feature extraction and pattern recognition capabilities. In this work, the proposed hyperparameter search method is applied to and evaluated on both architectures to understand the impact of different hyperparameter configurations on training speed and testing performance.

2.3.1 MLP

Multi-layer Perceptron is a simple but effective neural network architecture. It has one input layer, one output layer, and several hidden layers. Each layer has some nodes representing the neurons in the human brain. The nodes are interconnected with other nodes in the vicinity layers, and each connection is associated with a weight, which is constantly updated in the training phase. Figure 4 illustrates a schematic of a sample MLP model, with 4 input nodes, 2 output nodes, and 8 nodes in each of the 2 hidden layers. MLPs can learn any function given sufficient neurons and layers, making them broadly applicable to a range of SCA problems with various encryption algorithms. Also, their straightforward architecture makes them easier to implement and scale up or down across different datasets and platforms accordingly. However, MLPs with many parameters are prone to overfitting, especially with limited or noisy side-channel data, requiring careful tuning and regularization. In addition, they generally lack the ability of automatic feature extraction, which means critical preprocessing steps must be accurately designed to achieve optimal performance.

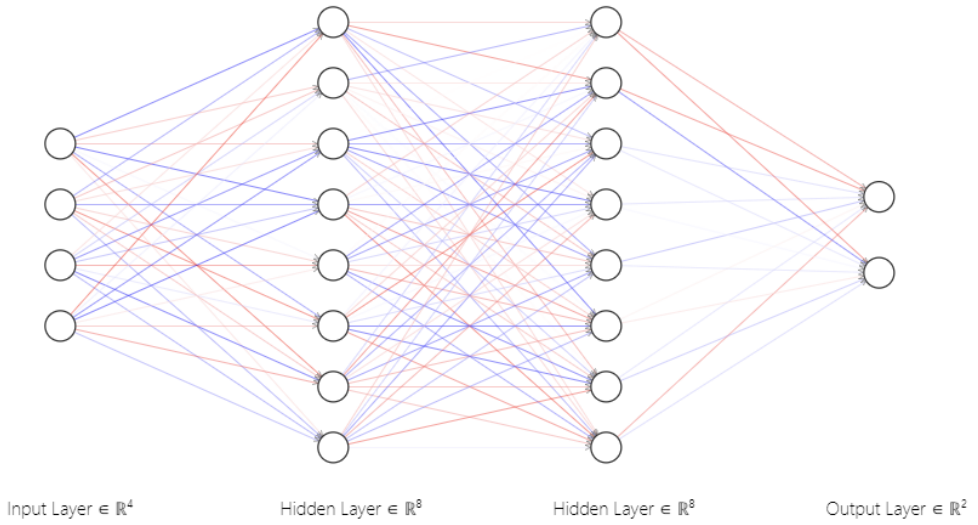

Figure 4: A schematic illustration of the architecture of a simple MLP network.

2.3.2 CNN

Convolution Neural Networks earn their name from the presence of convolution layers and have been a powerful tool in the field of SCA. Convolution layers make CNNs very suitable for performing feature extraction and image processing. They can autonomously identify and exploit localized and significant patterns within complex input data, which is common for side-channel analysis. CNNs can effectively focus on specific segments of a trace that are most indicative of information leakage [ZBHV19, WAGP20]. Unlike traditional machine learning models that typically require manual feature engineering and preprocessing, CNNs reduce the need for such preliminary steps through their inherent

convolution and pooling layers structure. Figure 5 illustrates a simple representation of a CNN network, with the front of the model consisting of a convolution layer and pooling layer, while attached by dense layers that share a similar principle of the MLP model. On the good side, CNNs are particularly well-suited for SCA because they can automatically detect important, localized features in input data, such as specific regions in a power trace where information leakage is high. Meanwhile, CNNs effectively reduce the data's dimensionality through convolution and pooling layers, improving computational efficiency. Yet, it also has some drawbacks. CNNs can be complex to set up and require careful tuning of their architecture (e.g., number of layers, types of layers, kernel sizes, etc) to perform well, which might demand extensive experimentation and expertise. Also, they are sensitive to small changes, which makes them vulnerable to platform or dataset changes. In addition, CNNs generally require more computational resources to train, which can be a limitation in resource-constrained environments.

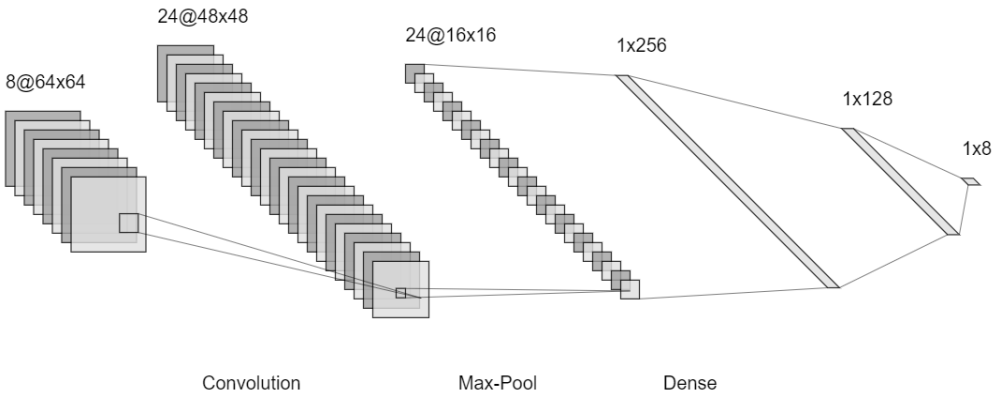

Figure 5: A schematic illustration of the architecture of a CNN network.

3 Related work

Since introducing deep-learning-based approaches to side-channel analysis in 2016 in [MPP16], more than 200 papers have emerged and carefully examined the possibility of performing such complex tasks with powerful deep neural networks. Deep learning models are complex and black-box, and they only work fine when proper training is conducted. Many factors influence model performance, including model architecture, hyperparameter tuning, evaluation metrics chosen, dataset coverage, etc.

For model architecture, in [ZBHV19], Zaid et al. proposed a methodology to design efficient CNNs that improve the robustness of SCA against common countermeasures such as desynchronization. They also introduced innovative visualization techniques such as Weight Visualization, Gradient Visualization, and Heatmaps to interpret the impact of hyperparameters on the network's feature selection capabilities. In [WAGP20], Wouters et al. revisited the previous CNN architecture. They demonstrated the effectiveness of classical preprocessing techniques, which reduce model complexity significantly—by an average of 52%, while maintaining similar performance levels. In [LZC+21], Lu et al. developed a novel neural network architecture that enables end-to-end profiling of cryptographic implementations without the need for manual feature extraction. In [PWP21], Perin et al. revealed that DNNs can achieve successful key recovery with minimal traces under each scenario, demonstrating the versatility and robustness of DNNs in handling different

feature selection and noise levels.

For evaluation metrics, in [MDP19] Loïc Masure et al. established a link between the Negative Log Likelihood (NLL) loss function used in deep neural networks and side-channel information metrics, demonstrating that minimizing NLL is asymptotically equivalent to maximizing Perceived Information (PI). Also, in [BCGR22], Béguinot et al. refined the understanding of how Guessing Entropy (GE) and Success Rate (SR) can predict each other's outcomes across different leakage models. Kerkhof et al. introduced the Focal Loss Ratio (FLR) in [KWPP22], designed to enhance the learning process from noisy or imbalanced datasets typical in SCA. It addressed limitations in existing loss functions by focusing more on hard and less on easy samples, thus optimizing learning effectiveness.

Regarding hyperparameter tuning, Wu et al. introduced a novel framework, AutoSCA, that employs Bayesian optimization to automate the tuning process and demonstrated that Bayesian optimization outperforms traditional random search methods in finding efficient neural network architectures in [WPP20].

Other machine learning methodologies are also applied and evaluated in side-channel analysis. For example, in [RWGP21], Rijsdijk et al. developed a Q-Learning-based reinforcement learning framework that automates the search for optimal CNN configurations, significantly reducing the complexity and size of the models while maintaining or even improving attack performance. In [WDR+23], Wang et al. proposed a novel deep learning-based SCA method called TripletPower, which reduces the number of training traces required for effective attacks. TripletPower utilizes triplet networks to learn robust embeddings for side-channel attacks with minimal data. It demonstrates its ability to effectively recover keys using as few as 250 training traces, compared to thousands typically required by CNNs.

4 Problems and objectives

Though quite thorough work has been carried out in deep learning-based SCA since its introduction in 2016, researchers are still lost in the mechanism of well-performing deep learning models for SCA tasks. It is frequently observed that by just tempering with certain hyperparameter settings in a network, the performance of the model will receive much influence and in some cases, the once well-performed model will cease to function even when the change is relatively trivial. Also, the interconnection between different kinds of hyperparameters is not comprehensively explored. Various research questions regarding the hyperparameters within competent SCA models are not properly answered, for example:

- Are learning hyperparameters more important than architectural hyperparameters?
- For learning hyperparameters, which has more impact on the overall performance, learning rate, regularization or batch size, etc.?

Meanwhile, search and tuning methods are not widely studied. Most work in deep-learning-based SCA still relies on simple and naive grid search and random search, which is easy to implement but computationally expensive and time-consuming. They are also not efficient and rely heavily on the definition of the search space. Work exists exploring more advanced hyperparameter tuning methodologies, such as Bayesian optimization (BO) covered in [WPP20]. However, BO is also computationally expensive, while other promising hyperparameter tuning approaches, such as the multi-armed bandit-based Hyperband, have not been extensively studied in the context of deep-learning-based SCA.

Therefore, this work aims to properly and comprehensively address the research questions, explore the uncharted territories in the domain, and achieve the following objectives based on previous related studies:

- Examine the level of impact of various kinds of hyperparameters, explore the interconnections between them, and identify high-influencing ones. Evaluate the best-working region of each hyperparameter, investigate the cause of such patterns, and formulate proper explanations.
- Propose, develop, and evaluate a new tuning strategy rather than the current domain baseline of random search to achieve the same or even better performance while requiring lower computational and time budgets. Verify the efficacy of the new algorithms proposed in various model and dataset setups.
- Draw suggestions for building and tuning deep neural networks to help reduce the time and computational resources needed to acquire well-performing DNN models for SCA tasks. These should be based on the observations made and results collected in the experiments and understanding of the DNN architectures.

5 Methodologies

This section discusses some of the primary methodologies employed in this work, including the hyperparameter tuning mechanism, the dataset involved in the experimentation, and evaluation metrics.

5.1 Hyperparameter optimization

Hyperparameter optimization (HPO), also known as hyperparameter tuning or searching, is a fundamental task and obstacle for nearly any task involving a neural network, as the hyperparameter configuration highly influences the model's performance. Manual selection and trial of these configurations require numerous efforts and time from the engineer and the outcome depends on the expertise and experience of the human expert significantly. Automated machine learning (AutoML) has been developed to speed up the process as an efficient alternative to replace human intervention. This bypasses the restrictions of the human body and enables automated and 24/7 searching for possible models.

Grid search and random search are among the most popular HPO approaches. Grid search works with a predefined grid of combinations of possible values for each hyperparameter, and, each time, the search engine samples a value in the pool to create a unique configuration to evaluate. For n hyperparameters and m values for each, a total of m^n configurations will be evaluated. However, in most hyperparameter spaces, the effective dimensionality is low: only a few hyperparameters influence the performance of a model with a real difference, and grid search is, hence, pretty inefficient. Therefore, random search has been proposed in [BB12], as it is not only computationally efficient but also very practical and flexible in design. This makes random search one of the preferable choices for hyperparameter optimization, especially when dealing with large hyperparameter spaces, for which grid search becomes computationally impractical. However, random search still requires much processing time and computational resources. Figure 6 illustrates the grid and random search coverage spaces.

5.2 Hyperband

An approach named Hyperband was developed to tackle the issues regarding the speed and resource demand of random search. Hyperband is a multi-armed bandit-based hyperparameter tuning strategy. Initially proposed in [LJD+18] by Li et al., it is a hyperparameter optimization algorithm that works more efficiently and is based on bandit-based strategies that use adaptive resource allocation and early stopping rules. The design of the Hyperband algorithm allows efficient exploration of hyperparameter space

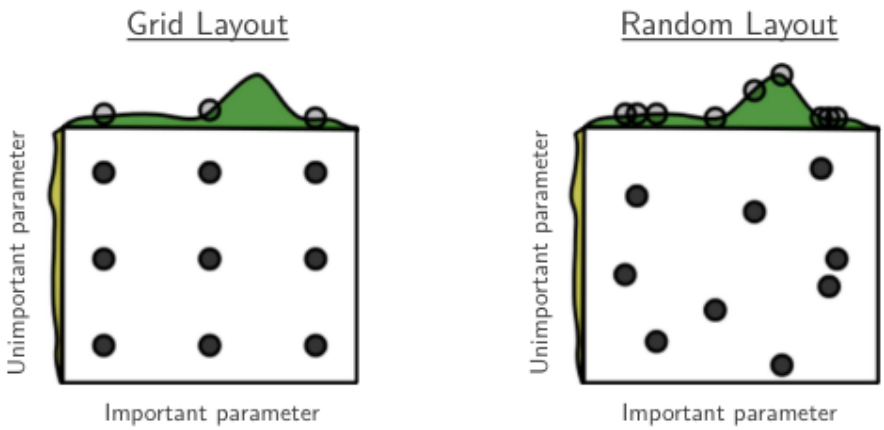

Figure 6: An illustration of the performance comparison of a nine-config evaluation with random search and grid search, where random search produces a better outcome. [BB12]

through dynamical resource allocations to more promising configurations using bandit-based approaches toward the trade-off between exploration and exploitation. This is done with no prior assumptions of the performance of the hyperparameters; hence, it is robust across the application space. It is shown that Hyperband can lead to significant computational savings in many different problems, from deep-learning to kernel-based learning, making it a powerful tool for hyperparameter optimization in machine-learning workflows [LJD+18] [AMH21].

At the core of Hyperband is Successive halving [JT15], an algorithm designed to dynamically allocate more resources to promising model configurations by killing bad-performing models at an earlier stage. It is based on the idea that if the model performs badly initially, it will most likely achieve bad results, even if trained with more epochs. Therefore, bad configurations should be identified at an early stage and ruled out to save the training resources and time for other promising ones. Figure 7 illustrates a scenario where eight configurations are evaluated with successive halving. It can be clearly identified that all 8 configurations are evaluated at the start of the process. When the budget consumption reaches 12.5% of the designated total budget, half of the configurations, the worst 4, are eliminated. The remaining budget is then equally distributed to the remaining half of the candidates until the next threshold is met and another half gets ruled out. This process continues until only one candidate is left, which is supposed to be the most promising one, and the remaining training budget is diverted to it fully. This approach tries to put more resources into more promising ones, thus saving the total training time as not-so-good ones are stopped very early and won't waste the training budget. To decide what candidate model deserves to continue in the search process, an appropriate metric should be computed during the training of each model. Although the metric in Figure 7 is the loss function value, for SCA application, as we will see in the experimental part, other SCA-metrics are more convenient to reach better results.

Successive halving is a powerful algorithm for HPO tasks. However, one question remains in some cases, especially in deep-learning-based SCA. For a fixed overall training budget, it is hard to decide whether to **search for fewer configurations but with a higher budget for each** or **search for more configurations but with fewer epochs for each**. This is also denoted as n **vs.** $\frac{B}{n}$ **issue**, where B is the total budget for the entire search and n is the number of configurations to search for. This is a complex dilemma for the expert without adequate prior knowledge about the essence of the task and search

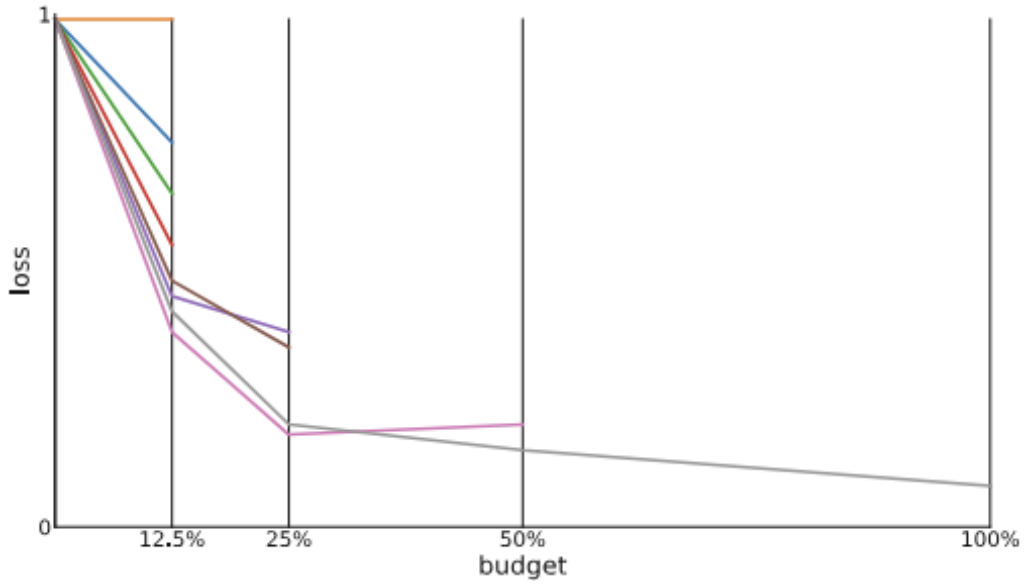

Figure 7: Illustration of the training process of a successive halving algorithm applied on eight configurations. The worst half is continuously dropped as the training budget gets consumed, and the remaining budget is concentrated on promising configurations. [FH19]

space. In the case of SCA, the search space for models intended for different datasets and architectures differentiates a lot. It is not fully known to the researchers yet, which means the answer to the previous selection question remains unknown. Yet Hyperband has the potential to bypass this limitation due to its special design.

Hyperband does so by taking many values of n in a single run, effectively performing a grid search over feasible values of n. This is done by composing SH runs, which are called "brackets" in [LJD+18], through various levels of aggressiveness in resource allocation. Algorithm 5.2 provides a general overview of the logic and structure of the original hyperband algorithm proposed in [LJD+18].

Hyperband requires two main inputs: R — the maximum amount of resources allocated to any single configuration, and η — a parameter controlling the fraction of configurations discarded in each round of Successive Halving (in practice, $\eta = 3$ has been found to work well). The Hyperband algorithm starts by evenly dividing the available budget into the brackets corresponding to a different value of n and r — the minimum resources allocated to each configuration before the first round of halving. The outer loop then iterates over a range of values for s from 0 to $s_{\max} = \lfloor \log_\eta R \rfloor$. For each value of s, it identifies the initial number of configurations $n = \lceil \frac{B}{R\eta^s} \rceil$, and the minimum amount of resource allocated to each configuration $r = R\eta^{-s}$. For each value of s, a Successive Halving algorithm is then executed, with the number of configurations n being trained using resources r. In contrast, SH fixes a given n in advance, and Hyperband dynamically changes n and r over several brackets. This design can let Hyperband explore a lot of different configurations with a small amount of initial resources and exploit promising configurations with more resources. Through the sweep over the values of n and r, Hyperband becomes a method that, in fact, attains a much better trade-off between exploration and exploitation. So, it will spend more resources systematically on promising configurations over different brackets, and it will make sure that there is a good enough search happening in the hyperparameter space without the too-premature dismissal of good configurations. In this way, Hyperband can

Algorithm 1 Default Hyperband for Hyperparameter Optimization

Require: R (maximum resources per configuration), η (default $\eta = 3$)

$s_{max} \leftarrow \lfloor \log_\eta(R) \rfloor$
$B \leftarrow (s_{max} + 1)R$
for $s \in \{s_{max}, s_{max} - 1, \ldots, 0\}$ **do**
  $n \leftarrow \lceil \frac{B}{R} \frac{\eta^s}{s+1} \rceil$
  $r \leftarrow R\eta^{-s}$
  $T \leftarrow \text{Sample}(n)$
  for $i \in \{0, 1, \ldots, s\}$ **do**
    $n_i \leftarrow \lfloor n\eta^{-i} \rfloor$
    $r_i \leftarrow r\eta^i$
    $L \leftarrow \{\text{Evaluate}(t, r_i) : t \in T\}$
    $T \leftarrow \text{TopK}(T, L, \lfloor \frac{n_i}{\eta} \rfloor)$
  end for
end for
return Best configuration observed

speed up hyperparameter optimization considerably, as it can be seen as an efficient way to run multiple SH instances in parallel with different resource allocation configurations. Such parallelism enables it to evaluate more configurations within the same budget than traditional SH or Bayesian optimization methods. Hypothetically, Hyperband can reach near-optimal performance because its total budget is guaranteed to be only logarithmically higher than the best SH strategy, and, on the other hand, the nature of the problem characteristics is unknown a priori. These strengths make Hyperband suitable and effective for many hyperparameter optimization problems, including finding a suitable configuration for SCA tasks.

5.3 HyperDrive: Hyperband with Gaussian Process

Though effective in searching hyperparameters in large search spaces with reduced computation costs, the original hyperband, by default, still relies on random search. This inherent characteristic makes hyperband an enhanced version of random search, but it cannot record trail results and intelligently search in more promising regions. Meanwhile, theories such as Gaussian process (GP) [WR96] and HPO methods like Bayesian Optimization (BO) [SLA12] had been proposed and intensively studied as an approach to perform hyperparameter tuning with an adaptive manner and they had been proven to be effective in finding competent configurations. In the context of deep-learning-based SCA, Bayesian Optimization has also been implemented and evaluated. [WPP20]. BO is a powerful tool in *exploitation*, but it is computationally expensive and may stuck in local optimal. On the contrary, Hyperband is good at *exploration* but cannot use newly obtained observations to refine the search process adaptively.

Therefore, this work adapts an integrated approach of combining Hyperband and a Gaussian process in determining proper training hyperparameters. Algorithm 5.3 defines the general logic flow of the HyperDrive algorithm (HB-GP) developed in this work. At the early stage of the search, the GP kernel can not properly guide the search process as it does not collect adequate observations to form a proper prior distribution. Therefore, in this phase, the GP kernel is taken offline, and the HB-GP algorithm works exactly as a default Hyperband. After enough observations are collected, the kernel is activated, and each time before a hyperband is run, the kernel will first randomly sample N candidates from the defined search space and conduct a screening to pick the most promising candidate based on its current distribution function. Then, the selected candidate is passed to the

Algorithm 2 HyperDrive: Hyperband with Gaussian Process for HPO (HB-GP)

Require: R (maximum resources per configuration), η (default $\eta = 3$), α (threshold controller), B (total budget), N (sample amount)

```
Initialize trial count and start the search process
while used budget not exceeded B do
    if trial count < threshold α then
        Perform default Hyperband to collect observations
    else
        Randomly sample N candidates from the defined search space
        Use GP kernel to recommend the most promising candidate from N samples
        Use Hyperband to train and evaluate recommended candidate
        Update the Gaussian Process kernel with new observations
        Increment trial count
    end if
end while
Return the best configuration found
```

Hyperband tuner for evaluation. The GP kernel in the process aims to better use the obtained observations by recommending more promising hyperparameter configurations instead of random sampling, which emphasizes exploitation in the HPO process. On the other hand, the Hyperband algorithm in HB-GP dynamically allocates more resources to promising candidates and terminates poor-performing configurations at an early stage, reducing unnecessary epoch wastes. By combining the advantage of GP and Hyperband, the HB-GP algorithm developed in this work aims to balance exploration and exploitation properly, thus improving the training speed while saving costs, as well as obtaining good hyperparameter configurations.

6 Experiments

To fully address the research questions and evaluate the advantages of the algorithm proposed in this work, multiple sets of experiments are designed and executed, including the following:

- **Single-hyperparameter characteristic scan.** This involves designating one hyperparameter and scanning the model's performance with different values of the designated hyperparameter within a search space while fixing other considered hyperparameters. This aims to reveal a general working region to keep the search spaces not too broad in future experiments, and it briefly shows the impact of each hyperparameter. Tuner is not activated in this phase as there is no tuning in this setup. A simple grid search is applied to the investigated hyperparameter for scanning.

- **Inter-hyperparameter relation screening.** This set of experiments utilizes the search tuners to search for the best model configuration candidate. For each hyperparameter, a selected search space is created based on the insights from the above experiment and includes both working and non-working regions. This is designed to reveal the correlation between different hyperparameters and distinguish the one with the most influence. This insight can be used to reduce redundant searches and instruct people to pay more attention to highly influential hyperparameters when performing deep-learning-based SCA, which should benefit other researchers in the field.

- **Tuner efficacy evaluation and comparison.** This aims to evaluate the performance and efficiency of the three tuner methods, namely the currently popular random search (baseline), Hyperband (default) that is being paid more attention to recently, and the HyperDrive (HB-GP) proposed in this work. All methods are tested with the ASCADf and ESHARD datasets in a wide and narrow search space. This attempts to reveal the performance difference between different tuning methods and prove the superiority of the proposed HyperDrive tuner in this paper.

- **A revisit to the configurations obtained.** This acts as a re-evaluation of the efficacy of the tuners. For each continuous hyperparameter, a single-variable re-scan in the search space is executed with settings similar to the first experiment set but with other hyperparameters fixed at the values in the best-performing model returned by the tuner. By comparing the positional relationship between the found hyperparameter value and the performance curve, we can further validate whether the tuner returns a configuration in proximity to the optimal choice.

The exact setup for each set of experiments is further discussed in section 6.1.

6.1 Experiments setup

6.1.1 Model architectures

For a more comprehensive study on the hyperparameters' impact and the effectiveness of the proposed HB-GP tuning strategy, MLP and CNN architectures are put to tasks of the ASCADf and the ESHARD datasets. To better evaluate the performance of distinctive architectures in various cases, three variants for each architecture were evaluated, noted as **Small**, **Medium**, and **Large** MLP/CNN, respectively. The specific configurations of the model architecture are detailedly described in Table 1. For MLP models, apart from a stationary output layer, a small variant consists of 3 fully connected layers, with 128 nodes each. A medium variant consists of 4 FC layers with 256 nodes each, and a large variant consists of 6 FC layers with 512 nodes. In the case of CNN models, a small CNN is equipped with one convolution layer with 8 filters. The medium and large CNN variant has 2 convolution layers with 12 and 16 filters, respectively. All variants have the same average pooling layer with a kernel size of 2×2. Small CNN is accompanied by 3 FC layers with 128 nodes each, medium CNN has 4 FC layers with 256 nodes each, and large CNN boasts 5 FC layers with 512 nodes each.

In the following sections, this paper uses small, medium, large models to address the model with the specification as illustrated in Table 1.

Table 1: Model variants architecture used in the experiments. Conv. refers to Convolution; FCL refers to the fully connected layer, which excludes the output layer and includes the input layer; Node count refers to the node count in the FC layer.

Arch.	Size	Conv. layer	Conv. filter	Pool. layer	FCL. count	Node count
MLP	Small	N/A			3	128
	Medium				4	256
	Large				6	512
CNN	Small	1	8; size = 1	1; size = 2	3	128
	Medium	2	12; size = 1	2; size = 2	4	256
	Large	2	16; size = 1	2; size = 2	5	512

6.1.2 Evaluation metrics

The primary metrics used to evaluate the deep learning model's performance in this work include Guessing Entropy (GE), the Number of Traces to Reach GE = 1 (NT), and Perceived Information (PI). Guessing Entropy measures the effort needed to find the correct key. It determines how many keys, on average, an attacker would have to test before finding the right one. The function calculates the GE for each key candidate across multiple traces. It sums probabilities for each key and sorts them to determine the rank of the correct key. The guessing entropy is then the position of the correct key in this sorted list, normalized across multiple executions. The formula for calculating GE is defined as the following:

$$\mathrm{GE} = \frac{1}{N_{exec}} \sum_{i=1}^{N_{exec}} \mathrm{Rank}_{k_{correct}}^{(i)}$$

Where: N_{exec} is the number of executions, $\mathrm{Rank}_{k_{correct}}^{(i)}$ is the rank of the correct key in the i-th execution, S_i is the set of traces selected in the i-th execution. The best possible GE is 1, indicating the model can make a correct guess at the first attempt. Since a 16×16 Sbox is used in the experiments, the worst case of GE will be 256. It should be noted that since an average is taken for GE, it may be a decimal value.

NT quantifies the minimum number of traces required for the Guessing Entropy of the correct key to be the highest, indicating the most probable key is the correct one. This is determined by periodically calculating the GE and identifying the first occurrence where the GE equals 1. It is calculated according to the following:

$$\mathrm{NT} = \min \{t : \mathrm{GE}_t = 1,\ t = n \times \mathrm{interval}\}$$

Where t represents the trace count where GE is calculated, and interval is the fixed number of traces, after which GE is reassessed to determine consistency and convergence. NT falls in the range between 1 and 2000, being a positive integer. For all working models (defined as GE = 1), NT will be in the range, and the lower, the better. If the model does not work (GE > 1), the NT will be capped at 2000.

Perceived Information measures the information content about the correct key, derived from the entropy of the key space adjusted by the predictive probabilities. This calculation takes the initial entropy of the key distribution. Then it adds the entropy reductions based on the log probabilities of the model's predictions for each key, adjusted by the frequency of each key. It is calculated according to the following formula:

$$\mathrm{PI} = H(K) + \sum_{k=1}^{K} p_k \left(\sum_{j \in J_k} \frac{\log_2(p_{k,j} + \epsilon)}{|J_k|} \right)$$

Where $H(K)$ represents the initial entropy of the key space; p_k is the probability of the k-th key being correct; N_k is the number of traces associated with the k-th key; $p_{k,j}$ is the predicted probability for the k-th key at the j-th trace, and ϵ is a small constant added to ensure numerical stability when calculating logarithms (e.g., $1e-36$). PI will be a negative value when a model performs poorly but not lower than -8 bits. On the contrary, the best possible model leads to the maximum possible value of a PI of +8 bits.

In this work, GE and NT were mainly used to evaluate the model's performance. Models with a GE = 1 are regarded as working, and their performance is determined by NT; the lower the NT, the better.

6.1.3 Datasets

This work primarily utilizes two datasets to evaluate the model's performance and fulfill the research questions described in Section 4. This section introduces these datasets and relevant preliminary knowledge.

Table 2: Properties of the datasets selected. P-set, V-set, and A-set correspond to the number of traces in the profiling (training), validation, and attack (test) set.

Dataset	P-set	V-set	A-set	Feature Count	Implementation
ASCADf	50000	5000	5000	700	ATMEGA boolean masked AES
ASCADr	200000	5000	5000	1400	ATMEGA boolean masked AES
ESHARD	90000	5000	5000	1400	STM32F4 SW masked AES-128

The **ASCAD** (ANSSI SCA Database) dataset [BPS+20] can be considered one of the most important open datasets in the domain of cryptographic security research. Specifically geared towards side-channel attacks through deep-learning methodologies, this dataset was made by the French National Cybersecurity Agency, ANSSI, and offers many power consumption traces collected from a microcontroller implementing AES. The dataset is engineered to simulate real-life scenarios of a side-channel attack that would create problems for current cryptographic security measures. All the traces in the ASCAD dataset are thoroughly annotated with the exact secret key that was used in an AES operation, which serves to provide a clear path for researchers to gauge the effectiveness of their analysis methods under fixed (ASCAD fixed key, ASCADf) or random key conditions (ASCAD random key, ASCADr). The ASCADf dataset comprises 60,000 traces, 50,000 of which belong to the profiling set, 5,000 for the validation set, and 5,000 for the attack set. Our experiments consider a trimmed version of ASCADf with 700 sampling points or features. Each acquired trace has been labeled with a fixed key and associated plain text and mask values. The ASCADr dataset contains 300,000 traces: 200,000 belong to the profiling set with variable keys, while the remaining 100,000 traces are set with a fixed key. From the fixed-key set, we select 5,000 for validation and 5,000 for attack. Traces in the ASCADr, compared to ASCADf, have an enlarged trace size. In our experiments, we consider a trimmed version of ASCADr with 1400 sampling points. ASCADr allows for a variable-key setting in which every trace is labeled with different keys in the profiling set and associated plaintext.

The **NUCLEO_SW_AES_MASKED_SHUFFLED** dataset is one of the datasets developed by the company ESHARD and is therefore sometimes referred to as the ESHARD dataset for simplicity [VTM23]. The ESHARD dataset is another popular SCA dataset widely used to evaluate attack performance. Compared with ASCAD, it contains much noise in the traces and incorporates countermeasures such as masking and shuffling, making it noticeably more difficult to tackle. In these ways, the obvious links among the key and observable side-channel emissions are blurred, and this complex dataset is provided to push the limits of testing the approaches of side-channel attacks. It contains electromagnetic side-channel traces of an SW AES implementation that runs on an STM32F4 microcontroller (Cortex-M4), a fairly representative of many secure embedded systems. The implementation uses Boolean masking and a shuffling of the SubBytes operation order to protect against side-channel attacks to a certain extent. It comprises 100,000 traces: 90,000 traces used in profiling, 5,000 in validation, and 5000 in attack set. Each trace has 1400 sampling points. For the experiments conducted in this work, we considered a dataset version with disabled shuffling operations.

This work elaborates on every designed experiment with both datasets to better evaluate the proposed algorithm's performance and answer the research questions. For simplicity, ASCAD fixed and random key datasets will be noted as **ASCADf** and **ASCADr**, respectively. The NUCLEO_SW_AES_MASKED_SHUFFLED dataset will be referred to as **ESHARD** dataset. Table 2 indicates the properties and implementation of the datasets selected in this work.

6.2 Single-hyperparameter characteristic scan

Regardless of the hyperparameter type, the model works in a finite space, and once it moves out of the working region, the model ceases to work properly. Acquiring a better understanding of such a working region makes it possible to limit the search space in a much narrower way, which helps to reduce the computation load and save time. Therefore, a single hyperparameter characteristic scan was first executed to reveal the feasible search region and define the search space for experiments in section 6.3 and 6.4. This scan was applied to learning rate, batch size, and regularization since they are all continuous variables. All but the one hyperparameter scanned is fixed, and the one is scanned from a starting value to the ending value in each run to reveal the performance curve of each test. Table 3 shows the default values for the hyperparameters involved. Each scan involves one hyperparameter and is evaluated in a predefined search space. All other hyperparameters in the process were fixed to the default value as documented.

Table 3: The default hyperparameter setting for the scan. Each test scans a defined search space for one hyperparameter and the remaining ones are fixed to the default values.

Hyperparameter	Learning Rate	Batch Size	L1	L2	Optimizer	Act. Func.
Default value	1e-4	200	1e-4	1e-4	Adam	elu

6.2.1 Batch size

Batch size (BS) is a vital hyperparameter in most, if not all, deep learning-based tasks, which considerably impacts the dynamics of the learning process. Batch size determines how many training samples are used in the loss calculation and gradient approximation before the model weights update. Notably, the batch size is essential for the convergence of the training algorithm and, at the same time, has a significant impact on the computational efficiency and memory requirements of the training process. A smaller batch allows for a closer approximation to the true gradient, potentially improving future generalization results. In other words, limited by the batch size, the overall loss computation is representative of fewer samples and, hence, more precise for these samples. On the other hand, larger batches benefit from computational efficiency improvements derived from built-in vectorized operations and require fewer weight updates in the long term, meaning faster convergence.

Figure 8 illustrates the performance curve with different batch size values. The left figure was obtained with a small MLP model, and the batch size is scanned from 8 to 1536, with a step of 16. The right one results from a small CNN model, with the batch size scanned through 16 to 1024 at a step of 32. Both models showed a similar trend: The model could not perform correctly at a very small batch size. When batch size is increased to a value roughly smaller than 200, the models achieve maximum performance, with NT being minimized. As batch size continues to increase, the performance of the models deteriorates until they cease to work correctly after a respective threshold. It is noticeable that when the batch size is large, the model's performance is extremely unstable, as it fluctuates significantly. However, an approximate threshold boundary can be drawn to distinguish the best working region for batch size, which, for the MLP, is around 1000, and for CNN, is around 600.

Yet, it should be noted that batch size's influence resonates with the model's size. An additional set of experiments was executed to investigate the performance curve for batch size with a different model size. Figure 9 demonstrates the differences and similarities between batch size scans in small and large MLP models. The small model is scanned through the same search space defined previously, and the large model is scanned through 16 to 10240 with a step of 64. It can be derived that the large MLP shares the same

trend as the small variant; both achieve optimal performance at a relatively small value and worsen as it grows. Meanwhile, a larger model pushes the threshold for the working region way further, from roughly 1000 to around 9000, and the same observation is made with CNN. However, both models archive their optimal performance at a small batch size setting, proving that **the batch size selection can be made at a relatively small search region regardless of model size.**

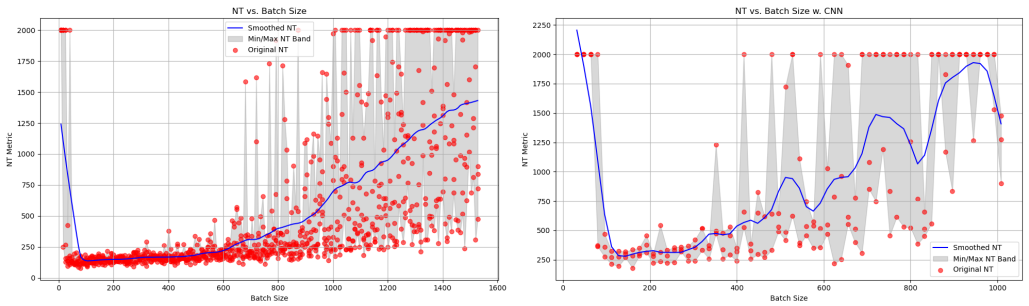

Figure 8: Model performance (NT) over different batch sizes. All other hyperparameters were fixed according to Table 3. A small MLP (left) and small CNN (right) were examined in the test with the ASCADf dataset.

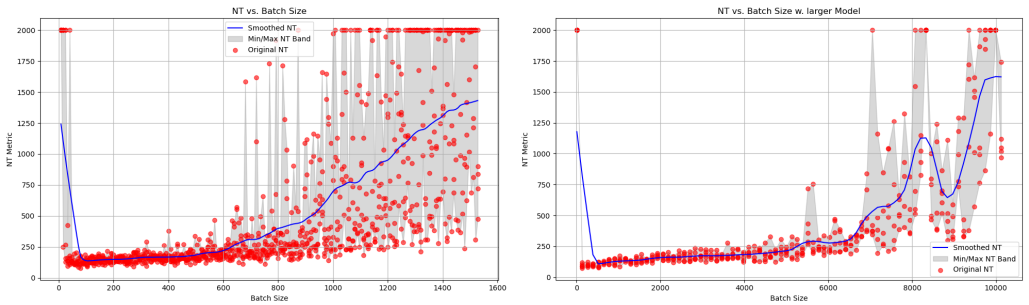

Figure 9: Model performance (NT) over different batch sizes in two sizes of MLPs. All other hyperparameters were fixed according to Table 3. A **small** MLP (left) and a **large** MLP (right) were investigated in the case with the ASCADf dataset.

6.2.2 Learning rate

The learning rate (LR) is one of the fundamental hyperparameters in deep learning model training used to adjust the weights based on the gradient calculated concerning the reduction in the loss function. Adequately setting the learning rate optimizes the convergence of the model to a good minimum in which the balance is between the convergence speed and the risk of overshooting the possible minimal points. If the learning rate is too high, the training process may become unstable, and the model could oscillate around the optimum weights or even diverge. On the other hand, a learning rate that is too small makes the convergence process unbearably slow. In deep-learning-based SCA, where a model has to detect and exploit fragile and often noisy signals to infer cryptographic keys or operations, getting a proper choice of the learning rate is essential. Figure 10 depicts the NT performance over different configurations of the learning rate scanned.

The MLP is tested with a learning rate between $5e-5$ and $5e-3$, and CNN is configured with a learning rate between $1e-4$ and $5e-3$. Both model architectures share a similar trend: the model won't work properly if the learning rate is too low or too high. Once the learning rate falls in the optimal region, model performance is significantly improved until it reaches the upper limit, and the model stops working afterward. For both cases examined, the optimal region lies roughly between $5e-4$ and $1e-3$, in which the model works fine. It could also be seen that both architectures suffer greatly from having a huge learning rate. Before the LR is too large to disable the model from performing correctly, there is a region where the model's performance fluctuates significantly, which should also be considered an infeasible region. Findings from the learning rate scan suggest that **both model architectures have a relatively narrow feasible region of learning rate configurations to work correctly, and therefore, the search space of the learning rate can be scaled down to reduce the time needed to converge to an optimal.**

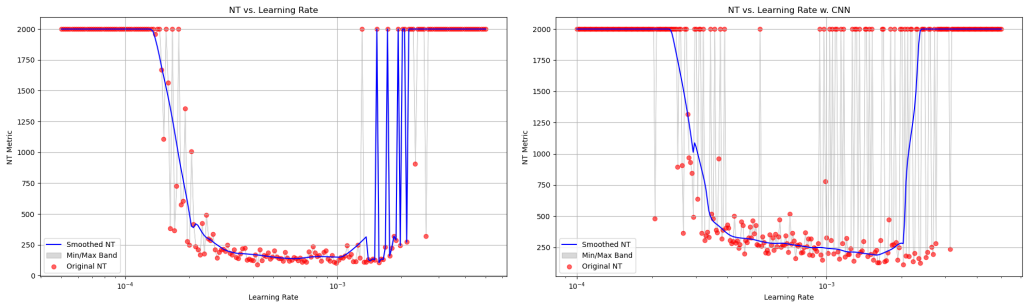

Figure 10: Model performance (NT) over different learning rates in a small MLP (left) and a small CNN (right) on the ASCADf dataset. All other hyperparameters were fixed according to Table 3. The X-axis is drawn in a log scale.

6.2.3 Regularization

Regularization techniques are important ingredients for training deep learning models. They mainly tackle overfitting, which occurs when a model learns the training data too well, including noise and outliers. For deep-learning-based SCA, noise is an influential yet unavoidable ingredient, mostly causing the model to learn poorly. L1 and L2 regularizations are two of the most popular approaches to regularizing the model.

L1 regularization is also known as Lasso regularization [Tib96]. Basically, the regularization penalizes the magnitudes of the coefficients of "weight" in a manner equivalent to the absolute value of the magnitudes. The L1 regularizer is particularly suitable for tasks where most features in a high-dimensional data structure are not useful for prediction, such as the noise in the side channel information. L2 regularization, also known as ridge regularization, applies a penalty equal to the square of the magnitude of the coefficients [Has20]. It keeps the errors distributed among all the terms that do not allow large weights. It further improves the model's robustness by ensuring that one particular weight does not influence the training process.

L1 and L2 regularizers are tested with a small MLP and a small CNN. The scan applies a range of 0 to $2.5e-4$ and 0 to $2e-4$ to MLP and CNN, respectively. Figure 11 and 12 illustrate the performance of both model architectures under different values of L1 and L2 regularizers. It can derived from the results that MLPs are more stable with regularizers compared with CNNs, and L1 regularizers have a much positive and higher boost in performance compared with L2. For L1, as the value increases, the model's performance improves with the process until it reaches a threshold, after which the model can not

function properly. The reason for the models to stop working after regularization exceeds a certain threshold may be that noise is a more intensified problem for the SCA domain, and the abundant noise inside the dataset makes models with a high penalty unable to distinguish between useful information and noise. The non-working region's threshold for MLP and CNN is around $1.2e-4$, before which the model's performance improves as L1 increases. L2, on the other hand, does not show any significant improvement for MLP and CNN models, as the performance fluctuates and models stay mostly in an inoperative state. This observation might be because the built-in feature selection ability gives the L1 regularization a chance to select the most important features that make an effective side-channel attack. L1 makes it possible to nullify less important weights so that the model can disregard non-informative features, stabilizing its predictions against the noises in the datasets.

Observations made in the scan of regularizer values suggest that **L1 generally is a better option for regularizing deep-learning models to perform SCA attacks, and models with an L1 regularizer perform better than the one that is not regularized, yet the L1 value must not be too large; otherwise, the model won't perform properly at all. Therefore, in hyperparameter tuning, L1 should be tuned in a restricted search space to avoid destabilizing the model.**

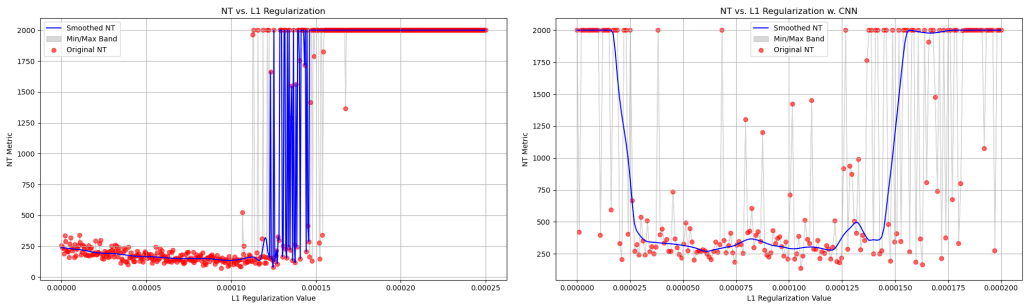

Figure 11: Model performance (NT) over different L1 values in a small MLP (left) and a small CNN (right) on the ASCADf dataset. All other hyperparameters were fixed according to Table 3.

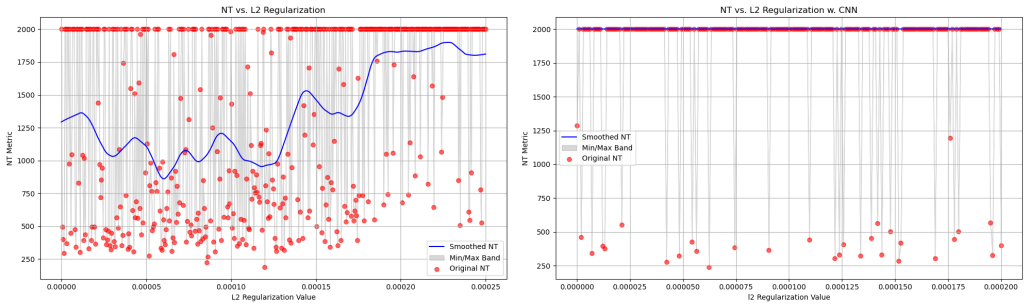

Figure 12: Model performance (NT) over different L2 values in a small MLP (left) and a small CNN (right) on the ASCADf dataset. All other hyperparameters were fixed according to Table 3.

6.2.4 Leakage model

In addition to hyperparameters, the leakage model plays a vital role in performance [DPRS11]. This work involves two leakage models: the Identify model (ID) and the Hamming Weight model (HW). In the ID model, the side-channel leakage is supposed to be directly proportional to the information (i.e., intermediate data processed by the encryption algorithm), with no transformation whatsoever. The assumption behind this model is that the leakage observed and measured (like power consumption or electromagnetic emissions) is expected to correlate directly with the actual binary values of the data being processed by the cryptographic device, as the following formula:

$$L(x) = x$$

The Hamming Weight model postulates that the side channel has an attack leakage function equal to the Hamming weight being operated on, which is the number of bits set to '1' in a binary representation. This model implies that the side channel information from a cryptographic device depends upon the number of active transistors, which is correlated to the number of '1's in the binary data. The computation of HW can be derived as the following:

$$L(x) = HW(x) = \sum_{i=1}^{n} x_i$$

Where x_i represents the i-th bit of x, and n is the total number of bits in x. The summation $\sum_{i=1}^{n} x_i$ calculates the total number of bits set to '1'.

The influence of the leakage model is also evaluated in this set of experiments. Figure 13 documents the comparison of the model's performance with the same hyperparameter configurations but with distinct leakage model selection for a small CNN model. It can be clearly observed that the ID achieves a smoother and higher performance when compared with the HW model in the same search space. Also, the tendency of 'going down first and back high' of the NT vs. L1 does not show with a HW model in the same search space. However, it should be noted that the selection of the leakage model should be based on the requirements and preferences of the dataset. For example, for the ESHARD dataset investigated, the model mostly ceases to operate with an ID model but works quite fine with the HW model. Therefore, it is concluded that **the choice of the leakage model should be tailored to the requirements of the dataset instead of being tuned as a hyperparameter. In this work, unless specified intentionally, models tested on the ASCADf dataset have an ID model, and those on EHARD have an HW model by default.**

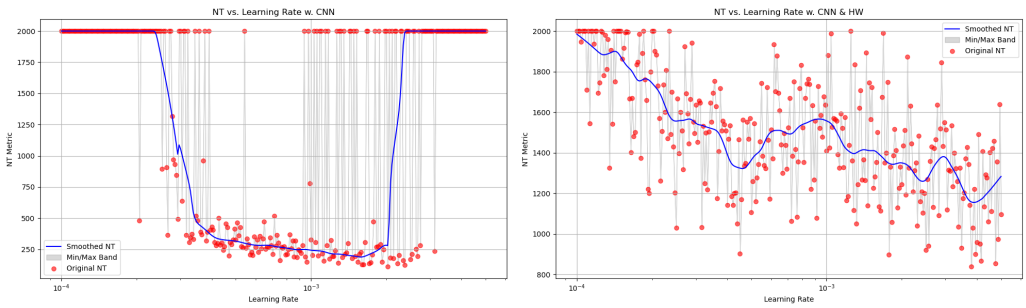

Figure 13: Model performance (NT) over L1 for an ID (left) and HW (right) leakage model with a small CNN on the ASCADf dataset. All other hyperparameters were fixed according to Table 3.

Overall, the single-hyperparameter characteristic scan investigates the influence of individual hyperparameters in a pre-defined search space to reveal their corresponding performance characteristic curves. The insights obtained are valuable for understanding the approximate best-performing region for each hyperparameter and selecting a narrower search space in the hyperparameter tuning phase to reduce the complexity and computation load in the search for a model that performs well. The search spaces defined for section 6.3 and 6.4 are based on the results obtained in this section.

6.3 Inter-hyperparameter relation screening

This section investigates the interconnections between different hyperparameters that lead to a model performing well. The assumption behind the design of this set of experiments is that **hyperparameter are not equal in terms of determining model's performance**. Some might have a larger impact on a good model that needs to be precisely fine-tuned, while others might be more resistant to changes and can be set in a broader range. Insights on this pattern can be used to define a more precise search space, reduce the computational load on fine-tuning deep-learning SCA models, and save time in finding a competent model.

Wide and narrow search spaces are defined to better evaluate the model's performance in various setups and provide more comprehensive coverage. Table 4 illustrates the configurations of the two search spaces defined. In terms of hyperparameter tuning for deep-learning-based SCA, two kinds of hyperparameters have an influence on the model's performance, which can be summarized as `learning hyperparameters` and `architectural hyperparameters`. Learning hyperparameters refer to the ones related to the training process, such as learning rate, optimizer kind, batch size, etc. Architectural hyperparameters are associated with the model's architecture, such as network type, layer count, node count, etc. This work mainly focuses on the **learning hyperparameters** since they are considered to have more impact on whether or not obtaining a good-performing model, while the layer count and node count won't be a big problem, as long as the architecture is defined in an acceptable range. In all four sets of experiments, **five learning hyperparameters** are examined: optimizer, learning rate, activation function, batch size, and regularization, to balance coverage and test feasibility. Small, middle, and large CNNs (Table 1) are constructed and tested in the same search space to reflect the performance more comprehensively. Specifications of the ranges are determined based on the findings in section 6.2.1, 6.2.2 and 6.2.3.

Table 4: Search space properties defined in this work. LR refers to learning rate, BS refers to batch size, and L1 refers to L1 regularization. For LR, BS, and L1, corresponding ranges apply. For the optimizer and activation function, a predefined list of choices applies.

Definition	Property	LR	BS	L1	Optimizer	Activation Function
Narrow	Min	1E-05	64	0	[adam,	[elu, relu,
	Max	1E-03	1024	8E-05	sgd,	selu,
	Steps	log	32	linear	rmsprop]	tanh]
Wide	Min	5E-06	32	0	[adam, sgd,	[relu, selu,
	Max	5E-03	2048	1.5E-04	rmsprop,	elu, tanh,
	Steps	log	64	linear	adadelta]	sigmoid]

Two methodologies are utilized to investigate the interconnection between different hyperparameters: a 3D scatter plot showing the spatial distribution of hyperparameter combination candidates in the search space and a parallel coordinate plot illustrating the interconnecting relationship between different hyperparameters. For the scatter plot, the three dimensions measured are learning rate, batch size, and l1 regularization, as they

are all continuous variables. For the parallel coordinate plot, all five hyperparameters are present.

6.3.1 Boundary and correlation

The 3D scatter plots are primarily used to reveal the possible virtual boundaries between the three involved hyperparameters: learning rate, batch size and L1 regularization value. Figure 14 illustrates two 3D scatter plots of the spatial distribution of NT performance in the search space, one with a small MLP (left) and one with a small CNN (right). The color bars of both plots share the same rule: lighter points indicate poor-performing models, and darker points lead to good-performing models. Yellow points represent models with an NT of 2000, considered not functioning. Two patterns exist for the three hyperparameters examined, which this work addresses as **boundary** and **correlation**.

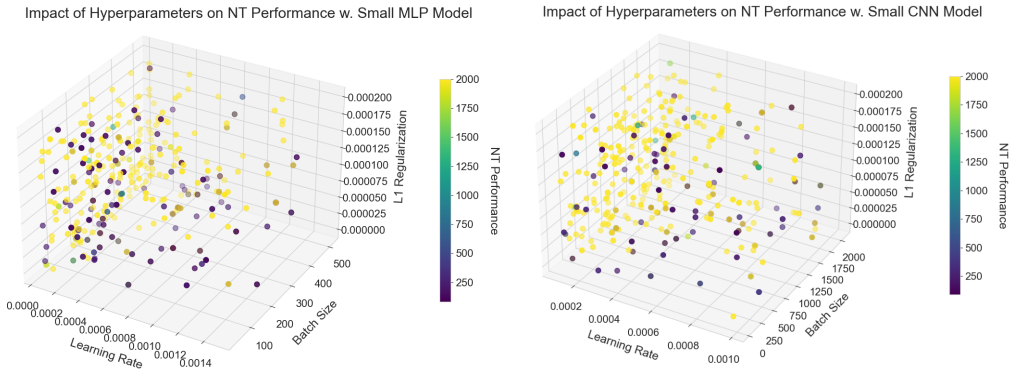

Figure 14: Spatial distribution of model performance (NT) in the search space obtained with a small MLP (left) and CNN (right). Point coordinates lead to the 3D configuration of the learning hyperparameters, and their color indicates corresponding performances. (Darker is better.)

Boundary refers to the virtual threshold determining whether a model functions properly. When the corresponding hyperparameter value exceeds it, the deep learning model will be disabled from performing SCA tasks. This observation is made with L1 in particular, as depicted in Figure 15. The boundary plane (in pink) is drawn to illustrate the barrier of L1, indicating that any L1 values above this threshold result in the failure of the deep learning model, regardless of other hyperparameters' values. The L1 threshold, in this case, is roughly $1.2e-4$. This observation is in accordance with and supplements the findings in section 6.2.3, that models stop working correctly when L1 increases and surpasses a threshold. The same pattern exists regardless of model architecture, leakage model, and dataset and can be seen as a common tendency for deep-learning-based SCA. This indicates that the selection of the L1 value is of great significance and should be fine-tuned with care to find a promising, competent model.

Correlation refers to the interconnections between two or more hyperparameters, from the belief that the hyperparameters are influential to each other. Generally, for competent deep-learning models, when the value of one hyperparameter moves, that for another will move accordingly to maintain the model's performance. Therefore, a correlation plane should be generalized to illustrate the interconnection between hyperparameters, as shown in Figure 16. The result suggests that a correlation exists between learning rate and batch size and that a large batch size combined with a small learning rate can maintain good performance. This is intuitive and, as such, is also true for many deep-learning tasks. Such

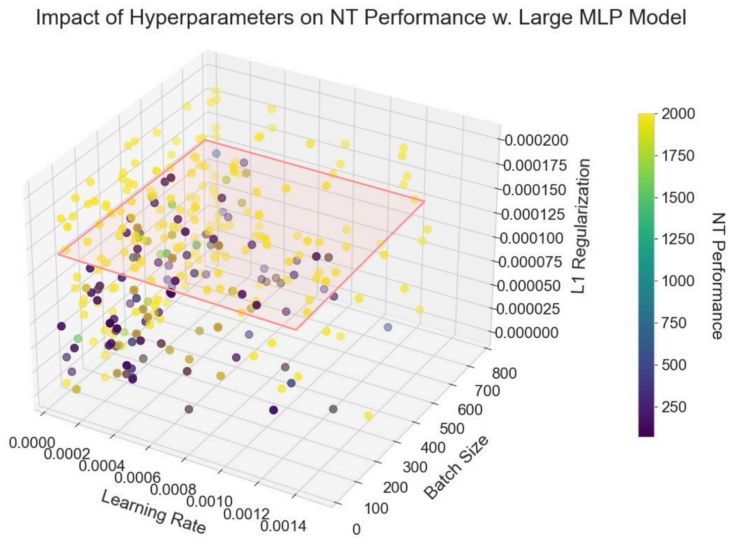

Figure 15: Virtual boundary (pink plane) of the L1 attached to the spatial performance distribution of a large MLP model. It can be clearly seen that all configurations above this hyperplane are invalid (NT = 2000).

a correlation plane covers most of the points in the search space, and it can be noticed that no matter what value is applied to the regularizer, such a correlation exists for BS and LR. This finding reminds us that **LR and BS are correlated to some extent, as most powerful models show an inverse-proportional trend between learning rate and batch size. This is further validated in the interconnection part in section 6.3.2. This correlation can be utilized in search strategies to tune them as a whole instead of individually, thus improving model performance and reducing search space.**

Such findings are observed regardless of the deep-learning model architecture and leakage model. Figure 17 shows the 3D spatial coordinate distribution of a medium CNN model with both ID (left) and HW (right) leakage models on the ASCADf dataset. Comparison between this and Figure 14 illustrates that the same pattern of barrier and correlation is documented, no matter the model's size and the leakage model's choice. It proves that connections exist between various hyperparameters, but only in the three continuous. To further analyze the relationship between all five learning hyperparameters, the parallel coordinate plots are introduced and utilized to analyze hyperparameters with continuous and discrete search space together, as discussed in section 6.3.2.

6.3.2 Interconnections

In the parallel coordinates plot, data points represented in multivariate data from connected line segments are placed in parallel across attributes or axes of variables, with each line segment representing a data point. It is a valuable asset for analyzing the effects of hyperparameters on models of deep-learning-based SCA. They give insight into how one hyperparameter affects the effectiveness of a model and how to address the optimization of these parameters for better model performance. As such, this type of plot finds much utility in model tuning [LDW21]. This work utilizes the Plotly library [Inc15] to visualize the interconnections between the five learning hyperparameters.

However, most parallel plot libraries can not directly deal with string data. The

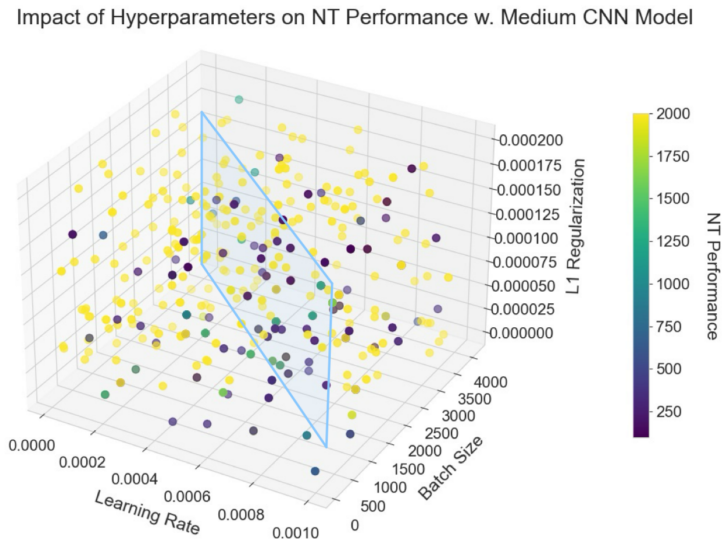

Figure 16: Virtual correlation plane (light blue) of LR and BS attached to the spatial performance distribution of a large MLP model. Hyperparameter configurations of competent models share the tendency that a large batch size with a small learning rate leads to a good model.

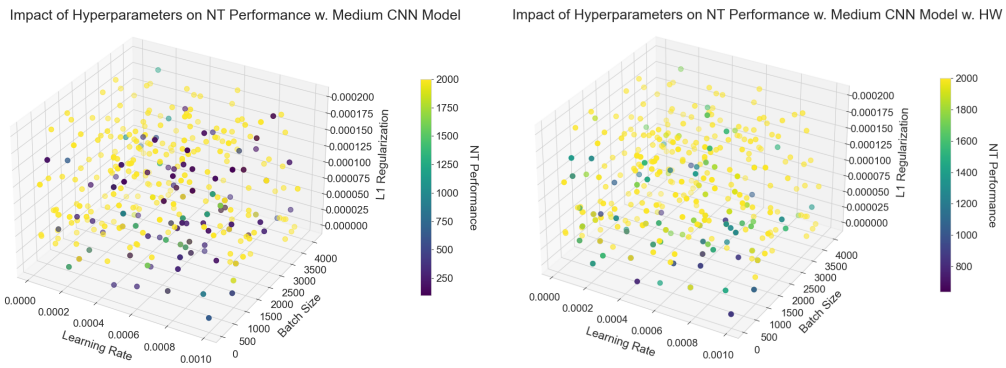

Figure 17: Spatial distribution of model performance (NT) in the search space obtained with a medium CNN with ID (left) and HW (right). Point coordinates lead to the 3D configuration of the learning hyperparameters, and their color indicates corresponding performances. (Darker is better.)

optimizers and activation functions are presented with not values but strings, which can not be directly processed by Plotly. Therefore, mapping is established to convert the strings to one of the categorical numbers, which is attached to the bottom of each figure for reference.

Figure 18 demonstrates the interconnections between the five learning hyperparameters investigated with a large MLP in a narrow search space on the ASCADF dataset. Each line represents a valid model selected by the search, and their intersections with each axis mark the corresponding value of each hyperparameter. The color bar indicates the performance of the model (NT metric); the darker the line, the better performance it is for the model. Only good models with an NT smaller than 1000 are selected to display in the figure to distinguish the pattern better. It can be observed that the selection of L1 value in this case is **highly concentrated** at around $6e-5$ to $8e-5$. This is because the cap of L1 in a narrow search space is set to be $8e-5$, which is lower than the maximum L1 threshold discovered in section 6.2.3, meaning that the larger the L1 in the defined search space, the better regularization performance it is for the model, which leads to a reduced NT metric. Meanwhile, the **inverse proportion pattern** between the batch size and learning rate can be observed: a larger batch size with a small learning rate and vice versa. This finding is per the observation in section 6.3.1. In addition, the effect of the activation function in the narrow search space is **not** significant, as shown in the figure, yet for the optimizer, `adam` and `rmsprop` have a **noticeable advantage** over `sgd`.

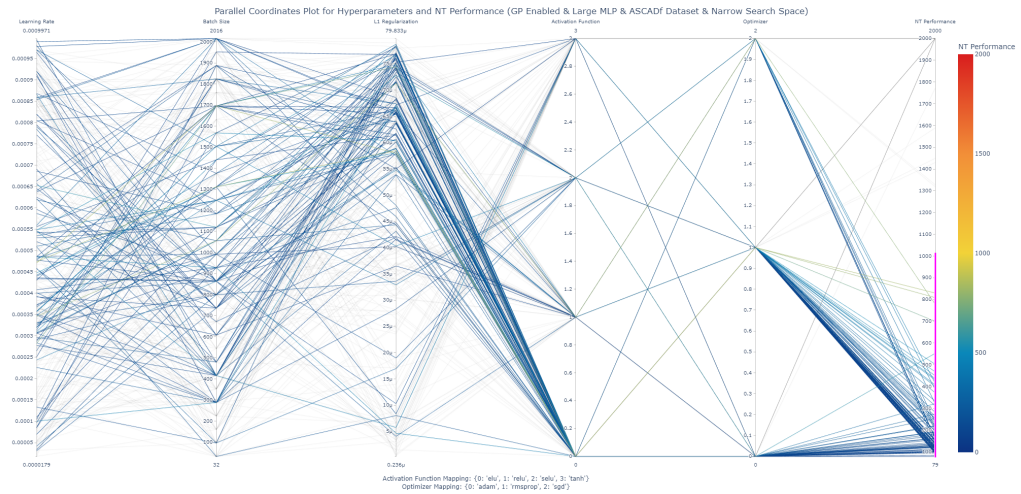

Figure 18: Parallel coordinates plot for a large MLP model searched in a narrow space on the ASCADf dataset. Only good models (NT < 1000) are selected to show.

Similar observations exist for the wide search space, as illustrated in Figure 19, obtained with a medium MLP tested in a wide search space. Compared with the previous outcome, this search clearly results in fewer competent models, as fewer lines exist in the figure. However, the same inverse-proportion pattern between batch size and learning rate, and the concentration of the L1 value are present. The observed tendency also applies to CNNs, illustrated in Figure 21, which is tested under a small CNN on a wide search space. It can be clearly noticed that the working models spread through a larger space and are less condensed in spatial distribution because more invalid options are available to the tuner. It may result in more invalid configurations that are filtered out in the figure. Tests on the ESHARD dataset are also executed and examined to understand the interconnections comprehensively. Figure 20 shows the plot for a medium MLP on the ESHARD dataset in a wide search space. Evidently, the model's performance is reduced compared with that of

the ASCADf dataset, which makes sense as the ESHARD dataset contains more noise and fewer sampling points, making it a harder target to attack. Yet, the similar tendency of the interconnections between the learning hyperparameters is valid in this dataset.

Overall, these experiments using the two plotting methodologies prove that **such interconnections between learning hyperparameters exist regardless of model architectures, model size, dataset, and search space size.** These findings suggest that understanding the relations between the hyperparameters can help reduce the search space and make the tuning more concentrated and targeted, increasing optimization efficiency.

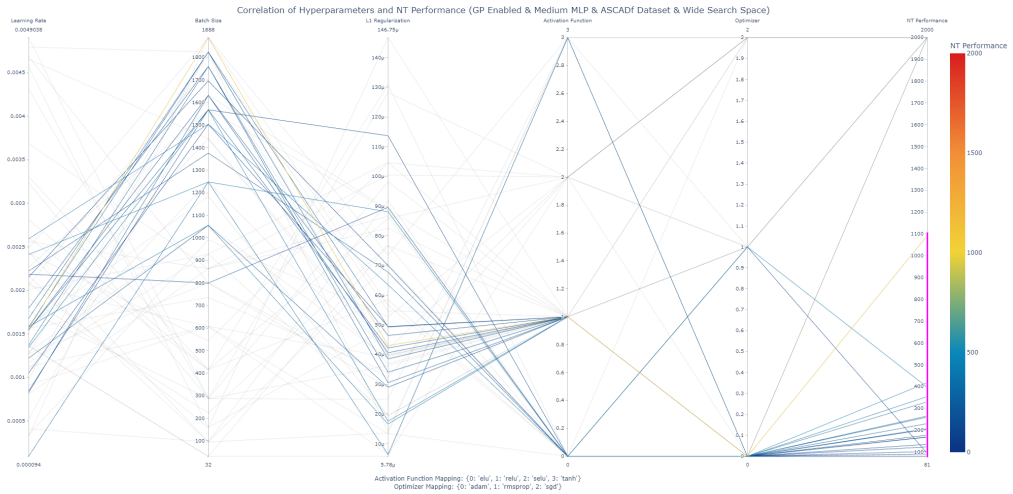

Figure 19: Parallel coordinates plot for a medium MLP model searched in a wide space on the ASCADf dataset. Only good models (NT $<$ 1000) are selected to show.

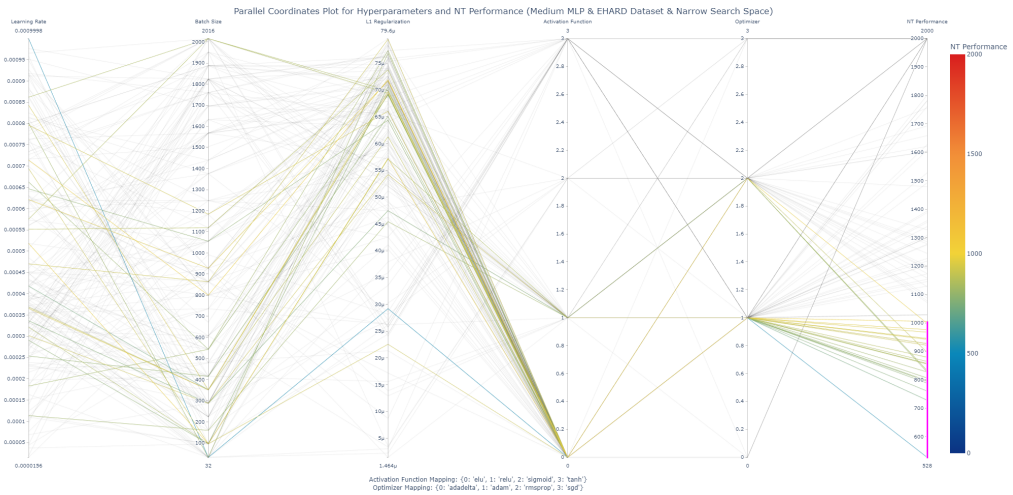

Figure 20: Parallel coordinates plot for a medium MLP model searched in a wide space on the ESHARD dataset. Only good models (NT $<$ 1000) are selected to show.

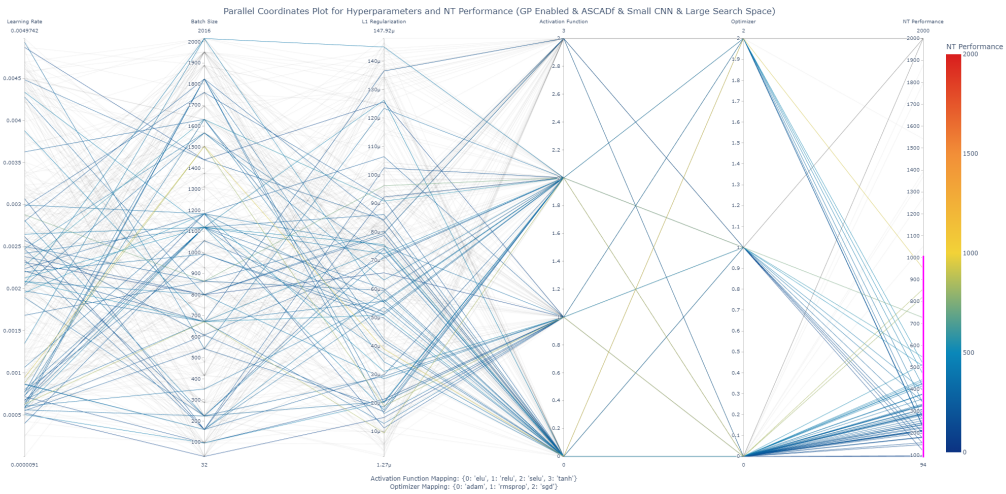

Figure 21: Parallel coordinates plot for a small CNN model searched in a wide space on the ASCADf dataset. Only good models (NT $<$ 1000) are selected to show.

6.4 Tuner efficacy evaluation and comparison

Tuner efficacy evaluation is at the core of this work. The previous analysis serves as the basis for understanding the impact of each hyperparameter, and the insights are utilized to define a proper search space for competent models. This paper proposes two algorithms, Hyperband (HB) and Hyperband with Gaussian Process (HB-GP or HyperDrive), to replace the current popular random search (RS) in the field. This section documents the findings and compares these methodologies.

Hyperparameter searches are executed with one of the three tuning algorithms defined in Table 5. All tuning algorithms are executed with a fixed seed (42 for MLP and 85 for CNN) to purge the impact of randomness in the process and address replicability. Trails performed with all tuners have a maximum budget limit of 40, with an early stopping mechanism kicking in after 8 non-improving epochs. Four full iterations are allowed for HB and HB-GP. At the same time, the iteration of RS is defined as the total budget divided by epochs per trail so that all three tuners have exactly the same total epoch budget to ensure a fair comparison.

Table 5: Settings of tuners used in the experiments. HB is the default Hyperband, HB-GP is the HyperDrive strategy proposed in this work, and RS is the widely used Random Search. E. P. T. stands for Epochs per Trail, and the GP threshold indicates the minimum observations needed to activate the GP process.

Tuner	Objective	E. P. T.	Iterations	Seed	HB Factor	GP Threshold
HB	CMOT	40 (ES=8)	4	42/85	3	N/A
HB-GP	CMOT	40 (ES=8)	4	42/85	3	10 Observations
RS	CMOT	40 (ES=8)	TB/E. P. T.	42/85	N/A	N/A

The CMOT function is defined to replace the default validation loss as the minimization objective. General DL tasks widely use validation loss as an intuitive and simple objective. However, low validation loss does not always lead to high model performance in deep-learning-based SCA tasks. Therefore, a customized simple logic is applied as the minimization objective for all three tuners, as described in Algorithm 3. The idea is based

on the definition of the evaluation metrics, that good models always have a GE = 1 and an NT as low as possible. For models that can not reach GE = 1, their NT value will be permanently capped at 2000. Therefore, the design of the CMOT objective allows the tuner first to try to minimize GE until it reaches 1, then move to reduce NT as much as possible, as there will be no sense in reducing NT if its GE is not 1, according to the definitions in section 6.1.2.

Algorithm 3 CMOT: Customized Minimization Objective for Tuners

```
function CMOT(ge, nt)
    if ge ≠ 1 then
        return 1000 + ge                ▷ High penalty for ge not being 1
    else
        return nt/2000                  ▷ Normalize nt; lower nt yields a drastically reduced score
    end if
end function
```

All tuners are evaluated with the same total budget for each run, with either a **large epoch setting (14400 epochs total)** or a **small epoch setting (4800 epochs total)**. This work aims to replace the time-consuming random search method widely used nowadays with a more efficient searching algorithm to find a more powerful model using the same time or spend less time finding an on-par model. Therefore, this work evaluates the performance of the tuners based on two metrics: the SCA performance metric, which determines the competency of the best-found model, and the searching time that illustrates the duration needed for tuners to find promising models. For performance metrics, NT and PI are used to judge the model's capability, as GE in these cases will always be 1 and can not be used to distinguish differences.

Table 6: Performance and time metric results for six model architectures (small/medium/large & MLP/CNN) under large budget setting on the **ASCADf dataset**. RS refers to baseline random search, HB and HB-GP refer to Hyperband and HyperDrive proposed in this paper. The best results are shown in **bold**. GE is omitted in the table for simplicity as all models listed achieve 1.

	NT (count)			PI (bit)			Search Time (min)		
Architecture	HB	HB-GP	RS	HB	HB-GP	RS	HB	HB-GP	RS
Small MLP	157	**151**	156	**0.243**	0.129	0.116	105	**101**	148
Medium MLP	85	**80**	126	0.265	**0.303**	0.270	**99**	132	157
Large MLP	117	**105**	108	0.196	**0.325**	0.272	**113**	119	208
Small CNN	**140**	148	190	0.164	0.189	**0.202**	**131**	134	256
Medium CNN	125	**107**	111	0.159	0.195	**0.237**	149	**148**	359
Large CNN	**72**	97	89	0.102	0.245	**0.260**	195	**187**	459

Table 6 demonstrates the detailed results of the performance and time metrics of six different model setups, including the small, medium, and large variants of MLP and CNN on the ASCADf dataset. Each configuration is executed three times, and the average is calculated and displayed to reduce the impact of a possible single abnormal run. The best-obtained results for each category are shown in **bold**.

It can be clearly noticed that for the NT and Search Time metrics, the proposed HB and its variant HB-GP in this paper achieve the best outcome compared with the currently widely adopted random search in every tested configuration. For MLPs, the HB-GP achieves the best model performance regarding the lowest NT, which applies to all

Table 7: Performance and time metric results for six model architectures (small/medium/large & MLP/CNN) under small budget setting on the **ESHARD dataset**. RS refers to baseline random search, HB and HB-GP refer to Hyperband and HyperDrive proposed in this paper. The best results are shown in **bold**. GE is omitted in the table for simplicity as all models listed achieve 1.

	NT (count)			PI (bit)			Search Time (min)		
Architecture	HB	HB-GP	RS	HB	HB-GP	RS	HB	HB-GP	RS
Small MLP	683	**470**	566	0.026	**0.129**	0.042	**37**	41	68
Medium MLP	576	**458**	724	0.054	**0.063**	0.057	43	**38**	79
Large MLP	**583**	669	702	0.037	**0.040**	-0.005	**56**	59	111
Small CNN	**391**	490	497	**0.046**	0.023	0.021	65	**63**	119
Medium CNN	741	**626**	642	0.040	0.018	**0.041**	104	**100**	129
Large CNN	733	**690**	775	**0.041**	0.038	0.026	**134**	139	169

three model sizes. For the medium MLP, both HB and HB-GP record much fewer NT needed to reach GE = 1 compared with the RS baseline, that the HB-GP requires an NT of 80 and HB requires 80, while RS requires 126. The small and large variants of MLP also show that HB and HB-GP achieve on-par or fewer NT, indicating a performance boost to the RS. Meanwhile, the two proposed algorithms require less time to find a competent model. HB utilizes significantly less time and finds an even better model, and it can be noticed that as the size of the model increases, the time-saving advantages grow even larger. For example, the HB method uses 105 minutes to find a small MLP model with roughly the same NT metric as the one found by RS, which uses 148 minutes. When it comes to large MLPs, the HB spends 113 minutes finding a model similar to the one found by RS, which takes 208 minutes. The HB and HB-GP methods also achieve a higher PI than the RS baseline.

A similar story describes the case for CNN models. The HB and HB-GP obtain a lower NT result for all CNNs with three sizes than the RS, and the search time difference between the proposed methods and random search is even larger. HB finds a better model using 131 minutes for a small variant, while RS finds a worse model consuming 256 minutes. Again, when the model size increases, the time saved grows significantly. HB-GP finds a slightly worse but on-par model within 195 minutes for a large CNN configuration, while RS finds a slightly better model yet uses a whopping 459 minutes. The RS baseline method, however, achieves a larger PI for all three sizes compared with the proposed new methodologies. However, it should be noted that the PI metric is sensitive to the dataset setup, model configuration, objective function, etc., and is less informative when determining the model's performance compared with other hard metrics like GE and NT. Therefore, for CNN, we can still conclude that the proposed methods can obtain models with on-par performance while using significantly less time.

The same experiment is conducted with the ESHARD dataset to demonstrate the performance difference between various tuners in more demanding situations. The ESHARD dataset implements both masking and shuffling countermeasures. These dual countermeasures significantly increase the resistance against side-channel attacks by introducing more randomness and noise into the measurements, making it harder to extract the cryptographic keys. The complex nature makes this dataset a harder target for models to break than the ASCADf dataset. Table 7 documents the performance detail of the model architectures evaluated. To further simulate a harder scenario for the models, the total training budget is set to the small variant, with 2400 epochs for the entire search. The results confirm the difficulties for models in making a correct guess brought by the complexity of the

dataset and reduced total budget, that for all configurations, the NT metric significantly deteriorates when compared with results on the ASCADf dataset. However, the proposed HB-GP and HB algorithms still perform better regarding a lower NT and a shorter search time in every configuration tested. For MLPs, models found by the two new methods mark a significantly lower NT metric than those obtained by random search, using around half the time. As for CNNs, the new approaches still obtain more competent models and consume less time than the baseline, though the time-saving advantage isn't as large as the ones shown in MLP or on the ASCADf dataset. Nevertheless, the results also advocate for the superiority of the HB-GP and HB tuners, as they can lead to better models for SCA tasks requiring less time searching.

Results in Table 6 and Table 7 have demonstrated that both the default Hyperband (HB) and the enhanced HyperDrive (HB-GP) show considerable performance boost compared with random search. The main modification and advantage of HB-GP is the Gassuian Process kernel, which suggests new candidates based on the observations made. In comparison, the core of the default HB is still a random search; though it can fasten the search process, it can not use the obtained observations to guide future searches. Therefore, this work further introduces HB-GP to simultaneously address performance and search speed. A performance comparison is conducted to demonstrate the advantage of HB-GP compared with the original HB.

Figure 22 and 23 shows the performance (NT metric) comparison of the HB-GP and HB algorithms on a large MLP and a CNN model, respectively. Results clearly exemplify the performance boost of the HB-GP method. Regardless of epoch usages, the enhanced variant (purple) achieves a lower NT value than the standard one (green). It can also be noticed that the HB-GP tuner obtains a less fluctuated outcome, which is evident by the generally narrower min-max band. These findings confirm the advantages of the GP kernel in the Hyperband, proving the superiority of the HB-GP method.

To sum up, this set of experiments extensively examines and compares the performance of the RS, HB, and HB-GP tuners. It has been proven that **Both HB-GP and HB tuners can achieve an on-par or even better model with lower NT metric while using significantly less time.** This is the case regardless of datasets, model architectures, and model sizes. It also suggests that HB-GP obtains an even better result than HB, benefitting from the learning capability on observations from the GP kernel. Therefore, **this work recommends replacing the current widely-used random search tuning methodology with HyperDrive tuner or Hyperband, which results in finding models with better performance while using significantly less time.** These proposed algorithms can drastically increase the efficiency of finding deep-learning models and benefit the domain of deep-learning-based SCA with the capability of obtaining more competent models swiftly.

6.5 A revisit to the configurations obtained

The proposed tuner's capacity to obtain competent models with proper hyperparameters has been proven in section 6.4. In addition to that, this work reviews the effectiveness of the tuners in terms of locating the best possible configuration. This involves comparing the found hyperparameter settings with results obtained in a search space scan, with one hyperparameter being probed. In contrast, others remain at the best value returned by the tuner. If the found configuration is located close to the global optimum discovered in the scan, it can be concluded that the tuner does obtain an adequate setup. The three continuous learning hyperparameters, L1, BS, and LR, are revisited and analyzed. The scan is executed in the same approach as the Single-hyperparameter characteristic scan shown in section 6.2 but with the best values found.

Figure 24 illustrates the scan plot of NT metrics over different L1 values in the search space and the relation to the L1 setup of the best-found model. Data is collected with

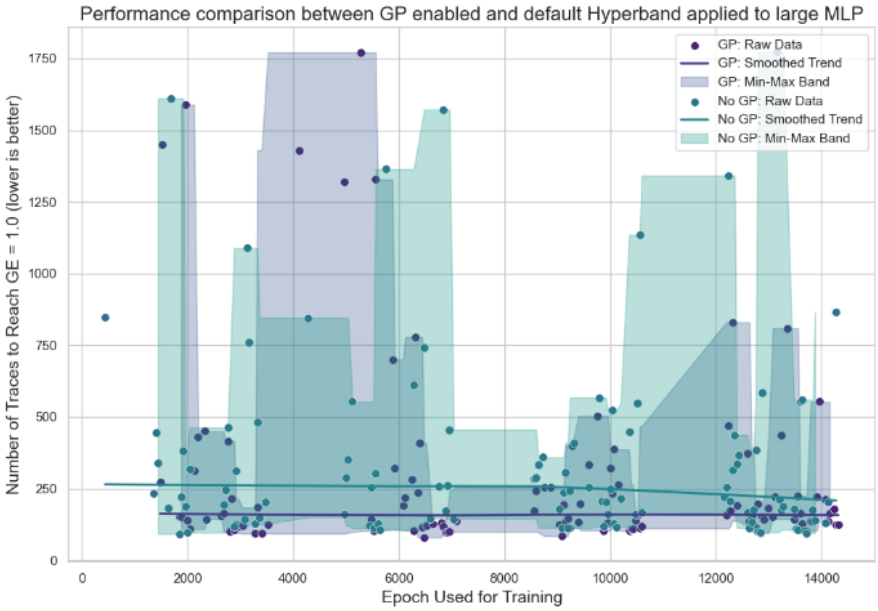

Figure 22: Performance comparison between HB-GP and default HB over different total budgets with a large MLP. The HB-GP (purple) achieves lower NT metrics than the default HB (green), regardless of the epoch used.

a small MLP model examined on the ASCADf dataset in a wide search space, and the orange dot represents the L1 value and performance of the model selected by the tuner (HB-GP). The test is performed with all other hyperparameters except L1, fixed at the value found by the tuner. It can be clearly noticed that the point lies close to the global optima of L1 in the search space, proving the tuner indeed boasts the ability to search for promising configurations for L1 regularization. The same observation is also made for LR and BS, as depicted in Figure 25 and 26.

Based on the results of re-scanning the hyperparameters in this experiment, it can be further confirmed and concluded that the hyperparameter configurations returned by the tuner are located in proximity to the global optima in the defined search space. This demonstrates the capacity of the proposed tuner to find extraordinary candidates and further validates the efficacy of the proposed methods.

7 Discussion

Since the introduction of deep-learning methods in the field of SCA, researchers have proven the effectiveness of such an approach in obtaining secrets in encrypted devices. However, few insights on the mechanism of the deep learning model performing SCA tasks and the corresponding hyperparameter tuning methodologies have been obtained. The widely adopted method to search for a competent model till this day in the domain is still the naive random search. Though, in most cases, the random search can eventually find a competent model, it requires much time to converge. Meanwhile, the absence of knowledge of the mechanism and interconnection of the hyperparameters makes it hard to rule out unnecessary variables and define a much more precise and narrower search space, which

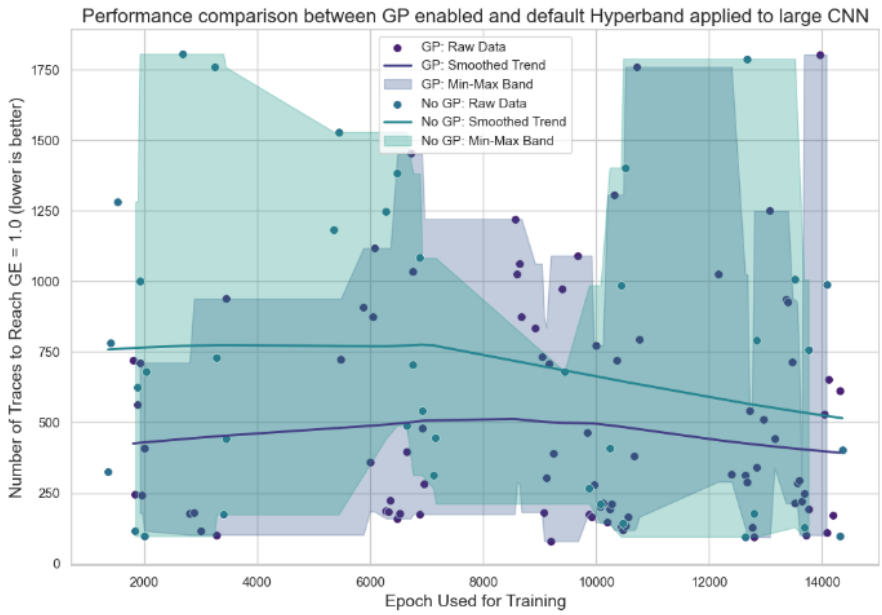

Figure 23: Performance comparison between HB-GP and default HB over different total budgets with a large CNN. The HB-GP (purple) achieves lower NT metrics than the default HB (green), regardless of the epoch used.

makes the already lengthy search process even more extensive.

This work aims to examine the interconnections between hyperparameters, propose new search algorithms to replace the random search, and draft guideline suggestions for performing hyperparameter tuning for SCA task-tailored deep-learning models, as discussed in section 4. After extensive experiments and analysis, this work claims the following findings and recommendations:

- **Learning hyperparameters impact the model's overall performance more than architectural hyperparameters.** It is observed that the size of the model does not significantly influence the model's performance. As for the three sizes tested for both MLP and CNN, no correlation is found between the architectural setting of the model and its corresponding performance. Also, the result proves that MLP can achieve on-par performance compared with CNN and uses noticeably less time. Therefore, it is recommended to focus on learning hyperparameters and avoid tuning the model's architectural configuration. Meanwhile, it is recommended that MLP would be sufficient in most cases.

- **Noticeably different weights of the learning hyperparameters exist.** It is observed that regularization exerts more influence on performance, followed by batch size and learning rate. Optimizer kind and activation functions influence the overall performance less as long as they are properly configured with a normal choice for the task. Also, it is noticed that L1 regularization works better compared with L2. Therefore, if the training budget is constrained, focusing on the L1 value, batch size setting, and learning rate configuration should be adequate to find a competent model.

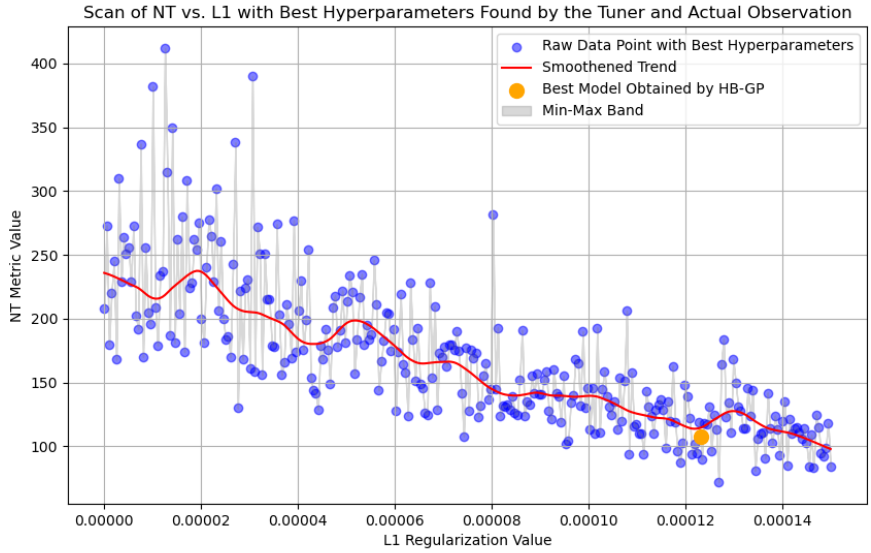

Figure 24: A re-scan of NT metric over L1 in the large search space defined with all other hyperparameters set at the value of the best small MLP model found. The L1 configuration obtained by the tuner is shown as the orange dot. This confirms that the found L1 setting is close to the global optima in the search space.

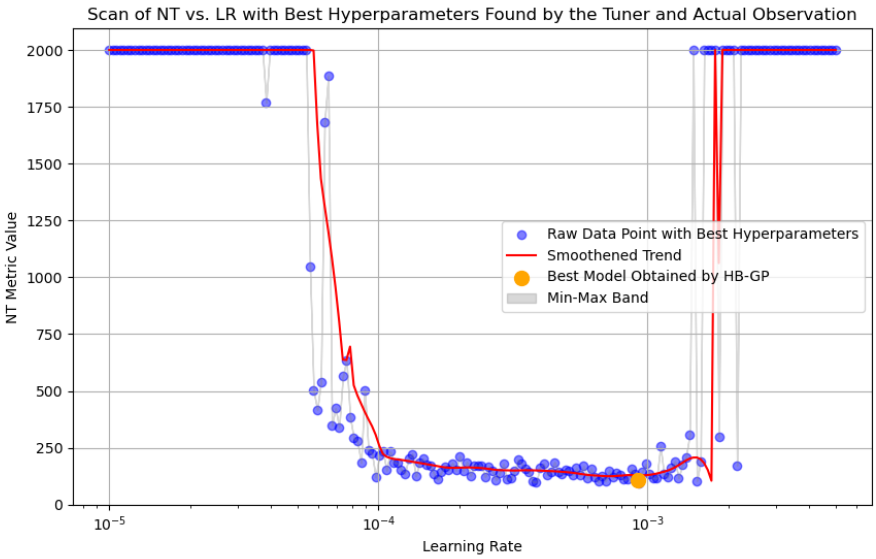

Figure 25: A re-scan of NT metric over LR in the large search space defined with all other hyperparameters set at the value of the best small MLP model found. The LR configuration obtained by the tuner is shown as the orange dot. This confirms that the found LR setting is close to the global optima in the search space.

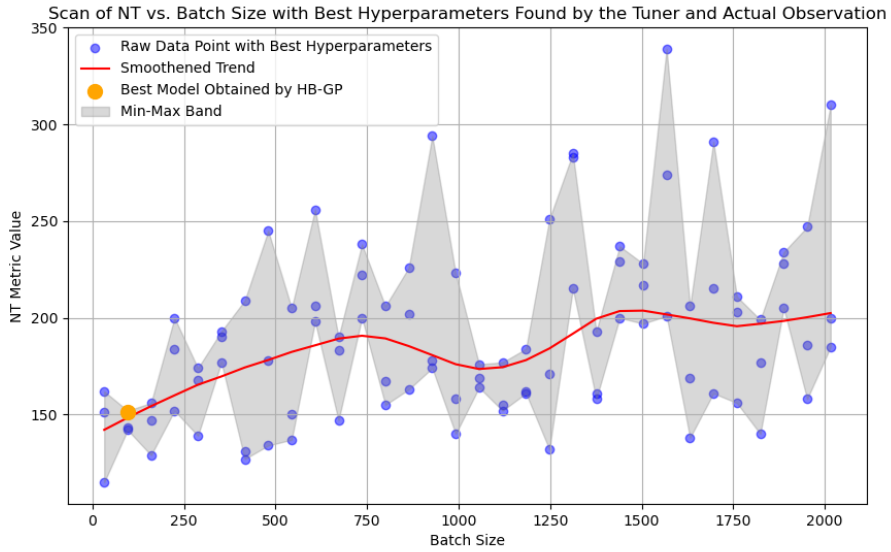

Figure 26: A re-scan of NT metric over BS in the large search space defined with all other hyperparameters set at the value of the best small MLP model found. The BS configuration obtained by the tuner is shown as the orange dot. This confirms that the found BS setting is close to the global optima in the search space.

- **Barrier, correlation, and interconnection exist for the learning hyperparameters.** It is found that the working region for the hyperparameter setting investigated is rather narrow. For L1 regularization, there exists a threshold before which the model's performance increases as the regularization intensifies, and the model is disabled after the L1 setting exceeds the threshold. For batch size and learning rate, an inverse-proportional relation is found. Therefore, in tuning these hyperparameters, it is recommended to utilize these characteristics to reduce the search space and save time.
- **The proposed HB and enhanced HB-GP show performance improvement and search time reduction compared with the current popular choice of random search.** In the six model configurations with distinct sizes and methodologies on two datasets, the proposed methods outperform random search by obtaining on-par or even better results with drastically less time, proving its capability to improve search efficiency. This allows researchers to search for more promising models using the same time or with the same number of candidates with significantly less time, in some cases, 1/3 of the time required by random search. It is therefore recommended to utilize the HB and HB-GP in SCA deep-learning model tuning.

However, this work also has some limitations. The first is the limitation of the model architectures evaluated. This work only examines MLP and CNN, two popular choices in the field. However, more state-of-the-art architectures like Transformers and time-series RNNs or LSTMs might perform better than traditional choices. Also, this work only evaluates the performance on two datasets, which may not generalize the tuner's and hyperparameters' behavior well. For example, though proven on the most popular datasets, the performance of the tuners and hyperparameter interconnections on datasets with significantly more noise may no longer be the case. Also, due to computational limitations,

this work only evaluates models with fewer epochs in total, which might not allow for the full potential of the HyperDrive method, as the Gaussian process component in it relies on the quality and quantity of observations and few total epochs result in fewer options examined and observations made. This explains why HyperDrive does not achieve significantly better results than the default HB. Yet, given the results obtained, there is reason to believe that the performance boost of HyperDrive will increase once more observations are made within a search.

8 Conclusion and future work

This work examined the fundamental component of deep-learning-based SCA: hyperparameter tuning, which has not been extensively explored. We studied and identified the most influential hyperparameters that determine model performance in the domain and revealed the existence of connections between these hyperparameters. These insights can be utilized to scale down the search space when performing model tuning to reduce the computational and time resources required, allowing for higher efficiency. Also, we proposed two new search algorithms, the Hyperband (HB) and HyperDrive, a Hyperband with Gaussian Process embedded, to replace the currently widely-used random search. We have demonstrated that these algorithms hold the capacity to obtain model candidates with on-par or even better performance while using significantly less time than the random search. We also observed the advantage of incorporating a GP kernel inside Hyperband to allow for real-time improvement and learning from previous observations, which benefits a more targeted and concentrated search. We made several recommendations for executing hyperparameter optimization for deep-learning models for SCA tasks based on the results obtained in four groups of experiments.

The future work of this paper mainly involves two directions: confirming the observations in other datasets and other model architectures and examining the behavior of the tuners in more budget configurations. As discussed in section 7, this work only investigates the domain's two most popular model architectures: MLP and CNN. We intend to incorporate more state-of-the-art model types in future work, such as transformers, to evaluate whether they will bring more power and produce more competent models. Also, we plan to apply the proposed methods on more complex datasets with more noise and more powerful countermeasures to evaluate their efficacy. In addition, we intend to test with more powerful computational resources on a larger scale to observe the corresponding behaviors. We hope this work and further investigations will help SCA researchers better understand how to tune their deep-learning models and provide them with more efficient and competent algorithms for finding the optimal model without hassle.

Acknowledgements

Ad astra per aspera.

To the boy who relentlessly pursues perfection in every endeavor to make a splash. Together with the tough moments, struggles and gloomy rainy days embraced by courage, perseverance, and cheerfulness.

My sincere appreciation goes out to my supervisor, Prof. Guilherme Perin, as his commitment to guiding me and generosity in sharing insights with me was nothing short of inspiring. I am also grateful to Sengim for his prompt and helpful responses to all my inquiries, coming very timely when I needed them. I would like to express my thanks to Prof. Nele Mentens as well, as her feedback and encouragement helped me reflect and stay on course. I must also extend my deepest admiration to myself for maintaining the courage to try and stay high in an era of uncertainty and unprecedented challenges. Despite encountering more failures than ever before, hope is never lost and I am always prepared to crush these hurdles with resilience. Words are beyond me for cherishing the togetherness presented across the continent eight thousand kilometers away that reached me every single moment. Connections haven't been, and will never be broken by distances that kept us apart for now. My family and wholehearted friends always welcomed me with a place to be myself and recharge, and assured me that I had unwavering support and companionship along the journey. They all made me, and they are truly a gift in my life. I shared this odyssey, and for that, I am deeply humbled and filled with gratitude.

From a first stride through the doors of Leiden, where before me spread a vast landscape of knowledge and seemed like alien territory, to now, where I leave as an ardent explorer, confident to share the insights I have drawn. All the learning moments and challenges enfolded have been steps toward growth as a student and adventurer. It has been a transforming journey I will cherish for life.

As the chapter of my master's journey draws to a close, I look to the future with a hopeful heart. Whenever I revisit these words, may I find that I am still the boy who was confident, fearless, and passionate about life—forever the explorer, courageously charting paths untrodden, as I have always held in my heart, to boldly go where no one has gone before.

References

[AARR02] Dakshi Agrawal, Bruce Archambeault, Josyula R. Rao, and Pankaj Rohatgi. The em side-channel(s). In *Revised Papers from the 4th International Workshop on Cryptographic Hardware and Embedded Systems*, CHES '02, page 29–45, Berlin, Heidelberg, 2002. Springer-Verlag.

[AMH21] Noor Awad, Neeratyoy Mallik, and Frank Hutter. Dehb: Evolutionary hyperband for scalable, robust and efficient hyperparameter optimization, 2021.

[BB12] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(10):281–305, 2012.

[BCGR22] Julien Béguinot, Wei Cheng, Sylvain Guilley, and Olivier Rioul. Be my guess: Guessing entropy vs. success rate for evaluating side-channel attacks of secure chips. In *2022 25th Euromicro Conference on Digital System Design (DSD)*, pages 496–503, 2022.

[Bon99] Dan Boneh. Twenty years of attacks on the rsa cryptosystem. *Notices of the American Mathematical Society*, 46:203–212, 1999.

[BPS+20] Ryad Benadjila, Emmanuel Prouff, Rémi Strullu, Eleonora Cagli, and Cécile Dumas. Deep learning for side-channel analysis and introduction to ASCAD database. *J. Cryptographic Engineering*, 10(2):163–188, 2020.

[DDR99] Joa Daor, Joan Daemen, and Vincent Rijmen. Aes proposal: rijndael. 10 1999.

[DPRS11] Julien Doget, Emmanuel Prouff, Matthieu Rivain, and François-Xavier Standaert. Univariate side channel attacks and leakage modeling. Cryptology ePrint Archive, Paper 2011/302, 2011. https://eprint.iacr.org/2011/302.

[EWTS14] Hassan Eldib, Chao Wang, Mostafa Taha, and Patrick Schaumont. Qms: Evaluating the side-channel resistance of masked software from source code. In *2014 51st ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6, 2014.

[FH19] Matthias Feurer and Frank Hutter. *Hyperparameter Optimization*, pages 3–33. Springer International Publishing, Cham, 2019.

[GST14] Daniel Genkin, Adi Shamir, and Eran Tromer. RSA key extraction via low-bandwidth acoustic cryptanalysis. In Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part I*, volume 8616 of *Lecture Notes in Computer Science*, pages 444–461. Springer, 2014.

[Has20] Trevor Hastie. Ridge regularizaton: an essential concept in data science, 2020.

[Her09] Simon Heron. Advanced encryption standard (aes). *Network Security*, 2009(12):8–12, 2009.

[HS13] Michael Hutter and Jörn-Marc Schmidt. The temperature side channel and heating fault attacks. In Aurélien Francillon and Pankaj Rohatgi, editors, *Smart Card Research and Advanced Applications - 12th International Conference, CARDIS 2013, Berlin, Germany, November 27-29, 2013. Revised Selected Papers*, volume 8419 of *Lecture Notes in Computer Science*, pages 219–235. Springer, 2013.

[Inc15] Plotly Technologies Inc. Collaborative data science, 2015.

[JT15] Kevin Jamieson and Ameet Talwalkar. Non-stochastic best arm identification and hyperparameter optimization, 2015.

[KFNO13] Takeshi Kumaki, Tomohiro Fujita, Mamoru Nakanishi, and Takeshi Ogura. Morphological pattern spectrum and block cipher processing based image-manipulation detection. *Nonlinear Theory and Its Applications, IEICE*, 4:400–418, 10 2013.

[KJJ99a] Paul Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In Michael Wiener, editor, *Advances in Cryptology — CRYPTO' 99*, pages 388–397, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.

[KJJ99b] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In Michael J. Wiener, editor, *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer, 1999.

[Koc96] Paul C. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In *Proceedings of CRYPTO'96*, volume 1109 of *LNCS*, pages 104–113. Springer-Verlag, 1996.

[KWPP22] Maikel Kerkhof, Lichao Wu, Guilherme Perin, and Stjepan Picek. Focus is key to success: A focal loss function for deep learning-based side-channel analysis. In Josep Balasch and Colin O'Flynn, editors, *Constructive Side-Channel Analysis and Secure Design*, pages 29–48, Cham, 2022. Springer International Publishing.

[LDW21] Jiaqi Lou, Ke Dong, and Maosen Wang. A parallel coordinates plot method based on unsupervised feature selection for high-dimensional data visualization. In *2021 International Wireless Communications and Mobile Computing (IWCMC)*, pages 532–536, 2021.

[LJD+18] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization, 2018.

[LZC+21] Xiangjun Lu, Chi Zhang, Pei Cao, Dawu Gu, and Haining Lu. Pay attention to raw traces: A deep learning architecture for end-to-end profiling attacks. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2021(3):235–274, Jul. 2021.

[MDP19] Loïc Masure, Cécile Dumas, and Emmanuel Prouff. A comprehensive study of deep learning for side-channel analysis. Cryptology ePrint Archive, Paper 2019/439, 2019. https://eprint.iacr.org/2019/439.

[MPP16] Houssem Maghrebi, Thibault Portigliatti, and Emmanuel Prouff. Breaking cryptographic implementations using deep learning techniques. *IACR Cryptol. ePrint Arch.*, 2016:921, 2016.

[NAHAB23] Khalid Nahar, Obaida Al-Hazaimeh, Moyawiah Alshanaq, and Mohammed Bawaneh. Analytical approach for data encryption standard algorithm. *International Journal of Interactive Mobile Technologies (iJIM)*, 17, 05 2023.

[PPM+22] Stjepan Picek, Guilherme Perin, Luca Mariot, Lichao Wu, and Lejla Batina. Sok: Deep learning-based physical side-channel analysis. *ACM Comput. Surv.*, oct 2022. Just Accepted.

[PWP21] Guilherme Perin, Lichao Wu, and Stjepan Picek. Exploring feature selection scenarios for deep learning-based side-channel analysis. Cryptology ePrint Archive, Paper 2021/1414, 2021. https://eprint.iacr.org/2021/1414.

[QS01] Jean-Jacques Quisquater and David Samyde. Electromagnetic analysis (ema): Measures and counter-measures for smart cards. In Isabelle Attali and Thomas Jensen, editors, *Smart Card Programming and Security*, pages 200–210, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

[RSA78] Ronald L. Rivest, Adi Shamir, and Leonard Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[RWGP21] Jorai Rijsdijk, Lichao Wu, Perin Guilherme, and Stjepan Picek. Reinforcement learning for hyperparameter tuning in deep learning-based side-channel analysis. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 677–707, 07 2021.

[SLA12] Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. Practical bayesian optimization of machine learning algorithms, 2012.

[Tib96] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, 58(1):267–288, 1996.

[VTM23] Aurélien Vasselle, Hugues Thiebeauld, and Philippe Maurine. Spatial dependency analysis to extract information from side-channel mixtures: extended version. *Journal of Cryptographic Engineering*, 13(4):409–425, 11 2023.

[WAGP20] Lennert Wouters, Victor Arribas, Benedikt Gierlichs, and Bart Preneel. Revisiting a methodology for efficient cnn architectures in profiling attacks. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2020(3):147–168, Jun. 2020.

[WDR+23] Chenggang Wang, Jimmy Dani, Shane Reilly, Austen Brownfield, Boyang Wang, and John M. Emmert. Tripletpower: Deep-learning side-channel attacks over few traces. In *2023 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 167–178, 2023.

[WPP20] Lichao Wu, Guilherme Perin, and Stjepan Picek. I choose you: Automated hyperparameter tuning for deep learning-based side-channel analysis. Cryptology ePrint Archive, Paper 2020/1293, 2020. https://eprint.iacr.org/2020/1293.

[WR96] Christopher K. I. Williams and Carl Edward Rasmussen. Gaussian processes for regression. In David S. Touretzky, Michael C. Mozer, and Michael E. Hasselmo, editors, *Advances in Neural Information Processing Systems 8*, pages 514–520, Cambridge, MA, USA, 1996. Max-Planck-Gesellschaft, MIT Press. Presented at the Ninth Annual Conference on Neural Information Processing Systems (NIPS 1995), Denver, CO, USA.

[ZBHV19] Gabriel Zaid, Lilian Bossuet, Amaury Habrard, and Alexandre Venelli. Methodology for efficient cnn architectures in profiling attacks. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2020(1):1–36, Nov. 2019.