# I Choose You

## Automated Hyperparameter Tuning for Deep Learning-based Side-channel Analysis

Wu, Lichao ; Perin, Guilherme; Picek, Stjepan

**DOI**
10.1109/TETC.2022.3218372

**Publication date**
2024

**Document Version**
Final published version

**Published in**
IEEE Transactions on Emerging Topics in Computing

IEEE TRANSACTIONS ON
EMERGING TOPICS
IN COMPUTING

Received 16 December 2021; revised 11 June 2022; accepted 16 October 2022.
Date of publication 7 November 2022; date of current version 7 June 2024.

*Digital Object Identifier 10.1109/TETC.2022.3218372*

# I Choose You: Automated Hyperparameter Tuning for Deep Learning-Based Side-Channel Analysis

**LICHAO WU, GUILHERME PERIN, AND STJEPAN PICEK**

Lichao Wu and Guilherme Perin are with the Delft University of Technology, 2628 Delft, XE, The Netherlands
Stjepan Picek is with the Delft University of Technology, 2628 Delft, XE, The Netherlands

CORRESPONDING AUTHOR: STJEPAN PICEK (s.picek@tudelft.nl.)

This work was supported in part by the NWA Cybersecurity Call with Project name PROACT with Project number NWA.1215.18.014 and in part by the Netherlands Organization for Scientific Research NWO Project DISTANT (CS.019).

**ABSTRACT** Today, the deep learning-based side-channel analysis represents a widely researched topic, with numerous results indicating the advantages of such an approach. Indeed, breaking protected implementations while not requiring complex feature selection made deep learning a preferred option for profiling side-channel analysis. Still, this does not mean it is trivial to mount a successful deep learning-based side-channel analysis. One of the biggest challenges is to find optimal hyperparameters for neural networks resulting in powerful side-channel attacks. This work proposes an automated way for deep learning hyperparameter tuning based on Bayesian optimization. We build a custom framework denoted AutoSCA supporting machine learning and side-channel metrics. Our experimental analysis shows that our framework performs well regardless of the dataset, leakage model, or neural network type. We find several neural network architectures outperforming state-of-the-art attacks. Finally, while not considered a powerful option, we observe that neural networks obtained via random search can perform well, indicating that the publicly available datasets are relatively easy to break.

**INDEX TERMS** Bayesian optimization, deep learning, hyperparameter optimization, side-channel analysis

## I. INTRODUCTION

Side-channel analysis (SCA) is a well-known and powerful type of implementation attack on cryptographic algorithms [1]. A common division of side-channel analysis is into direct attacks like Simple Power Analysis (SPA) and Differential Power Analysis (DPA) [2], and two-stage (profiling) attacks like template attack [3] and machine learning-based attacks [4]–[6]. Direct attacks do not require access to an identical and open copy of the device under attack, and they have no hyperparameters to tune. Simultaneously, to break a specific implementation, such attacks could require tens to hundreds of thousands of measurements. Profiling attacks assume an "open" device (or a copy of it), but the key recovery may require only a few measurements. Today, the most powerful representatives of profiling attacks come from the machine learning domain. Such machine learning-based (or deep learning-based) attacks can break targets protected with countermeasures, but to reach that level of performance, they also require a careful hyperparameter tuning [7], [8].

Hyperparameter tuning can differentiate a machine learning-based SCA that performs "only" satisfactorily from the one that breaks a target in a few measurements or even in a single measurement. In previous years, when simpler machine learning techniques were still commonly used in SCA, hyperparameter tuning was an important factor in the attack's success, but not the only one. For instance, feature engineering (e.g., dimensionality reduction like PCA [9] and LDA [10]) played an equally important role as hyperparameter tuning in mounting a successful attack. With deep learning, pre-processing and feature engineering lost most of their importance as deep learning techniques are powerful enough to work with raw traces [6], [11]. Thus, the security evaluator's (attacker's) attention shifted toward hyperparameter tuning as the core task for a successful deep learning-based side-channel analysis. Nevertheless, suppose one of the assumptions in the profiling phase involves an adversary restricted in terms of measurements. In that case, hyperparameter tuning plays a significant role in security evaluations

by allowing the discovery of models that can break targets with fewer side-channel traces [12].

The problem of hyperparameter tuning in SCA is a difficult one. In general, we do not know what would be the best hyperparameter tuning strategy for every setting. Deep learning architectures have many hyperparameters to tune, so it is impossible to check all options. Even a grid search becomes difficult for larger neural network models and datasets. In SCA, we encounter additional difficulties as we do not know what hyperparameters influence the attack performance compared to those that show little to no importance. Consequently, considering the number of different datasets, leakage models, neural network architectures, and hyperparameter options, it is evident that an exhaustive search is not an option. Simultaneously, random search and grid search offer good performance in many settings, but we are still left wondering how far those architectures are from the optimal ones. Finally, many SCA evaluators are not experts in machine learning, and for them, it is not easy to recognize the important hyperparameters without significant experience.

When considering machine learning and profiling SCA, several works discuss hyperparameter tuning, see, e.g., [13], [14]. While those works manage to (partially) answer the question of better performing neural network architectures for specific settings, they do not provide a methodology for tuning the hyperparameters. Still, by recognizing the less important hyperparameters, those works indirectly help make more efficient hyperparameter tuning. A few papers discuss how to provide a more structured way to build neural networks for SCA. More precisely, those works offer methodologies to build neural network architectures for SCA [7], [8]. Unfortunately, while such methodologies represent a good start, they are far from perfect as they require knowledge about the dataset to be attacked, and they are not easy to extend to other datasets. More recently, the SCA community experimented with reinforcement learning to design neural networks capable of breaking various targets [15]. Although this approach can find good hyperparameters for a successful side-channel attack, the search process is extremely time-consuming.

This paper proposes the hyperparameter tuning approach based on the Bayesian optimization optimized for side-channel analysis. More precisely, we start from a well-known Bayesian optimization paradigm for hyperparameter tuning, and we develop a custom SCA framework supporting both machine learning and SCA metrics. We manage to optimize neural networks (in this work, multilayer perceptron and convolutional neural networks) to reach excellent performance for several commonly used SCA datasets. By doing this, we offer a simple yet powerful approach that results in high-performing neural networks for SCA that do not require knowledge about the datasets to be attacked. Moreover, since our framework offers automated hyperparameter tuning, it is easy to switch to different datasets.

Our main contributions are:

1) To the best of our knowledge, we are the first to propose Bayesian optimization for hyperparameter tuning for deep learning-based SCA. We compare this approach with a common one in SCA, which is a random search. Our results show that for some commonly considered datasets in deep learning-based SCA, a random search can find top-performing neural networks, which raises the question of how justified it is to develop methodologies on such simple datasets. On the other hand, Bayesian optimization performs consistently well (and significantly better than random search in many settings), making it a natural choice to select when facing uncertainties about the attack's difficulty.
2) We develop a custom framework (AutoSCA) based on Keras Tuner [16] for hyperparameter tuning in SCA that optimizes machine learning and SCA metrics. The code is available at https://github.com/AISyLab/AutoSCA

   The results indicate that the metric is less important when the dataset is easy to attack. On the other hand, for more challenging settings, we found that the recently proposed $L_m$ metric performs the best [17].
3) We compare the neural network architectures obtained through our framework with the state-of-the-art results showing our architectures reach better performance.
4) We show that random search can find neural network architectures outperforming the state-of-the-art results obtained through recently proposed methodologies. We postulate that this happens as the datasets are relatively easy to attack, and many neural networks perform well. Such randomly obtained neural networks are commonly larger than those obtained through various methodologies (i.e., they have more trainable parameters), but we do not consider this a serious limitation.

## II. BACKGROUND

### A. NOTATION

We use calligraphic letters like $\mathcal{X}$ to denote sets and the corresponding upper-case letters $X$ to denote random variables and random vectors $\mathbf{X}$ over $\mathcal{X}$. The corresponding lower-case letters $x$ and $\mathbf{x}$ denote realizations of $X$ and $\mathbf{X}$, respectively.

We denote with $k$ a key candidate taking its value from the keyspace $\mathcal{K}$. The correct key is denoted with $k^*$. We use $\boldsymbol{\theta}$ to denote the vector of parameters learned in a profiling model.

Finally, we define a dataset as a collection of side-channel traces (measurements). Each trace $\mathbf{t}_i$ in the dataset is associated with an input value (plaintext or ciphertext) $\mathbf{d}_i$ and a key $\mathbf{k}_i$. We divide the dataset into disjoint subsets where the training set contains $N$ traces, the validation set contains $V$ traces, and the attack set contains $Q$ traces.

### B. SUPERVISED MACHINE LEARNING AND PROFILING SCA

Supervised machine learning represents the task of learning a function $f$ that maps an input to the output ($f : \mathcal{X} \rightarrow Y$)) based on examples of input-output pairs. The function $f$ is parameterized by $\boldsymbol{\theta} \in \mathbb{R}^n$, where $n$ denotes the number of trainable parameters. Supervised learning happens in two

phases: training and test. This corresponds to profiling SCA phases, commonly denoted as profiling and attack phases.

1) Training phase: the goal is to learn $\boldsymbol{\theta}$ minimizing the empirical risk represented by a loss function $L$ on a dataset of size $N$ (i.e., on the profiling (training) set).

   As usual in profiling SCA, we consider the $c$-classification task, where $c$ denotes the number of classes depending on the leakage model, as discussed in Section 2.4. More precisely, the classifier $f$ is a function that maps input features to label space ($f : \mathcal{X} \rightarrow \mathbb{R}^c$). As already stated, we consider deep learning techniques and, more precisely, multilayer perceptron (MLP) and convolutional neural networks (CNNs). The function $f$ is a (deep) neural network with the *Softmax* output layer. Additionally, we encode classes in one-hot encoding, where each class is represented as a vector of $c$ values that has zero on all the places, except one place, denoting the membership of that class, i.e., $\mathbf{y}_i = \mathbf{e}_{y_i} \in \{0, 1\}^c$ such that $1^T \mathbf{y}_i = 1, \ \forall i$.
2) Attack phase: the goal is to make predictions about the classes

$$y(x_1, k^*), \ldots, y(x_Q, k^*),$$

   where $k^*$ represents the secret (unknown) key on the device under the attack. The outcome of predicting with a model $f$ on the attack set is a two-dimensional matrix $P$ with dimensions equal to $Q \times c$. The cumulative sum $S(k)$ for any key candidate $k$ is then used as a maximum log-likelihood distinguisher:

$$S(k) = \sum_{i=1}^{Q} \log\left(\mathbf{p}_{i,y}\right). \tag{1}$$

   The value $\mathbf{p}_{i,y}$ is the probability that for a key $k$ and input $d_i$, we obtain the class $y$ (with $\sum_y^c \mathbf{p}_{i,y} = 1, \forall i$). The class $y$ is derived from the key and input through a cryptographic function $CF$ and a leakage model $l$.

We follow a standard assumption in supervised machine learning, stating that the training and test data are drawn independently from identical distributions (commonly called i.i.d. assumption). This means that the process that samples the data has no memory, i.e., we do not expect a higher correlation for any two traces compared to other traces.

The goal of a side-channel attacker is to reveal the secret key $k^*$. Commonly used metrics to estimate the effort required to reveal the secret key are the success rate (SR) and the guessing entropy (GE) [18]. This work uses guessing entropy to estimate the attack performance. For $Q$ traces in the attack phase, an attack results in a key guessing vector $\mathbf{g} = [g_1, g_2, \ldots, g_{|\mathcal{K}|}]$ in decreasing order of probability ($g_1$ is the most likely and $g_{|\mathcal{K}|}$ the least likely key candidate). Guessing entropy represents the average position of $k^*$ in $\mathbf{g}$. Commonly, averaging is done over a number of independent experiments to obtain statistically significant results. Guessing entropy can be observed for the whole key or separate key bytes, then denoted as partial guessing entropy. Since we attack only a single key byte, we calculate partial guessing entropy but denote it as guessing entropy for simplicity.

### C. HYPERPARAMETERS VERSUS PARAMETERS

It is common to differentiate between parameters and hyperparameters in machine learning algorithms. Hyperparameters are all configuration variables external to the model $f$, e.g., the number of hidden layers in a neural network. Neural networks (deep learning) have many hyperparameters, making their tuning very difficult and computationally intensive.

The parameter vector $\boldsymbol{\theta}$ represents the configuration variables internal to the model $f$, and those values are learned from data.

### D. DATASETS AND LEAKAGE MODELS

We consider two leakage models $l$ in this paper:

1) The Hamming weight (HW) leakage model. In this leakage model, the attacker assumes the leakage is proportional to the sensitive variable's Hamming weight. When considering the AES cipher (8-bit S-box[1]), this leakage model results in nine classes.
2) The Identity (ID) leakage model. In this leakage model, the attacker considers the leakage in the form of an intermediate value of the cipher. When considering the AES cipher (8-bit S-box), this leakage model results in 256 classes.

The ASCAD dataset contains the measurements from an 8-bit AVR microcontroller running a masked AES-128 implementation [13]. There are two versions of the ASCAD dataset: one with a fixed key with 50 000 traces for profiling and 10 000 for the attack. The second version has random keys, and the dataset consists of 200 000 traces for profiling and 100 000 for the attack. We attack the third key byte for both versions as that is the first masked byte. For ASCAD with the fixed key, we use a pre-selected window of 700 features, while for ASCAD with random keys, the window size equals 1 400 features [13]. These datasets are available at https://github.com/ANSSI-FR/ASCAD

### E. LEAKAGE DISTRIBUTION DIFFERENCE AND CORRELATION WITH THE KEY GUESSING VECTOR

Recently, Wu et al. proposed a new metric called Leakage Difference Distribution (*LDD*) to describe the correlation among various key guesses [17]. First, to calculate *LDD*, it is required to calculate a hypothetical leakage distribution for every key candidate and all plaintexts for a given dataset. The obtained leakage distribution variation between different key candidates gives Leakage Distribution Difference. *LDD* aims to provide an estimation of the hypothetical label distribution variation between different key candidates, where a specific key will have a smaller probability to be selected based on a (properly) trained profiling model if *LDD* is large

[1]Or any cipher with 8-bit S-box.

between that key guess $k$ and the correct key $k^*$:

$$\mathbf{LDD}(k^*, k) = \sum_{i=0}^{Q} \|f(d_i, k^*) - f(d_i, k)\|^2, k \in \mathcal{K}, \quad (2)$$

where $f(d_i, k)$ is the leakage model function that returns the leakage value according to a key candidate $k$ and data value $d_i \in Q$, where $Q$ denotes the number of attack traces in the dataset. Depending on the leakage model function, we use, e.g., *HWDD* for the Hamming weight leakage model or *IDD* for the Identity leakage model.

If the profiling model performs well, then we can expect that the correlation between *LDD* and the key guessing vector will be good, which can be used to define a metric estimating how well did the profiling model fit the leakage:

$$L_m(\mathbf{LDD}, \mathbf{g}) = \mathsf{corr}(\mathsf{argsort}(\mathbf{LDD}), \mathbf{g}). \quad (3)$$

### F. BAYESIAN OPTIMIZATION

Tuning hyperparameters for deep neural networks is a computationally expensive task. Various neural architecture search (NAS) methods aim to find the best architecture for the given learning task and dataset within the deep learning domain. The NAS algorithms are commonly costly as their computational complexity depends on the number of neural network architectures to evaluate and the time needed to evaluate each network. Therefore, it is crucial to have an efficient method to select optimal hyperparameters when the number of iterations $t$ is limited due to either computation power or time. In that context, Bayesian optimization (BO) can be used to optimize any black-box function [19], [20].

In general, Bayesian optimization aims to find the parameters $\mathbf{x}'$ that maximize the function $f(x)$ over a domain $\mathcal{X}$:

$$x' = \arg\max_{\mathbf{x} \in \mathcal{X}} f(x). \quad (4)$$

Let us consider that the Bayesian optimization works over $t$ iterations. Then, Bayesian optimization aims to find the maximum point on the function using the minimum number of iterations. Formally, the aim is to minimize the number of iterations $t$ before we can guarantee for a given $\mathbf{x}'$ that $f(\mathbf{x}')$ is less than $\epsilon$ from the true maximum $f'$.

If the problem is simple, e.g., we search in a small hyperparameter space, random search or grid search is often sufficient. When considering larger search spaces, we can benefit from the memory in the process (i.e., considering the results from previous measurements). This is commonly possible with sequential search strategies, represented by sequential model-based optimization (SMBO) in the Bayesian optimization.

To achieve good results with any search strategy, we need to account for both exploration (visiting search space regions not visited before) and exploitation (sampling from more promising regions based on observed results). In Bayesian optimization, the aim is to build a probabilistic model of the underlying function that will include exploitation and exploration. We first require a probabilistic model of a function (often referred to as the surrogate model), where there are several ways to model it. This work considers the Gaussian Process, a common choice for Bayesian optimization, especially considering Euclidean spaces [21]. A Gaussian Process is a collection of random variables, where any finite number of such random variables is jointly normally distributed. Gaussian Process is defined by the mean function and the covariance function. We can estimate the function's distribution at any new point $\mathbf{x}^*$, where the mean gives the best estimate of the function value, and the variance gives the uncertainty.

Second, we require an acquisition function for Bayesian optimization to generate the next neural network architecture to observe, i.e., to select what point to sample next. More precisely, the acquisition function takes the mean and variance at each point $\mathbf{x}$ on the function and computes a value that indicates how desirable it is to sample next at this position. We use a common example of the acquisition function in this work: the upper confidence bound [22]. Upper confidence bound action selection uses uncertainty in the action-value estimates to balance exploration and exploitation. The value of the upper confidence bound function is an estimation of the lowest possible value of the cost function given the neural network $f$:

$$\alpha(\mathbf{x}^*) = \mu(\mathbf{x}^*) - \beta\sigma(\mathbf{x}^*). \quad (5)$$

Here, $\beta$ is the balancing factor to regulate the exploration and exploitation (we use the default value from Keras Tuner, which equals 2.6). This acquisition function computes the likelihood that the function at $\mathbf{x}^*$ will return a result higher than the current maximum $f(\mathbf{x}')$. For further information about Bayesian optimization, possible models of the functions, and acquisition functions, we refer interested readers to [21], [23].

## III. RELATED WORKS

When discussing machine learning approaches, we can divide the corpus of works based on the complexity of the used techniques and the hyperparameter tuning treatment. The first works considered simpler machine learning techniques like random forest [24] and support vector machines [25]. The main focus was on the attack performance and how those techniques compare with the template attack [3] and its variants [26]. From 2016 [6], the SCA community shifted a large part of its attention to deep learning techniques. Since then, the two most explored approaches have been multilayer perceptron and convolutional neural networks. Both approaches reached top attack performance even considering implementations with countermeasures [11], [27].

The SCA community's maturity in using machine learning can also be examined from the hyperparameter tuning perspective. Indeed, the first works do not mention whether they conduct hyperparameter tuning or even what are the final hyperparameters selected [28], [29]. Afterward, numerous

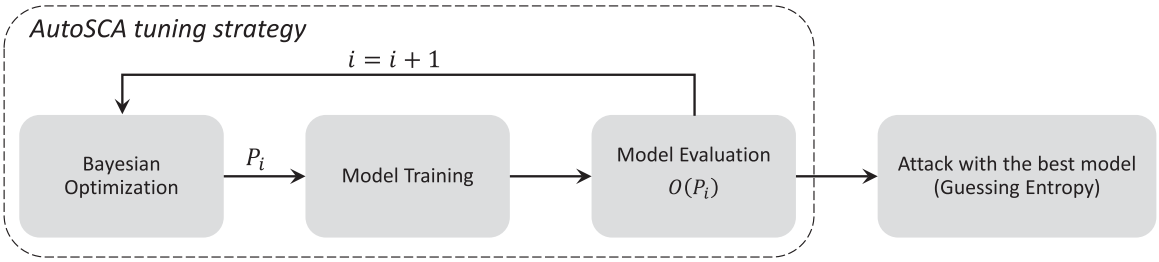



**FIGURE 1.** **AutoSCA framework.**

works investigate various machine learning techniques where hyperparameter tuning is done with a random or grid search. In [13], the authors conduct an empirical evaluation of different hyperparameters for CNNs when attacking the ASCAD dataset. Perin et al. conducted a random search in pre-defined ranges to build deep learning models to form ensembles [14]. Recently, reinforcement learning, with SCA-optimized learning objective, has been applied for hyperparameter tuning in deep learning-based profiling models [15]. The results achieved there managed to outperform previous state-of-the-art results, including those obtained by following specialized SCA methodologies.

Several works evaluate the influence of various hyperparameters systematically. L. Weissbart considered multilayer perceptrons and hyperparameter tuning for the number of layers/neurons and various activation functions [30]. Li et al. investigated the weight initialization options for MLP and CNN architectures [31]. These works discuss certain hyperparameters' influence in specific settings, but they do not offer the whole neural network architecture methodology. At the same time, they provide directions about the importance of specific hyperparameters, which can help improve hyperparameter tuning in the future.

Finally, some works aim to offer a methodology for building neural networks. For instance, Zaid et al. proposed a methodology to select hyperparameters related to the number of learnable parameters in CNNs [7]. Their investigation includes the number of filters, kernel sizes, strides, and neurons in fully connected layers. This is the first work to provide a systematic approach to building well-performing neural network architectures. Unfortunately, the proposed methodology has some drawbacks. While the authors presented CNN architectures that reach top performance, they also assume significant knowledge about datasets and consider only CNN architectures. What is more, it does not seem easy to extend this methodology to new datasets (or leakage models). Wouters et al. [8] improved upon the work from Zaid et al. [7]. The authors discussed several misconceptions in the first work, and they showed how to reach similar attack performance with significantly smaller neural network architectures. Thus, this shows that we are far from having established methodologies or ways to build them.

## IV. THE AUTOSCA FRAMEWORK

The AutoSCA framework can be divided into two steps:

1) characterizing the search space by testing different combinations of hyperparameters;
2) selecting the best candidate (profiling model) out of these attempts.

An illustration of the AutoSCA framework is shown in Figure 1. The framework allows tuning of various types of neural networks, e.g., multilayer perceptrons, convolutional neural networks, residual neural networks, or recurrent neural networks. Following the recent advances in deep learning-based SCA, this paper focus on MLP and CNN. Indeed, based on the literature, a prevailing number of works use MLP and CNN, and the results reported with those techniques are consistently breaking the investigated targets. To evaluate the efficiency of AutoSCA (we denote such experiments as BO since the framework uses Bayesian optimization), we compare it with random search (RS). In terms of search iterations, the iteration number is determined based on the extensive preliminary tuning phase. Specifically, during the preliminary tests, we observed that a higher number of searching iterations could help BO better characterize the search space, thus obtaining architectures with stronger attack performance than the one obtained by RS. To balance search performance and time consumption, eventually, our framework performs 200 iterations ($i = 200$) to test different hyperparameter combinations. In each iteration, the Bayesian optimization function outputs a set of hyperparameters $P_i$ to build the model, followed by the training process. Each profiling model is trained for ten epochs to speed up the training process. This also brings the additional benefit that the best model obtained from this setting would consume less training time for the real attack, increasing the attack efficiency. The search can be finalized within ten hours with a single CPU and an NVIDIA GTX 1080 Ti graphics processing unit (GPU) with 11 Gigabytes of GPU memory and 3 584 GPU cores. Note that we also tested the search efficiency with an increased number of training epochs (50), but the results are comparable to the 10-epoch training. Thus, 10-epoch training is more efficient, and we will show those results only.

The attack performance of each model is evaluated by calculating the score $O(P_i)$ of the different objective functions with 2 000 attack traces. Note that the score is only calculated in the validation phase to speed up the test procedure. After 200 iterations are finished, the best hyperparameter combination is selected based on its score, and the best model is constructed following this setting. Then, to evaluate

**TABLE 1. Hyperparameter search space for MLP.**

| Hyperparameter | min | max | step |
|---|---|---|---|
| Dense (fully-connected) layers | 2 | 10 | 1 |
| Neurons (for dense or fully-connected layers) | 8 | 1 024 | 8 |
| | Options | | |
| Learning Rate | 1e-3, 5e-4, 1e-4, 5e-5, 1e-5 | | |
| Activation function (all layers) | ReLU, Tanh, ELU, or SELU | | |

**TABLE 2. Hyperparameters search space for CNNs.**

| Hyperparameter | min | max | step |
|---|---|---|---|
| Convolution layers | 1 | 4 | 1 |
| Convolution Filters | 8 | 256 | 8 |
| Convolution Kernel Size | 2 | 10 | 1 |
| Pooling Size | 2 | 5 | 1 |
| Pooling Stride | 2 | 10 | 1 |
| Dense (fully-connected) layers | 1 | 3 | 1 |
| Neurons (for dense or fully-connected layers) | 8 | 1 024 | 8 |
| | Options | | |
| Pooling Type | max pooling, avg pooling | | |
| Learning Rate | 1e-3, 5e-4, 1e-4, 5e-5, 1e-5 | | |
| Activation function (all layers) | ReLU, Tanh, ELU, or SELU | | |

its actual attack performance, this model is trained for either 10 or 50 epochs and then used to attack 5 000 traces randomly selected out of 10 000 traces (the attack is repeated ten times). As a result, guessing entropy can be calculated by averaging the key rank of each attack.

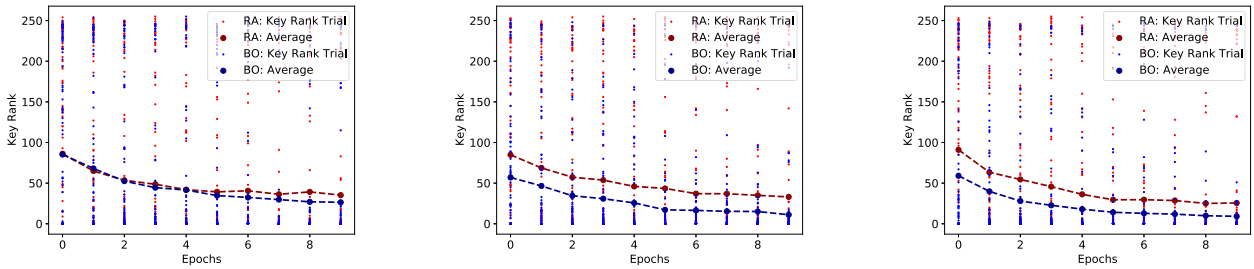
We do not add the neural network architecture size (measured by the number of trainable parameters) into our design considerations. We consider the neural network size less important than the attack performance. Additionally, the neural networks used in SCA are smaller than deep learning architectures used in other domains. Still, it is easy to extend the SCA framework to search for small neural networks that perform well in SCA.

## V. EXPERIMENTAL RESULTS

In Tables 1 and 2, we give the ranges for our search for MLP and CNN hyperparameters, respectively. We selected those ranges as a rough estimate to expect a good attack performance based on related works results. We could have selected even smaller ranges for certain settings, but that would make the search too easy for a random search and Bayesian optimization. Note that the ranges for MLP still result in a search space size of "only" 23 040 hyperparameter combinations (Table 1). On the other hand, for CNNs, the exhaustive search should evaluate 637 009 920 hyperparameter combinations (Table 2). Still, the AutoSCA framework can find models breaking the targets within hours. Although a larger search range would increase the search time, we believe the task can be completed with reasonable computation time (i.e., within a day). At the same time, the search ranges we investigated are sufficiently large to give even the latest state-of-the-art results that consider significantly larger datasets (i.e., more features in side-channel traces [32]. Recall, we randomly (RS) select a profiling model or run BO for 200 iterations to obtain a profiling model in all the experiments. The best profiling model is then trained for a certain number of epochs (10 or 50), and the test set evaluates the SCA performance. We use 50 000 traces for profiling, 2 000 for validation, and 5 000 for the attack for both versions of datasets. Besides profiling on the original dataset, we add different Gaussian noise levels to simulate a more difficult attack scenario (but also a more realistic one). The noise addition increases the search difficulty, and it could better demonstrate the performance difference between BO and RS. We note that we conducted experiments on one more dataset commonly used in the SCA domain (usually denoted as CHES CTF), but we do not show results due to the lack of space. Still, the obtained results align with the observations for the ASCAD datasets. For objective functions, we consider validation accuracy, validation key rank, and $L_m$. Accuracy and $L_m$ are maximized, while key rank is minimized. To increase the readability of tables, we present the results for the smallest architectures in italic style, while the best-performing ones are in bold style.

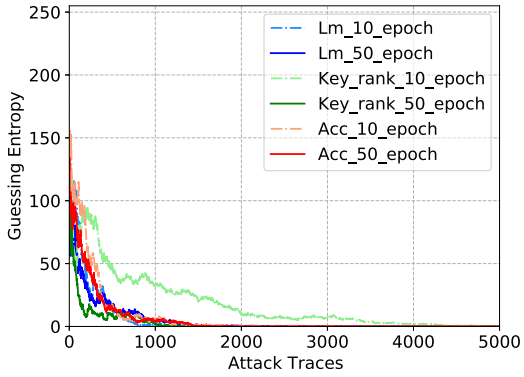
### A. ASCAD WITH THE FIXED KEY

First, in Figure 2, we depict the results for three objective functions (accuracy, key rank, and $L_m$), where we compare

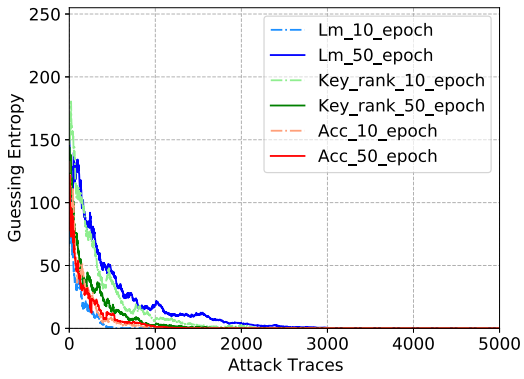



(a) Objective: accuracy. (b) Objective: key rank. (c) Objective: $L_m$.

**FIGURE 2. Search results for MLP with the HW leakage model on ASCAD with fixed key with no noise added.**

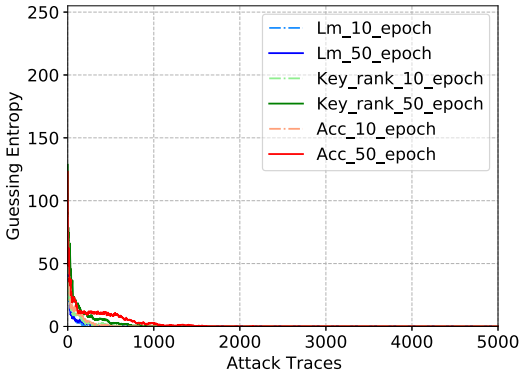



(a) Bayesian optimization.

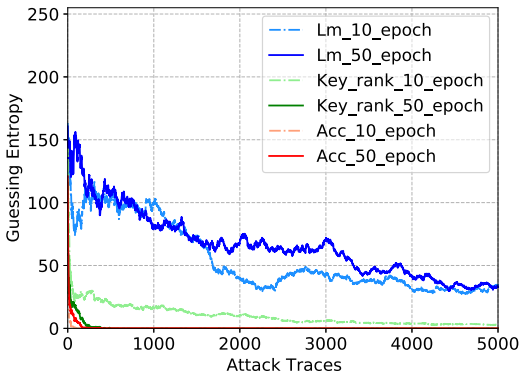



(b) Random search.

**FIGURE 3. The GE comparison with the best MLP models obtained by two search methods with the HW leakage model on ASCAD with the fixed key.**




(a) Bayesian optimization.




(b) Random search.

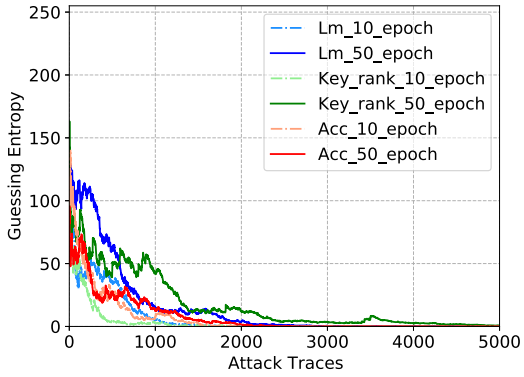
**FIGURE 4. The GE comparison with the best MLP models obtained by two search methods with ID leakage model on ASCAD with a fixed key.**

their performance for random search (RS) and Bayesian optimization (BO) when tuning MLP profiling models. Based on these results, one can decide what objective function is appropriate for the specific setting. Note that we do not depict more results for the objective functions due to the lack of space, but we discuss the results in the text. The profiling models are trained with ten epochs with the Hamming weight leakage model. We see that the key rank decreases regardless of the objective function. The performance of key rank and $L_m$ is better than accuracy, and for all three objective functions, BO converges faster than RS.

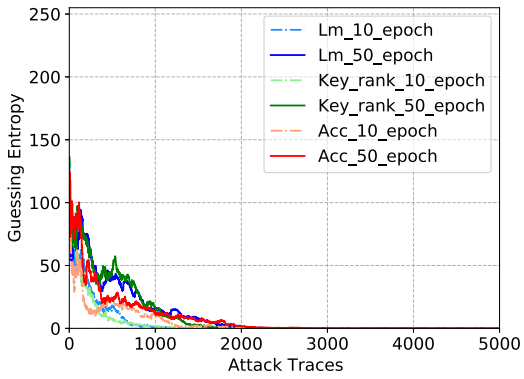
Based on the results from the best RS and BO profiling models, we show the guessing entropy results for several settings (Figure 3). As mentioned, we consider three objectives and two training durations (10 or 50 epochs), resulting in six settings. When using BO, $L_m$ with ten epochs works the best, as shown in Figure 2(a). This indicates that BO can find profiling models that generalize well. What is more, the best setting reaches GE of 1 for around 800 attack traces. The same observation also holds for RS, as shown in Figure 2(c) where $L_m$ objective manages to reach GE equal to 1 for around 400 attack traces. At the same time, the accuracy objective function with RS requires more traces to reach GE of 1. Interestingly, our results show very strong attack performance with ten epochs already, which is somewhat differing from related works where it is common to train for significantly more epochs [7], [11], [13]. Finally, we observe that multiple profiling models perform well, confirming that the ASCAD dataset with the fixed key is easy to attack.

For the ID leakage model, the results align with the HW leakage model results, and BO performs better for all three objective functions. The results for the attack dataset are shown in Figure 4, where the profiling model selected by BO performs, on average, significantly better than the one from RS. When training with ten epochs, the best model from BO requires around 191 attack traces, while the best model for RS requires only around 67 attack traces. Note that the best result from RS depends on chance, while BO obtains well-performing models consistently. Both results indicate (significantly) better attack performance than reported in state-of-the-art [7], [8]. The good results for random search indicate this dataset is easy to break, and we do not (necessarily) require any special methodologies to succeed in the attack.

Next, we show the results when optimizing CNN hyperparameters. The results for optimizing different objectives for CNN and the HW leakage model are significantly worse than MLP as now, the search space size is more than 27 000 times larger. Still, accuracy and $L_m$ reach a significantly

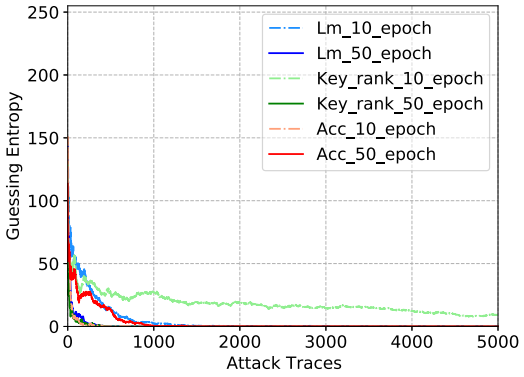



(a) Bayesian optimization.

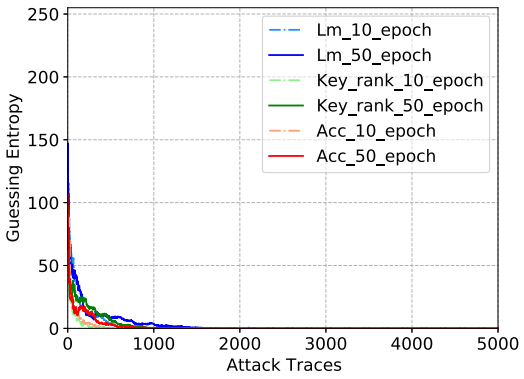



(b) Random search.

**FIGURE 5. The GE comparison with the best CNN models obtained by two search methods with the HW leakage model on ASCAD with the fixed key.**




(a) Bayesian optimization.




(b) Random search.

**FIGURE 6. The GE comparison with the best CNN models obtained by two search methods with the ID leakage model on ASCAD with the fixed key.**

better key rank with BO. The guessing entropy results depicted in Figure 5 show good performance, where around 1 100 attack traces are enough for most of the settings to reach a guessing entropy of 1. The best performing result is obtained with RS, where we need only 965 traces to break the target. Nevertheless, BO has a higher probability of finding good models as it converges faster than RS.

For the ID leakage model, BO performs better than RS with key rank and $L_m$ objectives. The best results are for BO and accuracy as the objective metric (155 traces to break the target). Class imbalance does not pose a problem when using the ID leakage model, and thus, accuracy is more stable. Considering the GE results in Figure 6, both models from BO and RS converge well: 500 traces on average are sufficient to break the target.

The obtained best architectures are retrained to validate their attack performance. Due to the random weight initialization, the attack performance may differ from the GE plots discussed before. In Tables 3 and 4, we compare the results for several architectures for both leakage models. Note that [7] provides results for the ID leakage model only, so we adapted the neural network models to work for the HW leakage model by changing the number of output classes. Thus, the adapted model is not an optimal architecture from [7], but it represents the best option for the comparison. We consider complexity (the number of trainable parameters) and the number of traces needed to reach GE of 1. To evaluate the attack performance of the obtained models, we train the model with 10 and 50 epochs separately, the corresponding GE are listed in the Tables (separated with the "/" symbol).

For the HW leakage model, both AutoSCA MLP and AutoSCA CNN reach top performance. Specifically, 447 traces are required to break the target for AutoSCA MLP with 10-epoch training, which is more than two times less than for [7]. Compared with the results obtained with reinforcement learning, we observe that AutoSCA produces neural networks with more trainable parameters, but they perform better. Note that more than a million trainable parameters for both models were obtained with AutoSCA (while those from related works are significantly smaller).

**TABLE 3. Comparison of performance on ASCAD with the fixed key and the HW leakage model.**

| | [7] | [15] | AutoSCA MLP | AutoSCA CNN |
|---|---|---|---|---|
| Complexity | 14 235 | *5 566* | 1 388 457 | 1 086 801 |
| Traces to reach $GE = 1$ | 1 246 | 906 | **447**/1 224 | 539/4 136 |

*Complexity denotes the number of trainable parameters.*

**TABLE 4. Comparison of performance on ASCAD with the fixed key and the ID leakage model.**

| | [13] | [7] | [15] | AutoSCA MLP | AutoSCA CNN |
|---|---|---|---|---|---|
| Complexity | 66 652 444 | *16 960* | 79 439 | 1 544 776 | 3 510 424 |
| Traces to reach $GE = 1$ | 1 476 | 191 | 202 | **120**/430 | 257/690 |

*Complexity denotes the number of trainable parameters.*

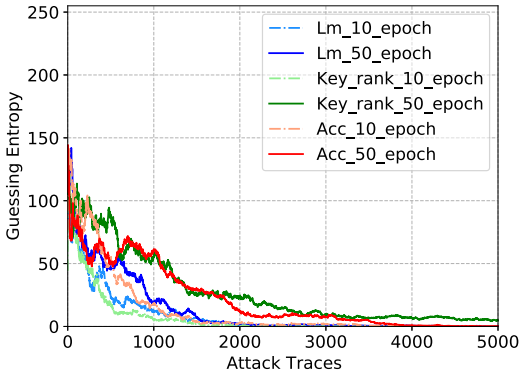
**TABLE 5. Comparison of performance of BO and RS with the addition of different noise levels (two and four) - ASCAD with the fixed key, both leakage models.**

| | Accuracy | Key rank | $L_m$ |
|---|---|---|---|
| $MLP_{HW}$ | RS/- | BO/BO | BO/BO |
| $MLP_{ID}$ | -/BO | BO/BO | RS/BO |
| $CNN_{HW}$ | RS/BO | BO/BO | BO/BO |
| $CNN_{ID}$ | BO/- | BO/BO | RS/RS |

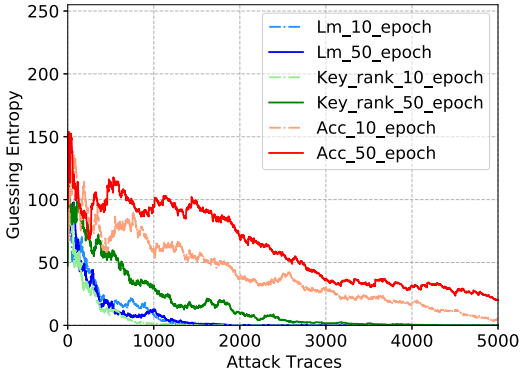
However, 10-epoch training is enough for a model to retrieve the secret key, thus efficiently erasing the time complexity. For the ID leakage model, Benadjila et al. [13] consider a significantly larger neural network, as evident through the training time and the number of trainable parameters. On the other hand, the architecture from Zaid et al. [7] is the smallest. However, it takes more time than AutoSCA MLP (MLP is simpler to train) and AutoSCA CNN when training with ten epochs. Indeed, the obtained best model outperforms state-of-the-art models from the literature for both HW and ID leakages models.

It is worth noting that 10-epoch training for the best AutoSCA models always performs better than 50-epoch training. This is because the models are trained and evaluated with 10-epoch training during the search process. As a result, the search algorithm selects the models with greater learning ability, as they could reach higher scores when training with ten epochs. Note that [7] and [15] reach very similar attack performance, but compared to BO, their performance is worse (around three times more traces are required to break the target in the best case scenario). Finally, we reach better performance with 10-epoch training for both leakage models, indicating that longer training causes overfitting and that it is possible to have a short training phase that results in top attack performance.

Next, we add Gaussian noise with a standard deviation of two and four to the dataset to investigate the hyperparameter tuning difficulty when dealing with more complex datasets. A brief overview is shown in Table 5. The averaged final GE at the tenth training epoch is used to compare BO and RS. If one search method is better than the other for a certain leakage model and objective function, the better search method (BO or RS) is noted in the table's corresponding position. If their key-rank difference is within five (thus, no significant performance difference), a sign '-' is added. Table 5 includes the comparison for two noise level ($noise_2$/$noise_4$). From the




(a) Bayesian optimization.




(b) Random search.

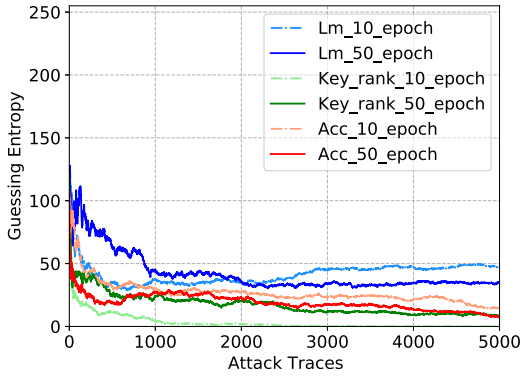
**FIGURE 7. The GE comparison with the best MLP models obtained by two search methods with the HW leakage model on ASCAD with random keys.**

results, when we exclude the cases where BO and RS are comparable, BO outperforms RS in 16 out of 21 cases, again indicating BO's superiority in hyperparameter tuning. Regarding the key rank difference, the performance variation between BO and RS increases with more noise added to the traces, indicating BO's capability to find strong models when training with more difficult datasets.

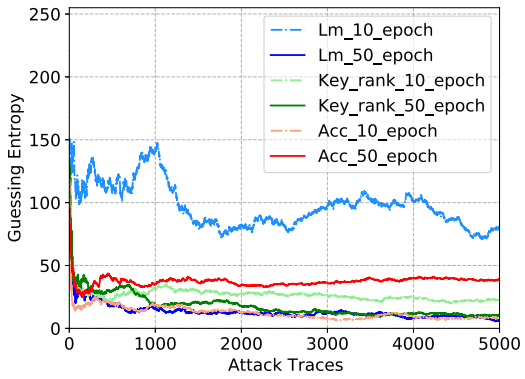
### B. ASCAD WITH RANDOM KEYS

For the HW leakage model and MLP, BO performances with all three objectives are slightly better than for RS and in line with the results in the previous section. The guessing entropy results are shown in Figure 7. Observe that the $L_m$ results are, in general, the best for both RS and BO. The best model is obtained for BO with the key rank objective and ten epochs: only around 600 traces are required to reach GE equal to 1. As this dataset is more difficult to attack than the ASCAD dataset with a fixed key, MLP with RS has more issues reaching top performance, and BO should be already considered a preferable option for hyperparameter tuning.

Next, we consider the ID leakage model and MLP for the ASCAD dataset with random keys. Note that there are more labels in this leakage model (256 classes), and the dataset is more difficult than ASCAD with a fixed key. The results

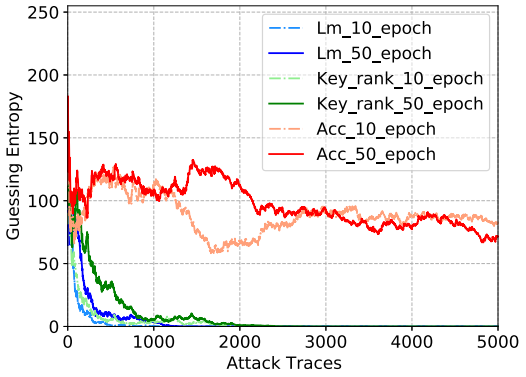



(a) Bayesian optimization.

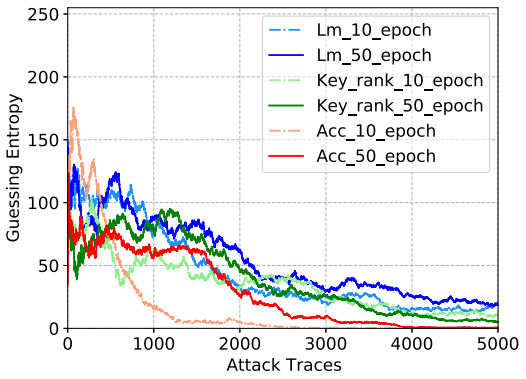



(b) Random search.

**FIGURE 8. The GE comparison with the best MLP models obtained by two search methods with the ID leakage model on ASCAD with random keys.**




(a) Bayesian optimization.




(b) Random search.

**FIGURE 9. The GE comparison with the best CNN models obtained by two search methods with the HW leakage model on ASCAD with random keys.**

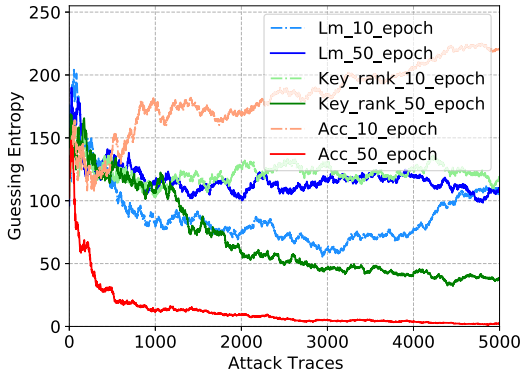
indicate that BO performs significantly better than RS with the key rank objective. Figure 8 shows corresponding GE results, where BO with key rank can break the target with 3 481 traces with 10-epoch training.

The obtained results suggest that accuracy is similar to the HW leakage model, while the key rank and $L_m$ objectives are somewhat better. Translating these into the attack performance, we show guessing entropy in Figure 9. Again, the profiling model selected by BO converges faster than RS in general, as the best-performing profiling model requires only 496 traces to break the target.

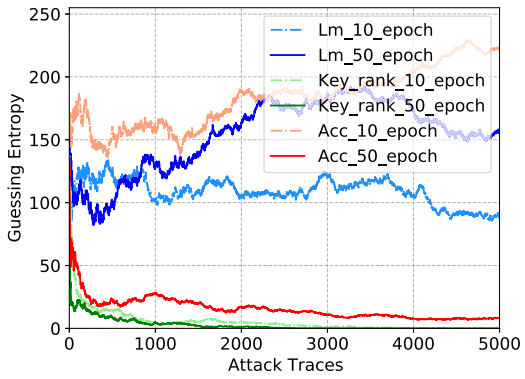
For the ID leakage model and CNN, all three objectives struggle to reach good performance, suggesting that our profiling models will have problems with generalization. Such intuition is confirmed in Figure 10, where we display the GE results. Here, RS works significantly better as it reaches GE of 1 for around 1 500 attack traces (key rank and 50 epochs). For BO, no results suggest we can break the target. We believe this happens as the search space is very large, and BO probably needs significantly more iterations to exhibit good performance.

Next, in Tables 6 and 7, we retrain and compare the results for several architectures for both leakage model. Again, the model complexity and the number of traces needed to reach GE of 1 are considered. For the HW leakage model, the attack performance of AutoSCA CNN is comparable with the model listed in [14]. Interestingly, the authors in [14] used an ensemble of neural networks to reach such an attack performance. On the other hand, we managed to find a single profiling model that performs similarly. Compared with the model listed in [15], our best models have more trainable parameters, but we argue that the model's attack performance should be prioritized when selecting the models. For the HW leakage model, the similar GE performance between [15] and the MLP models obtained in this work indicate that with a good hyperparameter tuning, MLP can represent a viable option even compared to CNNs. The AutoSCA CNN model performs significantly better than [15] for the HW leakage model. On the other hand, both benchmark models perform significantly better for the ID leakage model than those obtained with AutoSCA. One possible reason could be that the search space is not fully explored by the search algorithm, where more iterations may lead to better attack models. Also, additional training effort may be required to learn from this dataset with the ID leakage model, as a shorter training time (10 epochs) gives significantly worse results than 50 epochs.

Finally, we add Gaussian noise with standard deviations of two and four to the dataset, with a brief conclusion shown in




(a) Bayesian optimization.




(b) Random search.

**FIGURE 10. The GE comparison with the best CNN models obtained by two search methods with the ID leakage model on ASCAD with random keys.**

Table 8. In line with the tests on ASCAD with a fixed key, 15 out of 19 cases indicate that BO performs better than RS. With the increasing noise level added to the traces, the performance difference between BO and RS becomes larger, as observed from the key rank difference, indicating BO's ability to cope with more difficult datasets.

## VI CONCLUSIONS AND FUTURE WORK

This article proposed Bayesian optimization for the deep learning-based SCA hyperparameter tuning. We develop a custom framework that supports both machine learning and side-channel metrics, and we evaluate the performance of such obtained profiling models with random search and state-of-the-art results. We can observe that BO works well and produces a large number of highly-fit profiling models. This indicates that BO should be the first choice when running deep learning-based SCA, especially when the evaluator is more restricted concerning the number of measurements and wants to search for the strongest possible profiling model. We also see that random search can find excellent profiling models, especially for simpler datasets. Still, random search results need to be considered from a proper perspective as we pre-select some "reasonable" ranges. Extending the ranges makes the problem more difficult for a random search. Thus, there is a trade-off between the hyperparameter tuning complexity and the assumptions on the architectures one makes.

It is particularly interesting to observe that BO results can outperform the results obtained through a methodology approach [7] or reinforcement learning [15]. Considering that [15] reports on average 100 hours to perform a single experiment, Bayesian optimization requires on average 10× less time while having similar attack performance. However, one should keep in mind that the tuning complexity is positively correlated with the range of the search space. Indeed, a too large search range would increase the tuning complexity and more likely constrain the size of the generated models due to the limitation of the computation devices (i.e., GPU memory). The complexity will be governed by the number of models generated and their size (concerning the number of trainable parameters). Finally, we plan to extend our analysis to different types of Bayesian optimization in future work. We considered one surrogate model (Gaussian Process) and one acquisition function (upper confidence bound) in this work. While those choices are common options, further investigation should be done to judge specific design choices' merits.

**TABLE 6. Comparison of performance on ASCAD with random keys with the HW leakage model.**

| | [14] | [15] | AutoSCA MLP | AutoSCA CNN |
|---|---|---|---|---|
| Complexity | N/A | *15 241* | 1 783 425 | 4 128 753 |
| Traces to reach $GE = 1$ | **470** | 911 | 617/818 | 496/1 112 |

*Complexity denotes the number of trainable parameters.*

**TABLE 7. Comparison of performance on ASCAD with random keys with the ID leakage model.**

| | [14] | [15] | AutoSCA MLP | AutoSCA CNN |
|---|---|---|---|---|
| Complexity | N/A | *70 492* | 1 699 744 | 1 539 320 |
| Traces to reach $GE = 1$ | **105** | 490 | 3 481/1 596 | 2 945/1 568 |

**TABLE 8. Comparison of performance of BO and RS with the addition of different noise levels (two and four).**

| | Accuracy | Key rank | $L_m$ |
|---|---|---|---|
| $MLP_{HW}$ | RS/BO | RS/BO | BO/- |
| $MLP_{ID}$ | BO/- | BO/BO | BO/BO |
| $CNN_{HW}$ | -/RS | -/BO | BO/BO |
| $CNN_{ID}$ | -/BO | BO/BO | RS/RS |

*Complexity denotes the number of trainable parameters.*

## REFERENCES

[1] S. Mangard, E. Oswald, and T. Popp, *Power Analysis Attacks: Revealing the Secrets of Smart Cards*, Berlin, Germany: Springer, 2006.
[2] P. C. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," in *Proc. Adv. Cryptology 19th Annu. Int. Cryptology Conf.*, 1999, pp. 388–397.
[3] S. Chari, J. R. Rao, and P. Rohatgi, "Template attacks," in *Proc. Cryptographic Hardware Embedded Syst. 4th Int. Workshop Redwood Shores*, 2002, pp. 13–28.

[4] G. Hospodar, B. Gierlichs, E. D. Mulder, I. Verbauwhede, and J. Vandewalle, "Machine learning in side-channel analysis: A first study," *J. Cryptographic Eng.*, vol. 1, no. 4, pp. 293–302, 2011.

[5] L. Lerman, R. Poussier, G. Bontempi, O. Markowitch, and F.-X. Standaert, "Template attacks versus machine learning revisited (and the curse of dimensionality in side-channel analysis)," in *Proc. Int. Workshop Constructive Side-Channel Anal. Secure Des.*, 2015, pp. 20–33.

[6] H. Maghrebi, T. Portigliatti, and E. Prouff, "Breaking cryptographic implementations using deep learning techniques," in *Proc. Int. Conf. Secur. Privacy Appl. Cryptography Eng.*, 2016, pp. 3–26.

[7] G. Zaid, L. Bossuet, A. Habrard, and A. Venelli, "Methodology for efficient CNN architectures in profiling attacks," *IACR Trans. Cryptographic Hardware Embedded Syst.*, vol. 2020, no. 1, pp. 1–36, Nov. 2019.

[8] L. Wouters, V. Arribas, B. Gierlichs, and B. Preneel, "Revisiting a methodology for efficient CNN architectures in profiling attacks," *IACR Trans. Cryptographic Hardware Embedded Syst.*, vol. 2020, no. 3, pp. 147–168, Jun. 2020.

[9] C. Archambeau, E. Peeters, F. X. Standaert, and J. J. Quisquater, "Template attacks in principal subspaces," in *Cryptographic Hardware and Embedded Systems*, L. Goubin and M. Matsui Eds., Berlin, Germany: Springer, 2006, pp. 1–14.

[10] F.-X. Standaert and C. Archambeau, "Using subspace-based template attacks to compare and combine power and electromagnetic information leakages," in *Proc. Int. Workshop Cryptographic Hardware Embedded Syst.*, 2008, pp. 411–425.

[11] J. Kim, S. Picek, A. Heuser, S. Bhasin, and A. Hanjalic, "Make some noise. unleashing the power of convolutional neural networks for profiled side-channel analysis," *IACR Trans. Cryptographic Hardware Embedded Syst.*, vol. 3, pp. 148–179, 2019.

[12] S. Picek, A. Heuser, G. Perin, and S. Guilley, "Profiling side-channel analysis in the efficient attacker framework," Cryptology ePrint Archive, Report 2019/168, 2019.

[13] R. Benadjila, E. Prouff, R. Strullu, E. Cagli, and C. Dumas, "Deep learning for side-channel analysis and introduction to ASCAD database," *J. Cryptographic Eng.*, vol. 10, no. 2, pp. 163–188, 2020.

[14] G. Perin, L. Chmielewski, and S. Picek, "Strength in numbers: Improving generalization with ensembles in machine learning-based profiled side-channel analysis," *IACR Trans. Cryptographic Hardware Embedded Syst.*, vol. 2020, no. 4, pp. 337–364, Aug. 2020.

[15] J. Rijsdijk, L. Wu, G. Perin, and S. Picek, "Reinforcement learning for hyperparameter tuning in deep learning-based side-channel analysis," *IACR Trans. Cryptographic Hardware Embedded Syst.*, vol. 2021, no. 3, pp. 677–707, Jul. 2021.

[16] T. O'Malley et al., "Keras Tuner," 2019. https://github.com/keras-team/keras-tuner

[17] L. Wu et al., "On the attack evaluation and the generalization ability in profiling side-channel analysis," Cryptology ePrint Archive, Report 2020/899, 2020.

[18] F.-X. Standaert, T. G. Malkin, and M. Yung, "A unified framework for the analysis of side-channel key recovery attacks," in *Advances in Cryptology - EUROCRYPT 2009*, A. Joux Ed., Berlin, Germany: Springer, 2009, pp. 443–461.

[19] M. Pelikan et al., "BOA: The bayesian optimization algorithm," in *Proc. Genet. Evol. Comput. Conf.*, 1999, pp. 525–532.

[20] J. Snoek, H. Larochelle, and R. P. Adams, "Practical Bayesian optimization of machine learning algorithms," in *Proc. Adv. Neural Inf. Process. Syst.*, 2012, pp. 2951–2959.

[21] H. Jin, Q. Song, and X. Hu, "Auto-keras: An efficient neural architecture search system," in *Proc. 25th ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2019, pp. 1946–1956.

[22] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time analysis of the multiarmed bandit problem," *Mach. Learn.*, vol. 47, no. 2/3, pp. 235–256, May 2002.

[23] P. I. Frazier, "A tutorial on Bayesian optimization," 2018, *arXiv:1807.02811*.

[24] L. Lerman, S. F. Medeiros, G. Bontempi, and O. Markowitch, "A machine learning approach against a masked AES," in *CARDIS, Lecture Notes in Computer Science*, Berlin, Germany: Springer, 2013.

[25] A. Heuser and M. Zohner, "Intelligent machine homicide - breaking cryptographic devices using support vector machines," in *COSADE, ser. LNCS*, W. Schindler and S. A. Huss Eds., vol. 7275, Berlin, Germany: Springer, 2012, pp. 249–264.

[26] O. Choudary and M. G. Kuhn, "Efficient template attacks," in *Smart Card Research and Advanced Applications*, A. Francillon and P. Rohatgi Eds., Berlin, Germany: Springer, 2014, pp. 253–270.

[27] E. Cagli, C. Dumas, and E. Prouff, "Convolutional neural networks with data augmentation against jitter-based countermeasures," in *Cryptographic Hardware and Embedded Systems – CHES 2017*, W. Fischer and N. Homma Eds., Berlin, Germany: Springer, 2017, pp. 45–68.

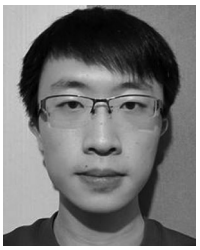
[28] Z. Martinasek, J. Hajny, and L. Malina, "Optimization of power analysis using neural network," in *Smart Card Research and Advanced Applications*, A. Francillon and P. Rohatgi Eds., Berlin, Germany: Springer, 2014, pp. 94–107.

[29] S. Yang, Y. Zhou, J. Liu, and D. Chen, "Back propagation neural network based leakage characterization for practical security analysis of cryptographic implementations," in *Information Security and Cryptology*, H. Kim, Ed, Berlin, Germany: Springer, 2012, pp. 169–185.

[30] L. Weissbart, "Performance analysis of multilayer perceptron in profiling side-channel analysis," in *Proc. Appl. Cryptography Netw. Secur. Workshops*, 2020, pp. 198–216.

[31] H. Li, M. Krček, and G. Perin, "A comparison of weight initializers in deep learning-based side-channel analysis," in *Proc. Appl. Cryptography Netw. Secur. Workshops*, 2020, pp. 126–143.

[32] G. Perin, L. Wu, and S. Picek, "Exploring feature selection scenarios for deep learning-based side-channel analysis," Cryptology ePrint Archive, Report 2021/1414, 2021.



**LICHAO WU** received the bachelor's degree with Northwestern Polytechnical University, in 2015 and the master's degree in microelectronic with the Delft University of Technology, in 2017. He is currently working toward the PhD degree in the cybersecurity research group with the Delft University of Technology. His main research interests are at the intersection of implementation attacks, cryptography, and machine learning.

**GUILHERME PERIN** received the graduated degree in electrical engineering from the Federal University of Santa Maria, in 2008, the master's degree in Informatics from the Federal University of Santa Maria, in 2011, and the PhD degree in microelectronics and automated systems with the University of Montpellier, in 2014. He is a postdoctoral researcher with the Delft University of Technology. His research areas include hardware security, cryptography, optimization algorithms, and machine learning.

**STJEPAN PICEK** received the PhD degree, in 2015, and from 2015 to 2017. He is an associate professor with Radboud University, The Netherlands. He was a postdoctoral researcher with KU Leuven, Belgium and MIT, USA. From 2017 to 2021, he was an assistant professor at the Delft University of Technology, The Netherlands. His research interests include cryptography, machine learning, and evolutionary algorithms.