Train Wisely: Multifidelity Bayesian Optimization Hyperparameter Tuning in Deep Learning-based Side-Channel Analysis

Trevor Yap[1,2][0000−0001−8651−574X], Shivam Bhasin[2][0000−0002−6903−5127], and Léo Weissbart[3][0000−0003−0288−9686]

[1] School of Physical and Mathematical Sciences, Nanyang Technological University, Singapore
[2] Temasek Laboratories, Nanyang Technological University, Singapore
trevor.yap, sbhasin@ntu.edu.sg
[3] Radboud University, Nijmegen, Netherlands
l.weissbart@cs.ru.nl

Abstract. Side-Channel Analysis (SCA) is critical in evaluating the security of cryptographic implementations. In recent years, the use of Deep Neural Networks (DNNs) in SCA has risen in popularity. However, DNNs consists of many hyperparameters and not every configuration of hyperparameters result in successful attack. Therefore, the search for DNN's hyperparameters poses a significant challenge, especially when resources are limited. In this work, we explore the efficacy of a multifidelity optimization technique known as Bayesian Optimization Hyper-Band (BOHB) in SCA. This introduces the notion of budget within SCA to encapsulate the idea of resources when tuning the hyperparmeters of the DNNs. Next, we proposed a new objective function called ge_{+ntge}, which could be incorporated into any Bayesian Optimization used in SCA. We show the capabilities of both BOHB and ge_{+ntge} on four different public datasets. Specifically, BOHB could obtain the least number of traces in the dataset called CTF2018 when trained in the Hamming weight and identity leakage models. Notably, this marks the first reported successful recovery of the key for the identity leakage model in CTF2018.

Keywords: Side-channel · Neural Network · Deep Learning · Profiling attack · Hyperparameter Search.

1 Introduction

Deep learning-based SCA have found great success when applying DNNs to analyze traces of physical leakages such as power consumption [13] or electromagnetic emanation [2]. These DNNs can retrieve the secret key after analyzing the traces given. It has been shown by previous works [16, 3] that using DNNs could recover the secret key with fewer traces compared to classical side-channel analysis such as Template Attack [6]. Furthermore, that even in the presence of countermeasures like masking and desynchronization, the DNN can still recover the secret key with little to no preprocessing needed [3, 5]. However, DNNs consist of many hyperparameters. Since not every configuration of hyperparameters will result in a successful attack, one has to tune these hyperparameters to find a well-performing DNN to retrieve the secret key.

Hyperparameters Optimization (HPO) search and Neural Architecture Search (NAS) for DNNs have become essential in finding the best-performing attack that a DNN could mount. This has become a crucial part of evaluating the security of a cryptographic implementation in evaluation labs. Many works have considered tackling the problem of finding a well-performing DNN. In the first work by Zaid et al. [29], authors provide some guidelines to manually create well-performing DNNs. [23] and [30] further discuss on these guidelines and provide a more precise methodology that helps to generate smaller and well-performing DNNs. On the other hand, various automated tools for SCA have been explored to tackle this issue like Bayesian Optimization [25], reinforcement learning [21] and evolutionary algorithm [1]. Although these techniques obtained favorable results, most techniques are slow and could run for days. Due to the large number of IT products to be evaluated, evaluating the security of these products in evaluation labs becomes very time-sensitive. An evaluator will naturally set a budget for any resources like the time needed to quantify the security of the primitive tested. Therefore, resources such as time are valuable assets to determine a device's security and a natural question arises:
Are there automated tools available can produce comparable results while allocating resources more efficiently?

Multifidelity optimization methods allow speed up in the optimization process by allocating more resources to promising configurations and stopping evaluations of poorly performing ones early. In this work, we explore a multifidelity optimization method known as Bayesian Optimization HyperBand (BOHB), which uses both Bayesian Optimization and HyperBand algorithms to search for hyperparameters.

Our Contributions. More precisely, the contributions of the paper are as follows:

1. We apply the multifidelity optimization method, BOHB, and show its effectiveness as an automated tool. We demonstrate that BOHB can recover the secret key for all four public datasets tested, while previous works tested only a few of these public datasets.
2. BOHB is shown to obtain the least number of traces for the public datasets called CTF2018 in the Hamming weight leakage model compared to other state-of-the-art methodologies. BOHB uses 82 attack traces to recover the key. Further, BOHB recovers the secret key of CTF2018 in the identity leakage model. This marks the first reported instance of a successful attack using an automated tool against CTF2018 in an identity leakage setting.
3. We propose a new objective function known as ge_{+ntge} to incorporate the computation of *number of attack traces needed for key recovery* into the objective function for the multifidelity optimization method. We show that

this objective function obtains better results compared to the other objective functions tested.

In this work, we target synchronized and desynchronized traces. We validate our approach on traces up to the first-order masking. We leave higher-order masking to future works. The results can be publicly accessed on the following weblink[4].

Paper Organization. The paper is organized as follows. Section 2 will summarize related works. In Section 3 provides the background necessary for SCA, DNN, and BOHB. We will recall previously used objective functions and present a new objective function ge_{+ntge} in Section 4. The experimental settings and datasets used will be shown in Section 5. We present the results using BOHB in Section 6. Lastly, we will conclude the paper and provide future works in Section 7.

2 Related Works

Both Multiayer Perceptrons (MLPs) and Convolutional Neural Networks (CNNs) were shown to outperform classical SCA attacks like Template attacks and other machine learning techniques in [16], resulting in the rise of popularity of DNN in SCA. In fact, [5] and [20] show that such CNNs can attain strong results even in the presence of countermeasures. But this also introduced a large number of hyperparameters to tune (e.g., the number of layers, kernel size, type of activation functions, etc.) compared to other machine learning or classical SCA. Furthermore, Maghrebi et al. have pointed out that the performance of DNNs is greatly influenced by their hyperparameters. This pushes for the need for methodologies to find good hyperparameters in the domain of SCA. However, finding and tuning hyperparameters is not an easy task, especially when resources are limited.

Various works have explored the influences of specific hyperparameters in the performance of the DNN within SCA. [22] examines how the number of layers and neurons and the type of activation function within a MLP would affect the performance. [14] explores the usage of weight initialization in SCA and found that weight initializers play a significant role in the training process, especially on datasets that are difficult to break. [24] investigates how the type of pooling layers affects the performance. In addition, new loss functions are customized for SCA [28, 31, 11], which are mostly evaluated in [12]. Introducing these new loss functions has also expanded the range of hyperparameters to tune in SCA.

Zaid et al. [29] proposed the first methodology to build CNN manually. The methodology gives some intuition of what the DNNs are looking at within the trace, providing some explainability of the neural network. [23] and [30] further examine these guidelines and improve on the methodology. For example, it has been shown by [23] that adding standardization on the traces could remove the first layer of kernel size 1 proposed by [29]. [10] further investigated the use of

[4] https://github.com/yap231995/BOHB-SCA.git

ResNets in SCA and provided methodology when constructing one by hand. However, manually tuning the hyperparameters could take a lot of time to find a well-performing DNN. Therefore, various studies considered using automated tools to search for well-performing DNNs.

Among the automated tools, the hyperparameters optimization and neural architecture search are two approaches explored in SCA. HPO fixed the architectures of the DNN, like MLP or CNN. The HPO problem is solved by searching for the best-performing DNN through a range of hyperparameters. Allowing one to define the range of hyperparameters of fixed architectures could enable human expertise to be included. On the contrary, to solve the NAS problem, a predefined architecture may not be given; instead, techniques proposed usually specifies certain building blocks and constructs a DNN based on them. [21] tackles the HPO problem by using reinforcement learning to find small CNN capable of key recovery. On the other hand, InfoNeat, developed by [1], focuses on the NAS problem to build DNNs based on information theoretic criteria as stopping criteria for each configuration. Although these automatic methods attain favorable results, they are slow and could run for more than a day. Wu et al. [25] consider the Bayesian Optimization based on Gaussian Processes to search for the best-performing DNN addressing the HPO problem. They proposed a framework called AutoSCA, where they train a DNN using 10 epochs for a given configuration or a set of hyperparameters. This trained DNN is then used within Bayesian Optimization to pick the next configuration. Although this framework could finish a run within a day, we observe a gap where 10 epochs may not be enough to determine if the configuration is good enough, as more epochs could be used to obtain better results on those more promising hyperparameters. Our aim is to propose the use of multifidelity optimization to allocate resources more efficiently. This will provide evaluators with another tool to use when resources like time are scarce.

3 Background

3.1 Profiling Attacks

One of the most common side-channel settings is known as the profiling attack. It assumes the worst-case scenario where the adversary has access to a clone device similar to the target device. The profiling attack is executed in two phases: profiling and attack phase. In the profiling phase, the adversary either knows or can manipulate the key of the clone device. Then, a distinguisher $\mathcal{F}$ can be built from the profiling traces of a known set of random public variables (plaintexts or ciphertexts) obtained from the clone device. In the second phase, the adversary performs the attack by collecting several attack traces from another set of known public variables of the target device. The trained distinguisher is applied to output a probability score for each hypothetical sensitive value $\boldsymbol{y}_i = \mathcal{F}(\boldsymbol{t}_i)$ for each attack traces $\boldsymbol{t}_i$ acquired from the target device. The log-likelihood score for each key $k \in \mathcal{K}$ is then computed:

$$s_{N_a}(k) = \sum_{i=1}^{N_a} \log(\boldsymbol{y}_i[z_{i,k}])$$

where N_a as the number of attack traces used and $z_{i,k} = C(p_i, k)$ are the hypothetical sensitive values based on the key k with p_i being the corresponding public variable to the trace $\boldsymbol{t}_i$. Here, C is the leakage model based on the cryptographic implementation. An example of C is the Hamming weight of the AES Substitution Box (Sbox), $HW(Sbox_{AES}(p_i \oplus k))$. The log-likelihood score is sorted in decreasing order: $\boldsymbol{G} = \{G_0, G_1, \ldots, G_{|\mathcal{K}|-1}\}$ with G_0 being the log-likelihood score for the most likely key candidate and $G_{|\mathcal{K}|-1}$ is the log-likelihood score for the least likely key candidate. The index of $\boldsymbol{G}$ is the rank of the key. We define the guessing entropy GE to be the average rank of the secret key over multiple experiments. In this work, we average over 100 experiments. If $GE = 0$, then the attack is successful. We also denote $NTGE$ as the least number of attack traces required to obtain $GE = 0$.

We train a DNN, $f_{\boldsymbol{\theta}}$, as the distinguisher where $\mathcal{F} = f_{\boldsymbol{\theta}}$ with $\boldsymbol{\theta}$ as the given hyperparameters. In this paper, we will explore two very commonly used architectures known as MLP and CNN.

3.2 Successive Halving, HyperBand and BOHB

The performance of a model in machine learning can be described as a function $f : \Theta \to \mathbb{R}$ with $\boldsymbol{\theta} \in \Theta$ as their hyperparameters. We note here that Θ is a predefined space that an expert with prior knowledge could consider to manipulate. We call the performance of a model f the objective function. The HPO problem can be defined as searching for $\boldsymbol{\theta}^*$ such that it satisfies

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta} \in \Theta}{\operatorname{argmin}}\, f(\boldsymbol{\theta}). \tag{1}$$

Since resources are limited, evaluators are often required to set budgets to evaluate a primitive. In the following, we shall present BOHB and the algorithms that are building blocks for BOHB. These algorithms incorporate budgets into their computation.

SuccessiveHalving [9] is one such hyperparameter optimization algorithm that employs the multi-armed bandit strategy. Given a budget b and a set of configurations, SuccessiveHalving evaluated the configurations' performance based on the budget b. Then it continues to evaluate the performance of the top η^{-1} configurations on a η times larger budget until the maximum budget is attained. [15] recommended taking $\eta = 3$ in practice. We illustrate how one full run of SuccessiveHalving is depicted in Figure 1. The evaluation of the top models with larger budgets helps to allocate the resources much more efficiently to more promising models.

Despite the efficient allocation within SuccessiveHalving, there is a trade-off in terms of the number of configurations to initialize and the initial budget to be

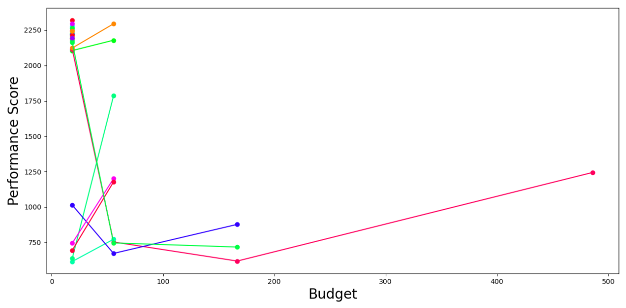

Fig. 1: Visualization representation of one full run of SuccessiveHalving with $b_0 = 18$ and $b_{max} = 486$. In the first iteration, 27 models are evaluated with budget of 18 each. In the second iteration, the top 9 models are rerun with a larger budget of 54 each. Subsequently, the top 3 models out of the previous 9 models are rerun with a larger budget of 162. In the last iteration, we choose the top performing out of the 3 models and run with the budget of 486. Here, the performance score corresponds to the value of $f(\boldsymbol{\theta})$ for the hyperparameter $\boldsymbol{\theta}$ which could be different metric like validation loss.

used. Li et al. [15] created HyperBand to solve this issue by repeatedly applying the SuccessiveHalving with different starting number configurations and initial budget. The pseudo-code of SuccessiveHalving and HyperBand are illustrated in Algorithm 1 and 2 respectively.[5] It has been shown that HyperBand's convergence to the global optimum is restricted as it is based on randomly sampled configuration.

Algorithm 1 SuccessiveHalving

```
Input: initial budget $b_0$, maximum budget $b_{max}$, $\eta$, $n$ different configurations
$HP = \{\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \ldots, \boldsymbol{\theta}_n\}$.

1: $b = b_0$
2: while $b \leq b_{max}$ do
3:     Evaluate all configuration in $HP$ with budget $b$, $L = \{f(\boldsymbol{\theta}, b) : \boldsymbol{\theta} \in HP\}$.
4:     Pick the top $\lfloor \frac{|HP|}{\eta} \rfloor$ performing configuration. $HP = top_k(L, HP, \lfloor \frac{|HP|}{\eta} \rfloor)$.
5:     Set the next round budget, $b = \eta \times b$.
6: end while
```

[5] The version of the HyperBand algorithm is slightly different from the original but is used in the original code of BOHB. Difference is how to compute the number of configuration n.

Algorithm 2 HyperBand

```
Input: minimum and maximum budgets per configuration $b_{min}$ and $b_{max}$, $\eta$

1: $s_{max} = \lfloor log_\eta \frac{b_{max}}{b_{min}} \rfloor$
2: for $s$ from $s_{max}$ to 0 do
3:     sample $n = \lfloor \lfloor \frac{s_{max}+1}{s+1} \rfloor \times \eta^s \rfloor$ configurations $HP = \{\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \dots, \boldsymbol{\theta}_n\}$
4:     run $SuccessiveHalving(b_0, b_{max}, \eta, HP)$ with budget $b_0 = \eta^s \times b_{max}$.
5: end for
```

In order to resolve the above issue, Falkner et al. combine Bayesian Optimization and HyperBand by proposing BOHB [8]. BOHB uses HyperBand to decide the number of configurations and the budget in which these configurations are to be evaluated. Furthermore, it replaced the randomly drawn sample at the beginning of each HyperBand iteration with Bayesian Optimization. For iteration i, Bayesian Optimization models the objective function f by using a probabilistic model $p(f|D)$ with currently observed data points $D = \{(\boldsymbol{\theta}_0, o_0), (\boldsymbol{\theta}_1, o_1), \dots, (\boldsymbol{\theta}_{i-1}, o_{i-1})\}$ where $o_i := f(\boldsymbol{\theta}_i)$. We note that $\boldsymbol{\theta}_j$ is a hyperparameter configuration in our use case. Given a probabilistic model $p(f|D)$, the Bayesian Optimization proceeds with the following steps in each iteration:

1. Select a new point $\boldsymbol{\theta}_{new}$ such that it maximizes a given acquisition function $a : \Theta \to \mathbb{R}$:
$$\boldsymbol{\theta}_{new} = \operatorname*{argmax}_{\boldsymbol{\theta} \in \Theta} a(\boldsymbol{\theta})$$
2. Evaluate the performance of $\boldsymbol{\theta}_{new}$ with the objective function f, $o_{new} = f(\boldsymbol{\theta}_{new})$.
3. Add the new data point into the dataset $D \leftarrow D \cup \{(\boldsymbol{\theta}_{new}, o_{new})\}$ and refit the probabilistic model $p(f|D)$.

We note that in each iteration, the acquisition function a is based on the current probabilistic model $p(f|D)$. BOHB uses a variant of Bayesian Optimization known as the Tree-structured Parzen Estimator (TPE) [4]. BOHB uses TPE with a multidimensional kernel density estimator (KDE) to estimate the following densities:

$$\begin{aligned} l(\boldsymbol{\theta}) &= p(o < \alpha | \boldsymbol{\theta}, D) \\ g(\boldsymbol{\theta}) &= p(o > \alpha | \boldsymbol{\theta}, D). \end{aligned} \tag{2}$$

It picks the next point $\boldsymbol{\theta}_{new}$ that maximizes the ratio $\frac{l(\boldsymbol{\theta})}{g(\boldsymbol{\theta})}$. [4] have shown that this is equivalent to maximizing a commonly used acquisition function known as expected improvement (EI) (i.e., $a(\boldsymbol{\theta}) = \int max(0, \alpha - f(\boldsymbol{\theta}) dp(f|D))$. We note that here, switching out the TPE with the commonly used Gaussian process as used in [25] is possible. However, Gaussian processes run in cubic time with respect to the number of data points while, due to the usage of multidimensional KDE (see Appendix A), the TPE scales linearly instead. We will leave the benchmarking of different probabilistic models and acquisition functions for future works.

Algorithm 3 Sampling in BOHB

Input: datasets $D = \bigcup_b D_b$, fraction ρ for random sampling, q percentage of models to consider, minimum number of data points N_{min} to build a model, number of sample N_s from KDEs, and bandwidth factor b_w.
Output: next configuration θ to evaluate.

```
1: if $rand() < \rho$ then return random configuration, $\boldsymbol{\theta}_{new}$.
2: end if
3: $b = \text{argmax}_b\{D_b : |D_b| \geq N_{min} + 2\}$
4: if $b = \emptyset$ then return random configuration, $\boldsymbol{\theta}_{new}$.
5: end if
6: Fit KDEs according to Equation 2 with $N_{b,l}$ number of best configurations to model $l(\boldsymbol{\theta})$ and
   $N_{b,g}$ number of worst configurations to model $g(\boldsymbol{\theta})$.
7: draw $N_s$ according to $l'(\boldsymbol{\theta})$ which is the same as $l(\boldsymbol{\theta})$ but with all the bandwidths of the KDE is
   multiplied by a factor of $b_w$.
8: return configuration $\boldsymbol{\theta}_{new}$ that maximizes the ratio $\frac{l(\boldsymbol{\theta})}{g(\boldsymbol{\theta})}$.
```

When sampling configurations in BOHB, a fraction of the random run ρ are randomly sampled while the rest are sampled using the Bayesian Optimization considered. This is to promote more exploration of configuration. We describe the sampling process within BOHB in Algorithm 3. This is performed within line 3 of Algorithm 2 when BOHB executes the HyperBand procedure. After initialized with $N_{min} + 2$ random configurations, $N_{b,l} = \max(N_{min}, q \cdot N_b)$ number of best configurations and $N_{b,g} = \max(N_{min}, N_b - N_{b,l})$ number of worst configurations are used to fit KDEs to model the two densities $l(\boldsymbol{\theta})$ and $g(\boldsymbol{\theta})$ respectively (line 6 of Algorithm 3). Then, it sample N_s from the modified KDE of $l(x)$ and pick the $\boldsymbol{\theta}_{new}$ that maximizes the ratio $\frac{l(\boldsymbol{\theta})}{g(\boldsymbol{\theta})}$. The modified KDE $l'(x)$ is simply $l(\boldsymbol{\theta})$ but with all the bandwidths is multiplied by a factor of b_w. For full details of the internal working of BOHB, we refer readers to its original paper [8].

4 Objective Functions

The type of objective function f is an essential component in an HPO problem. It determines which performance measure to minimize (according to Equation 1) in an HPO problem and guides the hyperparameter optimization algorithm. We want to emphasize that we do not include the network size in the objective function, as we prioritize the discovery of a DNN capable of recovering the secret key as significantly more important. In [25], they explored three different objective functions: key rank, validation loss function on attack traces, and a proposed objective function called L_m. In this section, we will first recall the definition of the objective functions L_m and the validation loss used previously. Then, we will propose a new objective function called ge_{+ntge}, which incorporates not just the guessing entropy GE but also $NTGE$ into the performance measure, which was never considered before.

4.1 Prior Objective Functions used: L_m and Val_loss

$\boldsymbol{L_m}$. In [25], they modify the Leakage Difference Distribution (**LDD**) created by [26] to create a metric called L_m as an objective function for their Bayesian

Optimization framework known as AutoSCA. The **LDD** is defined as

$$LDD(k, k^*) = \sum_{i=0}^{Q} ||LM(p_i, k^*) - LM(p_i, k))||^2, k \in \mathcal{K},$$

where LM is the leakage model, p_i is the public data and k is the corresponding key. An example of LM is the Hamming weight leakage model, where $LM(p_i, k) = HW(Sbox(p_i \oplus k))$. This provides an estimation of the hypothetical label distribution variation between the actual key and the other key candidates. [25] extend this metric as a correlation between the key guessing vector $\boldsymbol{G}$ and LDD:

$$L_m(\mathbf{LDD}, \boldsymbol{G}) = corr(argsort(LDD), \boldsymbol{G}).$$

It has been shown in [25] that using this L_m obtains superior results compared to other tested metrics like key rank and validation loss in the AutoSCA framework. Therefore, we will test BOHB with L_m as its objective function. Note that since we are minimizing f, we can simply set $f = -L_m$.

Val_Loss. Furthermore, it was previously demonstrated that minimizing the categorical cross-entropy loss is equivalent to maximizing the generalization of the mutual information between the leakage model and the trace (also known as perceived information) [17]. Similar to [25], we explore whether the minimization of the validation loss of the attack traces as an objective loss in BOHB can help to find good-performing DNNs. We denote this objective function as Val_loss.

4.2 A New Objective Function: ge_{+ntge}

In [25], they investigated three different objective functions: key rank, validation loss based on attack traces, and L_m. They concluded that L_m is the best objective function. However, the key rank did not consider $NTGE$ in its objective function. The main goal of the hyperparameter search is not just to recover the secret key within the given number of traces but also to get the best-performing DNN, especially if an evaluator wants to know the smallest $NTGE$ required. In other words, we want the DNN to recover the secret key with the least number of attack traces. Therefore, one should include $NTGE$ in the objective function. Hence, we proposed the objective function known as ge_{+ntge} to incorporate $NTGE$ into the optimization process.

For a given configuration/hyperparameters $\boldsymbol{\theta}$ and a fixed number of attack traces N_a for evaluation, we define the objective function ge_{+ntge} as follows:

$$ge_{+ntge}(\boldsymbol{\theta}) = \begin{cases} NTGE & \text{if } GE = 0, \\ GE + N_a + c & \text{otherwise} \end{cases}$$

where c is a small positive constant. When the $GE \neq 0$, this means that for N_a of attack traces, the trained DNN with hyperparameters $\boldsymbol{\theta}$ did not recover the key. Since we want to show how far off the given configuration is to recover the key (i.e., $GE = 0$), we add GE to N_a. The constant c is further added to give an extra penalty for not recovering the key within the given number of attack traces. We set $c = 100$ throughout this paper. Since we are minimize f according to Equation 1 and we want to minimize ge_{+ntge}, we simply set $f = ge_{+ntge}$.

5 Experimental Settings

5.1 Datasets and Leakage model

We consider four publicly available datasets running the Advanced Encryption Scheme (AES) [7]. These are scenarios commonly faced in SCA. We focus on attacking a single byte of the secret key.

ASCADf and ASCADr. The ASCAD dataset consists of a first-order masked AES implementation on an 8-bit AVR microcontroller (ATMega8515) [3]. We target the third byte of the first round AES Sbox. This is a first-order masked key byte. Two versions known as ASCADf and ASCADr are part of the ASCAD dataset. ASCADf contains traces corresponding to the same fixed key for both profiling and attack. ASCADr contains profiling traces generated from a random key setting, while the attack traces are obtained from the fixed key target device. We use 45000 profiling traces for both datasets. Moreover, we use 2000 attack traces for ASCADf and 10000 attack traces for ASCADr. The traces in ASCADf are composed of 700 sample points, while the traces in ASCADr consist of 1400 sample points.

CTF2018. At the Conference on Cryptographic Hardware and Embedded Systems (CHES) in 2018, a dataset called CTF2018 was released [6]. The CTF2018 dataset consists of traces from running a first-order masked AES on a 32-bit STM microcontroller. We use 45000 traces for profiling and 3000 traces for attack. Unlike the ASCADf dataset, both profiling and attack traces consist of different fixed keys. We attack the first byte of the key. The traces have 2200 sample points.

AES_HD. The AES_HD is an unprotected AES hardware implementation dataset executed on an FPGA in a round-based architecture. We target the last round leakage $Sbox^{-1}_{AES}(ct_{15} \oplus k^*_{15}) \oplus ct_{11}$ where ct_i is the i^{th} ciphertext byte and k^*_{15} is the 15^{th} byte of the last round secret key. We use 45000 profiling traces and 3000 attack traces.

Leakage Model. We investigate two common hypothetical leakage models: the identity (ID) and the Hamming weight (HW) leakage models for ASCADf, ASCADr, and CTF2018. In other words, the hypothetical sensitive variables we consider are $Sbox_{AES}(pt \oplus k)$ for the ID leakage model and $HW(Sbox_{AES}(p_i \oplus k))$. On the other hand, we only consider the HD leakage model for AES_HD dataset as with previous works [1, 29]: $Sbox_{AES}(ct_{15} \oplus k) \oplus ct_{11}$.

[6] https://chesctf.riscure.com/2018/news

Table 1: Hyperparameter search space.

Hyperparameter	**Options**
MLP	
Number of Dense Layers	1 to 8 in a step of 1
Neurons per layer	$10, 20, 50, 100, 200, 300, 400, 500$
CNN	
Convolution layers	1 to 4 in step of 1
Convolution filters	4 to 16 in step of 4
Kernel size	26 to 52 in step of 2
Padding	0 to 16 in step of 2
Pooling type	Average or Max
Pooling size	2 to 10 in step of 2
Number of Dense Layers	1 to 8 in a step of 1
Neurons per layer	$10, 20, 50, 100, 200, 300, 400, 500$
Others	
Batch size	100 to 1000 in a step of 100
Activation function	$ReLU, SeLU, ELU$ or $tanh$
Optimizer	Adam or RMSprop
Learning Rate	$1e-3, 1e-4, 5e-4, 1e^{-5}, 5e^{-5}$
Weight Initializer	Random Uniform or Glorot Uniform or He Uniform

5.2 DNN Architecture and Training Setting

The hyperparameter search space is provided in Table 1. Furthermore, we fix the polling stride equal to the pooling size for simplicity. The epochs to train each model are determined by the HyperBand algorithm used within BOHB. Furthermore, we fixed the iteration of BOHB to 50. Throughout the paper, we train the DNNs with categorical cross-entropy loss function similar to other works [1, 21, 25, 29]. We run BOHB by using the HpBandster library on top of PyTorch [18]. We run our experiments on a single CPU and one NVIDIA-GeForce-GTX-970 with 4 Gigabytes of GPU memory and 1664 GPU cores. Here, we consider a total of 76800 hyperparameters configurations for MLP and around 1.548×10^9 different configurations for CNN. This range is larger than [25] and [21]. This is because they did not consider hyperparameters like batch size and padding, which could affect the training of the DNN. Within the other parameters of BOHB, we fixed η to be 3 as recommended and the fraction of the run ρ to be randomly sampled as $\frac{1}{3}$.

5.3 Budget Considered: Number of Epochs

The minimum budget b_{min} and the maximum budget b_{max} of training one neural network is given to BOHB as parameters. Based on the values of both b_{min} and

Table 2: Total time taken to run BOHB.

Max Budget b_{max}	50	100	200	500
ASCADf	$\approx 3hrs$	$\approx 7.5hrs$	$\approx 12hrs$	$\approx 1day13hrs$
ASCADr	$\approx 4hrs$	$\approx 10hrs$	$\approx 14.5hrs$	$\approx 1day21hrs$
AES_HD	$\approx 5hrs$	$\approx 12hrs$	$\approx 17hrs$	$\approx 2day6hrs$
CTF2018	$\approx 4hrs$	$\approx 10hrs$	$\approx 14hrs$	$\approx 1day21hrs$

b_{max}, the HyperBand subroutine optimized the number of DNNs to train and the budget given to train these DNNs as described in Section 3.2. The budget could be any resource, like the amount of time taken or the number of epochs to train a neural network. We note that the number of epochs is essentially the same as the time taken for training. As a DNN is trained with more epochs, the more time it will take to finish training. *Therefore, we consider the number of epochs as the budget as the parameters for BOHB.*

Previous works have shown that a DNN could obtain the secret key even with a small number of epochs [1, 25]. Therefore, we fixed our minimum budget b_{min} to be 10 throughout. We will explore the impact of b_{max} in Section 6.

6 Experimental Results

In this section, we investigate the performance and run time of BOHB when the maximum budget b_{max} varies. We denote $NTGE_{best}$ as the best $NTGE$ attained by BOHB in the various settings given.

6.1 Synchronized Datasets

Time Taken. Firstly, we will show the total time in training for all the datasets. Here, we consider the CNN with ID leakage model, which will take the longest time to train. The total time taken for each of the datasets is given in Table 2. We see that the larger the max budget b_{max}, the longer the time required. This is expected, as a larger b_{max} will result in a larger s_{max}. Therefore, more hyperparameters/configurations are sampled for evaluation (see line 3 of Algorithm 2). It was reported that [21] uses around 4 days to finish their run when using reinforcement learning, while [1] uses 2 days. It has been reported by [25] that 10 hours on average are required for a single run. However, as stated above, AutoSCA simply fixed the number of epochs for every single configuration. Technically, the AutoSCA framework by [25] can be considered a special case of BOHB where $b_{min} = b_{max} = 10$. Therefore, AutoSCA will have a similar time required to run with BOHB, although they used Gaussian Processes instead of TPE. Recall that the Gaussian Process has a cubic time complexity w.r.t number of data, whereas TPE offers linear complexity.

Table 3: $NTGE_{best}$ on the ASCADf for HW leakage model. The best $NTGE_{best}$ among the MLP and CNN setting are marked in blue and red respectively.

Max Budget b_{max}	50	100	200	500
MLP: ge_{+ntge}	1097	1216	1314	849
MLP: Val_loss	1111	1132	1148	1139
MLP: L_m	1641	1314	918	1064
CNN: ge_{+ntge}	1388	1314	1157	1195
CNN: Val_loss	1461	1445	1193	1345
CNN: L_m	1812	1499	1379	1233

Table 4: $NTGE_{best}$ on the ASCADf for ID leakage model. The best $NTGE_{best}$ among the MLP and CNN setting are marked in blue and red respectively.

Max Budget b_{max}	50	100	200	500
MLP: ge_{+ntge}	358	304	201	260
MLP: Val_loss	376	315	323	251
MLP: L_m	400	272	335	321
CNN: ge_{+ntge}	416	359	307	244
CNN: Val_loss	473	297	322	255
CNN: L_m	317	424	352	322

ASCADf. Table 3 and 4 show the $NTGE_{best}$ for various max budgets when applying BOHB on the ASCADf dataset. Here, we observed that for MLP and CNN settings in both HW and ID leakage models, $NTGE_{best}$ is obtained using the proposed objective function ge_{+ntge}. Overall, even with a small max budget of 50, BOHB could find a competitive $NTGE$ compared to a larger max budget. For both ID and HW leakage models, $NTGE_{best}$ are found within a max budget of 200 or 500. Although we obtain competitive results with a max budget of 50 in the ID leakage model, it is observed that when the max budget b_{max} is increased, there is a higher probability that $NTGE_{best}$ attained will be smaller. This is because, with a larger budget, BOHB will be given more budget to explore more hyperparameters.

ASCADr. For ASCADr, we show $NTGE_{best}$ for various max budgets in Table 5 and 6. Unlike in ASCADf, we see that for both HW and ID leakage models, the least $NTGE_{best}$ is attained with a max budget b_{max} of 50 and 100. This is surprising as a lower budget means that BOHB attains better results despite searching for lesser hyperparameters. This means that when using BOHB, it is

Table 5: $NTGE_{best}$ on the ASCADr for HW leakage model. The best $NTGE_{best}$ among the MLP and CNN setting are marked in blue and red respectively.

Max Budget b_{max}	50	100	200	500
MLP: ge_{+ntge}	1548	1641	2019	1559
MLP: Val_loss	2323	1664	2076	1874
MLP: L_m	1866	2278	2167	2262
CNN: ge_{+ntge}	1890	942	1449	895
CNN: Val_loss	1582	2193	1721	1177
CNN: L_m	1411	879	1026	1836

Table 6: $NTGE_{best}$ on the ASCADr for ID leakage model. The best $NTGE_{best}$ among the MLP and CNN setting are marked in blue and red respectively.

Max Budget b_{max}	50	100	200	500
MLP: ge_{+ntge}	2788	2358	2169	2711
MLP: Val_loss	2579	1726	2976	1984
MLP: L_m	1568	1985	2443	2180
CNN: ge_{+ntge}	3103	2784	3101	2999
CNN: Val_loss	3108	3101	3101	2999
CNN: L_m	3103	2916	2910	3101

possible to use less time to recover the secret key (see Table 2). We hypothesize this is because a certain fraction of runs are randomly sampled for exploration during BOHB. Next, we observe that ge_{+ntge} attain the smallest $NTGE_{best}$ for MLP (HW) and CNN (ID) setting while L_m acquire the least $NTGE_{best}$ for CNN (HW) and MLP (ID) setting. However, we note that ge_{+ntge} obtain the second smallest $NTGE_{best}$ using a 500 max budget and the third smallest $NTGE_{best}$ using a max budget of 100. This shows that ge_{+ntge} is a valid objective function to be considered when using BOHB.

AES_HD. Next, we show the $NTGE_{best}$ of AES_HD for various max budgets in Table 7. We recall here that the ID leakage model here denotes the use of $Sbox_{AES}(ct_{15} \oplus k) \oplus ct_{11}$ as the label. From Table 7, we observe that using Val_loss as the objective function attains the smallest $NTGE_{best}$ in comparison to ge_{+ntge} and L_m. We note that ge_{+ntge} and L_m still acquire relatively similar $NTGE_{best}$ compared to those in Val_loss. The best max budget is 100, and we see that increasing the max budget does not necessarily mean a decrease in $NTGE_{best}$. This is most likely because some configurations are randomly

Table 7: $NTGE_{best}$ on the AES_HD for ID leakage model. The best $NTGE_{best}$ among the MLP and CNN setting are marked in blue and red respectively.

Max Budget b_{max}	50	100	200	500
MLP: ge_{+ntge}	1522	1255	1577	1409
MLP: Val_loss	1283	1030	1374	1343
MLP: L_m	1426	1119	1395	1367
CNN: ge_{+ntge}	1353	1314	1378	1204
CNN: Val_loss	1505	1132	1693	1382
CNN: L_m	1499	1319	1923	1492

Table 8: $NTGE_{best}$ on the CTF2018 for HW leakage model. The best $NTGE_{best}$ among the MLP and CNN setting are marked in blue and red respectively.

Max Budget b_{max}	50	100	200	500
MLP: ge_{+ntge}	180	450	936	200
MLP: Val_loss	713	742	269	288
MLP: L_m	1219	893	1237	773
CNN: ge_{+ntge}	141	122	135	82
CNN: Val_loss	101	149	185	89
CNN: L_m	115	147	140	91

sampled, affecting the overall BOHB results. This is favorable for evaluators, as this means there is a possibility the dataset could be broken with a smaller budget. As a result, less time is used to analyze the dataset.

CTF2018. We illustrate the $NTGE_{best}$ of CTF2018 for various max budget in Table 8 for HW leakage. We observe that ge_{+ntge} is the best objective for both MLP and CNN setting of HW leakage. For CTF2018, when running BOHB on a large max budget of 500, the $NTGE_{best}$ attained are all less than 100 attack traces. If we compare it with prior works, BOHB attains the best results (see Table 12).

Next, we consider the ID leakage model for CTF2018. Table 9 reports the $NTGE_{best}$ acquired for the different max budgets attained in CTF2018 with the ID leakage model. In [25] and [21], they reported that they could not find good-performing DNN. This is also discussed in [19]. However, we are able to find good-performing DNNs with the use of BOHB. In fact, with relatively small b_{max} of 50 and 100, BOHB could find a DNN that could retrieve the secret key with less than 3000 attack traces. This illustrates the effectiveness of BOHB in

Table 9: $NTGE_{best}$ on the CTF2018 for ID leakage model. The best $NTGE_{best}$ among the MLP and CNN setting are marked in blue and red respectively.

Max Budget b_{max}	50	100	200	500
MLP: ge_{+ntge}	$GE = 5$	2691	2975	2983
MLP: Val_loss	2991	$GE = 1$	2987	2955
MLP: L_m	$GE = 2$	2864	2999	2523
CNN: ge_{+ntge}	2992	$GE = 2$	2999	2927
CNN: Val_loss	2912	$GE = 1$	2987	2907
CNN: L_m	$GE = 1$	2989	$GE = 1$	2958

finding a well-performing DNN. Indeed, training with the ID leakage model is much inferior compared to training with the HW leakage model, as the number of traces to break is significantly more than in the HW leakage model. Next, we highlight that when we increase the b_{max}, finding a well-performing DNN becomes increasingly stable for the CTF2018 dataset.

6.2 Desynchronized Traces

In this section, we consider the setup where we train the DNN with desynchronized profiling traces and tackle attack traces that have the same desynchronization level as the profiling traces. We only consider the desynchronization level of 50 on ASCADf, which we will denote as ASCADf_desync50.

Table 10 and 11 both presents the $NTGE_{best}$ for various max budget. We observe that for $b_{max} = 50$, BOHB is unable to find any successful model when training in both HW and ID leakage models. This shows that the desynchronized datasets are more difficult to attack and require more resources for a successful attack. For the HW leakage model, we see that if we increase the max budget to more than 100, BOHB can find well-performing CNN for all objective functions. Furthermore, only when the max budget is 500 could BOHB find an MLP to attack the ASCADf_desync50 dataset successfully. This shows that it is easier to attack a desynchronized dataset with CNN compared to MLP and confirms the claims of [27]. This is most likely due to the shift-invariant property of CNN. For the ID leakage model of ASCADf_desync50, we observe that when the max budget is 100, BOHB could only find well-performing CNN for ge_{+ntge} and L_m as the objective function. But when the max budget is either 200 or 500, BOHB can find well-performing CNN for all objective functions. On the other hand, only the use of ge_{+ntge} with a max budget of either 200 or 500 can BOHB obtain a well-performing MLP. This shows that ge_{+ntge} is a preferred objective function for desynchronized datasets.

Table 10: $NTGE_{best}$ on the ASCADf_desync50 for HW leakage model. The best $NTGE_{best}$ among the MLP and CNN setting are marked in blue and red respectively.

Max Budget b_{max}	50	100	200	500
MLP: ge_{+ntge}	$GE = 129$	$GE = 139$	$GE = 210$	4507
MLP: Val_loss	$GE = 108$	$GE = 70$	$GE = 171$	3769
MLP: L_m	$GE = 106$	$GE = 118$	$GE = 85$	4861
CNN: ge_{+ntge}	$GE = 201$	3530	2841	3048
CNN: Val_loss	$GE = 216$	4278	3402	2469
CNN: L_m	$GE = 174$	3309	2778	2822

Table 11: $NTGE_{best}$ on the ASCADf_desync50 for ID leakage model. The best $NTGE_{best}$ among the MLP and CNN setting are marked in blue and red respectively.

Max Budget b_{max}	50	100	200	500
MLP: ge_{+ntge}	$GE = 6$	$GE = 7$	4951	4275
MLP: Val_loss	$GE = 6$	$GE = 6$	$GE = 10$	$GE = 5$
MLP: L_m	$GE = 9$	$GE = 7$	$GE = 2$	$GE = 8$
CNN: ge_{+ntge}	$GE = 5$	3182	1694	1744
CNN: Val_loss	$GE = 3$	$GE = 3$	2459	1311
CNN: L_m	$GE = 17$	2067	2054	1986

Which Objective Function and Max Budget to use? From the experiments, we highlight that ge_{+ntge} obtain the best $NTGE_{best}$ in 8 out of 14 scenarios. In comparison, L_m and Val_loss both attain best $NTGE_{best}$ in 3 different scenarios. This shows that ge_{+ntge} can be considered as a better objective function for BOHB. However, we note that the type of objective function could be dataset dependent as Val_loss obtain the best $NTGE_{best}$ for AES_HD for both MLP and CNN settings. Therefore, one should try each objective function with BOHB when resources permit. On the other hand, when resources/budgets are scarce, we suggest that ge_{+ntge} be the preliminary objective function to be used with BOHB.

Next, we recommend using a max budget of 100 if resources are limited as it obtains the best $NTGE_{best}$ for AES_HD. But if sufficient resources exist, a max budget of 500 should be considered. This is because BOHB managed to recover the secret key for all the scenarios with a max budget of 500 when tackling

CTF2018 with the ID leakage model, whereas there exist scenarios where no DNNs could recover the secret key when a lower budget is used (see Table 9).

For desynchronized datasets, we observed that a larger budget would be required to attain $GE = 0$. Therefore, a max budget of 500 would be preferred. Furthermore, we highlight that ge_{+ntge} is the only objective function to find a well-performing MLP on ASCADf with a desynchronization level of 50 when using the ID leakage model. Therefore, it is recommended to use ge_{+ntge} as an objective function for desynchronized datasets.

Comparing with other prior works. We present the results of $NTGE_{best}$ with other prior works in Table 12. Our goal was to give evaluators another tool to tackle more difficult datasets by allocating the resources more effectively through BOHB. We show that BOHB can obtain comparable results and still find a DNN with $GE = 0$ for every dataset tested. However, we want to highlight that the search space for each work is different. In fact, our search space is much bigger compared to those in [21] and [25]. We emphasize that BOHB can recover the secret key of CTF2018 of the HW leakage model with less than 100 attack traces. This is the least number of traces reported among the other automated tools. Furthermore, for the first time it was recorded, we could successfully recover the secret key of CTF2018 using the ID leakage model. Surprisingly, despite the BOHB did not consider the number of parameters in their objective function, we observe several instances where the number of parameters are relatively small. For instance, BOHB found a well-performing model with $10,596$ parameters in ASCADf (ID). This model is smaller than those found in [21], where they consider the size of the model in their methodology. Furthermore, this model by BOHB has a similar performance of compared to the model found in [21]. Similar observations can be seen in CTF2018 (HW) and ASCADf_desync50 (HW). Overall, we have demonstrated the capability of BOHB to recover the key for all datasets tested. These show that BOHB is effective in finding well-performing DNNs.

7 Conclusion and Future Works

In this work, we introduce a multifidelity optimization method known as BOHB into the domain of SCA. BOHB allocates resources efficiently and finds well-performing DNNs through the use of Bayesian Optimization when sampling. We show the capabilities of BOHB by showing that it could find a DNN that recovers keys on all the public datasets tested: ASCADf, ASCADr, AES_HD, and CTF2018. Although, in most datasets, BOHB did not attain the best results. This is most likely due to the differences in search space, as the tested hyperparameter search space is bigger than those used in other works. Despite that, BOHB obtained the best results for both CTF2018 (HW) and CTF2018 (ID). In addition, we also proposed a novel objective function known as ge_{+ntge}. We show the effectiveness of ge_{+ntge} compared to other objective functions, and also introduce the notion of budget when finding efficient architectures for DNN-based

Table 12: Comparing best $NTGE$ obtain by various works. $NTGE$ with colors are the best $NTGE_{best}$ provided by the dataset. Colored $NTGE_{best}$ are the best $NTGE$ for the particular dataset.

	Dataset	**Epochs**	**No. of parameters**	$NTGE_{best}$
[29]	ASCADf (ID)	50	16,960	191
	AES_HD	20	3,282	1,050
	ASCADf_desync50 (ID)	50	87,279	244
[1]	ASCADf (ID)	8	15,107	130
	ASCADr (ID)	8	317,408	120
	AES_HD	33	102,757	170
[25]	ASCADf (HW)	10	1,388,457	447
	ASCADf (ID)	10	1,544,776	120
	ASCADr (HW)	10	1,314,009	496
	ASCADr (ID)	50	1,539,320	1,568
	CTF2018 (HW)	50	2,418,085	618
[21]	ASCADf (HW)	50	8,480	1,246
	ASCADf (ID)	50	79,439	202
	ASCADr (HW)	50	15,241	911
	ASCADr (ID)	50	70,492	490
	CTF2018 (HW)	50	33,788	122
	ASCADf_desync50 (HW)	50	516,361	1,592
	ASCADf_desync50 (ID)	50	41,321	443
Ours	ASCADf (HW)	56	845,109	849
	ASCADf (ID)	200	10,596	201
	ASCADr (HW)	34	659,409	879
	ASCADr (ID)	17	1,465,056	1,568
	AES_HD	34	1,725,856	1,030
	CTF2018 (HW)	500	3,645	82
	CTF2018 (ID)	18	25,596	2,523
	ASCADf_desync50 (HW)	500	10,401	2,4698
	ASCADf_desync50 (ID)	500	91,976	1,311

SCA. For future work, it would be interesting to consider other Bayesian Optimization rather than TPE. One could explore the different probability models and acquisition functions used within BOHB. For another direction, one could look into the capability of different multifidelity optimization in the SCA domain, such as DEHB, in comparison to BOHB. As stated in the previous section, it is surprising that BOHB found smaller models despite the objective functions studied did not consider the number of parameters; it would be interesting to study objective functions that consider the number of parameters in the future.

References

1. Acharya, R.Y., Ganji, F., Forte, D.: Information theory-based evolution of neural networks for side-channel analysis. IACR Transactions on Cryptographic Hardware and Embedded Systems **2023**(1), 401–437 (Nov 2022). https://doi.org/10.46586/tches.v2023.i1.401-437, https://tches.iacr.org/index.php/TCHES/article/view/9957
2. Agrawal, D., Archambeault, B., Rao, J.R., Rohatgi, P.: The EM Side—Channel(s). In: Kaliski, B.S., Koç, ç.K., Paar, C. (eds.) Cryptographic Hardware and Embedded Systems - CHES 2002. pp. 29–45. Springer Berlin Heidelberg, Berlin, Heidelberg (2003)
3. Benadjila, R., Prouff, E., Strullu, R., Cagli, E., Dumas, C.: Deep learning for side-channel analysis and introduction to ASCADdatabase. J. Cryptogr. Eng. **10**(2), 163–188 (2020). https://doi.org/10.1007/s13389-019-00220-8
4. Bergstra, J., Bardenet, R., Bengio, Y., Kégl, B.: Algorithms for hyperparameter optimization. In: Shawe-Taylor, J., Zemel, R., Bartlett, P., Pereira, F., Weinberger, K. (eds.) Advances in Neural Information Processing Systems. vol. 24. Curran Associates, Inc. (2011)
5. Cagli, E., Dumas, C., Prouff, E.: Convolutional Neural Networks with Data Augmentation Against Jitter-Based Countermeasures. In: Fischer, W., Homma, N. (eds.) Cryptographic Hardware and Embedded Systems – CHES 2017. pp. 45–68. Springer International Publishing, Cham (2017)
6. Chari, S., Rao, J.R., Rohatgi, P.: Template Attacks. In: Kaliski, B.S., Koç, ç.K., Paar, C. (eds.) Cryptographic Hardware and Embedded Systems - CHES 2002. pp. 13–28. Springer Berlin Heidelberg, Berlin, Heidelberg (2003)
7. Dworkin, M., Barker, E., Nechvatal, J., Foti, J., Bassham, L., Roback, E., Dray, J.: Advanced Encryption Standard (AES) (2001-11-26 2001). https://doi.org/https://doi.org/10.6028/NIST.FIPS.197
8. Falkner, S., Klein, A., Hutter, F.: BOHB: robust and efficient hyperparameter optimization at scale. CoRR **abs/1807.01774** (2018), http://arxiv.org/abs/1807.01774
9. Jamieson, K.G., Talwalkar, A.: Non-stochastic best arm identification and hyperparameter optimization. CoRR **abs/1502.07943** (2015), http://arxiv.org/abs/1502.07943
10. Karayalcin, S., Perin, G., Picek, S.: Resolving the doubts: On the construction and use of ResNets for side-channel analysis. Mathematics **11**(15), 3265 (jul 2023). https://doi.org/10.3390/math11153265
11. Kerkhof, M., Wu, L., Perin, G., Picek, S.: Focus is key to success: A focal loss function for deep learning-based side-channel analysis. In: Balasch, J., O'Flynn, C. (eds.) Constructive Side-Channel Analysis and Secure Design. pp. 29–48. Springer International Publishing, Cham (2022)
12. Kerkhof, M., Wu, L., Perin, G., Picek, S.: No (good) loss no gain: systematic evaluation of loss functions in deep learning-based side-channel analysis. Journal of Cryptographic Engineering **13**(3), 311–324 (may 2023). https://doi.org/10.1007/s13389-023-00320-6
13. Kocher, P.C., Jaffe, J., Jun, B.: Differential Power Analysis. In: Proceedings of the 19th Annual International Cryptology Conference on Advances in Cryptology. p. 388–397. CRYPTO '99, Springer-Verlag, Berlin, Heidelberg (1999)
14. Li, H., Krček, M., Perin, G.: A comparison of weight initializers in deep learning-based side-channel analysis. In: Zhou, J., Conti, M., Ahmed, C.M., Au, M.H.,

Batina, L., Li, Z., Lin, J., Losiouk, E., Luo, B., Majumdar, S., Meng, W., Ochoa, M., Picek, S., Portokalidis, G., Wang, C., Zhang, K. (eds.) Applied Cryptography and Network Security Workshops. pp. 126–143. Springer International Publishing, Cham (2020)

15. Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A., Talwalkar, A.: Hyperband: A novel bandit-based approach to hyperparameter optimization. Journal of Machine Learning Research **18**(185), 1–52 (2018), http://jmlr.org/papers/v18/16-558.html
16. Maghrebi, H., Portigliatti, T., Prouff, E.: Breaking Cryptographic Implementations Using Deep Learning Techniques. pp. 3–26 (12 2016). https://doi.org/10.1007/978-3-319-49445-6_1
17. Masure, L., Dumas, C., Prouff, E.: A comprehensive study of deep learning for side-channel analysis. IACR Transactions on Cryptographic Hardware and Embedded Systems pp. 348–375 (2020)
18. Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., Lerer, A.: Automatic differentiation in pytorch (2017)
19. Perin, G., Chmielewski, L., Picek, S.: Strength in numbers: Improving generalization with ensembles in machine learning-based profiled side-channel analysis. IACR Transactions on Cryptographic Hardware and Embedded Systems **2020**(4), 337–364 (Aug 2020). https://doi.org/10.13154/tches.v2020.i4.337-364, https://tches.iacr.org/index.php/TCHES/article/view/8686
20. Picek, S., Heuser, A., Jovic, A., Bhasin, S., Regazzoni, F.: The Curse of Class Imbalance and Conflicting Metrics with Machine Learning for Side-channel Evaluations, volume=2019. IACR Transactions on Cryptographic Hardware and Embedded Systems (1), 209–237 (Nov 2018). https://doi.org/10.13154/tches.v2019.i1.209-237, https://tches.iacr.org/index.php/TCHES/article/view/7339
21. Rijsdijk, J., Wu, L., Perin, G., Picek, S.: Reinforcement learning for hyperparameter tuning in deep learning-based side-channel analysis. IACR Transactions on Cryptographic Hardware and Embedded Systems **2021**(3), 677–707 (Jul 2021). https://doi.org/10.46586/tches.v2021.i3.677-707, https://tches.iacr.org/index.php/TCHES/article/view/8989
22. Weissbart, L.: Performance analysis of multilayer perceptron in profiling side-channel analysis. In: Zhou, J., Conti, M., Ahmed, C.M., Au, M.H., Batina, L., Li, Z., Lin, J., Losiouk, E., Luo, B., Majumdar, S., Meng, W., Ochoa, M., Picek, S., Portokalidis, G., Wang, C., Zhang, K. (eds.) Applied Cryptography and Network Security Workshops. pp. 198–216. Springer International Publishing, Cham (2020)
23. Wouters, L., Arribas, V., Gierlichs, B., Preneel, B.: Revisiting a Methodology for Efficient CNN Architectures in Profiling Attacks. IACR Transactions on Cryptographic Hardware and Embedded Systems **2020**(3), 147–168 (Jun 2020). https://doi.org/10.13154/tches.v2020.i3.147-168, https://tches.iacr.org/index.php/TCHES/article/view/8586
24. Wu, L., Perin, G.: On the importance of pooling layer tuning for profiling side-channel analysis. In: Zhou, J., Ahmed, C.M., Batina, L., Chattopadhyay, S., Gadyatskaya, O., Jin, C., Lin, J., Losiouk, E., Luo, B., Majumdar, S., Maniatakos, M., Mashima, D., Meng, W., Picek, S., Shimaoka, M., Su, C., Wang, C. (eds.) Applied Cryptography and Network Security Workshops. pp. 114–132. Springer International Publishing, Cham (2021)
25. Wu, L., Perin, G., Picek, S.: I Choose You: Automated Hyperparameter Tuning for Deep Learning-based Side-channel Analysis. IEEE Transactions on Emerging Topics in Computing pp. 1–12 (2022). https://doi.org/10.1109/TETC.2022.3218372

26. Wu, L., Weissbart, L., Krček, M., Li, H., Perin, G., Batina, L., Picek, S.: On the attack evaluation and the generalization ability in profiling side-channel analysis. Cryptology ePrint Archive, Report 2020/899 (2020), https://ia.cr/2020/899
27. Wu, L., Won, Y.S., Jap, D., Perin, G., Bhasin, S., Picek, S.: Ablation analysis for multi-device deep learning-based physical side-channel analysis. IEEE Transactions on Dependable and Secure Computing (2023)
28. Zaid, G., Bossuet, L., Dassance, F., Habrard, A., Venelli, A.: Ranking loss: Maximizing the success rate in deep learning side-channel analysis. Cryptology ePrint Archive, Report 2020/872 (2020), https://ia.cr/2020/872
29. Zaid, G., Bossuet, L., Habrard, A., Venelli, A.: Methodology for Efficient CNN Architectures in Profiling Attacks. IACR Transactions on Cryptographic Hardware and Embedded Systems **2020**(1), 1–36 (Nov 2019). https://doi.org/10.13154/tches.v2020.i1.1-36, https://tches.iacr.org/index.php/TCHES/article/view/8391
30. Zaid, G., Bossuet, L., Habrard, A., Venelli, A.: Understanding methodology for efficient cnn architectures in profiling attacks. Cryptology ePrint Archive, Paper 2020/757 (2020), https://eprint.iacr.org/2020/757, https://eprint.iacr.org/2020/757
31. Zhang, J., Zheng, M., Nan, J., Hu, H., Yu, N.: A novel evaluation metric for deep learning-based side channel analysis and its extended application to imbalanced data. IACR Transactions on Cryptographic Hardware and Embedded Systems **2020**(3), 73–96 (Jun 2020). https://doi.org/10.13154/tches.v2020.i3.73-96, https://tches.iacr.org/index.php/TCHES/article/view/8583

A Kernel Density Estimator

In this section, we describe the KDE used in BOHB. Suppose $h = (h_1, \ldots, h_q) \in \mathbb{R}^q$ and let $X_1, \ldots, X_n$ be independent and identically distributed samples drawn from an unknown density distribution. The multivariate KDE at a given point $x \in \mathbb{R}^q$ is describe as

$$f(x) = \frac{1}{nh_1 \cdots h_q} \sum_{i=1}^{n} K\left(\frac{X_i - x}{h}\right)$$

where

$$K\left(\frac{X_i - x}{h}\right) = k\left(\frac{X_{i1} - x_1}{h_1}\right) \times k\left(\frac{X_{i2} - x_2}{h_2}\right) \times ... \times k\left(\frac{X_{iq} - x_q}{h_q}\right)$$

with the function k as the kernel function. It is stated in [8], BOHB uses Gaussian kernel for the continuous parameters. On the other hand, the Aitchison-Aitken kernel are used for categorical parameters.