IACR Transactions on Cryptographic Hardware and Embedded Systems
ISSN 2569-2925, Vol. 2023, No. 1, pp. 401–437. DOI:10.46586/tches.v2023.i1.401-437

Information Theory-based Evolution of Neural Networks for Side-channel Analysis

Rabin Y. Acharya[1], Fatemeh Ganji[2] and Domenic Forte[1]

[1] University of Florida, Gainesville, USA, rabin.acharya@ufl.edu,dforte@ece.ufl.edu
[2] Worcester Polytechnic Institute, Worcester, USA, fganji@wpi.edu

Abstract. Profiled side-channel analysis (SCA) leverages leakage from cryptographic implementations to extract the secret key. When combined with advanced methods in neural networks (NNs), profiled SCA can successfully attack even those cryptocores assumed to be protected against SCA. Despite the rise in the number of studies devoted to NN-based SCA, a range of questions has remained unanswered, namely: how to choose an NN with an adequate configuration, how to tune the NN's hyperparameters, when to stop the training, etc. Our proposed approach, "InfoNEAT," tackles these issues in a natural way. InfoNEAT relies on the concept of neural structure search, enhanced by information-theoretic metrics to guide the evolution, halt it with novel stopping criteria, and improve time-complexity and memory footprint. The performance of InfoNEAT is evaluated by applying it to publicly available datasets composed of real side-channel measurements. In addition to the considerable advantages regarding the automated configuration of NNs, InfoNEAT demonstrates significant improvements over other approaches for effective key recovery in terms of the number of epochs (e.g.,×6 faster) and the number of attack traces compared to both MLPs and CNNs (e.g., up to 1000s fewer traces to break a device) as well as a reduction in the number of trainable parameters compared to MLPs (e.g., by the factor of up to 32). Furthermore, through experiments, it is demonstrated that InfoNEAT's models are robust against noise and desynchronization in traces.

Keywords: Side-channel Analysis · Neural Networks · Multi-layer Perceptrons · Evolutionary Strategies · Stacking · Information Theory

1 Introduction

While promising candidates have been proposed in the literature to deal with secure generation and key storage, the issue with secure execution continues to exist. Secure execution is a challenging goal to attain due to the leakage of information from implementations, often referred to as a side-channel. Attacks leveraging such secret-key leakages are strong in the sense that they have broken the security of various real-world devices. The involvement of standardization and certification bodies in the activities related to the development of tools for leakage detection and side-channel analysis (SCA) further highlights the importance of this matter, see, e.g., [Nat20, Eur20].

Prominent examples of side-channels include execution time, power consumption, and electromagnetic (EM) radiations collected from a target, e.g., a device embodying a cryptographic module such as an advanced encryption standard (AES) crypto-core. To analyze such side-channels, profiled attacks are common practice due to their effectiveness [BPS+18]. In the first phase of this attack (so-called profiling), the leakage from an open copy of the target device, is modeled under a controlled condition (e.g., known secret keys) cf. [CRR02, HGDM+11, LBM15, MHM13]. Traditionally, such a model has been obtained by characterizing the leakages precisely through statistical techniques, e.g.,

Received: 2022-07-15 Accepted: 2022-09-15 Published: 2022-11-29

linear regression [SLP05, DPRS11]. Afterward, in the second phase (so-called attack), this model is used to launch a key-recovery attack on the target device. These phases are in line with the specifics of machine learning (ML) tasks in the sense that the profiling and attack steps correspond respectively to the training and testing sub-tasks in the context of supervised ML. Interestingly enough, ML-enhanced SCA can defeat not only unprotected, see e.g., [LPB$^+$15, HZ12, HGDM$^+$11], but also protected cryptographic implementations [LBM15, GHO15, CDP17, PHJ$^+$19, KPH$^+$19, WPP20]. Furthermore, when only a limited number of traces with noise is available, ML-enhanced SCA approaches outperform the template attack [CRR02], known to be optimal from an information-theoretic perspective if a large enough number of traces are available [PHJ$^+$19, BCH$^+$20]. The scenario, where only a few noisy traces are available, is of great importance as it reflects a more realistic scenario.

In this regard, it is not surprising that deep learning and NNs are now playing an active role in the SCA literature [HGG20, PPM$^+$21]. In particular, it has been demonstrated that NNs can further defeat some countermeasures designed to protect a cryptographic implementation. Specifically, the jitter-based misalignments in the side-channel traces, i.e., creating an array of asynchronous measurements, cannot stop an attacker from launching SCA through NNs [CDP17, BPS$^+$18]. Even masked implementations, with countermeasures that randomize the intermediate values, can be successfully attacked by NNs [KPH$^+$19, WPP20, PHJ$^+$19]. In fact, methodologies, techniques, and solutions have been proposed to help evaluators at the analysis stage of NN-enabled SCA cf. [RPBA21]. An expensive profiling step can account for the cost of NN-enabled SCA [ABB$^+$20], where the NN configuration and hyperparameters should be determined. Therefore, although it seems tempting to consider NN-enabled SCA a step toward automating SCA, the design of NNs needs to become less expert-dependant. More concretely, these questions need to be answered, as they have remained partially open so far: *Which configurations and hyperparameter combinations should be used? When can the training process be stopped to achieve better generalization in the attack phase?*

InfoNEAT: Our paper introduces InfoNEAT that is the *first* NN-enhanced SCA taking advantage of neural architecture search (NAS) at its ***full*** capacity (see Section 2.3 for comparison with the most relevant approaches). InfoNEAT is a novel algorithm that revolves around the notion of the evolution of NNs, so-called "neuroevolution." The cornerstone of this concept is to evolve the *configuration of multiple networks (so-called augmenting topologies), and simultaneously tune their hyperparameters and parameters*. Specifically, it has been demonstrated that although NN-based attacks can extract the secret from even protected implementations, hyperparameter tuning imposes serious challenges to the application of NNs for SCA [WPP20, KPH$^+$19, ZBHV20]. Interestingly, although neuroevolution can be employed in various domains, InfoNEAT is tailored to the specific requirements of SCA. The workflow of InfoNEAT[1] is similar to common profiled attacks, see Figure 1. For existing NN-based attacks, in the training process, selecting the adequate configuration of the network and tuning the hyperparameters are often based on trial and error (see Section 2 for more details). InfoNEAT, on the contrary, evolves various NNs and their hyperparameters and eventually delivers the one selected based on sound information-theoretic metrics. Regarding our **contributions** listed below and expounded upon in the paper, InfoNEAT is particularly powerful for SCA.

(1) InfoNEAT tailors NAS for SCA. InfoNEAT takes advantage of the entire range of modern NN design elements, including hyperparameter tuning and multi-branch configuration. In doing so, the irregular topology of NNs (compared to the structure of multi-layer perceptrons–MLPs) automatically discovered by NEAT is one of the most important aspects of our framework, which can lead to the robustness against desynchronization as defined in SCA (see Section 6.3). The original NEAT algorithm [SM02] cannot cope

[1] The source code is available in the Github https://github.com/rachary00/InfoNEAT

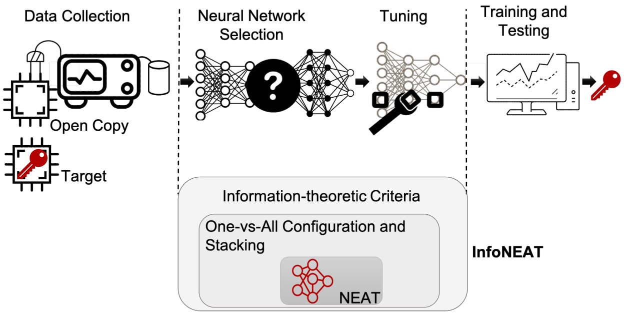


Figure 1: Overview of the proposed "InfoNEAT" which extends the NEAT algorithm to meet SCA requirements. Compared to conventional ML-enabled SCA, network selection and hyperparameter tuning are handled automatically.

with complex multi-class classification tasks, specifically when many output neurons are involved. To address this, InfoNEAT employs One-vs-All (sometimes called "One-vs-Rest") multi-class classification followed by stacking ensemble learning. This is indeed an elegant and effective method to *combine* classifiers for all sub-keys, although it has not been studied in the SCA-related literature. This is one of the novel aspects of InfoNEAT that conforms with the specification of SCA, where the number of classes is much larger than what has been considered in neuroevolution-related literature (see Section 5.3 for more details).

(2) Criteria for stopping the training/evolution processes in InfoNEAT. When introducing NEAT, defining such criteria has been identified as a challenge [SM02], however this issue has not been resolved, but rather handled through trial and error. It is evident that evolving NNs could be a time-consuming and memory-hungry process as done by the original NEAT algorithm. To address this, InfoNEAT is equipped with mechanisms relying on information theory. Intuitively, the information-theoretic conditions indicate whether adding a new connection/node can help improve InfoNEAT's output. If not, the evolution process is stopped. Moreover, the proposed criteria are useful to guide the evolution process by selecting the best NN among ones evolved by InfoNEAT. For SCA, the implication of this is that stopping early enough avoids overfitting and obtains better generalization to unseen testing traces.

(3) Key-recovery attacks using InfoNEAT. Last but not least, we evaluate the effectiveness of InfoNEAT against state-of-the-art (SOTA) side-channel trace databases and ML models (MLPs and CNNs) that have become the standard benchmarking system for SCA. InfoNEAT achieves competitive results compared to its counterparts, see Section 6.3, without relying on additional techniques (e.g., validation step) employed against challenging datasets in SOTA studies. InfoNEAT can also enhance the memory and time efficiency of SCA by reducing the number of trainable parameters in NNs and epochs[2], e.g., by factors of up to 32 and 6, respectively, when compared to the approach in [WPP20] that considers hyperparameter tuning for MLPs, counterparts of NNs delivered by InfoNEAT (for comparison with other SOTA attacks, see Section 6).

Outline: The rest of the paper is organized as follows. Section 2 gives a brief overview of the most relevant studies in the literature. In Section 3, profiled SCA along with our notations and mathematical principles are discussed. Afterward, in Section 4, we introduce the NEAT algorithm as a building block of our proposed InfoNEAT algorithm described in Section 5. This is then followed by Section 6, where the results of our experiments are presented and discussed. Finally, Section 7 concludes the paper.

[2] The number of epochs is the number of times a learning algorithm sees the entire training dataset.

2 Related Work

Machine learning techniques have become quite popular and successful in profiled SCA in recent years. Among them, multilayer perceptron (MLP) and convolutional neural networks (CNNs) are the two most widely used [Wei20, PSK+18, WPB19, HGG20]. Leaving aside the selection of network type, tuning the hyperparameters of NNs remains an issue of concern as inappropriate choices may lead to *overfitting*. In this section, we explain these issues as well as the relevant works in more detail. A summary of the related works, as well as the comparison with InfoNEAT, are also included in Table 7 (see Appendix A).

2.1 Type of Neural Networks

Similar to other ML tasks, selecting the model, e.g., NNs with proper configuration, is the first step and maybe, one of the most challenging ones. When combining CNNs with data augmentation techniques, one could launch successful SCA against even protected designs [CDP17, PYW+17]. As an example, a recent study has applied multi-scale CNN along with various data transformations (e.g., phase-only correlation, principal component analysis–PCA, alignment methods) to allow a better attack performance [WHJ+21]. In this regard, it has been demonstrated that advantages offered by CNNs can be most enjoyed when the level of noise is small and the number of measurements and features is high [PSK+18]. From another perspective, one of the reasons behind the success of CNNs in image classification is maintaining the ordering of features. Interestingly enough, this feature could be less important for SCA, cf. [PSK+18]. In line with this, in [HGG20, HHO20], the authors have mentioned that CNNs could outperform other NN-based attack techniques because of their effectiveness with raw data especially in the presence of jitter or desynchronization. This comparison, however, should be considered carefully in the field of SCA as some of the techniques have traditionally only considered ML metrics to evaluate their models [PHJ+19, PSK+18]. This, of course, does not rule out the adoption of CNNs in this domain but opens up new avenues for other NNs. From this perspective, the authors in [PHJ+19] have shown that when considering SCA-related metrics, both models (CNN and MLP) perform similarly well, especially when combined with SMOTE (Synthetic Minority Oversampling Technique). Picek et al. in [PSK+18] showed that MLPs equipped with XGBoost outperform CNN when considering pure SCA metrics such as the guessing entropy.

Another feature of CNNs that could be beneficial to SCA is their implicit feature selection part. This has been well-studied in [WAGP20], where the first convolutional layer of the SOTA CNN configured by Zaid et al. [ZBHV20] is replaced by a classical preprocessing technique to reduce the model's complexity effectively. Such combinations of preprocessing and CNNs can reduce the cost of training CNNs. In some scenarios, this cost could be justified if the performance of the attack is improved, e.g., when a trained CNN (e.g., VGG16) is used for SCA [BPS+18, MS21, HHO20]. In doing so, an interesting case is discussed in [HHO20], where a model trained on a dataset is applied against the desynchronized traces in that dataset. To our knowledge, MLPs have not been considered in this case; therefore, a careful cost assessment of training MLPs and CNNs under such circumstances is needed (see Section 6.3.1 for our results). This discussion leads us to one of the major issues in the SCA community, *generalization*.

2.2 The Issue of Generalization

Generalization is the ability of a trained SCA model to adapt well to new, unseen data. In our framework, in line with other relevant studies, generalization from training to test sub-datasets is considered. In this context, the authors in [vdVPB20] demonstrate attacks on different SCA datasets and show how MLPs are internally the same despite changing the device or the key because MLPs are considered universal approximators. Furthermore, even small MLP networks were shown to roughly learn the same function without overfitting and generalize to different datasets [PHPG21, vdVPB20]. Generalization has also been addressed in [PCP20], where the authors show how ensembles of models based on averaged class probabilities can further improve the network generalization for various datasets. For this purpose, [PCP20] has applied the bagging ensemble technique to combine models trained on all classes. In contrast to that, the model selection performed in the context of InfoNEAT is similar to the "bucket of models" ensemble method enhanced with stacking to combine sub-models generated per class (see Section 5.3 for more details). Accordingly, InfoNEAT is further boosted to improve the generalization among various classes. Tuning of the *hyperparameters* also has a direct impact on generalization. Next, we consider the most relevant literature devoted to this matter.

2.3 Hyperparameter Tuning and NAS

Before reviewing the studies on designing NNs for SCA, it is useful to distinguish between hyperparameter optimization and NAS. A NAS approach attempts to discover neural architectures by searching a specific space. The following items typically parameterize a search space: (a) The (maximum) number of layers that can be possibly unbounded; (b) the type of operation for each layer, e.g., pooling and convolution; (c) hyperparameters associated with the operation per layer, e.g., the number of filters, kernel size and strides for a convolutional layer, the number of neurons in a layer of MLPs, and the number of epochs; (d) finally, more advanced methods incorporate multiple branches and skip connections [EMH19]. InfoNEAT is the *first* NN-enhanced SCA that considers all these items through neuroevolution. Nevertheless, the most relevant studies are discussed below.

In the same vein as other ML tasks, for SCA, simpler models could be preferred when available resources are limited; however, the attack performance should not be compromised. The trade-off between these has been studied in the SCA-related literature. Conventionally, it has been thought that when attacks are harder to mount, e.g., in cases with noisy traces and countermeasure-protected designs, small (not using too many layers) NNs could not achieve the performance desired for a successful SCA; nonetheless, it has been shown that having a deeper network is not necessarily an advantage [vdVKPB20, WPP20]. Hence, for SCA, one of the hyperparameters to tune is the maximum number of layers. Besides the number of layers (a), the type of operation for each layer and their hyperparameters (b and c) have been considered in recent works. Random search within pre-defined ranges [PCP20, WPP20], and grid search[Wei20] have been considered to tune the hyperparameters. An example of hyperparameter tuning for choosing the type of operations is given in [PP20, KWPP21], where numerous choices for optimizers and loss functions have been explored. In [PBP20], the authors suggest tuning hyperparameters, while trying to prevent underfitting or overfitting by defining stopping criteria (rather than a pre-defined number of epochs) by relying on the mutual information between output layer activation functions (Softmax) and the data labels (for a comparison with InfoNEAT, see Section 5.2).

Arguably, for SCA, Zaid et al. [ZBHV20] proposed the first methodology devoted to the explainability and interpretability of model hyperparameters. For this, visualization techniques have been applied to determine the impact of CNN model hyperparameters related to the convolutional part, e.g., the number of filters. With regard to their impact, one attempts to find a suitable architecture with a minimized complexity, cf. [ZBHV20]. Nevertheless, their method suffers from issues discussed in [WAGP20] where the authors improve the configuration of CNNs and significantly reduce their size without compromising the attack performance. Similarly, towards making the NN-enhanced SCA more automated, [RWPP21] uses Reinforcement Learning (RL) while [WPP20] uses Bayesian optimization to explore different network architectures. Specifically, [WPP20] has done so through layer-level network morphism and Bayesian optimization as offered by Auto-Keras [JSH19], the core of their methodology. The rationale behind layer-level network morphism is to modify a trained neural network into a new architecture by applying different operations, such as inserting a layer or adding a skip-connection between layers. When employing such approaches, attention should be paid as obtained models might overfit [WPP20]. The work in [RWPP21] is devoted to finding CNNs that are small (in terms of the number of trainable parameters), but exhibit good attack performance; however, choosing the configuration is still (to some extent) guided by the expert through providing a random range of the hyperparameters' values (see Sections 4.2 and 4.5 in [RWPP21]).

InfoNEAT, on the other hand, tunes the whole range of parameters (a-d) considered for NAS and additionally, the weights (see Table 8 for a summary of the comparison, and Section 6.5 for a comparison between the time complexity of InfoNEAT and existing methods). An important feature of NEAT, at the core of InfoNEAT, is that it starts from a small network to optimize the networks more efficiently [SM02]. With the aid of information-theoretic measures, InfoNEAT is guided to stop without posing any limitation on the number of layers, but when the network is well-trained (Section 5.2). Moreover, for the purpose of SCA, an adequate fitness function is chosen (see Section 5.1). The stacking technique is also incorporated to scale up the number of classes that NEAT can handle as profiled SCA is a multi-class ML task with a high number of classes (Section 5.3).

3 Background and Mathematical Foundations

3.1 Notations

In this paper, the calligraphic letters, e.g., $\mathcal{X}$, are used to denote sets. Moreover, bold letters (e.g., $\mathbf{y}$) correspond to matrices and vectors. We use the standard notations for mathematical operators defined in the respective sections. For a random variable X, the corresponding lower-case letter x denotes realizations of X. Shannon's definition of MI and CMI is $\mathbf{I}(x;y) = \mathbf{H}(x) + \mathbf{H}(y) - \mathbf{H}(x,y)$ and $\mathbf{I}(x;y|x') = \mathbf{H}(x,x') + \mathbf{H}(y,x') - \mathbf{H}(x,y,x') - \mathbf{H}(x')$, where $\mathbf{H}$ denotes entropy or joint entropy: $\mathbf{H} = -\sum_{x \in X} p(x) \log p(x)$ and $p(x)$ denotes the probability of random variable X taking value x.

3.2 Profiled Side-channel Analysis

One of the most powerful forms of side-channel attack is the profiled SCA in which the adversary uses a device that he can control to build a *profiling model* that can then later be used to extract the encryption key from similar devices. After the introduction of profiling attacks in [CRR02], due to their close conceptual connection with ML tasks, ML-based attacks were proposed to enhance them. Thus, the profiled SCA has two steps: a *profiling* step and an *attack* (or training and testing as known in ML domain) step. During the first step (profiling), the adversary has access to a test device and can control its (guessable or public) inputs, which are a chunk of plaintext P, as well as a part of the cryptographic algorithm's secret key S that the attacker aims to disclose. The observations made by feeding these inputs can be seen as an estimation $\hat{\varphi}_s$ of the conditional probability distribution function for every possible $s \in \mathcal{S}$ as defined below cf. [BPS⁺18].

$$\varphi_s : (\mathbf{x}, s) \mapsto \Pr[\mathbf{X} = \mathbf{x} \mid (P, S) = (p, s)].$$

In other words, the traces can be used to estimate $\hat{\varphi}_s$. Precisely, for a given set of $\{p_i, s_i\}_{i=1}^{n}$, the attacker collects n traces $\{x_1^i, x_2^i, \cdots, x_k^i\}_{i=1}^{n}$, where each trace contains k features ($k \geq 2$). Based on this "profiling dataset", the adversary constructs a ML model (i.e., a leakage model) that can estimate the probability of inputs for each trace: $\hat{\varphi}_{X,P} : (\mathbf{x}, p) \mapsto \Pr[(P, S) = (p, s) | \mathbf{X} = \mathbf{x}]$ for every $s \in \mathcal{S}$.

During the second step, the attacker attempts to classify a set of N_{test} traces, the so-called test set corresponding to an unknown s, based on the above leakage model. Formally, the attacker should derive the label for a trace, similar to a ML classification problem: $\mathbf{y} = \hat{\varphi}_{X,P}(\mathbf{x}, p)$, for $\hat{s} \in \mathcal{S}$ so that $\hat{s} = \arg\max_{s\in\mathcal{S}} \mathbf{y}_k$, where $\mathbf{y}_k$ is the k^{th} entry in the vector $\mathbf{y}$. For N_{test} traces, a *score* based on the maximum-likelihood of each hypothetical key can be obtained as $\mathbf{d}_k = \prod_{i=1}^{N_{test}} \mathbf{y}_k^i$, where $\mathbf{y}_k^i$ is the k^{th} entry in the vector $\mathbf{y}^i$ corresponding to the i^{th} trace. Based on this score, the key hypotheses are ranked in a decreasing order based on the rank function (Equation (1)), from which the attacker chooses the key that is ranked first. The rank function is defined as follows cf. [BPS+18].

$$Rank(\hat{\varphi}, N_{test}) = |\{k \mid \mathbf{d}_k > \mathbf{d}_{k^*}\}|, \tag{1}$$

where k^* represents the key that has been used during the acquisition of the profiling traces. Note that the rank is computed for a collection of N_{test} traces from the test dataset, where N_{test} is increased gradually until the rank is minimized (lower the rank, higher the score). Since the rank is dependent on the traces used, it is common practice to compute the rank over different chunks of datasets and compute an *average rank*, also called the *guessing entropy* [MPP16]. We use the term "average rank" throughout this paper.

Realistic attacker: A successful and effective attack results in the average rank that equals zero, whereas the guessing entropy staying at r after mounting the attack means that the attacker must brute force 2^r different keys to recover the key [BCH+20]. In more realistic scenarios, it is suggested that the attacker is computationally powerful; however, instead of simply breaking the target by achieving the average rank equal to 0, the attacker attempts to break the target with a minimal number of traces [PHPG21]. An example of this is the attacker's power measured as the minimum number of traces tried by the attacker to reach the average rank < 20 given a model trained using profiling traces [PHJ+19].

3.3 Datasets

3.3.1 ASCAD Dataset

ASCAD is a dataset introduced in [BPS+18] which consists of electromagnetic (EM) radiations emitted from the software implementation of AES-128 protected with first-order Boolean masking and running on an 8-bit AVR microcontroller ATMega8515. This dataset is structured similar to a typical ML dataset and consists of training (or profiling) traces and testing (or attack) traces. There are two different versions of this dataset which we will name ASCAD$^{\text{fixed}}$ and ASCAD$^{\text{var}}$. ASCAD$^{\text{fixed}}$ dataset contains 50,000 profiling traces and 10,000 attack traces where each trace is acquired using the same fixed key and contains 700 sample points or features that represent various intermediate values related to the processing of the third S-box. ASCAD$^{\text{var}}$ on the other hand, contains 200,000 profiling traces and 100,000 attack traces. Here, one out of three profiling traces is acquired using a fixed key, while the rest are acquired using a random key. All attack traces are acquired using a fixed key as well. Each trace in the ASCAD$^{\text{var}}$ dataset consists of 1400 features. In addition, both of the datasets contain the *labeled* data, which represents the output of the third S-box during the first round as labels for each trace, leading to 256 classes. Note that throughout this paper, we consider this leakage and the *ID leakage model*, i.e, the leakage in the form of an intermediate value of the cipher leading to 256 classes; hence, to compare our results with SOTA approaches, we also take into account their results for ID leakage model.

Both of ASCAD$^{\text{fixed}}$ and ASCAD$^{\text{var}}$ datasets are publicly available [BPS+17] along with traces desynchronized by 50 and 100 samples window maximum jitter, which we will refer to as $\boldsymbol{ASCAD^{fixed}_{desync50}}$, $\boldsymbol{ASCAD^{fixed}_{desync100}}$, $\boldsymbol{ASCAD^{var}_{desync50}}$, and $\boldsymbol{ASCAD^{var}_{desync100}}$.

3.3.2 AES_HD Dataset

This dataset consists of EM traces acquired from an unprotected implementation of AES-128 on a Xilinx Virtec-5 FPGA with each encryption taking a total of 11 cycles [PHJ+19, KPH+19]. A total of 500,000 traces were captured with each trace consisting of 1,250 features. The dataset has been extended and made available [BJP20], which is used in this paper. The labels for each of the trace were generated using a HD based leakage model targeting the last round of AES encryption[3]. In this paper, we use 45,000 traces for profiling and 4,500 traces for testing and attack purposes.

3.4 Matrix-based Rényi Entropy

One of the recent and major breakthroughs in information theory is the new estimator of Rényi entropy in a matrix form as defined below. In particular, using this estimator, it has become feasible to understand the information flow without knowing the probability density functions [YP19, YWJP20]. This estimator is the basis of the NN selection and algorithm stopping criteria introduced in InfoNEAT.

Definition 1. *(cf. [YP19]) Given a set of n samples $\{x_1^i, x_2^i, \cdots, x_k^i\}_{i=1}^n$, where each sample contains k ($k \geq 2$) measurements, we define the kernels $\kappa_1 : \mathcal{X}_1 \times \mathcal{X}_1 \mapsto \mathbb{R}$, $\cdots$, $\kappa_k : \mathcal{X}_k \times \mathcal{X}_k \mapsto \mathbb{R}$ ($X_z = x_z^1, \cdots, x_z^n$ with $1 \leq z \leq k$), which are real-valued positive definite and infinitely divisible [Bha06, YP19]. The Rényi's α-order joint-entropy among k variables is*

$$\boldsymbol{J}_\alpha(\boldsymbol{X}_1, \cdots, \boldsymbol{X}_k) = \boldsymbol{S}_\alpha\left(\frac{\boldsymbol{A}_1 \odot \cdots \odot \boldsymbol{A}_k}{tr(\boldsymbol{A}_1 \odot \cdots \odot \boldsymbol{A}_k)}\right), \tag{2}$$

where $(\boldsymbol{A}_1)ij = \kappa_1(x_1^i, x_1^j)$, $\cdots$, $(\boldsymbol{A}_k)ij = \kappa_k(x_k^i, x_k^j)$. $tr(\cdot)$ and $\odot$ denote the transpose and Hadamard product operators, respectively. Furthermore, the function $\boldsymbol{S}_\alpha(\cdot)$ is defined as follows.

$$\boldsymbol{S}_\alpha(\boldsymbol{A}) = \frac{1}{1-\alpha}\log_2(tr(\boldsymbol{A}^\alpha)) = \frac{1}{1-\alpha}\log_2\sum_{i=1}^n \lambda_i(\boldsymbol{A}^\alpha), \tag{3}$$

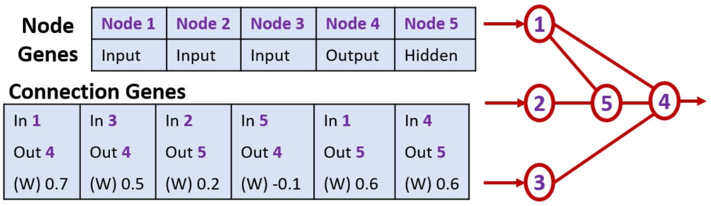
where $\lambda_i(\boldsymbol{A})$ denotes the i^{th} eigenvalue of $\boldsymbol{A}$.

There is a relationship between the matrix $\mathbf{A}$ as defined in Equation (3) and the Gram matrix $\mathbf{K}$. In this regard, $\mathbf{A}_{ij} = \frac{\mathbf{K}_{ij}}{n\sqrt{\mathbf{K}_{ii}\mathbf{K}_{jj}}}$. Moreover, the information quantities are estimated using the above mentioned matrix-based Rényi entropy with with $\alpha = 1.01$ to approximate Shannon's entropy as suggested in [GP13, GRP14].

4 NeuroEvolution of Augmenting Topologies

This section gives an overview of how the neuroevolution of augmenting topologies (NEAT) algorithm works as part of InfoNEAT. For more details, the reader is referred to [SM02]. The fundamental process taking place in NEAT is similar to the evolution of organism's genomes, where genomes represent NNs, see Figure 2. In particular, the main idea behind NEAT is the concept of neuroevolution, which searches for network topologies by means of evolutionary algorithms (EAs). In doing so, the EA optimizes hyperparameters of deep networks, specifically, the weights of individual neurons and their inter-connections are

[3] Although there are dissimilarity between the leakage model, round key and label calculation in relevant studies, i.e., [ZBHV20, WAGP20, PHJ+19], for the sake of comparison, the results as presented in their papers are considered here.



Node Genes	Node 1	Node 2	Node 3	Node 4	Node 5
	Input	Input	Input	Output	Hidden

Connection Genes

In 1	In 3	In 2	In 5	In 1	In 4
Out 4	Out 4	Out 5	Out 4	Out 5	Out 5
(W) 0.7	(W) 0.5	(W) 0.2	(W) -0.1	(W) 0.6	(W) 0.6

Figure 2: A typical genome or network created in NEAT. (W) denotes the weight assigned to each connection.

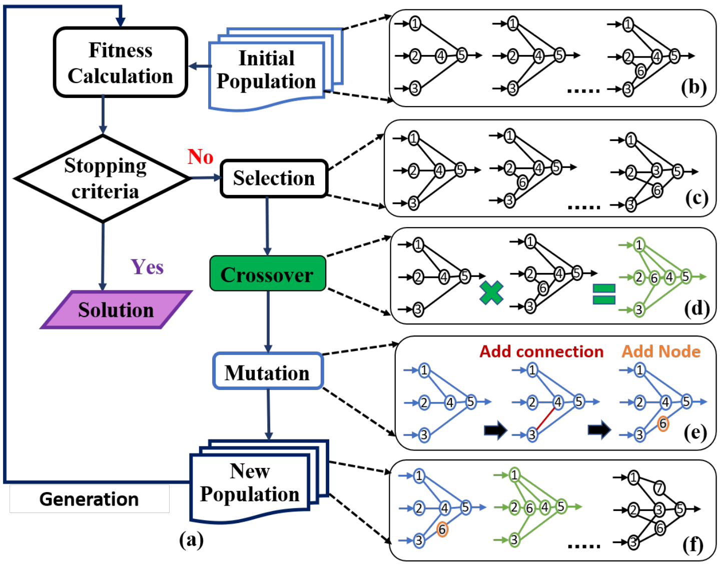


Figure 3: A general overview of the evolution mechanism involved in NEAT, and consequently, InfoNEAT. Note the multi branches and skip connections that have been used in step (e). Note that the population depicted here includes only one species.

evolved in each step, called a *generation*, see Figure 3(a). In the context of SCA, the number of generations is equivalent to the minimum number of attempts made by the attacker to tune the hyperparameters, cf. [PHPG21].

Although NEAT variants can be comprised of various types of NNs, e.g., convolutional NNs, we stick to a variant evolving MLP-like NNs in this paper. In fact, the configuration of NNs (so-called genomes) is different from typical multi-layer perceptrons (MLPs), where each hidden neuron is connected to every neuron in the previous and next layers. This irregular topology (compared to the structure of MLPs) discovered by NEAT is one of the most important aspects of NNs and contributes to their special behavior. As explained in, e.g., [SM02], the selection of topology throughout evolution as performed by NEAT brings various advantages, including (1) saving time spent on selecting the topology manually, (2) faster learning speed due to topology evolution, i.e., minimizing and growing it incrementally, and (3) along with the evolution of weights, it significantly enhances the performance of the NNs.

NNs generated by NEAT are grouped in species based on their randomly chosen structures. In each generation, several species (i.e., sub-populations) exist that share a similar topology (i.e., the way that the nodes are connected to each other). In this regard, the genomes in different species can have different sizes. Independent from other species, each species evolves proportionally to its fitness so that the weights of NNs are learned. Learning the weights of NNs has conventionally been performed without backpropagation. Figure 3 illustrates the main steps involved in NEAT (see Algorithm 1) as explained below.

Initial population (line #1 in Algorithm 1): The population considered by NEAT contains NNs, i.e., the genomes (see Figure 2) that are denoted by $g^j_{t,i}$, where superscript j $(1 \leq j \leq \ell)$ indicates the j^{th} layer. Moreover, the subscripts t and i denote the generation and the i^{th} NN in the species, respectively. In the first generation $t = 1$, the total number of NNs in all species is N and each species contains $N_{s,t=1}$ NNs. For the sake of simplicity and without loss of generality, we explain the learning/evolution phase for one species. Features at the input of each genome and the output class labels are denoted by $\mathbf{X}$ and $\mathbf{Y}$, respectively. The evolution process begins with a set of NNs with minimal complexity. In the first generation, in a species, each NN $g^1_{1,i}$ $(1 \leq i \leq N_{s,1})$ is composed of $j = 1$ hidden layer. The output of each layer is a function of the input $\mathbf{X}$, the matrix of weights between j^{th} and $(j+1)^{\text{th}}$ layers $\mathbf{W}_j$ and the vector of biases for the $(j+1)^{\text{th}}$ layer $\mathbf{b}_{j+1}$ $(1 \leq j \leq \ell)$. More specifically, for a set of weights and biases $\theta_j = \{\mathbf{W}_j, \mathbf{b}_{j+1}\}$, the function $f : \mathbb{R}^{|\theta_j|} \to \mathbb{R}^{|\theta_{j+1}|}$ is applied to the weights, biases and inputs to generate the output at the $(j+1)^{\text{th}}$ as follows: $\hat{\mathbf{y}}_{j+1} = f(\hat{\mathbf{y}}_j \mathbf{W}_j + \mathbf{b}_{j+1})$, where $\hat{\mathbf{y}}$ and $f(\cdot)$ denote the estimated label and the activation function, respectively.

Fitness evaluation and selection (lines #3–5 in Algorithm 1): One of the differences between the NNs usually applied in various domains and NEAT as a neuroevolutionary algorithm is that all the NNs in a species are evaluated based on a fitness function. According to this probabilistic evaluation, the algorithm decides which NNs would be successful in the next generations. This is accomplished by assigning a fitness value to each genome. Afterward, the fitness values are taken into account by a selection operator, e.g., tournament selection [GM21] that examines a small random subset of the population to find the fittest individual (see Figure 3(c)).

Crossover (lines #6 in Algorithm 1): In the next step, crossover operation, also known as recombination, is performed, where the genetic information (e.g., the parameters of the NNs) of two selected individuals are combined as shown in Figure 3(d). For this, genomic distance is considered to define species within which the combination happens. This distance is a combination of the number of disjoint genes or nodes (i.e., derived from different ancestors), as well as the average weight differences of matching genes or nodes between two networks [SM02]. Within a species, the crossover is based on the historical marking, used to label each connection based on their ancestry. For crossover between two parents, their connection genes are lined up according to their historical markings. The genes that have the same markings are swapped at random while the remaining genes or disjoint genes are stacked at the end to create a new offspring, cf. [SM02].

Mutation (lines #7 in Algorithm 1): For each non-crossover individual, random changes are made into the NNs based on the mutation operator, as shown in Figure 3(e). Specifically, nodes and connections are added incrementally to these NNs in each generation to update configurations and parameters. The steps above are repeated until a condition is met, e.g., typically when a maximum number of generations are executed.

5 InfoNEAT

This section gives details on InfoNEAT, built upon NEAT. Variants of the NEAT algorithm have been developed accordingly, which are used for various tasks, including regression [HMA17], classification [HMA17, SM02], and reinforcement learning (RL) [SM15]. Nevertheless, in this paper, we stick to NEAT's application to classification. First, we explain which fitness function can be applied to tailor NEAT for SCA. It is followed by how the information-theoretic stopping criteria are defined for InfoNEAT. Finally, we discuss the challenges that were faced when designing and implementing InfoNEAT.

Algorithm 1: NEAT Algorithm cf. [GM21]

```
Input: Batch size data of input dataset $D$; Population size $N$;
Initial number of hidden nodes $n_h$; Fitness threshold $L_{TH}$;
Connection add probability $P_c$; Node add probability $P_n$;
Number of generations $T$; Compatibility threshold $t_c$;
Weights and bias mutation rate $P_w$ and $P_b$, respectively;
Initialization: Generate a set of genomes or networks $g_{t=1,i}$ $(1 \leq i \leq N)$ randomly
  based on $N$ and $n_h$;
for $t = 1, 2, ..., T$ do
    Fitness evaluation: Compute the fitness (e.g., cross-entropy loss) for $g_{t,i}$;
    if fitness values $L_{t,i} \leq L_{TH}$ then break; else continue;
    Selection: Select the best individuals and produce a new generation $g_{t+1,i}$ from $g_{t,i}$;
    Crossover: Individuals with genomic distance $< t_c$ are part of the same species and
      are selected for crossover;
    Mutation: For each individual $g_{t,i}$, the mutation of weights and bias is performed
      based on $P_w$ and $P_b$ respectively and the structural mutation is performed based on
      $P_c$ and $P_n$;
    end
end
```

5.1 Adequate Fitness Function for SCA

The fitness function evaluating the performance of the NNs in each generation helps to select the NNs that will be evolved in the next generations. For a given genome (e.g., i^{th} NN) in the t^{th} generation, InfoNEAT employs the categorical cross-entropy loss function to compute the loss $L_{t,i}$ formulated as

$$L_{t,i} = -\sum_{k=1}^{|\mathbf{y}_{t,i}^{\ell}|} y_k \log \hat{y}_k,$$

where y_k and $\hat{y}_k$ are the k^{th} entries in $\mathbf{y}_{t,i}^{\ell}$ and $\hat{\mathbf{y}}_{t,i}^{\ell}$, respectively. When it comes to computing the cross-entropy for classification tasks, the terms "cross-entropy" and "negative log-likelihood" are used interchangeably [Mur12]. This is of great importance to SCA as it has been proven that negative log-likelihood (NLL) is inversely related to "perceived information" [RSVC+11, MDP20, BHM+19]. The latter refers to the generalization of the mutual information between the side-channel traces and the leakage profiling model (i.e., the ML trained on the traces). In other words, the perceived information quantifies how well the ML model is trained. More interestingly, minimizing the NLL loss function (similarly, cross-entropy) during NN training is asymptotically equivalent to maximizing the perceived information and improving the trained NN performance cf. [MDP20].

5.2 Information Theoretic Criteria

Conventionally, a maximum number of generations is defined, usually accompanied by another stopping criterion relying on the heuristic, e.g., if the accuracy of the best NN (i.e., the NN with the highest accuracy) does not improve after some generations, halt the evolution. To address this, InfoNEAT applies information theory-based approaches to select the genomes (NNs) to be evolved (see the part highlighted in gray in Algorithm 1) as well as to stop the evolution process. Although InfoNEAT shares a similarity with [PBP20], namely relying on an information-theoretic measure to stop the training, we neither train NNs by minimizing the information bottleneck (IB) function [AG19], nor consider the so-called fitting and compression phases [SBD+19] as applied in [PBP20]; hence, the issues inherent to these methods [SBD+19] are avoided by InfoNEAT.

In our approach, each mutated NN is seen as a randomly permuted one. Intuitively, making any change in a NN (i.e., evolving the NN in the next generation) should result in a

Algorithm 2: Genome selection based on fitness and CMI

```
   Input: Number of genomes within a species $N_s$; All the genomes in a species $g^j_{t,i}$
          ($1 \leq i \leq N_s$ and $1 \leq j \leq \ell$);
   Result: Best genome within a species $g^j_{t,*}$;
1  Initialization: Calculate the loss values $L_{t,i}$ for $1 \leq i \leq N_{s,t}$;
2  if number of genomes with $\min_{1 \leq i \leq N_s} (L_{t,i})$ is one then
3      break;
4  else
5      $G = \{g_{t,i} | \, L_t = \min_{1 \leq i \leq N_{s,t}} (L_{t,i})\}$ ;
6      for $g_{t,i} \in G$ do
7          for $j = \ell, \ell - 1, \cdots, 1$ do
8              CMI $= \mathbf{I}\left(\hat{\mathbf{y}}^j_{t+1,i}; \mathbf{y} | \hat{\mathbf{y}}^j_{t,i}\right)$;
9              if number of genomes with $\min_{1 \leq i \leq N_{s,t}} (CMI)$ is one then
10                 Output $g_{t,i}$ and break;
11             else
12                 continue;
13             end
14         end
15     end
16 end
```

monotonic decrease in the conditional mutual information (CMI) that is $\mathbf{I}\left(\hat{\mathbf{y}}^j_{t+1,i}; \mathbf{y} | \hat{\mathbf{y}}^j_{t,i}\right)$, where $\hat{\mathbf{y}}^j_{t,i}$ denotes the output of a given layer in $g^j_{t,i}$. This is according to permutation tests [Goo13] and well-described in [FRWV07] as an impact of adding useless variables (i.e., parameters associated with evolved NN). Below, we formalize this more precisely.

5.2.1 Selection of the Best Genomes

The selection criterion offered in NEAT is based on the loss function as explained in Algorithm 2, line #2. However, if the genomes perform almost similarly well as can happen in the last generations, loss-based selection criteria would not be effective. For this purpose, we combine this with CMI-based criterion explained below.

As discussed before, from each species, a set of NNs are evolved in the next generation. The evolution of NNs from one generation to the next is considered as a permutation (without permuting the corresponding $\mathbf{y}$). Precisely, when a genome is evolved to another one in the next generation, the mutual information between the output of the $(j)^{\text{th}}$ layer and the output class labels in generation t and $t+1$ are $\mathbf{I}\left(\hat{\mathbf{y}}^j_{t,i}; \mathbf{y}\right)$ and $\mathbf{I}\left(\hat{\mathbf{y}}^j_{t+1,i}; \mathbf{y}\right)$, respectively. The difference between these two is the CMI $\mathbf{I}\left(\hat{\mathbf{y}}^j_{t+1,i}; \mathbf{y} | \hat{\mathbf{y}}^j_{t,i}\right)$, which is non negative and rarely equals zero in practice because of the statistical variation [Cov99, VCB14]. Hence, from one generation to the next, making an unnecessary change to the well-trained genome (almost surely) increases the mutual information between $\hat{\mathbf{y}}^j$ and $\mathbf{y}$ that is against the training goal. The higher this increase is, the less useful the change made to the genome would be.

With regard to this principal, to choose one or more NNs from a species, InfoNEAT discards NNs (e.g., $(i+1)^{\text{th}}$ genome), when the CMI $\mathbf{I}\left(\hat{\mathbf{y}}^j_{t+1,i+1}; \mathbf{y} | \hat{\mathbf{y}}^j_{t,i+1}\right)$ is not significantly smaller than $\mathbf{I}\left(\hat{\mathbf{y}}^j_{t+1,i}; \mathbf{y} | \hat{\mathbf{y}}^j_{t,i}\right)$ (see lines #6–15 in Algorithm 2). This means that if the mutated NNs cannot outperform the respective ones in the previous generation, those NNs are discarded. To implement this CMI-based criterion, however, technical difficulties regarding the computation of the CMI should be resolved as discussed below.

CMI computation based on Matrix-based Rényi's α-entropy: This notion encompasses the extension of Shannon's entropy; however, in its traditional form, the probability distribution function (PDF) should be accurately estimated. To cope with this, we follow the procedure presented in [YGJP19, GRP14] that relies on the principle of the Gram matrix obtained from evaluations of a positive definite kernel from data samples, see Definition 1. This allows a direct estimation of the entropy and joint entropy between two or multiple variables from data without PDF estimation. The multivariate matrix-based Rényi's α-entropy can be applied to estimate the CMI in high-dimensional space as follows cf. [YP19] (see Equation (2)). Suppose that $|\hat{\mathbf{y}}^j_{t+1,i}| = k_{t+1}$ and $|\hat{\mathbf{y}}^j_{t,i}| = k_t$ denoting the number of outputs at the j^{th} layer of the i^{th} genome in the generations $t+1$ and t, respectively. Then, the CMI is

$$\begin{aligned}\mathbf{I}_\alpha(\{\mathbf{C}_1, \mathbf{C}_2, \cdots, \mathbf{C}_{k_{t+1}}; \mathbf{B}|\{\mathbf{A}_1, \mathbf{A}_2, \cdots, \mathbf{A}_{k_t}\}) =\\ = \mathbf{S}_\alpha\left(\frac{\mathbf{A}_1 \odot \cdots \odot \mathbf{A}_{k_t} \odot \mathbf{C}_1 \odot \cdots \odot \mathbf{C}_{k_{t+1}}}{\operatorname{tr}(\mathbf{A}_1 \odot \cdots \odot \mathbf{A}_{k_t} \odot \mathbf{C}_1 \odot \cdots \odot \mathbf{C}_{k_{t+1}})}\right) +\\ +\mathbf{S}_\alpha\left(\frac{\mathbf{A}_1 \odot \cdots \odot \mathbf{A}_{k_t} \odot \mathbf{B}}{\operatorname{tr}(\mathbf{A}_1 \odot \cdots \odot \mathbf{A}_{k_t} \odot \mathbf{B}}\right) - \mathbf{S}_\alpha\left(\frac{\mathbf{A}_1 \odot \cdots \odot \mathbf{A}_{k_t}}{\operatorname{tr}(\mathbf{A}_1 \odot \cdots \odot \mathbf{A}_{k_t}}\right) -\\ -\mathbf{S}_\alpha\left(\frac{\mathbf{A}_1 \odot \cdots \odot \mathbf{A}_{k_t} \odot \mathbf{B} \odot \mathbf{C}_1 \odot \cdots \odot \mathbf{C}_{k_{t+1}}}{\operatorname{tr}(\mathbf{A}_1 \odot \cdots \odot \mathbf{A}_{k_t} \odot \mathbf{B} \odot \mathbf{C}_1 \odot \cdots \odot \mathbf{C}_{k_{t+1}})}\right). \end{aligned} \tag{4}$$

In Equation (4), $\mathbf{A}_1, \cdots, \mathbf{A}_{k_t}, \mathbf{B}, \mathbf{C}_1, \cdots, \mathbf{C}_{k_{t+1}}$ denote the Gram matrices evaluated over $\hat{\mathbf{y}}^j_{t,i}$, $\mathbf{y}$, and $\hat{\mathbf{y}}^j_{t+1,i}$, respectively. Moreover, $\mathbf{S}_\alpha(\cdot)$ and $\odot$ denote the Rényi's α-entropy and the Hadamard product (see Definition 1). This equation is the core of Algorithm 2 (line #8) for how InfoNEAT selects the genomes to be evolved. Note that this algorithm is run in every generation when the genomes' parameters are updated.

As can be seen in Algorithm 2, the CMI is computed to compare the genomes in two consecutive generations in a layer-wise manner. Interestingly, this comparison is first applied to the last layers of the genomes in the sense that if the CMI values are the same for the last layers of the genomes $j = \ell$, the second to last layer $j = \ell - 1$ is considered and so on. This is due to the fact that the layer-wise mutual information between the labels and the outputs of a layer is minimized at the last hidden layer. This value increases so that its maximum can be observed at the first hidden layer [SZT17, SBD+19, ZBH+16]. Therefore, our layer-wise comparison implies that for the best genome selected through Algorithm 2, the mutual information between the labels and the outputs of the last layer can stay minimized [Fle04].

5.2.2 Stopping Criteria

Our method to deal with the definition of a stopping criterion relies on the notions discussed for selecting the best genome from a species. In fact, the stopping mechanism can be seen as a continuation of the process associated with best genome selection. Compared to one of the most relevant approaches presented in [YP19], no threshold is needed to stop the algorithm. InfoNEAT makes a decision based on the change in the CMI value, following the so-called CMI-permutation concept. The permuted NNs are evolved by InfoNEAT automatically and randomly evolved from one generation to the next. To halt the process, we monitor not only the CMI values, but also the fitness values that are the cross-entropy loss (see Section 4).

Algorithm 3 explains this further. In the initialization phase (line #1), the loss function for the genomes in the current generation $t+1$ and the best genome $g^\ell_{t-1,*}$ delivered by the Algorithm 2 is computed. An interesting observation is that to define the stopping criteria, similar to Algorithm 2, the last layers of the genomes are taken into account. This is due to the fact that the last hidden layer carries the highest level of mutual information between the output of hidden layers and the labels [ZBH+16, SZT17]. In this regard, and according to the CMI-permutation principle, the stopping criteria encompasses the CMI between the output of the last hidden layer and the labels conditioned on the best genome created in the previous generations (lines #2,7 in Algorithm 3). Moreover, the

Algorithm 3: Stopping criteria based on fitness and CMI

```
   Input: Best genomes in the current generation $g^\ell_{t,*}$ and in the generation $g^\ell_{t-1,*}$ (i.e., the
          output of Algorithm 2); Genomes evolved from $g^\ell_{t,*}$ in the next generation $g^\ell_{t+1,i}$
          $(1 \leq i \leq N_{s,t+1})$;
   Result: Stop the training or evolution process and deliver the best genome $g^\ell_{T,*}$ $(T \geq 2)$;
 1 Initialization: Calculate the loss $L_{t+1,i}$ for $g^\ell_{t+1,i}$ as well as $L_{t,*}$, i.e., the loss for the
     best genome $g^\ell_{t,*}$;
 2 $\mathrm{CMI}_t = \mathbf{I}\left(\hat{\mathbf{y}}^\ell_{t,*};\mathbf{y}|\hat{\mathbf{y}}^\ell_{t-1,*}\right)$;
 3 for $i = 1, \cdots, N_{s,t+1}$ do
 4     if $L_{t,*} < L_{t+1,i}$ then
 5         break and $g^\ell_{T,*} = g^\ell_{t,*}$;
 6     else
 7         $\mathrm{CMI}_{t+1} = \mathbf{I}\left(\hat{\mathbf{y}}^\ell_{t+1,i};\mathbf{y}|\hat{\mathbf{y}}^\ell_{t,*}\right)$;
 8         if $CMI_t < CMI_{t+1}$ then
 9             break and $g^\ell_{T,*} = g^\ell_{t,*}$;
10         else
11             continue;
12         end
13     end
14 end
```

degradation in the loss (i.e., an increase in the cross-entropy loss) is considered as in the line #4 in Algorithm 3. If the degradation is not observed, the CMI values are compared to ensure that the permuted genomes, i.e., the NNs in generation $t+1$, outperform their respective descendant $L_{t,*}$. If not, the algorithm halts and outputs $L_{T,*} = L_{t,*}$ that is the best genome at the last generation denoted by T. *Note that no stopping criterion has been previously defined for the NEAT algorithm, as shown in Algorithm 1.*

5.3 Training and Testing Phases

The training phase of InfoNEAT involves running the algorithms corresponding to evolution process enhanced by the best genome or network selection, and training stopping, i.e., Algorithms 1-3. Neuroevolution techniques in Algorithms 1 evolve the weights and structure of a network from a simple starting point, and usually develop a minimal and a generalizable NN. However, it is challenging for the NEAT to optimize all parts of the network if the learning task concerns multi-class classification [MAB$^+$18]. To tackle this, InfoNEAT uses the One-vs-All (OvA) classification technique to develop m sub-models (i.e., NNs or genomes) corresponding to the number of labels or classes (i.e., 256 sub-keys). This is similar to what has been proposed in the context of neuroevolution in [MAB$^+$18]. However, the methods for obtaining the final classification result (i.e., the class an unseen trace belongs to) are different. The approach employed in [MAB$^+$18] simply selects the class, whose corresponding sub-models outputs the highest probability. This is not helpful when considering SCA with a larger number of classes (256 vs. 26 considered in [MAB$^+$18]) making the classification more prone to noise [PHJ$^+$19]. Under such a complicated scenario, soft classifiers, e.g., logistic regression, work better [LZW11]. Hence, InfoNEAT takes advantage of the stacking method, where a meta-learner (i.e., a classifier at the second layer, see Figure 4) is trained to combine the output of the sub-models so that the final, stacked model outperforms each sub-model. Notice the difference between stacking of ML models trained for a multi-class task and our approach, where a set of m OvA models are stacked to combine them and further improve the predictive performance. For this, we apply logistic regression, trained to best combine the predictions from each of the sub-models. This is, of course, advantageous as the class label prediction can be improved through training not only the sub-models, but also the meta-learner (as an example, compare the green and orange curves in Figure 7).

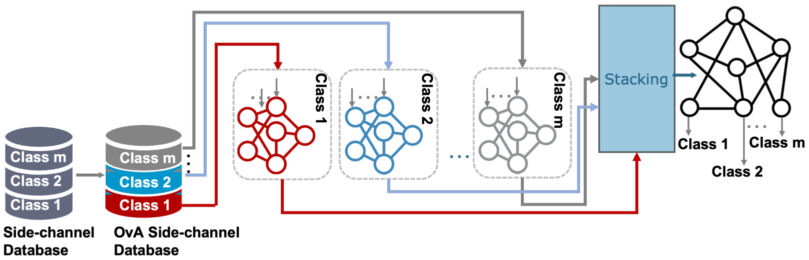


Figure 4: Schematic of the NEAT algorithm configured to deliver a stacked model. For each class, a sub-model is trained by feeding batches from One-vs-All (OvA) database.

OvA vs. ensemble learning proposed in [PCP20]: Although the bagging ensemble technique applied in [PCP20] and our stacking method share some similarities, i.e., being from the same ensemble learning category of ML algorithms, there is a crucial difference between them. InfoNEAT combines $m = 256$ sub-models, corresponding to sub-keys, and deliver *one* model (i.e., stacked model, here denoted by M). [PCP20] has suggested combining 50 models trained on *all sub-keys*, i.e., $M_1, \cdots, M_{50}$, through the bagging technique; hence, their proposed technique is different in nature from ours.

Training the sub-models: We train the sub-models by running InfoNEAT algorithm m times using m different, respective sub-datasets, see Figure 4. If k-fold cross-validation is used, each of these m different datasets contains k folds. For each label or class, we make sure that the dataset contains an equal number of traces belonging and not belonging to that particular class as required by the OvA method. The labels in these datasets are then modified accordingly using one-hot encoding where the length of each label is m. If the data belongs to the particular class, only the index corresponding to that class is 1. All the indices in the labels for traces not belonging to the class are set to 0.

Training the stacked model: After m number of sub-models are trained, a stacked model is trained which basically combines the predictions from all the sub-models and outputs a final prediction as shown in Figure 4. To train the stacked model (meta-learner), a dataset is prepared by involving the predictions from all the sub-models.

Testing phase: The trained meta-learner is used against the test dataset to return a set of predictions (or **average ranks** in the case of SCA). Notice that we do not use the common ML metrics to test our model due to the fact that the ML metrics, e.g., the accuracy, do not provide relevant information to the attackers; hence, it is not guaranteed that a model with good performance based on ML metrics necessary performs well in the case of SCA [PHJ$^+$19]. If k-fold cross-validation is applied, the training/testing process is repeated k times, and the average of the average ranks is reported.

5.4 Discussion of Practical Implementation

Various practical challenges were encountered and resolved during InfoNEAT's design and execution. They are itemized and discussed below along with pointers to our results.

Bias and variance in NNs: A common problem with stochastic algorithms involved in training the NNs is the high variance in the network similar to the case of SCA [vdVP19]. In other words, every time the model is fit to a new data point, the network has different sets of weights and parameters which in turn makes different predictions. To deal with this, the weight initialization method is chosen carefully for InfoNEAT (see Appendix B) and the evolution is further controlled by monitoring the fitness and CMI values. On top of this, stacking is used that has also been shown to produce models with lower bias compared to the sub-models used. Furthermore, for datasets with fixed key, we employ the k-fold cross-validation technique to reduce the variability of the stacked model associated with learning new data points (see Figure 12 in Appendix B showing an example of our

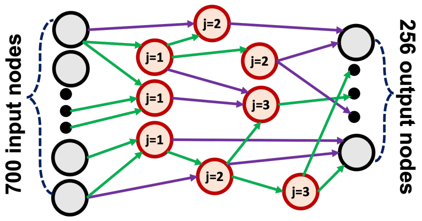


Figure 5: Example of the irregular InfoNEAT architecture used for SCA with 5 layers (3 hidden layers based on the node connections). The nodes in one layer can have a direct connection to a layer other than the immediate next layer (as highlighted in purple), e.g., the top input node has a direct connection to a node in hidden layer $j = 2$, but not $j = 1$.

sub-models that is neither underfitting nor overfitting).

Search space explored by NEAT: The search space in the case of NNs is the set of all possible weight configurations – the wider and deeper the network is, the larger the search space is. In that sense, the architecture of the network (i.e., the number of layers and number of hidden nodes) can also be considered as part of the search space since they affect the size of the network. Neuroevolution techniques such as NEAT evolve both the weights and NN architecture through crossover and mutation [GM21]. Thus, compared to traditional optimizers (such as the stochastic gradient descent, SGD) which only optimize the weight, the search space for neuroevolution techniques is considerably larger [AWD19]. In InfoNEAT, this challenge is simplified. InfoNEAT starts the evolution process from small sized network and uses the evolutionary operators, namely the mutation and crossover operators, to explore the search space. But unlike NEAT, InfoNEAT uses the CMI values rather than just the fitness values to help guide evolution. This also helps to achieve an optimally sized network as the hidden nodes are only added if the CMI values do not decrease monotonically.

Evaluation of multiple *distinct* networks: A salient feature of NEAT that makes it preferable over other network optimization methods is that NEAT evaluates a population of different networks all at the same time. However, this is only true if the networks are inherently distinct. One of the NEAT methods that help maintain diversity among networks is *speciation*. Speciation has been typically effective for penalizing similar networks by looking at the fitness and the structure of the network. The speciation parameters thus have to be tuned very carefully because of the associated time complexity of evaluating a typical SCA network and converging towards a solution. We discuss some of these parameters in more detail in Section 6.1.1, namely the *compatibility threshold* which helps divide a population of genomes into different species where the genomes between two species are inherently distinct.

Automatic evaluation of CMI for analyzing and finding the best genome every generation and implementing the stopping criteria: The methods proposed in the literature calculate the CMI for typical CNNs and MLPs to determine the number of filters and select features, respectively [YP19, YWJP20]. However, the NN evolved using NEAT is an unusual network as nodes from multiple layers can have a direct connection to each other as opposed to a typical MLP network, see Figure 5. We have addressed this by calculating the CMI for each node based on all of its connections.

6 Results

6.1 Experimental Setup

Experiments in this section are conducted on different databases as introduced in Section 3.3. Moreover, the datasets are shuffled to improve the learning process by avoiding any visible or invisible bias induced during the data collection phase. This step has been considered

Table 1: InfoNEAT parameters to be set by the user.

Parameters	**Description**	**Values**
Parameters related to evaluation of multiple distinct networks		
Population size (N)	Number of genomes or networks considered in a generation.	16
Initial Compatibility threshold (t_c)	Individuals whose genomic distance is less than this threshold are considered to be part of the same species.	1.8
Initialization of weights and biases - using Xavier technique		
Weights and biases {init_mean, init_variation min, max}	*Init_mean* and *init_variation* refer to the Xavier-based distribution of the weights at the start of the algorithm. Weights and biases beyond {min, max} range are clamped.	{0, *init_variation*, -3×*init_variation*, 3×*init_variation*}

optional in the SCA-related literature [BPS+18], although from ML point of view, it is helpful to incorporate into the learning process. All the experiments are run, without any special pre-processing of the datasets, on a high-computing cluster with a total of 8 CPUs allocated per task and a total memory of 80 GB. The CPUs are the Skylake Dell C6420 model with Xeon Gold 6142 processors. The rank function provided in the ASCAD package [BPS+18] is used in all the experiments described in this section.

Training and testing sub-dataset Preparation: First, to address the imbalance in the dataset, we create new balanced datasets by performing a data-level method, namely the random undersampling [PHJ+19]. This means that only the maximum possible number of data equally for each class is used. For instance, for ASCAD$^{\text{fixed}}$ dataset, we found this number to be 150, i.e., 150 traces per class are used to create new balanced datasets. Interestingly, InfoNEAT is able to recover the sub-key even when the number of traces per sub-key is reduced (see Section 6.3.1). Therefore, it is not needed to rely on oversampling that may lead to overfitting [PHJ+19]. After dealing with the imbalance in the dataset, for datasets with a fixed key for both training and testing data, we perform the *k-fold cross-validation* (see Section 5.3); otherwise, the trained model is applied against a separate testing set. Regardless of whether k-fold cross-validation is applied, the training set is used for training the sub-models as well as the stacked model. To train the stacked model, a 9:1 split of the training batch size is used as suggested in the literature [SKJ21].

6.1.1 Network and Configuration Parameters

As discussed in Section 4, the resultant NNs evolved using NEAT are usually quite small (in terms of the number of layers and nodes), yet still quite effective depending on the problem at hand. To obtain such NNs, some parameters as summarized in Table 1 should be set before training the NNs through InfoNEAT. After setting these, no change in them is required during the evolution of NNs. In other words, per learning task (i.e., per dataset), these parameters are set once and remain unchanged (see Appendix B for more details). Furthermore, for output nodes, we have selected the *softmax* as our activation function which is one of the commonly used output layer activation function in multi-class classification problems. Softmax assigns probability values to each class or output node such that the sum of these probabilities is 1. We apply the numerically stable softmax proposed in [GBC16] in order to deal with the overflow issue (i.e., exploding gradient problem) as observed in ML- and SCA-related studies [GBC16, KWPP21].

Parameters related to the initialization of weights and biases, and the batch size: These should be chosen carefully by the user, as explained in Appendix B (see Table 1). For ASCAD$^{\text{fixed}}$ dataset, for instance, the batch size is 150. Note that this should be done for *any* deep learning task regardless of the algorithm to be employed. After the initialization, the weights and biases are tuned by InfoNEAT over generations. Additionally, what is promised and offered by InfoNEAT is the automatic selection of the configuration and tuning of the hyperparameters, including the number of hidden layers, the number of nodes per layer, etc. Comparing these to the parameters to be tuned by the user (only the ones in Table 1), we can observe a drastic reduction in user effort.

Parameters related to evaluation of multiple distinct networks: *Population size* can be set based on how many NNs the user wants to evaluate at the same time.

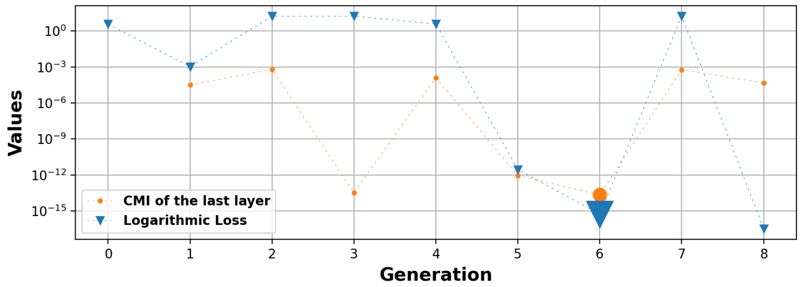


Figure 6: Log loss and CMI-based values for the last layer versus different generation. These values are obtained for the best genome at each generation. For better readability, the results for one randomly selected class from ASCAD$^{\text{fixed}}_{\text{sync}}$ dataset is presented. Note that if the CMI-based criterion has not been considered, the learning might have been stopped in generation 3, where the fitness (log-loss) value does not decrease; however, the reduction in the CMI indicates that the model is still learning. Dashed lines are drawn for the sake of readability.

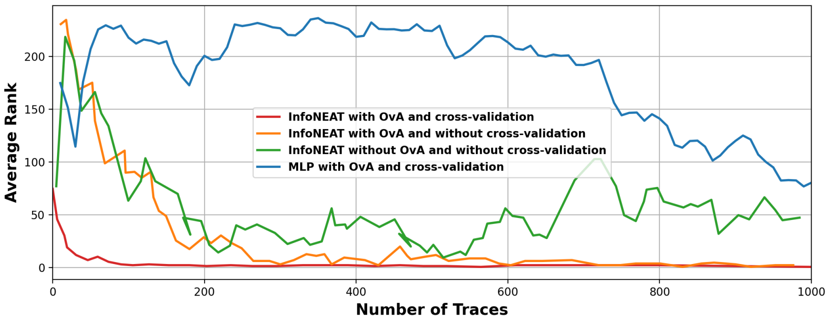


Figure 7: Comparison of average rank obtained for different models obtained with and without stacking and cross-validation. For this experiment, ASCAD$^{\text{fixed}}_{\text{sync}}$ dataset is used.

Since most of the experiments were conducted on a 16-core machine, we set this to 16 to efficiently evaluate multiple NNs. The higher the population size is, the longer a generation takes. Intuitively, this value shows how many NNs are evaluated simultaneously. *Initial compatibility threshold* is a NEAT parameter that helps maintain diversity in a population of genomes or NNs by comparing the genomic distance of the network with this value (see Section 4). Here diversity means how different NNs are configured and tuned in a generation. Larger thresholds result in having less number of species, i.e., if the population size is small, a modest threshold should be chosen (see [Ome19] for more information). Note that the user only defines the initial value of this threshold, and NEAT adjusts that in the next generations automatically.

6.2 Examining the Characteristics of InfoNEAT

Impact of CMI-based Criteria: As explained in Algorithm 2, to select the best genome, using log loss as a metric would not be sufficient. This is due to the subtle difference between the configurations of NNs in a species that can result in obtaining more than one NN with the same performance. Figure 6 shows the progression of both the fitness and CMI values for the best genome (or network) during each generation for a randomly chosen class. We also employ the CMI-based criteria to stop the evolution process at the right time as described in Algorithm 3. Whenever the CMI value and log loss both start to degrade, then we select the best genome from the previous generation as our ultimate model for the class. Thus, in Figure 6, we select our ultimate model from generation 6 rather than generations 7 or 8.

Table 2: Comparison of the time taken for InfoNEAT versus NEAT to train a sub-model for each ASCAD$^{\text{fixed}}$ dataset.

	With InfoNEAT		**With NEAT**	
ASCAD$^{\text{fixed}}$ dataset	**Average no. of generations**	**Average generation time [s]**	**Average no. of generations**	**Average generation time [s]**
Sync	8	67.384	17	91.426
Desync50	9	69.805	19	103.754
Desync100	11	68.361	20	120.682

Training an Effective Stacked Model: After the configuration parameters are set and the weights and biases appropriately initialized, we now enter the *training phase*. As discussed in Section 6.1, we train 256 different sub-models for the 256 classes using the balanced ASCAD databases and by employing the OvA technique. For the experiment on ASCAD$^{\text{fixed}}$, the stacked model is trained using 20 traces per class as discussed before. As shown in Figure 7, with stacking and cross-validation involved, the test metric (the average rank in our experiments) improves drastically compared to without stacking and cross-validation.

Impact of Irregular Architecture of NNs: To give a better understanding of different aspects of InfoNEAT, especially the importance of architecture delivered by our algorithm, we have designed the following experiment. Similar to InfoNEAT, the OvA strategy is used to make 256 binary classification tasks from the multi-class (i.e., 256-class) classification problem underlying the SCA. Each binary task associated with a sub-key is handled by training an MLP on a set of traces from the corresponding class and other classes as performed in experiments on InfoNEAT. In order to provide a comparison, the number of hidden layers and nodes in the MLPs are set equal to the maximum of those in the NNs generated by InfoNEAT. Moreover, the number of epochs, number of traces in a batch, and number of training traces are similar to what has been set for InfoNEAT. The average rank presented in Figure 7 shows a clear difference between InfoNEAT and MLPs combined through OvA strategy. This result indeed indicates the major role played by the irregular configuration of the NNs generated by InfoNEAT.

Time-complexity for InfoNEAT Training: All the steps mentioned in Section 6.1, such as the configuration of NEAT parameters, initialization of weights and batch size, and the inclusion of CMI-based criteria are part of the InfoNEAT algorithm, and they highly contribute to the improvement of the time-complexity associated with training an effective SCA model for different ASCAD datasets. Table 2 summarizes the average time involved in training a particular sub-model using the InfoNEAT algorithm (column labeled as "With InfoNEAT") and compares this time to the average time taken while using the default NEAT without the CMI-based criteria (column labeled as "With NEAT"). For the default NEAT equipped with OvA, we select the batch size of 200 as recommended in [BPS$^+$18], whereas it is 150 for InfoNEAT (the reason is explained in Appendix B). InfoNEAT could outperform in terms of the number of generations thanks to the CMI-based best genome selection. The matrix-based operations for CMI computation are fast, and the difference in the average generation time is mainly due to the difference in the batch size (for more discussion and results, see Appendix C).

6.3 Side-channel Analysis through InfoNEAT

6.3.1 ASCAD$^{\text{fixed}}$ Dataset

Figure 8 shows the comparison of SOTA networks and InfoNEAT for the ASCAD$^{\text{fixed}}_{\text{sync}}$ dataset. For the results presented in [WPP20], the best results for MLPs and CNNs are depicted. Since feature scaling between 0 and 1 is applied in InfoNEAT, in addition to the best result from [WAGP20], the result for the similar scaling is also presented. It is remarkable that compared with the most relevant approaches, applying MLPs

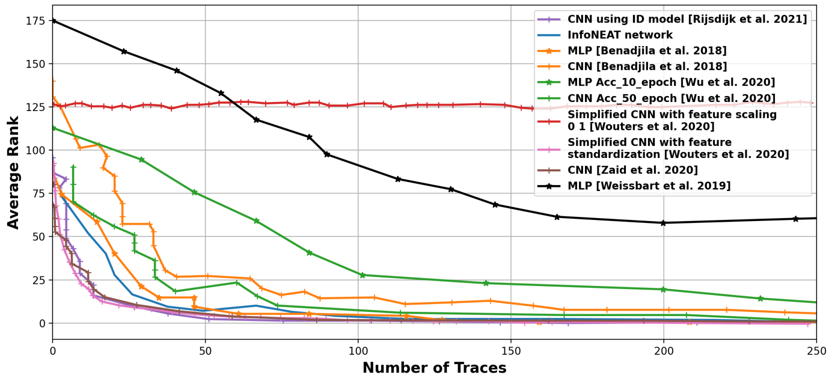


Figure 8: SOTA NNs vs. InfoNEAT applied against $\text{ASCAD}^{\text{fixed}}_{\text{sync}}$

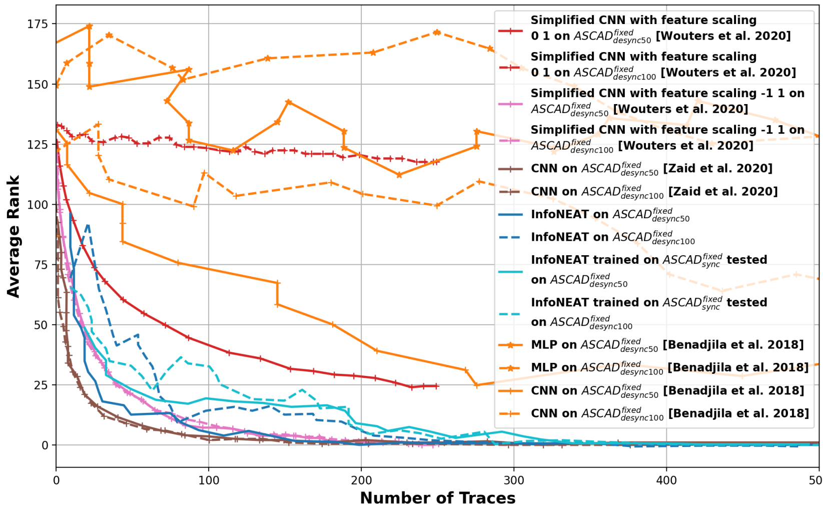


Figure 9: Average rank for the $\text{ASCAD}^{\text{fixed}}_{\text{desync}}$ dataset.

with hyperparameter tuning [WPP20], InfoNEAT's attack performance is slightly better. Considering CNNs with hyperparameter tuning [ZBHV20, RWPP21, WAGP20], InfoNEAT outperforms the model in [WAGP20] trained on traces, whose features are scaled between 0 and 1 (similar to InfoNEAT). For this type of feature-scaling, the model in [ZBHV20] and InfoNEAT yield similar performance. InfoNEAT and the model trained on standardized features (zero-mean unit-variance) in [WAGP20] show similar performance. In a nutshell, comparing InfoNEAT's results with the best results in [ZBHV20, WAGP20], they have presented $T_{GE0} = 191, 155$, respectively, whereas InfoNEAT achieves $T_{GE0} = 130$, where T_{GE0} denotes the least number of attack traces required to break the target. Nevertheless, InfoNEAT's largest potential for improving SCA performance becomes evident when $\text{ASCAD}_{\text{desync}}$ dataset is considered.

Training on $\textbf{ASCAD}^{\textbf{fixed}}_{\textbf{sync}}$ and testing against $\textbf{ASCAD}^{\textbf{fixed}}_{\textbf{desync}}$: Figure 9 show the results of using InfoNEAT model against different $\text{ASCAD}^{\text{fixed}}_{\text{desync}}$ dataset. As can be seen, InfoNEAT is capable of learning different desynchronized datasets when trained and tested on the same dataset. As it might not always be easy to align traces using alignment techniques (see,e.g., [WHJ+21]), it is useful to employ ML models robust against desynchronization [HHO20, PBP20]. In our case, once the model is trained against synchronized traces (during the profiling phase where the attacker works on an open copy), there is no need to re-train/fine-tune the model, even in the presence of desynchronization. More interestingly, the average rank computed by using the model trained on synchronized data to attack or test the desynchronized data is comparable to the other average ranks

Table 3: Comparison between InfoNEAT and SOTA attacks against ASCAD$^{\text{var}}$ dataset.

Metric	[WPP21]	[HHO20]	[PCP20]	[WPP20]	[RWPP21]	[PHPG21]	InfoNEAT
T_{GE0}	188†	F	56* / 450†	2000* / 3144†	490†	1040* / 2854†	**120**
T_{GE1}	81†	F†	50* / 400†	1250* / 3000†	265†	NR	**30**
T_{GE20}	40†	6†	6* / 50†	150* / 100†	38†	56* / 318†	**10**

$*$ MLP, † CNN, F: failed to reach NR: Not reported.

Table 4: Qualitative comparison between important characteristics of the methodologies applied against ASCAD$^{\text{var}}$ dataset.

Required/used	[WPP21]	[HHO20]	[PCP20]	[WPP20]	[RWPP21]	[PHPG21]	InfoNEAT
Plaintext	No	Yes	No	No	No	No	No
Template attack	Yes	No	No	No	No	No	No
Additional model validation step	No	Yes	Yes	Yes	Yes	No	No

(trained and tested on the same type of dataset). In this case, this shows that the models trained using InfoNEAT are *generalizable* in the sense that they are not sensitive to desynchronization/jittering (see also the results for ASCAD$^{\text{var}}_{\text{desync}}$ in Section 6.3.2). This is due to the non-typical NN architecture created by NEAT that can handle time-dependent, or in general, order-dependent phenomena [CBD15]. This is made possible thanks to the feedback connections between neurons automatically inserted using the add connection operator. In other words, networks generated and trained by InfoNEAT can adapt to sequences of events, even if they exhibit a lag, e.g., time-delay.

InfoNEAT's performance in the face of more challenging datasets is considered next, where it showcases its potential even better.

6.3.2 ASCAD$^{\text{var}}$Dataset

In the same vein as ASCAD$^{\text{fixed}}$, the batch size is set as explained in Appendix B. For ASCAD$^{\text{var}}_{\text{sync}}$ dataset, it is determined that the batch size of 256 is an adequate choice. We performed the experiments with 20 and 25 traces per class and observed no significant improvement if the latter is chosen; hence, the results for the former are reported here.

For SCA, in the case of fixed key, k-fold cross-validation has been employed as a powerful tool for making a more realistic estimate of the model's skill than other methods, such as a simple train/test split [BPS$^+$18, PHJ$^+$19, KPH$^+$19, BCH$^+$20]. For variable key cases, to report a good estimation of the average rank, the attack should be launched multiple times to assess the attack performance properly [WWK$^+$20, WPP22]. This is due to the fact that if a number of independent experiments (key rank evaluations) are conducted in the attack phase, the attack performance can vary. Although this variation might be small, it is needed to mount the attack multiple times (e.g., around 40 independent attacks [WPP22]) using a model with properly optimized hyperparameters, as for InfoNEAT. To account for the performance variation, summary statistics should be taken into account, e.g., the median has been suggested to be the most effective in this respect [WPP22]; hence, our results present the median, in addition to the minimum, calculated over 50 independent attacks.

In Table 3, the least number of attack traces required to break the target (T_{GE0}), obtain an average rank 1 (T_{GE1}), and < 20 (T_{GE20}) are provided. Note that the results of other studies are directly derived from their papers, and the *minimum* number of traces for different metrics is provided. Noteworthy is that [PHPG21] has not reported T_{GE1}. In this table, "F" indicates that according to the results in the respective papers, the proposed attack could not reach the average rank of 0, 1, or below 20. We should add that these results have been obtained for the specific setting considered in [HHO20]. Their model is trained using ASCAD$^{\text{var}}$ training dataset, whereas it is tested in 100 runs, each with 100 traces, resulting in a total of 10,000 testing traces. The results in [HHO20] are the average over these 100 runs. We stress that although their attack could not reach the average rank 0 or 1, it achieved rank 2 with only 40 traces, comparable to InfoNEAT,

Table 5: NNs trained on ASCAD$^{\mathrm{var}}_{\mathrm{sync}}$ against ASCAD$^{\mathrm{var}}_{\mathrm{desync}}$.

Metric	CNN [HHO20]		VGG16 [BPS+18]		**InfoNEAT**	
	desync50	desync100	desync50	desync100	desync50	desync100
T_{GE0}	F	F	F	F	880	960
T_{GE1}	F	F	F	F	590	140
T_{GE20}	F	F	300	F	170	70
T_{GE50}	935	F	90	F	70	30

desync50 and desync100 denote ASCAD$^{\mathrm{var}}_{\mathrm{desync50}}$ and ASCAD$^{\mathrm{var}}_{\mathrm{desync100}}$, respectively. F: failed to reach

which reaches the average rank 1 with 30 traces.

[PCP20] reported one of the best results not only for the ensemble of models, but also for the *best* model. In their framework, the best model(s) is selected with regard to the validation key rank, i.e., an additional step is taken to fine-tune the model after some epochs by applying the trained model against a validation set which is created using the attack traces. This directly affects the performance of the model cf. [PCP20]. The same holds for approaches in [WPP20, RWPP21, HHO20]. As this additional step enhances the performance of the attack, it is important to differentiate between these approaches and InfoNEAT and [PHPG21], which do *not* take advantage of the validation step. Moreover, [WPP21] has combined DL and template attacks, whereas [HHO20] has incorporated plaintext as a feature (see Table 4 for a summary). Additionally, for InfoNEAT, the *median* values are $T_{GE0} = 310$, $T_{GE1} = 240$, and $T_{GE20} = 145$.

Training on ASCAD$^{\mathbf{var}}_{\mathbf{sync}}$ and testing against ASCAD$^{\mathbf{var}}_{\mathbf{desync}}$: As demonstrated for ASCAD$^{\mathrm{fixed}}$ dataset, it is possible to train the model on synched traces and apply it against desynchronized traces, thanks to the irregular configuration of NNs evolved by InfoNEAT. Interestingly enough, the same observation is made when the model trained on ASCAD$^{\mathrm{var}}_{\mathrm{sync}}$ is applied against ASCAD$^{\mathrm{var}}_{\mathrm{desync50}}$ and ASCAD$^{\mathrm{var}}_{\mathrm{desync100}}$. As the models in [HHO20, BPS+18] could not deliver results aligned with the metrics T_{GE0}–T_{GE20}, we add T_{GE50} to this table, also considered as a performance metric in the relevant literature cf. [KPH+19]. Table 5 compares InfoNEAT's performance with the results presented in [HHO20]. Similar to Table 3, "F" indicates that the attack could not achieve rank 0,1,20, or 50. In order to test the model, 10 runs, each with 1000 traces, are performed in [HHO20] and the results are the average over the runs. Interestingly, as can be seen in Table 5, InfoNEAT can reach rank 0 within 1000 testing traces, while the minimum rank is 48 obtained for 935 testing traces [HHO20] and 10 for 700 traces (using VGG16 [BPS+18]). For InfoNEAT, for ASCAD$^{\mathrm{fixed}}_{\mathrm{desync50}}$ (ASCAD$^{\mathrm{fixed}}_{\mathrm{desync100}}$) the *median* values are $T_{GE0} = 970$ (975), $T_{GE1} = 870$ (675), $T_{GE20} = 480$ (395), $T_{GE50} = 310$ (185). Note that, to the best of our knowledge, no other results for NNs trained on ASCAD$^{\mathrm{var}}_{\mathrm{sync}}$ to attack ASCAD$^{\mathrm{var}}_{\mathrm{desync50}}$ and ASCAD$^{\mathrm{var}}_{\mathrm{desync100}}$ have been reported.

6.3.3 AES_HD Dataset

To train the NNs, we follow the procedure explained in Appendix B to determine the size of batches, which is set to 256, whereas the number of traces used to train the stacked model is 30. The average number of generations to train a sub-model is 19. The number of folds for cross-validation is set to 5, similar to [PHJ+19]. For the MLP results from this study, no feature scaling has been applied; hence, a direct comparison between our results and ones in [PHJ+19] is not possible. For the sake of better comparison, we also take into account their best results obtained after applying SMOTE, i.e., a resampling method that adds synthetic examples with the random shift, as well as applies other transformations. Similar to undersampling, SMOTE can deal with imbalanced datasets.

As shown in Figure 10, we compare our results with the SOTA methods based on k-fold cross validation shown in Figure 10(a) and the best model (fold) as depicted in Figure 10(b). This is with regard to the observation made before in the literature [KPH+19, WWK+20] and by us during the testing phase. For some folds, the average rank decreases and reaches

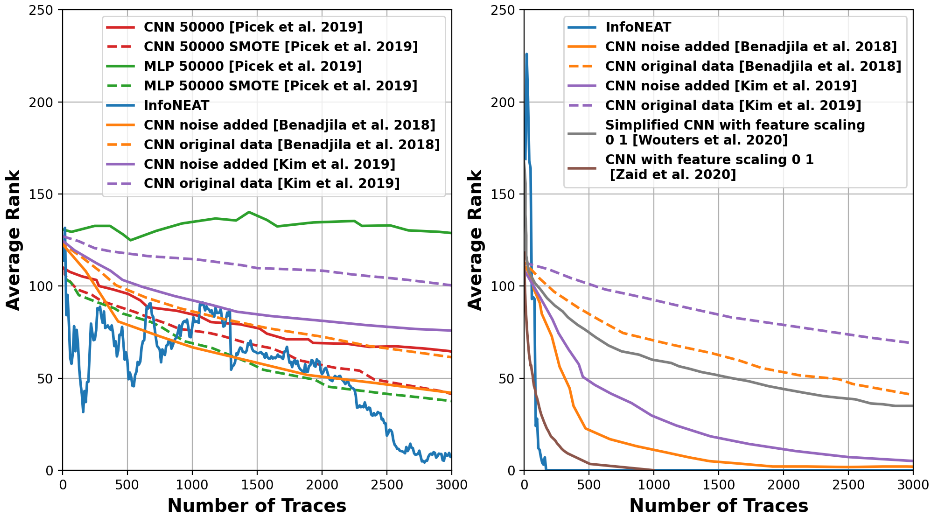


Figure 10: SOTA NNs vs. InfoNEAT for AES_HD (a) with cross-validation: T_{GE50} is 2000 (InfoNEAT), 2000 (MLP with SMOTE and CNN in [BPS+18] with noise regularization [KPH+19]) and 2500 (CNN with SMOTE) [PHJ+19]; (b) best models.

0 by increasing the number of testing traces, whereas, for others, it decreases and remains below a threshold, e.g., 50 in our experiments. Note that the trend observed here and in the prior studies is similar, although the minimum average rank obtained by us is less than what has been reported before. Comparing to [PHJ+19] that considers k-fold cross validation, $T_{GE50} = 2000, 2500$ is observed for MLP and CNN, respectively, whereas $T_{GE50} = 2000$ in our case (see Figure 10(a)). It is interesting to compare the trend in the results obtained by using InfoNEAT and the SOTA approaches. The latter has shown an almost smooth trend, contrary to InfoNEAT. This can be explained by the differences between these methods when being employed against relatively more noisy measurements in the AES_HD dataset. First, algorithms applying SMOTE and adding noise to the traces leverage regulation-like techniques, which are useful in the test phase as well as training [KPH+19, PHJ+19]. On top of this, and more importantly, we should stress that the k-fold validation method in [KPH+19] is different from ours. In that study, repeated random subsampling has been used to reduce the instability. Moreover, [PHJ+19] has considered feature extraction on traces. InfoNEAT has applied none of these techniques, and they are left for future work.

Our best model achieves $T_{GE0} = 170$, whereas using CNNs combined with pre-processing leads to $T_{GE0} \approx 800$ (averaged over 100 trials) [WAGP20, ZBHV20] (see Figure 10(b)). Similarly, multi-scale CNN along with various data transformation is used to obtain $T_{GE50} = 2000$ and $T_{GE10} = 5000$ [WHJ+21], whereas $T_{GE10} = 130$ for InfoNEAT. Besides those results depicted in this figure, one can consider the ones presented in [vdVKPB20]. In that work, it has been observed that if a compact network (MLP or CNN), called the student network, is trained on the output of a big (teacher) network, better results can be obtained when compared to complex networks. Nevertheless, the minimum number of traces for which the average rank goes below 100 (best case presented) is 10,000.

Summary of the results obtained for different datasets: The most important message conveyed here is that InfoNEAT could achieve high attack performance, even *without* combining DL with a template attack or leveraging the plaintext as a feature or model validation. The models trained using InfoNEAT also enjoy another advantage, namely being resilient against desynchronization.

Table 6: SOTA approaches vs. InfoNEAT.

Ref.	No. of trainable parameters	No. of epochs	No. of training traces	Hyperparameter tuning	Dataset	T_{GE0}
[BPS$^+$18]	393,936* 66,652,444†	400* 75†	40,000	Trial and error	ASCAD$^{\text{fixed}}$	410* 480†
[Wei20]	740,136*	200	20,000	Trial and error	ASCAD$^{\text{fixed}}$	1630
[WPP20]	478,656* 54,752†	[10,50]	50,000	Bayesian optimization and Random search	ASCAD$^{\text{fixed}}$	129*/158†
					ASCAD$^{\text{var}}$	2000* / 3144†
[PHJ$^+$19]	49,024*	-	50,000	Randomly selected	AES_HD	10,000
[PCP20]	493,480*	-	200,000	Random Search	ASCAD$^{\text{var}}$	56* / 450†
[ZBHV20]	16,960†	50	50,000	Visualization techniques	ASCAD$^{\text{fixed}}$	191†
	3,282†	20	45,000		AES_HD	800†
[WAGP20]	6,436†	50	45,000	Taken from[ZBHV20]	ASCAD$^{\text{fixed}}$	155†
	2,020†		50,000		AES_HD	800†
[RWPP21]	79,439†	50	50,000	Reinforcement learning	ASCAD$^{\text{fixed}}$	202†
	70,492†		200,000		ASCAD$^{\text{var}}$	490†
InfoNEAT	15,107	8	38,400	Automatic Neuroevolution	ASCAD$^{\text{fixed}}$	130
	317,408	8	161,280		ASCAD$^{\text{var}}$	120
	102,757	33	38,400		AES_HD	170

$*$ MLP, † CNN.

6.4 Memory and Epoch-wise Efficiency

Table 6 compares SOTA networks with InfoNEAT when attacking various datasets. Note that this table presents the number of *trainable* parameters, different from the number of hyperparameters (for the computational complexity of hyperparameter tuning and NAS, see Section 6.5). It is worth mentioning that for the sake of comparison, we consider the size of the individual models because we train them one by one. Similar to the bagging ensemble technique used in [PCP20], the complexity of the stacked model is linear in the number of models (see Appendix C for additional information on the number of nodes and number of species). In general, InfoNEAT requires fewer epochs to train an NN. As can be seen in Table 6, the number of trainable parameters for CNNs is less than that for MLPs and InfoNEAT. Compared to MLPs, InfoNEAT shows improvement. Note that the comparison with [PCP20] is based on the average number of layers and neurons reported in this study (no concrete number is reported). Nevertheless, when comparing the average rank obtained by all types of NNs, they could perform well (see Figures 8-10 and Tables 3-5). Yet, InfoNEAT can achieve the competitive result *without* applying any additional technique as also summarized in Table 6. It also delivers desynchronization-robust NNs in our experiments. Next, we discuss other technical aspects of InfoNEAT and provide a cost-benefit evaluation.

6.5 Discussion

InfoNEAT vs. approaches for hyperparameter optimization: When it comes to tuning the hyperparameters, the main proposals in SCA-related literature include random search and Bayesian optimization. Random search evaluates uniformly random points in the hyperparameter space to select the one that provides the best performance. This implementation is easier to comprehend and parallelize. Despite its ease of use, random search is burdened by the high variance between runs. Moreover, from one run to another, this algorithm cannot use previous observations leading to a long convergence time in some scenarios. An example of this is reported in [WPP21] when unsuccessfully mounting the ensemble learning attack [PCP20] with a non-optimal hyperparameter space caused by the input dimensions. If the number of runs is restricted to avoid this, random search may not deliver the optimum result [And15].

Another promising candidate for hyperparameter tuning is Bayesian optimization, which is, on one hand, data-efficient, but on the other hand, computationally expensive. Concretely, for a given number of evaluations of candidate solution N, its complexity is $O(N^3)$, and even after applying mechanisms devised to reduce this cost, it grows substantially with the number of evaluations [LTRE22]. The complexity of InfoNEAT is similar to the NEAT algorithm. This is due to the fact that while the stopping and best-genome selection criteria can reduce the number of candidates to be evaluated per submodel, the maximum number of evaluations does not exceed that of the NEAT algorithm. It is also obvious that when training all submodels, since the evaluation of candidate solutions is repeated for a constant number (i.e., 256 for SCA), the complexity remains the same compared to NEAT. Therefore, the complexity of the InfoNEAT algorithm is $O(N^2)$, similar to NEAT [LGP00, BSI20]. Given this complexity, Bayesian optimization could take longer to converge to the solution. Nevertheless, Bayesian optimization is more efficient than Grid search with the complexity $O(N^k)$, which is used in, e.g., [Wei20]. Moreover, Bayesian optimization follows a more systematic search mechanism compared to random search [LNK$^+$17].

InfoNEAT vs. approaches for NAS (beyond hyperparameter optimization): When formalizing NAS as a reinforcement learning task, different RL approaches with various policies and optimization techniques could be taken into account [EMH19]. Baker et al.'s Q-learning algorithm [BGNR16] is one of such proposals that has found application in SCA as well [RWPP21]. [BGNR16] has proposed to train a policy that sequentially chooses not only the type of each layer, but also its corresponding hyperparameters. The main drawback of such a technique is its high exploration cost and slow convergence time [BGNR16, WG21, RWPP21]. Precisely, the complexity of Q-learning proposed in [BGNR16] is $\tilde{\mathcal{O}}(T)$, where T is the total number of steps and $\tilde{\mathcal{O}}(\cdot)$ denotes that $T \geq \text{polylog}(S, A, H)$ with S, A, and H denoting the number of states, actions, and steps. This means that the complexity depends on the number of states, actions, and steps up to a poly-log degree. Although a direct comparison with InfoNEAT computational complexity may not be possible, InfoNEAT could exhibit less computational complexity in practical scenarios as demonstrated for NEAT cf. [GM21].

Scalability of InfoNEAT: Neuroevolutionary approaches are alternative solutions to RL, where recent case studies have demonstrated their comparably-well performances [RAHL19]. In addition, neuroevolution is more advantageous because it avoids backpropagation and has comparatively less time complexity, although one challenge that remains to be addressed is the scalability. Neuroevolution at scale is an active research area, with many methods being inspired by NEAT's ability to increase complexity over generations. In this regard, adding layers instead of individual neurons is one of the promising solutions [RM18, RAHL19]. Our observation is that InfoNEAT taking advantage of meta-learning, can generate compact networks that are effective under SCA scenarios, in line with the conclusion made in [WPP20, RWPP21]. Our future work will also consider other variants of neuroevolution.

Is InfoNEAT a silver bullet? InfoNEAT should be considered another step toward fostering research into the application of NAS in SCA. In fact, methods devised in [RWPP21] and our paper are computationally expensive, although the effort made by the attacker/evaluator launching SCA and the reliance on their own expertise can be reduced. From a more technical perspective, NAS and especially InfoNEAT bring the advantage of configuring NNs that human experts might not easily develop manually. From this perspective, InfoNEAT could demonstrate its potential, in particular, when more challenging datasets have been considered. Our results show that such networks and their applications in SCA can be an interesting research topic.

The natural question to be answered is under which scenarios InfoNEAT and its counterparts [WPP20, RWPP21] should be used. In a nutshell, similar to these studies, training time would constitute the cost of applying InfoNEAT. Since we trained submodels one by one, training took 2 days on average (including training of the stacked model), comparable to 4 days and approximately 3.5 days reported in [RWPP21] and [WPP20], respectively. The stacking technique used in InfoNEAT facilitates the parallelization of the algorithm, as discussed next.

Parallelization of InfoNEAT: Generally, NEAT and its variants can enjoy the benefits offered by parallel evolutionary algorithms to optimize memory allocation and time complexity [CG19]. Compared to NEAT, InfoNEAT enhanced by integrating a stacked model (converting a multi-class task to a number of binary classification problems) can greatly benefit from parallelization. In fact, InfoNEAT fits the purpose of distributed learning in asynchronous mode [Ben13] since submodels can be trained individually and in parallel. The results presented in this paper, however, are not obtained using any specific parallelization. Running parallel InfoNEAT can be considered as future work. For SCA, our experiments, however, suggest that InfoNEAT is scalable to the number of classes, i.e., 256 classes (see Appendix C for an example of InfoNEAT's time complexity in our experiments).

7 Conclusion

In this paper, we introduce a first of its kind, information theory-based automatic neuroevolution technique that successfully profiles the SCA traces of 256 different classes. This is achieved through use of the CMI-heuristic, NEAT algorithm, OvA decomposition technique, and consequent stacking ensemble learning to train an effective SCA model which is simple, compact, and yet deep enough to successfully extract the sub-key using less than a hundred traces. In addition, we provide a detailed study on how InfoNEAT can be implemented, trained, and tested in practice. The effectiveness and feasibility of InfoNEAT are demonstrated by applying it against widely-used side-channel datasets. For future work, particular research directions will be considered, including the application of InfoNEAT against other datasets, e.g., [Age21, Ais21]. Furthermore, the stacking technique used in InfoNEAT facilitates the parallelization of the algorithm, which has not yet been explored by us and is left for future work.

Acknowledgment

The authors would like to thank Mr. Mohammad Hashemi and Dr. James Kingsley (Worcester Polytechnic Institute) as well as the HyperGator Team (University of Florida) for their support and help with the experimental setup. Results in this paper were obtained in part using a high-performance computing system acquired through NSF MRI grant DMS-1337943 to WPI. This study has also been supported in part by NSF under award number 2138420, Air Force's Center of Excellence for Enabling Cyber Defense in Analog and Mixed Signal Domain (CYAN) under the contract number FA8650-19-1-1741, and AFOSR MURI Grant under Award FA9550-14-1-0351.

References

[ABB$^+$20] Melissa Azouaoui, Davide Bellizia, Ileana Buhan, Nicolas Debande, Sébastien Duval, Christophe Giraud, Éliane Jaulmes, François Koeune, Elisabeth Oswald, François-Xavier Standaert, et al. A systematic appraisal of side channel evaluation strategies. In *International Conference on Research in Security Standardisation*, pages 46–66. Springer, 2020.

[AG19] Rana Ali Amjad and Bernhard C Geiger. Learning representations for neural network-based classification using the information bottleneck principle. *Trans. on Pattern Analysis and Machine Intelligence*, 42(9):2225–2239, 2019.

[Age21] Agence nationale de la scurit des systmes d'information (ANSSI). ASCADv2: the STM32 SCA traces databases. [Online]https://github.com/ANSSI-FR/ASCAD/tree/master/STM32_AES_v2 [Accessed: Jul.12, 2022], 2021.

[Ais21] Aisylab . Portability dataset. [Online]http://aisylabdatasets.ewi.tudelft.nl [Accessed: Jul.12, 2022], 2021.

[And15] Sigrún Andradóttir. A review of random search methods. *Handbook of Simulation Optimization*, pages 277–292, 2015.

[AWD19] Ahmed Aly, David Weikersdorfer, and Claire Delaunay. Optimizing deep neural networks with multiple search neuroevolution. *arXiv preprint arXiv:1901.05988*, 2019.

[BCH+20] Shivam Bhasin, Anupam Chattopadhyay, Annelie Heuser, Dirmanto Jap, Stjepan Picek, and Ritu Ranjan. Mind the portability: A warriors guide through realistic profiled side-channel analysis. In *NDSS 2020*, 2020.

[Ben13] Yoshua Bengio. Deep learning of representations: Looking forward. In *International conference on statistical language and speech processing*, pages 1–37. Springer, 2013.

[BGNR16] Bowen Baker, Otkrist Gupta, Nikhil Naik, and Ramesh Raskar. Designing neural network architectures using reinforcement learning. *arXiv preprint arXiv:1611.02167*, 2016.

[Bha06] Rajendra Bhatia. Infinitely divisible matrices. *The American Mathematical Monthly*, 113(3):221–235, 2006.

[BHM+19] Olivier Bronchain, Julien M Hendrickx, Clément Massart, Alex Olshevsky, and François-Xavier Standaert. Leakage certification revisited: Bounding model errors in side-channel security evaluations. In *Annual Intrl. Cryptol. Conf.*, pages 713–737. Springer, 2019.

[BJP20] Shivam Bhasin, Dirmanto Jap, and Stjepan Picek. AES HD dataset - 500 000 traces. AISyLab repository, 2020. https://github.com/AISyLab/AES_HD_2.

[BPS+17] Ryad Benadjila, Emmanuel Prouff, Rémi Strullu, Eleonora Cagli, and Cécile Dumas. ASCADv1 Dataset: the atmega8515 sca campaigns. [Online]https://github.com/ANSSI-FR/ASCAD/tree/master/ATMEGA_AES_v1 [Accessed: Jan.15, 2022], 2017.

[BPS+18] Ryad Benadjila, Emmanuel Prouff, Rémi Strullu, Eleonora Cagli, and Cécile Dumas. Study of deep learning techniques for side-channel analysis and introduction to ascad database. 2018.

[BSI20] Alejandro Baldominos, Yago Saez, and Pedro Isasi. On the automated, evolutionary design of neural networks: past, present, and future. *Neural computing and applications*, 32(2):519–545, 2020.

[CBD15] Pilar Caamaño, Francisco Bellas, and Richard J Duro. τ-neat: Initial experiments in precise temporal processing through neuroevolution. *Neurocomputing*, 150:43–49, 2015.

[CDP17] Eleonora Cagli, Cécile Dumas, and Emmanuel Prouff. Convolutional neural networks with data augmentation against jitter-based countermeasures. In *Intrl. Conf. on Cryptographic Hardware and Embedded Systems*, pages 45–68. Springer, 2017.

[CG19] John Runwei Cheng and Mitsuo Gen. Accelerating genetic algorithms with gpu computing: A selective overview. *Computers & Industrial Engineering*, 128:514–525, 2019.

[Cov99] Thomas M Cover. *Elements of information theory*. John Wiley & Sons, 1999.

[CRR02] Suresh Chari, Josyula R Rao, and Pankaj Rohatgi. Template attacks. In *Intrl. Workshop on Cryptographic Hardware and Embedded Systems*, pages 13–28. Springer, 2002.

[DPRS11] Julien Doget, Emmanuel Prouff, Matthieu Rivain, and François-Xavier Standaert. Univariate side channel attacks and leakage modeling. *Journal of Cryptographic Engineering*, 1(2):123, 2011.

[EMH19] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. *The Journal of Machine Learning Research*, 20(1):1997–2017, 2019.

[Eur20] European Commission. The framework programme for research and innovation. [Online]https://ec.europa.eu/programmes/horizon2020/en [Accessed: Jan.15, 2021], 2020.

[Fle04] François Fleuret. Fast binary feature selection with conditional mutual information. *Journal of Machine learning research*, 5(9), 2004.

[FRWV07] Damien François, Fabrice Rossi, Vincent Wertz, and Michel Verleysen. Resampling methods for parameter-free and robust feature selection with mutual information. *Neurocomputing*, 70(7-9):1276–1288, 2007.

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[GHO15] Richard Gilmore, Neil Hanley, and Maire O'Neill. Neural network based attack on a masked implementation of aes. In *Intrl. Symposium on Hardware Oriented Security and Trust (HOST)*, pages 106–111. IEEE, 2015.

[GM21] Edgar Galván and Peter Mooney. Neuroevolution in deep neural networks: Current trends and future challenges. *Trans. on Artificial Intelligence*, 2021.

[Goo13] Phillip Good. *Permutation Tests: A Practical Guide to Resampling Methods for Testing Hypotheses*. Springer Science & Business Media, 2013.

[GP13] Luis G Sanchez Giraldo and Jose C Principe. Rate-distortion auto-encoders. *arXiv preprint arXiv:1312.7381*, 2013.

[GRP14] Luis Gonzalo Sanchez Giraldo, Murali Rao, and Jose C Principe. Measures of entropy from data using infinitely divisible kernels. *Trans. on Information Theory*, 61(1):535–548, 2014.

[HDR19] Soufiane Hayou, Arnaud Doucet, and Judith Rousseau. On the impact of the activation function on deep neural networks training. In *Intrl. Conf. on Machine Learning*, pages 2672–2680. PMLR, 2019.

[HGDM+11] Gabriel Hospodar, Benedikt Gierlichs, Elke De Mulder, Ingrid Verbauwhede, and Joos Vandewalle. Machine learning in side-channel analysis: a first study. *Journal of Cryptographic Engineering*, 1(4):293, 2011.

[HGG20] Benjamin Hettwer, Stefan Gehrer, and Tim Güneysu. Applications of machine learning techniques in side-channel attacks: a survey. *Journal of Cryptographic Engineering*, 10(2):135–162, 2020.

[HHO20] Anh-Tuan Hoang, Neil Hanley, and Maire O'Neill. Plaintext: A missing feature for enhancing the power of deep learning in side-channel analysis? *IACR Trans. on Cryptographic Hardware and Embedded Systems*, pages 49–85, 2020.

[HMA17] Alexander Hagg, Maximilian Mensing, and Alexander Asteroth. Evolving parsimonious networks by mixing activation functions. In *Proceedings of the Genetic and Evolutionary Computation Conf.*, pages 425–432, 2017.

[HZ12] Annelie Heuser and Michael Zohner. Intelligent machine homicide. In *Intrl. Workshop on Constructive Side-Channel Analysis and Secure Design*, pages 249–264. Springer, 2012.

[JSH19] Haifeng Jin, Qingquan Song, and Xia Hu. Auto-keras: An efficient neural architecture search system. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 1946–1956, 2019.

[KMN+16] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. *arXiv preprint arXiv:1609.04836*, 2016.

[KPH+19] Jaehun Kim, Stjepan Picek, Annelie Heuser, Shivam Bhasin, and Alan Hanjalic. Make some noise. unleashing the power of convolutional neural networks for profiled side-channel analysis. *IACR Trans. on Cryptographic Hardware and Embedded Systems*, pages 148–179, 2019.

[Kum17] Siddharth Krishna Kumar. On weight initialization in deep neural networks. *arXiv preprint arXiv:1704.08863*, 2017.

[KWPP21] Maikel Kerkhof, Lichao Wu, Guilherme Perin, and Stjepan Picek. No (good) loss no gain: Systematic evaluation of loss functions in deep learning-based side-channel analysis. *IACR Cryptol. ePrint Arch., 2021/1091*, 2021.

[LBM15] Liran Lerman, Gianluca Bontempi, and Olivier Markowitch. A machine learning approach against a masked aes. *Journal of Cryptographic Engineering*, 5(2):123–139, 2015.

[LGP00] Fernando G Lobo, David E Goldberg, and Martin Pelikan. Time complexity of genetic algorithms on exponentially scaled problems. In *Proceedings of the 2nd annual conference on genetic and evolutionary computation*, pages 151–158, 2000.

[LNK+17] Pablo Ribalta Lorenzo, Jakub Nalepa, Michal Kawulok, Luciano Sanchez Ramos, and José Ranilla Pastor. Particle swarm optimization for hyper-parameter selection in deep neural networks. In *Proceedings of the genetic and evolutionary computation conference*, pages 481–488, 2017.

[LPB$^+$15] Liran Lerman, Romain Poussier, Gianluca Bontempi, Olivier Markowitch, and François-Xavier Standaert. Template attacks vs. machine learning revisited (and the curse of dimensionality in side-channel analysis). In *Intrl. Workshop on Constructive Side-Channel Analysis and Secure Design*, pages 20–33. Springer, 2015.

[LTRE22] Gongjin Lan, Jakub M Tomczak, Diederik M Roijers, and AE Eiben. Time efficiency in optimization with a bayesian-evolutionary algorithm. *Swarm and Evolutionary Computation*, 69:100970, 2022.

[LZW11] Yufeng Liu, Hao Helen Zhang, and Yichao Wu. Hard or soft classification? large-margin unified machines. *Journal of the American Statistical Association*, 106(493):166–177, 2011.

[MAB$^+$18] Tyler McDonnell, Sari Andoni, Elmira Bonab, Sheila Cheng, Jun-Hwan Choi, Jimmie Goode, Keith Moore, Gavin Sellers, and Jacob Schrum. Divide and conquer: Neuroevolution for multiclass classification. In *Proceedings of the Genetic and Evolutionary Computation Conf.*, pages 474–481, 2018.

[MDP20] Loïc Masure, Cécile Dumas, and Emmanuel Prouff. A comprehensive study of deep learning for side-channel analysis. *IACR Trans. on Cryptographic Hardware and Embedded Systems*, pages 348–375, 2020.

[MHM13] Zdenek Martinasek, Jan Hajny, and Lukas Malina. Optimization of power analysis using neural network. In *Intrl. Conf. on Smart Card Research and Advanced Applications*, pages 94–107. Springer, 2013.

[MPP16] Houssem Maghrebi, Thibault Portigliatti, and Emmanuel Prouff. Breaking cryptographic implementations using deep learning techniques. In *Intrl. Conf. on Security, Privacy, and Applied Cryptography Engineering*, pages 3–26. Springer, 2016.

[MS16] Gregory Morse and Kenneth O Stanley. Simple evolutionary optimization can rival stochastic gradient descent in neural networks. In *Proceedings of the Genetic and Evolutionary Computation Conf. 2016*, pages 477–484, 2016.

[MS21] Loïc Masure and Rémi Strullu. Side channel analysis against the anssi's protected aes implementation on arm. *Cryptology ePrint Archive*, 2021.

[Mur12] Kevin P Murphy. *Machine Learning: A Probabilistic Perspective*. MIT press, 2012.

[Nat20] National Institute of Standards and Technology. Threshold cryptography project. [Online]https://csrc.nist.gov/projects/threshold-cryptograph [Accessed: Jan.15, 2021], 2020.

[Ome19] Iaroslav Omelianenko. *Hands-On Neuroevolution with Python: Build high-performing artificial neural network architectures using neuroevolution-based algorithms*. Packt Publishing Ltd, 2019.

[PBP20] Guilherme Perin, Ileana Buhan, and Stjepan Picek. Learning when to stop: a mutual information approach to fight overfitting in profiled side-channel analysis. *IACR Cryptol. ePrint Arch.*, 2020:58, 2020.

[PCP20] Guilherme Perin, Łukasz Chmielewski, and Stjepan Picek. Strength in numbers: Improving generalization with ensembles in machine learning-based profiled side-channel analysis. *IACR Trans. on Cryptographic Hardware and Embedded Systems*, pages 337–364, 2020.

[PHJ$^+$19] Stjepan Picek, Annelie Heuser, Alan Jovic, Shivam Bhasin, and Francesco Regazzoni. The curse of class imbalance and conflicting metrics with machine learning for side-channel evaluations. *IACR Trans. on Cryptographic Hardware and Embedded Systems*, 2019(1):1–29, 2019.

[PHPG21] Stjepan Picek, Annelie Heuser, Guilherme Perin, and Sylvain Guilley. Profiled side-channel analysis in the efficient attacker framework. In *International Conference on Smart Card Research and Advanced Applications*, pages 44–63. Springer, 2021.

[PP20] Guilherme Perin and Stjepan Picek. On the influence of optimizers in deep learning-based side-channel analysis. In *International Conference on Selected Areas in Cryptography*, pages 615–636. Springer, 2020.

[PPM$^+$21] Stjepan Picek, Guilherme Perin, Luca Mariot, Lichao Wu, and Lejla Batina. SoK: Deep learning-based physical side-channel analysis. *IACR Cryptol. ePrint Arch., 2021/1092*, 2021.

[PSK$^+$18] Stjepan Picek, Ioannis Petros Samiotis, Jaehun Kim, Annelie Heuser, Shivam Bhasin, and Axel Legay. On the performance of convolutional neural networks for side-channel analysis. In *Intrl. Conf. on Security, Privacy, and Applied Cryptography Engineering*, pages 157–176. Springer, 2018.

[PYW$^+$17] Sihang Pu, Yu Yu, Weijia Wang, Zheng Guo, Junrong Liu, Dawu Gu, Lingyun Wang, and Jie Gan. Trace augmentation: What can be done even before preprocessing in a profiled sca? In *International Conference on Smart Card Research and Advanced Applications*, pages 232–247. Springer, 2017.

[RAHL19] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. In *Proceedings of the aaai conference on artificial intelligence*, volume 33, pages 4780–4789, 2019.

[RM18] Aditya Rawal and Risto Miikkulainen. From nodes to networks: Evolving recurrent neural networks. *arXiv preprint arXiv:1803.04439*, 2018.

[RPBA21] Unai Rioja, Servio Paguada, Lejla Batina, and Igor Armendariz. The uncertainty of side-channel analysis: A way to leverage from heuristics. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 17(3):1–27, 2021.

[RSVC$^+$11] Mathieu Renauld, François-Xavier Standaert, Nicolas Veyrat-Charvillon, Dina Kamel, and Denis Flandre. A formal study of power variability issues and side-channel attacks for nanoscale devices. In *Annual Intrl. Conf. on the Theory and Applications of Cryptographic Techniques*, pages 109–128. Springer, 2011.

[RWPP21] Jorai Rijsdijk, Lichao Wu, Guilherme Perin, and Stjepan Picek. Reinforcement learning for hyperparameter tuning in deep learning-based side-channel analysis. *Cryptol. ePrint Arch., Report 2021/071*, 2021.

[SBD$^+$19] Andrew M Saxe, Yamini Bansal, Joel Dapello, Madhu Advani, Artemy Kolchinsky, Brendan D Tracey, and David D Cox. On the information bottleneck theory of deep learning. *Journal of Statistical Mechanics: Theory and Experiment*, 2019(12):124020, 2019.

[SKJ21] Seba Susan, Aishwary Kumar, and Anmol Jain. Evaluating heterogeneous ensembles with boosting meta-learner. In *Inventive Communication and Computational Technologies*, pages 699–710. Springer, 2021.

[SLP05] Werner Schindler, Kerstin Lemke, and Christof Paar. A stochastic model for differential side channel cryptanalysis. In *Intrl. Workshop on Cryptographic Hardware and Embedded Systems*, pages 30–46. Springer, 2005.

[SM02] Kenneth O Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary Computation*, 10(2):99–127, 2002.

[SM15] Jacob Schrum and Risto Miikkulainen. Discovering multimodal behavior in ms. pac-man through evolution of modular neural networks. *Trans. on Computational Intelligence and AI in Games*, 8(1):67–81, 2015.

[SZT17] Ravid Shwartz-Ziv and Naftali Tishby. Opening the black box of deep neural networks via information. *arXiv preprint arXiv:1703.00810*, 2017.

[VCB14] Nguyen Xuan Vinh, Jeffrey Chan, and James Bailey. Reconsidering mutual information based feature selection: A statistical significance view. In *Twenty-Eighth AAAI Conf. on Artificial Intelligence*, 2014.

[vdVKPB20] Daan van der Valk, Marina Krcek, Stjepan Picek, and Shivam Bhasin. Learning from a big brother-mimicking neural networks in profiled side-channel analysis. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2020.

[vdVP19] Daan van der Valk and Stjepan Picek. Bias-variance decomposition in machine learning-based side-channel analysis. *IACR Cryptol. ePrint Arch.*, 2019:570, 2019.

[vdVPB20] Daan van der Valk, Stjepan Picek, and Shivam Bhasin. Kilroy was here: The first step towards explainability of neural networks in profiled side-channel analysis. In *Intrl. Workshop on Constructive Side-Channel Analysis and Secure Design*, pages 175–199. Springer, 2020.

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

[Wei20] Leo Weissbart. Performance analysis of multilayer perceptron in profiling side-channel analysis. pages 198–216, 2020.

[WG21] Ke Wang and Ping Guo. A robust automated machine learning system with pseudoinverse learning. *Cognitive Computation*, 13(3):724–735, 2021.

[WHJ$^+$21] Yoo-Seung Won, Xiaolu Hou, Dirmanto Jap, Jakub Breier, and Shivam Bhasin. Back to the basics: Seamless integration of side-channel preprocessing in deep neural networks. *IEEE Transactions on Information Forensics and Security*, 16:3215–3227, 2021.

[WPB19] Leo Weissbart, Stjepan Picek, and Lejla Batina. One trace is all it takes: Machine learning-based side-channel attack on eddsa. In *Intrl. Conf. on Security, Privacy, and Applied Cryptography Engineering*, pages 86–105. Springer, 2019.

[WPP20] Lichao Wu, Guilherme Perin, and Stjepan Picek. I choose you: Automated hyperparameter tuning for deep learning-based side-channel analysis. *IACR Cryptol. ePrint Arch.*, 2020:1293, 2020.

[WPP21] Lichao Wu, Guilherme Perin, and Stjepan Picek. The best of two worlds: Deep learning-assisted template attack. *IACR Cryptol. ePrint Arch., 2021/959*, 2021.

[WPP22] Lichao Wu, Guilherme Perin, and Stjepan Picek. On the evaluation of deep learning-based side-channel analysis. In *International Workshop on Constructive Side-Channel Analysis and Secure Design*, pages 49–71. Springer, 2022.

[WWK+20] Lichao Wu, Léo Weissbart, Marina Krč, Huimin Li, Guilherme Perin, Lejla Batina, Stjepan Picek, et al. On the attack evaluation and the generalization ability in profiling side-channel analysis. *IACR Cryptol. ePrint Arch., 2020/899*, 2020.

[YGJP19] Shujian Yu, Luis Gonzalo Sanchez Giraldo, Robert Jenssen, and Jose C Principe. Multivariate extension of matrix-based rényi's α-order entropy functional. *Trans. on Pattern Analysis and Machine Intelligence*, 42(11):2960–2966, 2019.

[YP19] Shujian Yu and José C Príncipe. Simple stopping criteria for information theoretic feature selection. *Entropy*, 21(1):99, 2019.

[YWJP20] Shujian Yu, Kristoffer Wickstrøm, Robert Jenssen, and José C Príncipe. Understanding convolutional neural networks with information theory: An initial exploration. *Trans. on Neural Networks and Learning Systems*, 2020.

[ZBH+16] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. *arXiv preprint arXiv:1611.03530*, 2016.

[ZBHV20] Gabriel Zaid, Lilian Bossuet, Amaury Habrard, and Alexandre Venelli. Methodology for efficient cnn architectures in profiling attacks. *IACR Trans. on Cryptographic Hardware and Embedded Systems*, 2020(1):1–36, 2020.

Appendix A. Related Work at a Glance

Table 7 summarizes some of the most relevant studies discussed in detail in Section 2. Moreover, Table 8 compares InfoNEAT with existing studies considering NAS for SCA.

Appendix B. Network and Configuration Parameters

Here we expand on why and how some of the parameters in NNs are set in the InfoNEAT algorithm, which is tuned to launch key-recovery attacks against side-channel traces. Besides parameters set by the user and summarized in Table 1, some parameters given in Table 9 remain in their default setting as defined in the NEAT algorithm.

The fitness function and the fitness threshold: One of the major functions that guides the evolution process within the NEAT framework is the objective function or commonly known as the *fitness function.* The fitness function which is specified by the user is used to evaluate the quality of the solution or the genome. In the case of InfoNEAT, we have selected *logistic loss* (hereafter called log loss) as our fitness function as shown in Section 4. This is due to the fact that calculating log loss can give the same quantity as calculating the cross-entropy. Specifically, log loss is one of the most commonly used loss functions which measures the performance of a classification model whose output is a probability value ranging from 0 to 1. The lower the log loss value, the better the model is. With the fitness function selected, we must also specify the fitness threshold, the

Table 7: Table showing the most recent and state-of-the-art works in the field of SCA. The table also elaborates on the problem investigated as well as the comparison to our approach, InfoNEAT. In this table, RF, SVM refer to random forest and support vector machine algorithms, respectively.

Ref.	Problem investigated	Method	Comparison to our approach (InfoNEAT)
[vdVPB20]	Explains why MLP works, how the internal nodes are working, and how different they are compared to other models in terms of the internal representation	MLP	This work ends up using a smaller network and smaller training dataset similar to our approach.
[HHO20]	Uses the plaintext feature to make the ML models more powerful. Claims that CNN is most successful because of their effectiveness with raw data.	CNN	Without considering the plaintext, our approach along with [vdVPB20, PCP20, Wei20] have been quite effective and comparable to the works using CNN.
[PBP20]	Discusses a stopping criterion based on the analysis of mutual information.	MLP	We use Matrix-based Rényi's α-entropy CMI rather than usually applied mutual information to define stoping criterion. Network architecture and other hyperparameters are evolved automatically.
[HGG20]	Surveys the state-of-the-art techniques used for the purpose of SCA. And explains that most of the techniques including CNN and MLP are quite comparable when it comes to SCA.	CNN, MLP	In line with this survey, we use our methodology to train compact MLP-like networks.
[PHJ$^+$19]	Discusses the metrics that should be used for evaluating the models developed for the purpose of SCA.	MLP	We use the same metrics especially the average rank to evaluate our InfoNEAT model.
[WPB19]	Shows how all ML techniques are effective in attacking the EdDSA, especially CNN which was able to break the implementation with a single measurement.	RF, SVM, CNN	They attack a different dataset compared to us, but nevertheless, show that only few traces are required for an effective SCA.
[PSK$^+$18]	Evaluates the performance of CNN and MLP for the purpose of SCA. Also shows that considering the minor performance gains that CNN offers, it is not worthy to invest time and resource to design such a complex network.	CNN, MLP	We therefore use InfoNEAT to design simple and compact MLP-like networks and show that they are quite effective.
[BPS$^+$18]	Introduces the ASCAD dataset structured in a similar way as compared to a typical ML dataset.	CNN, MLP	We use the ASCAD dataset to evaluate our model and our results are comparable to the ones presented in this paper.
[Wei20]	Evaluates the performance of MLP and shows that they are quite effective for the purpose of SCA.	MLP	We also show that MLP-like network is very effective for SCA.
[PCP20]	Explains that output class probabilities are sensitive to small changes and thus the developed model is not generalizable. They use ensemble of different models to create a generalizable model.	MLP	We also use ensembles of models (stacking) to develop an effective model. Our stacked model involves submodel for each class, devised using the One-vs-All classification technique. And the whole training process is automated.
[WPP20]	This work tunes the hyperparameters using Bayesian optimization and random search.	CNN, MLP	Usual techniques are used to output a set a hyperparameters, where each set used to train a model. The mechanism to find the best model (i.e., without underfitting or overfitting) is partially automatic (i.e., searching a pre-defined set of parameters), in contrast to InfoNEAT. Our approach starts from a minimal size network and gradually evolves the network size and parameters automatically based on cross-entropy loss and CMI-values.
[RWPP21]	Talks about the problems related to traditional deep learning models- they are complex and there are too many hyperparameters to tune.	CNN	This work uses reinforcement learning (RL) techniques to tune the hyperparameters of CNN.

value when if reached halts the evolutionary algorithm. This threshold value is set as 0 as specified in Table 1.

Network parameters and hyperparameters: The following hyperparameters can be considered when running NEAT.

Initial number of hidden nodes (n_h) indicates how many hidden neurons are involved in the NNs in the first generation. NEAT starts out minimally to ensure that the solution in the lowest-dimensional weight space is searched first. This further enhances the time-complexity since smaller structures optimize faster than larger structures [SM02]. For this, the NNs in the first generation have one hidden layer with the number of neurons n_h. Over the generations, the number of neurons in each layer increases until the stopping criteria are fulfilled. These criteria include the *maximum number of generations* defined to stop NEAT if other criteria are not fulfilled. In our experiments, InfoNEAT always stops before reaching to this number of generations, thanks to the CMI-based criterion.

To determine with what probability the mutation operation adds a connection between existing nodes, *connection add probability* (P_c) can be set. This probability is not changed over the generations. The larger this probability is set, the more diverse range of NNs can be evolved. In doing so, the larger number of species and greater topological diversity lead to obtaining a larger population, although the per generation training time would increase. Nevertheless, we observe that $P_c = 0.8$ has been suitable in all our experiments. Similarly, *node add probability* (P_n) indicates the probability of inserting a new node in an existing connection. The high P_n encourages NEAT to explore new architecture per generation; hence, it is set to its maximum, i.e., $P_n = 1$. This probability remains the same over generations.

One of the most important network parameters is the *activation function*. In this paper,

Table 8: Different aspects of NAS considered in SCA-related studies and InfoNEAT. (✓, ✓*, and ✗ are used to denote a task that has been completely, partially (e.g., with a range defined for the hyperparameter values), and by no means performed, respectively.

Ref.	Max. no. of layers	Type of operation	Hyperparameters	Connection config.
[PBP20]	✗	✗	✓*	✗
[PP20]	✓*	✓	✓	✗
[KWPP21]	✓*	✓	✓	✗
[ZBHV20]	✓*	✓	✓	✗
[Wei20]	✗	✓	✓	✗
[PCP20]	✗	✓	✓	✗
[WPP20]	✓*	✓	✓	✗
[RWPP21]	✗	✓	✓	✗
InfoNEAT	✓	✓	✓	✓

Table 9: InfoNEAT hyper/parameters, mainly set to the default values defined in NEAT.

Parameters	Description	Values
fitness threshold (L_{TH})	When the fitness value meets or exceeds this value, the algorithm halts.	0.0
Network parameters		
initial num_hidden (n_h)	Refers to the initial number of hidden nodes in the network.	10
Maximum number of generations (T)	Refers to the maximum number of generations the algorithm runs until the fitness criteria is not met. Usually, the algorithm converges before it reaches T generations.	30
Connection add probability (P_c)	The probability that mutation will add a connection between existing nodes.	0.8
Node add probability (P_n)	The probability that mutation will add a new node, essentially replacing an existing connection with a node.	1.0
Activation function	Part of every node and helps calculate the output of the node when given an input or a set of inputs.	Leaky ReLU
Fitness function	Guides the evolution process within the NEAT framework.	Log loss

we have selected the *Leaky ReLU* or Leaky Rectified Linear Unit as our activation function for all the nodes (except the output nodes) rather than the widely used ReLU activation function (see Table 1). This was done to prevent the well-known "dying ReLU" problem – the scenario when a large number of the nodes output zero because of their inputs being negative, which causes the network to saturate early and train very slowly [HDR19]. Furthermore, Leaky ReLU has been shown to have better performance compared to ReLU and other activation functions such as tanh, sigmoid, etc. [HDR19].

Initialization of weights and batch size for training: To prevent convergence issues, including the *saturation problem* [Kum17], we look at two initialization techniques, namely He and Xavier. Figures 11a and 11b show the results of using the aforementioned initialization techniques in terms of log loss values for models trained and tested with different batch sizes. Note that we select the batch data from the dataset randomly, and we make sure that the data is balanced and unbiased. Moreover, for the sake of readability, we present these results for one class (sub-key), although similar observations are made for other classes. In this case, we use the same batch size for training and testing. Based on our results, the He initialization can lead to underfitting (see the curve in Figure 11a for batch size being greater than 250). Moreover, a sharp minimum is observed when batch size is equal to 100 that can result in poor generalization [KMN$^+$16]. Although it can be thought that the Xavier initialization is not a suitable method for networks with ReLU activation function, as our MLPs are not considerably deep, Xavier initialization can be as powerful as He initialization [Kum17]; hence, we choose Xavier initialization.

Batch size selection: To select the **batch size**, one has to make a trade-off between the size of the batches and the speed of training and generalization error. On the one hand, reducing the size of the batches can have a noise-like effect that is useful for regularizing the data. On the other hand, choosing a larger batch size has been considered as a solution

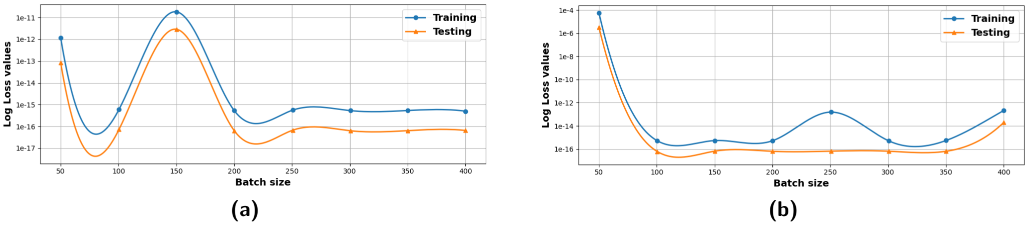


Figure 11: Log loss values for different batch sizes. The weights and biases are initialized based on (a) He, and (b) Xavier initialization techniques. The results are shown for a randomly chosen class.

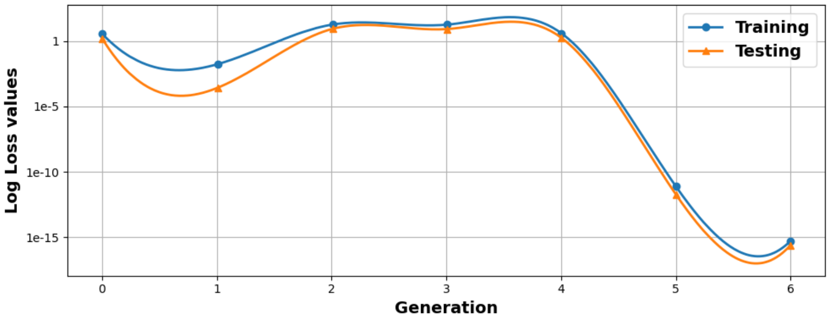


Figure 12: Learning curve showing the log loss values for different generations. The training and the testing curve values are obtained using the best trained model against the corresponding training and testing dataset in each generation respectively. The result is obtained for a randomly chosen class and the batch size is set to 150.

to speed up the training by parallelizing computations. In contrast to this, as reported in [MS16], EAs can run significantly faster by using smaller batch sizes. Hence, to keep the generalization gap as small as possible, we focus on the batch sizes 100 and 150. To speed up the training, we measure the average time spent to evolve the genomes before the algorithm halts, which is 653 s and 582 s for batch size equals to 100 and 150, respectively. Therefore, we use the batch size of 150. To observe the impact of this, the diagnostic learning curves are drawn, where the Xavier initialization technique is applied to train the model. The fittest genome or network in each generation is then tested against the test data, and the corresponding log loss values are plotted in Figure 12. The figure shows that the models trained with InfoNEAT are neither overfitting nor underfitting. This is a good sign that the trained model will be generalizable (see Section 6.3 for more detail). Additionally, the selection of Xavier initialization is further justified based on this result.

Appendix C. Examples Illustrating the Time and Memory-efficiency of InfoNEAT

Time-complexity: To further elaborate on the time complexity of InfoNEAT, Figure 13 illustrates the average time spent per generation and the average number of generations for each sub-key in $\text{ASCAD}^{\text{fixed}}_{\text{sync}}$ (a similar trend has been observed for $\text{ASCAD}^{\text{fixed}}_{\text{desync50}}$ and $\text{ASCAD}^{\text{fixed}}_{\text{desync100}}$). As can be understood from the graphs, although a set of NNs including the species and genomes is trained per generation, the training remains feasible thanks to the CMI-based stopping criterion. Interestingly, per submodel, the average training time (536.95s) is much less than what has been reported in [BPS$^+$18] (5475s) cf. [WPP20], and its order is comparable to the studies focused on hyperparameter tuning (405s) [WPP20]. Nevertheless, InfoNEAT offers additional benefits as discussed in Section 2.3, and as a trade-off, the training time is slightly increased.

Memory-efficiency: Additional information on the number of nodes and number

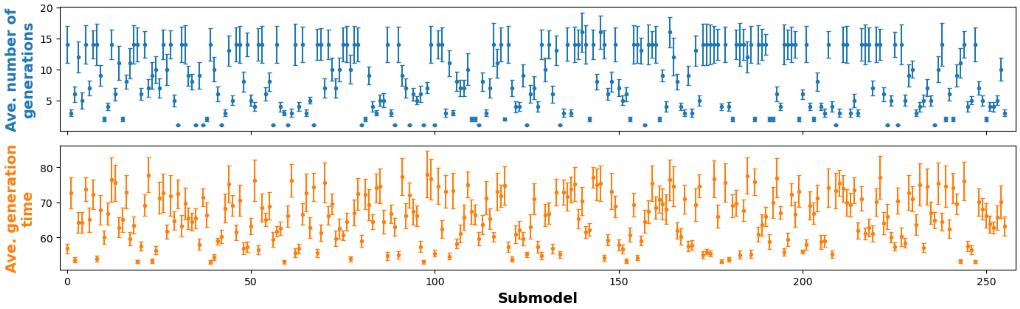


Figure 13: An example of the average number of generations (top) and generation time [s] (bottom) for all the 256 sub-models trained on ASCAD$^{\text{fixed}}_{\text{sync}}$. These values are obtained from across all ten folds to be trained. The error bars represent the Standard Error Mean (SEM) with 95% confidence interval.

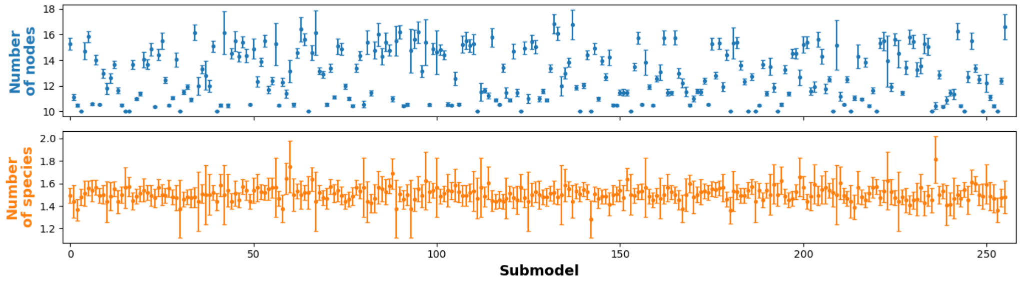


Figure 14: The number of nodes (top) and species (bottom) for all the 256 sub-models. These values are obtained for all N genomes across different generations throughout the training process. The error bars represent the SEM with 95% confidence interval.

of species for all classes (corresponding to 256 sub-keys and sub-models) is depicted in Figure 14. Note that although the genomes are relatively small, they perform well and recover the sub-key efficiently (see Figures 9 and 8).