Auto-tuning of Hyper-parameters for Detecting Network Intrusions via Meta-learning

Omar Anser, Jérôme François, Isabelle Chrisment

Auto-tuning of Hyper-parameters for Detecting Network Intrusions via Meta-learning

Omar Anser
Inria Nancy Grand Est, Université de Lorraine, France
omar.anser@inria.fr

Jérôme François, Isabelle Chrisment
Inria Nancy Grand Est, France
jerome.francois@inria.fr, isabelle.chrisment@loria.fr

***Abstract*—In recent years, machine learning-based Network Intrusion Detection Systems have been widely investigated to detect network attacks. The performance of such systems is strongly affected by their configuration, *i.e.* the setting of the hyper-parameters, usually based on human expertise. Few efforts have been made towards automatic methods except using a long process of trials. Besides, the resulting configuration is specific to the network where the system is deployed or the type of attacks to detect. To address these issues, we define a method using meta-learning which learns from the past experiences. By extracting useful information from the previous optimized tuning tasks, a model is trained in order to quickly infer a new configuration. In comparison with Bayesian optimization, our evaluation based on the CSE_CIC_IDS2018 and CIC_IDS2017 datasets demonstrates that our lightweight technique does not degrade attack detection accuracy in 88% of cases but is on average 9 times faster.**

***Index Terms*—Network Security; NIDS Auto-tuning; Machine Learning; Meta-learning; Bayesian Optimization**

I. Introduction

A Network Intrusion Detection System (NIDS) is in charge of analyzing network traffic to detect malicious activities. With the sophistication of attacks and the full encryption of network traffic, conventional signature-based detection approaches may fail. Hence, leveraging machine learning (ML) has been widely adopted [1].

An important step lies in the configuration of the ML algorithm, known as hyper-parameter optimization [2]. In this paper, the hyper-parameter optimization (HPO) of an ML-based NIDS is referred to *configuration optimization*. Several strategies such as Grid Search and Bayesian Optimization have been successfully applied [1]. However, most studies have overlooked two important issues. First, the use of these conventional methods lead to significant costs in terms of computation and time as they solve the configuration optimization problem starting from scratch. Second, a ML-based NIDS, once built, is only effectively applicable in a similar network context that includes the network topology, services and applications running on computers, applications or types of attacks from which the data was obtained for its initial generation. Thus, for a new network context, building and configuring a new ML model is recommended.

For instance, during preliminary experiments with the 17-day dataset used in this paper, learning a unique optimal configuration over the 17 aggregated days leads to a performance degradation up to 21% (based on the Matthews Correlation Coefficient metric) for one day compared to a daily updated configuration. The results are even worse if a day's NIDS is trained with the optimal configuration of a different day. Globally, 6 out of 17 days highlight an average MCC degradation between 3% and 56%. This is also observed when proceeding each dataset hourly. The degradation can reach 16%, 26% or even 100% on specific days. Hence, on one hand, although a configuration may be valid on different days or at different times, there is clearly room for improvement in order to optimize the NIDS configuration. On the other hand, increasing the reconfiguration frequency to maintain optimal performance using a conventional HPO method would waste computational resources.

To address both computation expensive and the need of executing HPO from scratch at each context change for better performance, we propose to optimize the NIDS configuration on various contexts with HPO and reuse these past experiences with meta-learning. The latter is applied during an early phase to acquire a meta-dataset containing (1) meta-features characterizing an ensemble of heterogeneous datasets, and (2) information about the corresponding optimal NIDS configurations. The meta-dataset is then used to learn a regression model denoted as the meta-model. Once a new dataset is collected, the NIDS is configured on the fly by inferring its configuration using the trained meta-model. The evaluation shows that the detection performance is close to that of an HPO technique with a significant gain in terms of resource usage (nine times faster on average).

This paper is organized as follows: the problem is formalized in Section II. Section III discusses the related work. Our proposed method is introduced in Section IV. We describe the experimental setup in Section V and analyze the results in Section VI. We conclude this paper in Section VII.

II. Problem definition

A. Network Intrusion Detection System

We assume a NIDS detecting attacks in network flows. Each flow is characterized by statistical features such as duration, number of packets, number of bytes, etc. and an attack can be detected through specific deviations in these features. For example, a flow with a number of packets significantly higher may reveal a brute-force attack. Assuming datasets with flows labeled with particular attacks, a ML algorithm can be trained on the aforementioned features to detect attacks in new

network traffic. We focus on a binary decision to determine whether an attack occurs.

B. ML-based NIDS configuration problem

Assuming a ML-based NIDS with a set of k (hyper-)parameters (HP), our optimization problem is defined as follows:

Definition 1: Let $C = C_1 \times C_2 ... \times C_k$ be the configuration space. Each C_i represents a set of possible values for the i^{th} configurable HP and can be real-valued, integer-valued, binary or categorical ($i \in 1, ..., k$).

Example 1: If SVM is used, it is mostly affected by the values of four HPs: the kernel function (C_1), its width (C_2) or polynomial degree (C_3), and the regularized constant (C_4). All these are real-valued except C_1 that is categorical.

Definition 2: Additionally, let $D = D_1 \times D_2 ... \times D_m$ be a set of datasets. The function $p : D_j \times c \rightarrow \mathbb{R}$ measures the attack detection performance on the dataset $D_j \in D$ given a configuration $c = (c_1, c_2, ..., c_k) \in C$. For example, precision, recall or f1-score are possible outputs of p.

Definition 3: Given C and $D_j \in D$, the goal is to find $c^* = (c_1^*, c_2^*, ..., c_k^*)$ such that p is maximal:

$$c^* = \arg\max_{c \in C} p(D_j, c) \tag{1}$$

For practical reasons, it is approved to search reduced C (C^{red}). Also, c^* that leads to a NIDS with high detection performance for D_i may not lead to the same performance for other datasets $D_j, j \neq i$. Finally, in order to properly react against attacks, a NIDS must be rapidly reconfigured.

III. Related work

A. Conventional HPO methods

To configure a ML-based NIDS, experts can manually tune HPs. Ad-hoc trial-and-error techniques are often used but Grid Search (GS) and Random Search (RS) are the two most used strategies.

GS is a brute-force method that evaluates all combinations of C_i ($i \in 1, ..., k$) according to an initialization of C^{red}. Instead of trying all possible combinations, RS randomly selects a predefined number of samples for each C_i between their upper and lower bounds. Actually, a NIDS can be viewed as a traffic classification problem. The authors in [3] use a RS solution that took more than 56 hours to classify attacks. Lim et al [4] rely on GS to configure a RNN-based model to classify network packets. Different algorithms are tuned also with GS in [5] to classify encrypted TLS traffic. Labonne et al. [6] have first chosen RS but due to lack of performance, they have given this method up in favor of Bayesian optimization (BO).

Bayesian optimization (BO) is a sequential method that determines the future configuration to be evaluated based on the previously obtained results. BO uses two key components: a *surrogate model* $\widetilde{p}$ which is a probabilistic model approximating the true function p of the ML model, and an *acquisition function* that aims to detect $\widetilde{c^*}$ of $\widetilde{p}$. BO starts with a small non empty *observation* set $O \subset C^{red}$ of configurations for which $p(D_j, c)$ are being computed for each $c \in O$. For each iteration and until a stopping criteria, O is extended by a new $\widetilde{c}$, such that the expected value of $p(D_j, \widetilde{c})$ is maximal according to $\widetilde{p}$. The acquisition function determines the utility of different $\widetilde{c}$, trading off exploration and exploitation.

All these techniques can help to configure a ML-based NIDS but they are also very resource consuming.

B. Meta-learning

There are different interpretations of the term meta-learning. In this paper, meta-learning refers as a learning paradigm in which a ML model gains experience over several learning episodes based on many datasets [7]. Thus, meta-learning can be adapted to particular applications such as AutoML & HPO and few-shot learning. It can also be meaningfully combined with ensemble learning.

Xu et al. [8] apply a few-shot learning algorithm to perform network traffic classification. The aim of few-shot learning is to train an accurate model using few labeled examples. The authors classifies network flows using two deep neural networks for feature extraction and comparison and they manually set the HPs values of their model. Although the authors' use of meta-learning differs from ours, it may have a positive impact on the configuration time, but the authors do not specify any information about this.

Mantovania et al. [9] propose a HPO recommendation framework for a SVM classifier based on meta-learning. Their method is a binary recommender system that attempts to minimize the time cost of optimizing the HPs of an SVM classifier by predicting whether or not using a conventional HPO method would be more advantageous in terms of accuracy compared to using the default values. Unlike our work, their framework does not propose an alternative to the use of a classical HPO method but can be complementary to our solution.

Possebon et al. [10] present different architectures for network traffic classification that combine meta-learning and ensemble learning. In this case, meta-learning involves combining information generated from previous learning systems (e.g. with a voting process). It thus differs from our approach by definition as our goal is to optimize HPs.

Bui et al. [11] apply meta-learning for Intelligent Transportation System (ITS) traffic classification and select the previous best configuration as the one to apply to all. So, this assumes homogeneous datasets, what differs from our case.

The novelty of our approach arises from combining meta-learning and automated tuning to directly generate a new and specific HPs configuration of a ML-based NIDS. Our work can be categorized as an AutoML & HPO solution as it automates the process of finding near-optimal HPs.

IV. Method

To find an accurate configuration over a large space in a timely manner, we rely on meta-learning (MtL) to approximate c^* (Section II-A).

The core idea is to build a knowledge base of prior experiences in attack detection, called *meta-dataset*, to train

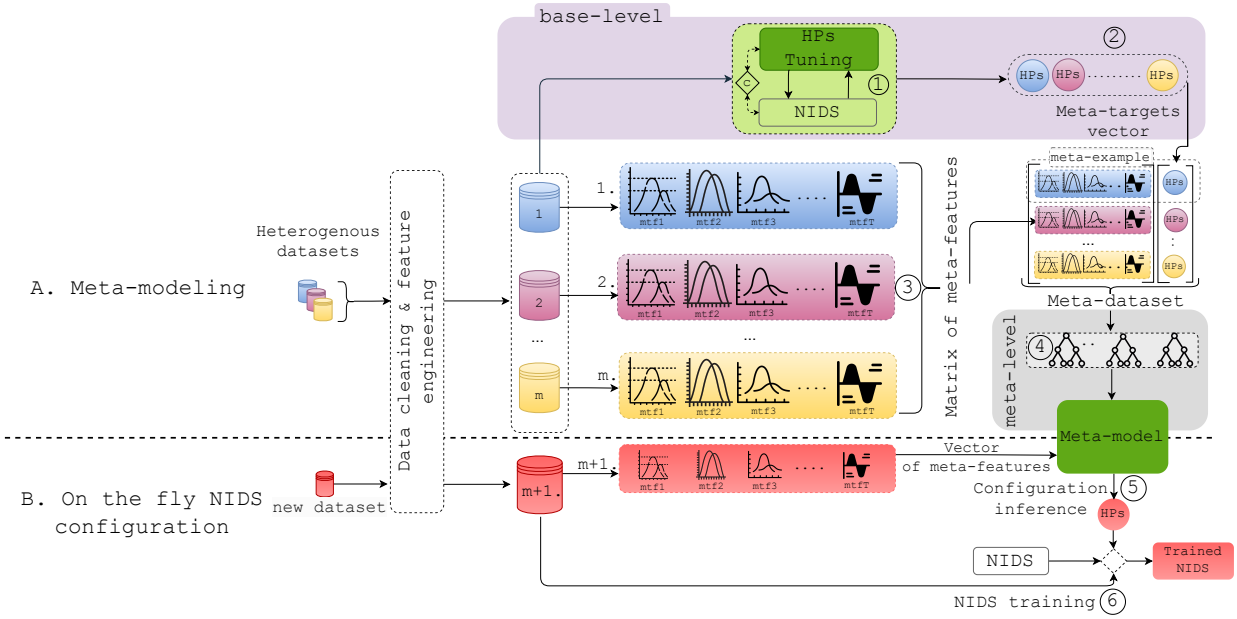

Fig. 1. Meta-learning framework for NIDS configuration optimization

a regression model, named *meta-model*, able to predict on the fly the NIDS configuration for an unseen dataset. We consider that these detection experiments have been performed on datasets representative of heterogeneous contexts (network topology, types of attacks ...). The meta-dataset's construction and the meta-model's learning have to be done upstream in a *meta-modeling phase*, while the prediction of the new NIDS configuration can be done *on the fly* as shown in Figure 1, detailed in following subsections.

A. Meta-modeling phase

The goal of this phase is to build, from a meta-dataset, a meta-model that emulates the behavior of a conventional HPO solution. A meta-dataset is composed of a set of meta-examples. Each meta-example stores: (1) the features characterizing a given dataset, called meta-features; and (2) the NIDS's optimal configuration used for this particular dataset. Our technique relies on 36 meta-features extracted from [12] and listed in Table I. They are divided into:

- *Simple meta-features*, such as the number of attributes or the number of instances.
- *Statistical meta-features* such as kurtosis which measures the dispersion of numeric attributes.

They have been selected in order to allow fast computation and to be independent of the nature of underlying data. By using the *pymfe* [1] open source library, the reproducibility of meta-features generation is ensured.

The meta-modeling phase is illustrated in Figure 1, Part A. This phase is composed of two steps: a base-level step (colored in purple) where the HP tuning process is performed for each dataset, and a meta-level step (colored in grey) where the meta-model is constructed. First, for the base-level's input, each heterogeneous dataset is processed individually to find the best corresponding configuration using an HPO solution as described in Section III-A ①. We obtain a vector of optimal HPs values called *meta-target* vector ②.

Second, in parallel and aside from base- or meta-level, meta-features are extracted for each dataset ③, and they are all consolidated into a single matrix M. The vector of meta-targets and M constitute the meta-dataset associating a target configuration with meta-feature values.

Finally, at the meta-level, a Random Forest regressor (RFr), called *meta-learner*, is learned over the meta-dataset ④ and constitutes the meta-model. The latter is able to predict a meta-target (a configuration) based on meta-features calculated from any dataset. The meta-learner (Rfr) was tuned with a RS technique with a budget of 100 evaluations.

TABLE I
LIST OF IMPLEMENTED META-FEATURES

Simple meta-features:	The eigenvalues of covariance matrix
Ratio attributes instances	Attribute geometric means
Ratio categoric numeric features	Attribute harmonic means
Relative frequency of distinct class	Attribute interquartile ranges
Ratio instances attributes	Attribute kurtosis
Total number of attributes	Attribute median absolute deviation
Number of binary attributes	Attribute maximum values
Number of categorical attributes	Attribute mean values
Number of distinct classes	Attribute median values
Number of instances	Attribute minimum values
Number of numeric features	Number of distinct correlated attributes
Number of numerical and categorical features.	Number of attributes with at least one outlier.
	Attribute ranges
Statistical meta-features:	Attribute standard deviations
Attribute skewness	Canonical correlations of the data
Gravity of the numeric dataset	Attribute sparsities
Pillai's trace value	Lawley-Hotelling trace value
Wilk's lambda value	Attribute trimmed means
Attribute variances	Absolute value of correlation between distinct attributes.
	Number of canonical correlations between each attribute and class.

[1] https://github.com/ealcobaca/pymfe, accessed on Jan. 30 2023

B. On the fly NIDS configuration phase

During this phase (Figure 1, Part B), the previously generated meta-model (RFr in our case) is used based on a new context described again through a particular dataset. A new meta-feature vector is generated and the RFr predicts almost instantaneously well-adapted HPs for the NIDS without using an expensive solution ⑤ (time of the inference only). Finally, the NIDS is trained with these HPs ⑥.

The NIDS does not have to be evaluated multiple times to find a suitable configuration (unlike GS for instance). It is now a straightforward operation as it only requires an inference from previously trained regressor. In this lightweight step, the NIDS is solicited only once in the learning stage with the processed dataset and the predicted configuration to produce the final detection model in a single shot. Our approach is agnostic to the tuning method used in the base-level.

V. Experimental Setup

Experiments were carried out on a single core of an Intel Xeon CPU E5-2650 0 @ 2.00GHz. The detection process was performed offline based on previously collected data.

A. Intrusion detection datasets

We use two publicly available datasets, the refined version of the cic-ids2017 [13] and the cse-cic-ids2018 [14]. Each record is a flow, described with its features and labeled as either a normal or an attack. The datasets contain 80 features with a strong correlation, suggesting that many of them are redundant. Feature engineering is based on: (1) The feature selection analysis provided by the datasets's authors, (2) The most common features found in NIDS solutions in practice [15]. As a result, we consider the features listed in Table II.

TABLE II
Features considered in each dataset

Feature name	Description
dst_port	Destination port numbers (one hot encoding). Used to specify the services offered on remote hosts
protocol	Network protocol of the flow
flow_duration	Flow duration
tot_fwd_pkts	Total packets in the forward direction
tot_bwd_pkts	Total packets in the backward direction
totlen_fwd_pkts	Total size of packet in forward direction
totlen_bwd_pkts	Total size of packet in backward direction
flow_byts_s	Number of bytes transferred per second
flow_pkts_s	Number of packets transferred per second
fwd_iat_tot	Total time btw two pkts sent in the forward direction
bwd_iat_tot	Total time btw two pkts sent in the backward direction
fwd_pkts_s	Number of forward packets per second
bwd_pkts_s	Number of backward packets per second

17 different attacks scenarios have been run. They can be grouped into 6 categories: DoS, Bruteforce, Web attack, Botnet, DDoS and Infiltration as shown in Table III. The datasets are split per day. A day contains a single category of attacks but possibly multiple occurrences. Normal traffic is generated continuously whereas attacks are executed over short periods of time.

Since our goal is to have multiple contexts, each day is considered a single dataset, as each differs from the others due to a different network context or attack category. The exception is for cse-cic-ids2018 where the Web and Infiltration attacks are repeated on two different days, as shown in Table IV.

B. Network Intrusion Detection System

Our goal is to train a NIDS capable of detecting attacks independently of their type and is therefore based on a binary classifier. Since we are not looking to design a new NIDS with high detection performance, but to assume a particular one and configure it properly, we consider a Random Forest (RFc) classifier. We configure it in the base-level using Tree Parzen Estimator (TPE), which is a single objective BO method (using the Hyperopt implementation[2]). We focus on the two following HPs as they are the most influential on the performance of a RFc [16]:

- *max features* (numerical): the number of features to consider each time to make the split decision.
- *min samples leaf* (numerical): the minimum number of samples required to be at a leaf node.

To estimate the performance and avoid overfitting, most of the current studies on HPs tuning adopted Nested cross-validation [17]. We adopt this methodology for the tuning step in the base-level with a 10-folds for the outer loop and 3-folds for the inner loop.

VI. Results and Analysis

For evaluation purposes, a positive sample corresponds to an attack flow and a negative sample to a normal flow. We calculate four evaluation metrics: MCC, f1-score, precision and recall. We compare the MtL-based NIDS performance against two configurations: a default configuration, which is provided by the ML tools and is independent of the nature of the data (we use Scikit-learn which initializes by default the max features to 3 [3] and the min samples leaf to 1), and an optimal configuration returned by the TPE method (HPO).

Each day (17 in total) is considered as a new dataset on which a configuration must be returned. For MtL, the other 16 days are used to learn the meta-model. By rotating over the days, we obtain 17 results. The MtL-based NIDS is trained using 80% of the day's data and tested on the remaining 20% in a stratified manner.

On each day, the HPO is performed with a budget of 300 iterations. Thus, to build the whole meta-target vector, 4800 HPO iterations are required (with one iteration always slightly less or equal to 10 seconds). In practice, this number is not known in advance, and we select it after several attempts.

When applying HPO, we considered the MCC as the objective metric to be maximized. MCC is known to be an appropriate metric for measuring overall classification performance, especially for extremely unbalanced datasets.

In the next sections, we present our experimental results and expect our technique to lead to a minimal degradation of detection performance in comparison with an optimal configuration (based on HPO) while being more resource efficient.

[2]http://hyperopt.github.io/, accessed on Jan. 30 2023

[3]The result of the floor function of the square root of the number of features

TABLE III
CATEGORY OF THE ATTACKS IN CIC-IDS2017 AND CSE-CIC-IDS2018

Main categories	Attacks
DoS	Hulk, GoldenEye, Slowloris, Slowhttptest, Heartbleed
DDoS	HTTP flood, UDP flood, TCP flood
Bruteforce	FTP-Patator, SSH-Patator
Web attack	XSS, HTTP bruteforce, SQL injection
Botnet	Ares, Zeus
Infiltration	IP sweep, port scan

TABLE IV
LIST OF THE DATASETS

day id:Name	Attack	% Bening	% Attack	#flows
0:bot_02-02-2018	Botnet	82.37	17.63	819904
1:bruteforce-ftp-ssh_14-02-2018	Bruteforce	85.97	14.03	671126
2:bruteforce-web-xss-sql-injection_22-02-2018	Web attack	99.96	0.04	901548
3:bruteforce-web-xss-sql-injection_23-02-2018	Web attack	99.94	0.06	905177
4:ddos-loic-http-loic-udp_20-02-2018	DDoS	90.52	9.8	6073495
5:ddos-loic-udp_hoic_21-02-2018	DDoS	64.26	35.74	561405
6:dos-goldeneye-slowloris_15-02-2018	DoS	94.14	5.86	876946
7:dos-slowhttp-hulk_16-02-2018	DoS	75.45	24.55	591901
8:infiltration_01-03-2018	Infiltration	72.31	27.69	279818
9:infiltration_28-02-2018	Infiltration	90.31	9.69	347198
10:Friday2017-Afternoon-DDos	DDoS	57.38	42.62	223082
11:Friday2017-Afternoon-PortScan	Infiltration	57.57	42.43	213777
12:Friday2017-Morning-Bot	Botnet	98.94	1.06	184044
13:Thursday2017-Afternoon-Infilteration	Infiltration	99,98	0.02	252790
14:Thursday2017-Morning-WebAttacks	Web attack	98.69	1.31	164173
15:Tuesday2017-WorkingHours	Bruteforce	97.82	2.18	421626
16:Wednesday2017-workingHours	DoS	68.26	31.74	610445

A. Detection Performance

The experimental results are shown in Table V. The day identifiers are the ones from the Table IV. 10 out of 17 days are not reported because the detection performance obtained for these days using all the configuration techniques is higher than 0.99. Hence, during these days, MtL is as efficient as HPO and default.

On day 12, the configurations returned by the three methods give the same level of performance. Their values are 0.801 (MCC), 0.783 (f1-score), 1.0 (precision) and 0.644 (recall). Note that even if the inferred configuration differs (HPs values), it means that attack and normal traffic flows are classified similarly by the different RFcs generated.

On the days 2, 3, 8, 9 and 14 and in comparison to the default, HPO improves the MCC with an average percentage increase of 27.79% (with values ranging from 5.54% for day 2 to 80.24% for day 9). This is an expected result since HPO's objective is to maximize this metric. MtL manages to improve it also in a similar order of magnitude with an average of 25.28% (with values ranging from 4.47% for day 2 to 79.01% for day 9). Although the precision was not taken into account in the HPO's objective function, it increases with an average of 62.03% (with an improvement of up to 220% on day 9). For MtL, the improvement is 58.69%. Regarding the recall, HPO and MtL bring a gain at day 14 where it increases from 0.767 to 0.97 for HPO and to 0.946 for MtL.

MtL is the least efficient on day 13. Actually, HPO manages to improve the NIDS's performance while MtL performs poorly on three of the four metrics. HPO increases the MCC's default configuration from 0.771 to 0.847, the f1-score from 0.769 to 0.855 and the recall from 0.714 to 0.865. HPO increases the precision from 0.833 to 0.847. MtL outperforms HPO on precision (= 1) but at the cost of a low recall. In that case the NIDS misses a lot of attacks but never raises a false positive alarm. As seen in Table IV, day 13 is the only day combining both a relative low number of flows and a high imbalance between normal and attack traffic. This might have a major impact on the values of constructed meta-features and so on the learning phase of the meta-model.

Generally, HPO and MtL improve the performance on almost every day in comparison with the default configuration. Over the five relevant days (2, 3, 8, 9 and 14), MtL is at the same level of HPO performance. The difference is minimal with only 2.62% gain in MCC and 3.37% gain in precision for HPO compared to MtL. Thus, MtL manages to learn well from previous optimal configurations (based on the 17 other days) with a little degradation (less than 3.40%) in NIDS performance in comparison to HPO.

B. Resource usage

We do not consider the time needed to learn the meta-model, *i.e* the meta-modeling phase, since it is done in advance. In our case, we have a new dataset every day, and after a number of uses of HPO (16 in this example), we expect to be able to have the new day's configuration (the 17th) quickly.

The time MtL takes to infer a configuration is equivalent to the meta-model's inference time (in our case a RFr) but, unlike HPO, it requires first to extract the meta-features for the new dataset. We thus compare the time to generate a configuration for MtL and HPO by measuring the number of HPO iterations.

For MtL, we converted the meta-features's extraction time in an equivalent number of HPO iterations. In our experiments, the meta-model's inference time is between 7 and 9 seconds. For simplicity, we consider it to be equal to one HPO iteration, which can be considered as the worst case since one iteration can be up to 10 seconds. For HPO, knowing the minimum number of iterations to achieve the maximum NIDS performance cannot be predicted. Hence, we consider the number of iterations performed by HPO to reach the same detection performance as MtL.

As shown in Figure 2, MtL is executed in quasi-constant number of iterations between 5 and 8. So, even for days 1, 5 and 8 where HPO is slightly better (about 3 to 5 iterations), the practical impact for the NIDS is negligible, *i.e* using MtL will take 30 or 50 seconds longer than HPO at most in these days. In contrast, HPO takes significantly more iterations to find an equivalent configuration for the other days. This value highly varies with extreme values above 100. Globally, MtL

TABLE V
COMPARISON PERFORMANCE RESULTS (BEST VALUE IN GREEN, WORST VALUE IN RED, INTERMEDIATE VALUE IN BLACK)

	MCC			f1-score			precision			recall		
day id	Default	HPO	MtL	Default	HPO	MtL	Default	HPO	MtL	Default	HPO	MtL
2	0.715	**0.757**	0.747	0.702	**0.729**	0.717	0.87	**1.0**	**1.0**	**0.588**	0.574	0.559
3	0.558	**0.619**	0.596	0.525	**0.554**	0.524	0.792	**1.0**	**1.0**	**0.393**	0.383	0.355
8	0.333	**0.443**	0.424	**0.465**	0.441	0.457	0.6	**0.916**	0.831	**0.379**	0.29	0.315
9	0.081	**0.146**	0.145	**0.121**	0.069	0.069	0.22	**0.705**	0.698	**0.084**	0.036	0.036
12	0.801	0.801	0.801	0.783	0.783	0.783	1.0	1.0	1.0	0.644	0.644	0.644
13	0.771	**0.847**	0.368	0.769	**0.855**	0.23	0.833	0.847	**1.0**	0.714	**0.865**	0.141
14	0.767	**0.845**	0.837	0.77	**0.84**	0.834	**0.772**	0.74	0.745	0.767	**0.97**	0.946

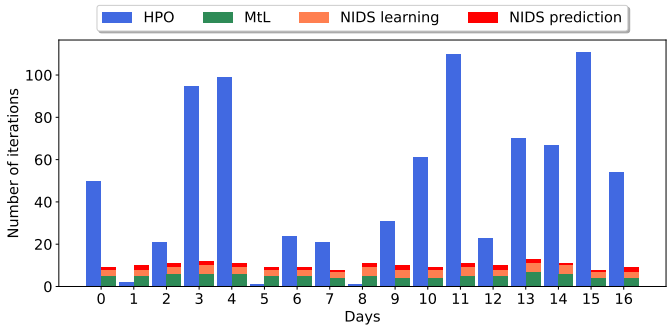

Fig. 2. Configuration generation, learning and prediction time of the NIDS

is thus faster with a factor of 9 in average over all days. Besides, it would be difficult to deduce the right number of HPO iterations in advance.

With our method, the model still needs to be trained with the configuration and used on the test set to predict whether the flows are malicious or not. This is not the case for HPO because training and prediction are done (multiple times) in parallel with the search for optimal HPs. Figure 2 also includes the learning and prediction time of the MtL-based NIDS. The learning time is fairly constant and represents between 3 and 5 iterations. For the prediction time, it is between 1 and 2 iterations. Hence, in a normal scenario where the number of iterations of HPO is not known in advance, building a NIDS and predicting the type of new flows takes 300 iterations for HPO while MtL takes between 9 and 15 iterations.

Regarding memory usage, the whole day dataset is necessary for HPO. The best case is day 11 with a size of 18.79 MB, and the worst case is day 4 with 837.33 MB. Since MtL only uses meta-features, less than 36KB is used for any day.

VII. CONCLUSION AND FUTURE WORK

To avoid performance degradation of a ML-based NIDS, adapting its configuration to a new context is necessary. Using a common HPO technique is costly. In this paper, we propose an alternative that leverages meta-learning to infer a configuration automatically from past experiences. The results demonstrate an acceptable degradation of detection accuracy while the configuration is in average 9 times faster.

In the experimental datasets, each day is a particular scenario. However, in practice, the changes in traffic profiles might be more frequent and less periodic. Hence, we plan to investigate how to determine when the configuration must be revised according to changes in collected features within shorter and variable time windows.

ACKNOWLEDGMENT

This work was partially funded by the H2020 AI@EDGE project under grant agreement No 101015922.

REFERENCES

[1] A. L. Buczak and E. Guven, "A survey of data mining and machine learning methods for cyber security intrusion detection," *IEEE Communications Surveys Tutorials*, 2016.
[2] L. Yang and A. Shami, "On Hyperparameter Optimization of Machine Learning Algorithms: Theory and Practice," *Neurocomputing*, 2020.
[3] Y. N. Kunang, S. Nurmaini, D. Stiawan, and B. Y. Suprapto, "Attack classification of an intrusion detection system using deep learning and hyperparameter optimization," *Journal of Information Security and Applications*, 2021.
[4] H.-K. Lim, J.-B. Kim, J.-S. Heo, K. Kim, Y.-G. Hong, and Y.-H. Han, "Packet-based Network Traffic Classification Using Deep Learning," in *2019 International Conference on Artificial Intelligence in Information and Communication*.
[5] B. Anderson and D. McGrew, "Machine Learning for Encrypted Malware Traffic Classification: Accounting for Noisy Labels and Non-Stationarity," in *SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2017.
[6] M. Labonne, A. Olivereau, B. Polvé, and D. Zeghlache, "A Cascade-structured Meta-Specialists Approach for Neural Network-based Intrusion Detection," in *IEEE Annual Consumer Communications Networking Conference (CCNC)*, 2019.
[7] J. Vanschoren, "Meta-learning: A survey," 2018.
[8] C. Xu, J. Shen, and X. Du, "A method of few-shot network intrusion detection based on meta-learning framework," *IEEE Transactions on Information Forensics and Security*, 2020.
[9] R. G. Mantovani, A. L. Rossi, E. Alcobaça, J. Vanschoren, and A. C. de Carvalho, "A meta-learning recommender system for hyperparameter tuning: Predicting when tuning improves svm classifiers," *Information Sciences*, 2019.
[10] I. P. Possebon, A. S. Silva, L. Z. Granville, A. Schaeffer-Filho, and A. Marnerides, "Improved Network Traffic Classification Using Ensemble Learning," in *IEEE Symposium on Computers and Communications (ISCC)*, 2019.
[11] K.-H. N. Bui and H. Yi, "Optimal Hyperparameter Tuning using Meta-Learning for Big Traffic Datasets," in *2020 IEEE International Conference on Big Data and Smart Computing* , 2020.
[12] E. Alcobaça, F. Siqueira, A. Rivolli, L. P. F. Garcia, J. T. Oliva, and A. C. P. L. F. d. Carvalho, "MFE: Towards reproducible meta-feature extraction," *Journal of ML Research*, 2020.
[13] G. Engelen, V. Rimmer, and W. Joosen, "Troubleshooting an intrusion detection dataset: the cicids2017 case study," 2021.
[14] I. Sharafaldin, A. Habibi Lashkari, and A. Ghorbani, "Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization," in *4th International Conference on Information Systems Security and Privacy*, 2018.
[15] A. Khraisat, I. Gondal, P. Vamplew, and J. Kamruzzaman, "Survey of intrusion detection systems: techniques, datasets and challenges," *Cybersecurity*, 2019.
[16] J. N. van Rijn and F. Hutter, "Hyperparameter importance across datasets," in *International Conference on Knowledge Discovery & Data Mining*, 2018.
[17] G. C. Cawley and N. L. C. Talbot, "On Over-fitting in Model Selection and Subsequent Selection Bias in Performance Evaluation," *Journal of ML Research*, 2010.