# Certainly Uncertain: Demystifying ML Uncertainty for Active Learning in Network Monitoring Tasks

Katharina Dietz*, Mehrdad Hajizadeh†, Nikolas Wehner*, Stefan Geißler*, Pedro Casas‡, Michael Seufert§, Tobias Hoßfeld*
**University of Würzburg, Germany*, †*Technical University of Chemnitz, Germany*
‡*AIT Austrian Institute of Technology, Vienna, Austria*, §*University of Augsburg, Germany*
*{katharina.dietz, nikolas.wehner, stefan.geissler, tobias.hossfeld}@uni-wuerzburg.de,
†mehrdad.hajizadeh@etit.tu-chemnitz.de, ‡pedro.casas@ait.ac.at, §michael.seufert@uni-a.de

*Abstract*—**Artificial Intelligence (AI), particularly Machine Learning (ML), has become prominent in network monitoring, yet its practical adoption, such as for anomaly and intrusion detection, remains limited. Standard AI/ML methods often exclude experts, reducing trust and hindering practical implementations. Active Learning (AL) allows to integrate admins and their expert knowledge into the ML loop by leveraging expert-labeled data. Together with self-training and automated decisions, AL can enhance model performance, trust, and the ability to adapt to system changes. In this work, we evaluate uncertainty-based AL in network monitoring, offering a comprehensive parameter study for best practices in real-world AI/ML adoption. To this end, we evaluate stream-based and pool-based AL across four datasets for various monitoring use cases and conduct a parameter study on ten uncertainty measures, thereby identifying scenarios benefiting from self-training. By analyzing the impact of admin competence on model performance, we offer actionable guidelines towards the practical implementation of AL.**

*Index Terms*—**Active Learning, Machine Learning, Traffic Classification, Intrusion Detection**

## I. Introduction

In the past decade, Artificial Intelligence (AI), especially Machine Learning (ML), has enjoyed increasing popularity and has been widely applied in network monitoring and management. However, when it comes to sensitive topics such as network security, practical adoption has been poor, e. g., for anomaly and intrusion detection [1]. Academic research on security-related topics lacks a holistic view [2], focusing on either human or technical aspects, while neglecting the interconnection of both [2]. Incorporating standard AI/ML approaches further widens this gap and creates barriers [3], as they take away the decision-making from the admins without giving any information about the confidence or severity of their decision, and provide no means to give feedback to the model, ultimately reducing the trust of the admins. Having an admin in the loop can be beneficial not only for the performance of the model by incorporating expert knowledge, but also increase its trustworthiness – one of the goals of the European AI Act[1].

Originally, Active Learning (AL) aims to increase the performance of an ML model by manually labeling as few instances as possible via queries to an *oracle* [4], e. g., a (human) expert or a more powerful model, and add these few selected instances to the (re)training data. A complementary technique to AL is self-training [5], which consists of a base classifier that classifies unlabeled data and, subsequently, adds the instancs it predicts with high confidence to the (re)training datasets. While AL aims for the most informative data (e. g., highest uncertainty), self-training targets the most confident data to add to the model. Combining both techniques can yield great benefits [5], i. e., making automated decisions for obvious choices, while relaying less confident decisions to the oracle, thus incorporating a human in the AI/ML loop.

In the context of communication networks, these approaches help enable the real-world deployment of AI/ML solutions by giving the users decisive power over critical administrative decisions, while enriching the monitored data with expert knowledge. When unseen devices, applications, or even zero-day attacks start appearing in the network (i. e., equating to potential labels the model was not trained on), the model may not be very confident in its decision and thus relay it to the expert admin for further inspection and relabeling. Ultimately, this also helps keeping the model up-to-date and adapt to network changes over time, e. g., mitigating concept drift [6].

Thus, the goal of this paper is to evaluate different approaches to evaluate ML uncertainty in the context of a wide range of network monitoring tasks. We aim to fill the gap in literature by providing an extensive simulative parameter study, including the admin competence level, confidence thresholds, or the impact of self-training. With this parameter study, we establish best practices towards enabling real-world adoption of AI/ML-based monitoring with humans in the loop and self-training models. The main contributions of this paper are[2]:

1) An evaluation and comparison of stream-based and pool-based AL on the basis of four different datasets.
2) A parameter study on different uncertainty measures in combination with the above stream-based and pool-based AL with hundreds of thousands retrain loops.
3) The identification of scenarios that benefit from the application of self-training in combination with AL.
4) An analysis of the influence of competence levels of admins/oracles on the overall performance of the model.

The remainder of this work is structured as follows. In

[1]https://eur-lex.europa.eu/eli/reg/2024/1689

[2]https://github.com/lsinfo3/cnsm2024-active-learning-evaluation

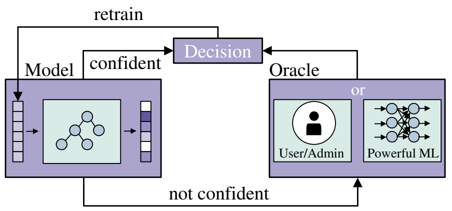


Fig. 1: Overview of the AL training loop.

Section II, we give background information about AL, as well as related work. In Section III, we specify our parameter study and then we evaluate general influence factors on AL in Section IV. Lastly, we summarize our findings and provide an outlook for future work in Section V. This work is an extension of our recently published extended abstract [7], in which we depicted an outline of the study for this work.

## II. Background and Related Work

**AL and Self-Training.** Figure 1 shows a schematic AL loop and the involved components. A base classifier outputs label probabilities for newly seen instances. These probabilities allow us to calculate the confidence of each prediction and handle it accordingly. In case the model is confident in its prediction, pseudolabels can be created by including the instances to augment datasets for (re)training, at risk of instances that were (confidently) wrongly labeled. In case the model is not confident in its prediction, the instance is relayed to the oracle, either represented by a human annotator or a more powerful ML model. After potentially relabeling the instances, they are fed back into the training data.

AL can be divided into three different approaches, namely pool-based AL, stream-based selective sampling, and membership query synthesis [4], [8]. The pool-based approach ranks all instances regarding predefined criteria and relays the top ranked instances to the oracle, while the stream-based approach makes an independent judgement for each instance [4], i. e., pool-based AL operates in an offline manner, while stream-based AL operates more in an online fashion [9]. A membership query generates new instances on its own. The former two approaches make the most sense in the context of our use case, since we are not interested in generating synthetic instances. In the end, our goal is to improve the model's performance, while keeping the workload low.

**AL in Networking.** Traditional AL methods typically involve retraining the model after each query [8], which can be inefficient in practice [10]. Training and deploying a new model constantly might not be feasible. Instead, we employ AL in batches, where the model is retrained after multiple new instances have been classified. In addition, the pool of all unlabeled instances is often assumed to be fixed/known beforehand [8], which is also impractical for network monitoring, as new data is constantly flowing through the network, potentially at gigabyte or even terabyte scales. Storing all this data indefinitely is not feasible. Thus, we assume dynamic pools, potentially discarding data. So, in the context of this work, we deviate from traditional AL, by assuming that for the pool-based approach we relay the top $n$ most uncertain instances in a batch before retraining, i. e., static query sizes. In practice, this could be implemented in constant memory. For the stream-based approach, we relay all instances below a certain threshold in a batch before retraining, i. e., dynamic query sizes. Summarizing, we relax the definitions for both approaches, since the pool-based approach does not know all unlabeled instances beforehand, and the delayed retraining reduces the online nature of the stream-based approach slightly.

**Querying Strategies.** Regardless of the chosen AL approach, many querying strategies exist, i. e., how to choose which elements to relay. In [4], the main querying strategies have been identified as diversity-based approaches, approaches based on the expected model change, and uncertainty-based approaches. The latter is a natural fit due to the self-training aspect of our use case, since it favors instances where the model was most uncertain in its decision. The other two focus more on picking the most informative instances, e. g., picking mostly diverse instances, which do not necessarily need any admin supervision, and thus, are out of scope for this work.

**ML Confidence/Uncertainty.** ML uncertainty refers to predictive uncertainty, linked to not only assigning a label to an instance, but a likelihood for each available class. Thus, uncertainty is directly related to the model's confidence in its prediction. Luckily, many traditional ML but also Deep Learning (DL) algorithms are capable of doing so, including Random Forests (RFs) via majority voting or Neural Networks (NNs) via softmax layers. Given these probabilities, we may now formulate measures of confidence/uncertainty, e. g., by simply looking at the highest class probability or calculating the entropy of the class probabilities [8]. Given these measures, we can then establish rankings or select appropriate thresholds to relay interesting instances to an expert. Determining how to configure these parameters and selecting the appropriate strategy is a fundamental challenge.

**Related Work.** Table I summarizes existing approaches with respect to the utilized AL and sampling strategies, analyzed parameters, datasets and their corresponding use cases. The majority of papers only focus on either pool- or stream-based strategies, without explicitly comparing the two approaches. In conjunction with the AL strategy, uncertainty sampling is the most common sampling strategy. Here, many works focus on making sophisticated additions to uncertainty-based AL, e. g., via Out-of-Distribution (OoD) [20] or outlier detection (OD) [22]. Other sampling strategies include Query-by-Committee (QBC) sampling [11], [22], which makes use of an ensemble of classifiers and their label probabilities, density/similarity-based sampling [6], [9], which focuses on selecting representative instances, or ranked batch mode, which combines uncertainty with similarity [9], [11], [23]. Our work, in comparison, focuses on both stream- and pool-based AL and performs a systematic comparison. We test ten measures for uncertainty-based sampling to quantify the impact the selected measure has on the model evolution.

Regarding different benchmarking parameters for parameter studies, many works focus on the initial training size only or simply have no variable parameters, e. g., thresholds are

TABLE I: Comparison of our work to related work, table adapted/extended from Guerra-Manzanares and Bahsi [11].

| Ref. | AL Strats. | Sampl. Strats. | Benchmark. Params. | Datasets | Use Cases | Self-Train. | Human Err. |
|---|---|---|---|---|---|---|---|
| [12] | Pool | Uncertainty+Rare Category | n/a | Contagio[a]<br>NSL-KDD [13]<br>NetFlow Data[b] | IDS | ✗ | ✗ |
| [14] | Stream | Uncertainty+Clustering | n/a | CICIDS2017 [15]<br>CTU-13 [16] | IDS | ✓ | ✗ |
| [6] | Stream<br>Pool | Uncertainty<br>Density<br>QBC<br>Exp. Error Reduction | Initial Training Size | TRAbID [17]<br>ISCXVPN2016 [18]<br>ISCXTOR2016 [19] | IDS<br>App Det. | ✗ | ✗ |
| [20] | Pool | Uncertainty+OoD | Num. Unknown Classes | NF-ToN-IoT-v2 [21] | IDS | ✗ | ✗ |
| [22] | Pool | Uncertainty+OD<br>QBC (3x)<br>Ranked Batch Mode | Pool/Batch Size<br>Num. Outliers | NIDS Alert Data[b] | IDS | ✗ | ✗ |
| [11] | Pool | Uncertainty (3x)<br>QBC (3x)<br>Ranked Batch Mode | Pool/Batch Size<br>Initial Training Size<br>Noise Level | MedBIoT [23] | IDS | ✗ | ✓ |
| [9] | Stream<br>Pool | Extended Strats. of [6]<br>Uncertainty+RL<br>Ranked Batch Mode | n/a | CTU-13 [16]<br>DeCrypto [24]<br>IP Flows[b] | Proto. Det.<br>IDS<br>Crypto. Det. | ✗ | ✗ |
| [25] | Stream | Uncertainty+RL | Initial Training Size | MAWILAB [26] | IDS | ✗ | ✗ |
| [27] | Pool | Uncertainty (2x) | Initial Training Size | NSL-KDD [13]<br>SWaT [28] | IDS | ✓ | ✗ |
| [29] | Stream | Uncertainty | Uncertainty Threshold<br>Initial Training Size | CICIDS2017 [15]<br>CTU-13 [16] | IDS | ✓ | ✗ |
| [30] | Stream | Uncertainty+Drift Det. | Initial Training Size | NSL-KDD [13] | IDS | ✓ | ✗ |
| [31] | Pool | Uncertainty | Initial Training Size<br>Noise Level | CTU-13 [16]<br>CTU-19 [16] | IDS | ✗ | ✓ |
| **Ours** | Stream<br>Pool | Uncertainty (10x) | Uncertainty Threshold<br>Query Size<br>Admin Competence Level | ISCXTOR2016 [19]<br>ISCXVPN2016 [18]<br>CICIOT2022 [32]<br>CICIDS2017 [15] | IDS<br>App. Det.<br>Dev. Det. | ✓ | ✓ |

[a]http://contagiodump.blogspot.fr/, [b]Real-world data collected by authors; not available. [c]https://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html

TABLE II: Overview of utilized datasets.

| Name | Abbr. | Use Case | Size | Features | Classes |
|---|---|---|---|---|---|
| ISCXTOR2016 | TOR | App. Det. | 3360 | 23 | 8 |
| ISCXVPN2016 | VPN | App. Det. | 18 758 | 23 | 7 |
| CICIOT2022 | IOT | Dev. Det. | 229 757 | 44 | 3 |
| CICIDS2017 | IDS | Intr. Det. | 692 703 | 77 | 6 |

often fixed. Additionally, the effect of self-training is seldom explored. The majority of works also assumes that the admin always makes the correct decision. While this may be true for use cases in the visual domain, classifying network traffic with respect to detecting anomalies or network intrusions is a much more complex task. Thus, besides an expert oracle, we also look at noisy or even malicious oracles, as proposed by Miller et al. [33]. In summary, our work depicts an overarching parameter study regarding the many factors that influence the performance of AL, including human errors.

## III. Methodology

**Use Cases/Datasets.** As we have seen in Table I, IDS is the most common use case when it comes to network monitoring. This work also looks at application and device detection, giving a more well-rounded view over the topic of AL. Table II depicts a summary of the datasets we utilize.

For the first use case of application detection, we utilize two datasets: ISCXTOR2016 [19] and ISCXVPN2016 [18]. Both datasets provide two scenarios ($A$ and $B$). Scenario $A$ concerns the distinction between VPN/Tor and non-VPN/Tor traffic, while scenario $B$ is about the classification into different applications in the network, such as VoIP or chatting. Both datasets contain the same amount of features and labels, except the *streaming* class, which is subdivided into audio and video streaming for the TOR dataset. We utilize scenario $B$, as the multi-label classification is a more complex problem, especially since it contains applications with and without the use of a VPN/Tor. The contained features are extracted in $15\,\mathrm{s}$ timeslots, and contain statistics over inter-arrival times (IATs), flow duration, bytes, and more.

For the second use case of device detection, we utilize the CICIOT2022 [32] dataset that comes with six types of scenarios, including interactions of devices or even malicious devices. For this work, we utilize the already preprocessed dataset provided by the authors, which contains only traffic from non-anomalous devices and includes devices from classes like home automation, camera, and audio. It contains features about utilized protocols, information in relation to preceeding packets (e. g., size of the last 20 packets or IAT), and more.

For the last use case of intrusion detection, we utilize the CICIDS2017 [15] dataset. It contains five days, which all represent different attack scenarios. We utilize the Wednesday-subset, which contains five types of Denial-of-Service (DoS) and distributed DoS (DDoS) attacks. The dataset contains flow-based features, including the flow duration, and statistics about packet sizes and IATs. Here, we excluded features such as IP addresses/ports to avoid overfitting on artifacts, which may not be representative of real-world attack scenarios.

**ML/AL Workflow.** Firstly, like in traditional ML, the data is split into a training and a test set. For this, we create three different training and test splits while ensuring that each instance is used for testing and training at least once. The training set is split again, into the initial training data (i. e., the data which the pre-AL model is trained on as a baseline), as well as several batches. These batches are used to simulate newly arriving data instances. The data is preprocessed by min-max scaling and extracting the top features. The scaler and top features are re-evaluated after every batch iteration.

To clarify the full AL loop in-depth, Figure 2 depicts the overall workflow. At the start, the initial training data is used to train the initial ML model, which serves as the baseline (step ⓪). With this initial model, the instances in the first batch are classified (steps ① + ②). The corresponding predictions are then evaluated regarding the confidence of the ML model. If a decision was not made with confidence, the instance is

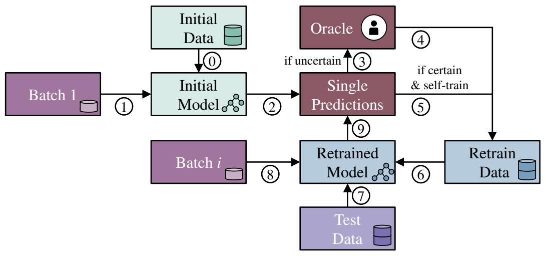


Fig. 2: Overall workflow of the AL loop simulation.

forwarded to the oracle (step ③), and after examination by the oracle it is included into the retraining data (step ④). If the decision was made with confidence, however, it is included directly into the retraining data if self-training is enabled (step ⑤). This means, that if self-training is disabled, all instances that are over the confidence threshold are dismissed for the stream-based approach. Similarly, for the pool-based approach, all instances that are not contained in the ranking of the most uncertain instances are also dismissed. After processing the whole batch, the retraining data is then used to train a new model (step ⑥). After every iteration, the model is tested via the test data, to evaluate its prediction performance (step ⑦). The test data is not included in the (re)training in any way. It is an independent part of the data, only utilized for this performance evaluation. After this performance evaluation, the next batch is used as input to the retrained model and all the instances get classified (steps ⑧ + ⑨). Then, this loop (steps ③-⑨) begins anew, until all batches have been included.

**ML/AL Parameters.** Table III depicts the simulation settings. To obtain initial training sets that contain a similar amount of instances, the initial training data is set to 10 %, 1 %, 0.1 %, 0.01% of the full training data for the smallest to largest dataset, respectively. TOR and VPN scenarios utilize twice as many features as IOT and IDS scenarios, since the former are more complex datasets, while the latter already show high accuracies for less initial training data and fewer features. We utilize a Random Forest (RF) with 100 Decision Trees (DTs) and tree depth of 20 to avoid overfitting. It enables the calculation of probabilities by simply taking the percentage of DTs, that voted for a specific class label. RFs perform reasonably well with default values [34], even without any hyperparameter tuning, enabling our large AL parameter study. The rest of the original training data, that is not used for the initial ML model, is divided into 25 batches in all scenarios.

Our virtual admins are also parameterized. Since we are merely interested in the fraction of (in)correct decisions, each admin is represented by a simple probability that they know the label. Otherwise, a random guess is taken. We have seven admin types, evenly spaced out between 0 and 1: *best* (expert admin, 1), *better* $\left(\frac{5}{6}\right)$, *good* $\left(\frac{2}{3}\right)$, *mediocre* $\left(\frac{1}{2}\right)$, *bad* $\left(\frac{1}{3}\right)$, *worse* $\left(\frac{1}{6}\right)$, and *worst* (0, random guess). We also model an eighth adversary admin, who permutes the labels in a pattern.

**Uncertainty Measures.** Various uncertainty measures may be utilized to either define a threshold, when an instance is deemed not confidently classified (stream-based) or to rank all instances (pool-based). The first and simplest uncertainty-based approach we utilize is querying instances whose preferred class label is least confident [8]. Henceforth, we refer to this strategy as *MAX*, since the uncertainty for an instance $x$ is calculated by simply taking its maximum label probability, so $\max_{i\in\mathcal{Y}} P(\hat{Y}=i|x)$, where $\mathcal{Y}$ is the set of possible labels. Even though the maximum label probability will never exceed 1, we can still normalize it, as the minimum probability will always be at least $\frac{1}{|\mathcal{Y}|}$. Since simply taking the maximum label probability may not be sufficient, another possibility is the margin-based approach [11]. Henceforth, we refer to this as *DIFF*, since the uncertainty for $x$ is the difference of the highest two label probabilities, that the classifier assigned to that instance. This already lies between 0 and 1.

An uncertainty measure that implicitly includes all label probabilities is the Shannon-Entropy (SE). The SE is a measure of "chaos", thus it is minimized for events that are 100% certain. In contrast, a vector of uniformly distributed probabilities maximizes the SE. Henceforth, we refer to it as $ENT$, defined via $-\sum_{i\in\mathcal{Y}} P(\hat{Y}=i|x)\log P(\hat{Y}=i|x)$. We can normalize this measure by dividing by the maximum SE.

Another possibility to include all label probabilities in the uncertainty measure is by taking the distance of the vector to a uniform distribution, since a uniformly distributed label probability equates to highest uncertainty. We calculate $dist(\hat{\mathcal{Y}}, \mathcal{U}_{|\mathcal{Y}|})$, where $\hat{\mathcal{Y}}$ is the probability vector of a single instance put out by the ML model and $\mathcal{U}_{|\mathcal{Y}|}$ is the vector of uniform probabilities. In general, we henceforth refer to this measure as $DIST$. The question how the function $dist(x)$ is defined arises now. There are many generic distance metrics, so we opt to include seven, namely Euclidean (*EUC*), Manhattan (*MAN*), Chebyshev (*CHE*), and Wasserstein (*WS*) distance, as well as the Kolmogorov-Smirnov statistic (*KS*), Kullback-Leibler (*KL*) and Jensen-Shannon (*JS*) divergence. For brevity's sake, we refrain from providing detailed descriptions of the above metrics. Instead, we will explain them on-the-fly if they are of relevance in the remainder of this work. We can normalize all metrics by dividing through the maximum distance to a uniformly distributed probability vector, which occurs when the probability vector has a label probability of 1 for one of the labels.

For stream-based AL and since we normalized all measures to take values between 0 and 1, we search this range in 0.1 increments, resulting in nine threshold values. For the pool-based approach this normalization is not necessary, since only the ranking is important. Here, we search in relation to the batch size, i. e., 10% are forwarded, 20% and so on. So, we also have nine different parameters here.

TABLE III: Parameter settings for all four scenarios.

| Dataset | Initial Training Size | # Features | ML Model | # Batches |
|---|---|---|---|---|
| TOR | 10 % | 20 | RF$_{20}$ | 25 |
| VPN | 1 % | | | |
| IOT | 0.1 % | 10 | | |
| IDS | 0.01 % | | | |

## IV. Evaluation

**Pool- vs. Stream-based AL.** For our evaluation, we follow a top-down approach and start with an overall comparison of the

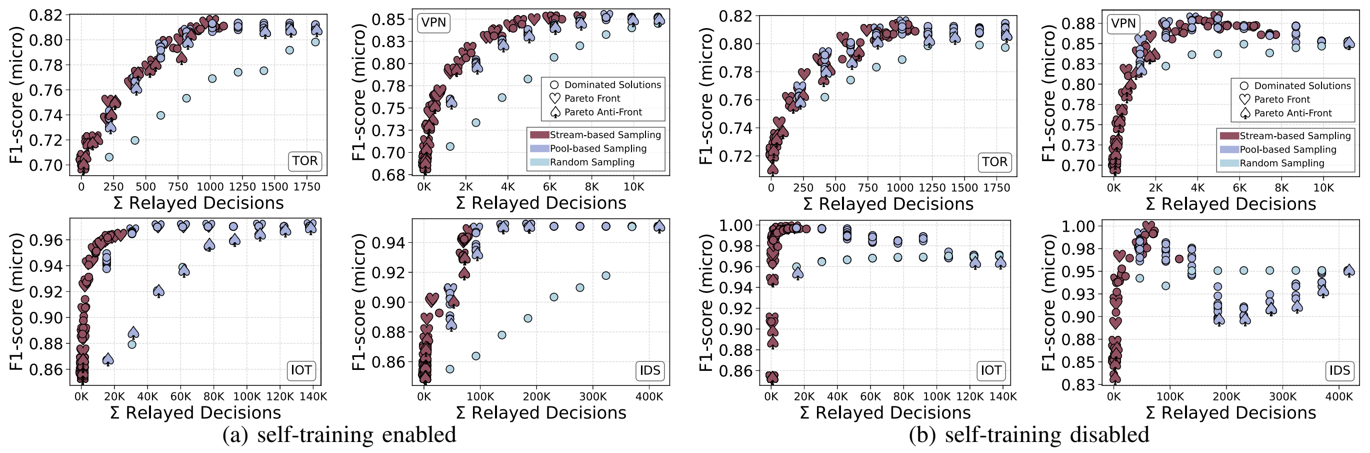


Fig. 3: Scatter-plots illustrating the trade-off between accuracy and admin workload after termination of all 25 batches.

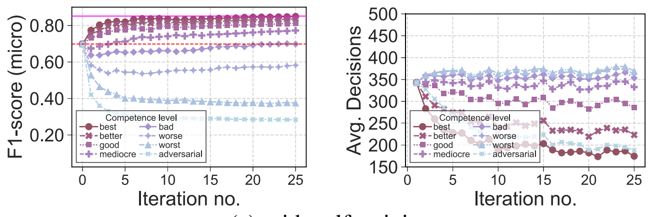
performance of pool- and stream-based AL. Figure 3 depicts all possible parameter combinations of thresholds/query sizes and uncertainty measures for the expert admin. The x-axis depicts the sum of relayed decisions after termination of the 25 batches, while the y-axis depicts the final F1-score (micro)[3]. The various plots also contain Pareto fronts and Pareto anti-fronts. The Pareto front depicts all strategies, that provide the best trade-off between maximizing the F1-score while minimizing the admin workload. Analogously, the anti-front depicts all strategies, that provide the worst trade-off. The figures also include random sampling as a baseline, which is not considered in the front calculation, since we are mainly interested in the comparison of stream- and pool-based AL.

Generally, the TOR and VPN use cases have a lower accuracy than the IOT and IDS datasets. For all scenarios, the strategies lay on a curve that rises fast at first and then flattens out or even decreases again in some cases, and there is no clear "winner" at first sight. Both stream- and pool-based AL outperform random sampling. Especially when self-training is enabled, stream-based AL has a slight edge over pool-based AL. If self-training is disabled, both strategies are more intertwined and the overall accuracy is generally higher. Naturally, pool-based AL is evenly spaced out w.r.t. the x-axis, whereas for stream-based AL, it is dependent on the dataset.

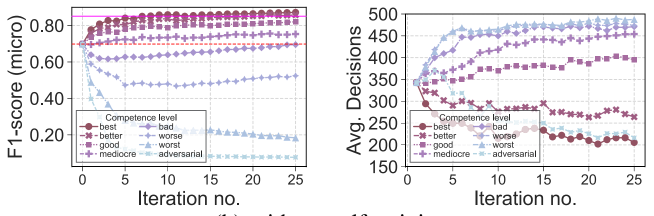
*Discussion:* In general, TOR and VPN perform worse, since they are smaller datasets, and the differences between the labels (i. e., the applications) might be less obvious than for device classes or intrusions vs. normal traffic. Generally, if self-training is enabled, stream-based AL performs better since it is adaptive, i. e, if there are only a few "interesting" instances in a batch, pool-based AL still chooses a fixed amount of instances, potentially including less interesting ones. Analogously, if there are many interesting instances in a batch, pool-based AL will miss out on forwarding them to the admin, while also at risk at being pseudolabeled wrongly.

The results without self-training are generally better. The model autolabels only instances it has seen before, so the information gain is minimal and the pseudolabels always come

[3]We also looked at the macro-avg. F1-score, but trends generally remained similar, while the score itself was slightly lower due to imbalances.



(a) with self-training



(b) without self-training

Fig. 4: Influence of different admin types.

with the possibility of errors. This is also the reason, why for non-self-training scenarios the accuracy sometimes drops off again with more instances relayed. If the query size is set too high or the threshold is too strict, the admin will also see more uninteresting instances, which dilute the actual interesting information when including them in the retraining.

In summary, even though the stream-based AL seems to have a slight edge over the pool-based approach, since it implicitly adapts the amount of relayed elements, i. e, if the model gets more confident over time, fewer elements are relayed, stream-based AL only makes sense if there is an attentive admin that verifies the relayed instances in a timely manner. Otherwise, the queries will queue up, equating to an unranked pool-based approach, taking away the online nature of this approach. Depending on how much traffic is flowing in the network, this might not be feasible.

**Self-Training.** To investigate self-taining further, Figure 4 illustrates exemplary how the accuracy and workload (i. e., the number of decisions forwarded to the admin) change over the course of the AL simulation for the VPN dataset, for stream-based $DIFF$ with a threshold of 0.8 (henceforth: $DIFF^S_{0.8}$). The x-axis contains the batch iterations, starting at 0 for the initial model. The y-axis shows the micro-averaged F1-score and workload, respectively. The red dotted line illustrates the

F1-score for the initial model, while the solid magenta line depicts the model performance when using the full training data as ground truth. The rest of the lines depict admin profiles.

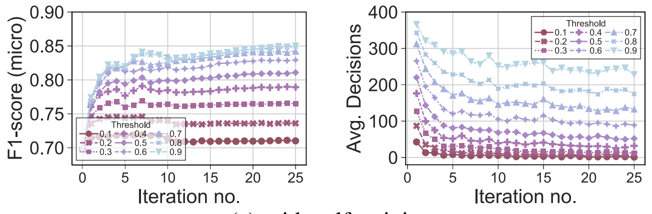
Figure 4a illustrates the changing F1-score and workload when self-training is enabled. Naturally, the more competent admins increase the accuracy over time, while the worse ones show mixed results. Interestingly, even the *bad* and *worse* admins show a slight upwards trend after initially dropping below the baseline. The *bad* admin is even able to surpass the initial model after some time. Regarding the workload, the better the admins are, the more they decrease their workload over time. One exception is the adversarial admin, which also decreases the workload over time. Analogously, Figure 4b illustrates the changing F1-score and workload when self-training is disabled. The trends for the accuracy remain similar. Though, the better admins here, as mentioned in the previous section, benefit from disabling the self-training. Here, the best admin even surpasses the ground truth model, even though the model was trained with less data. In contrast, the performance for worse admins, including the adversary, is now drastically decreased. Regarding the workload for the admins, more decisions are now relayed to the admins. Only the two best admins and the adversary are able to decrease their workload, while the others now show a prominent upwards trend.

*Discussion:* Generally, the self-training aspect is not necessarily needed the better the admin is. The confidently pseudolabeled instances dilute the expertise of the admin due to low information gain and the risk of being wrongly autolabeled. Without self-training the best admin was even able to surpass the model trained on the full ground truth, since only the interesting instances were added to the retraining, acting like a sort of implicit weighting of the instances. For less competent admins, the self-training can be of guidance, as seen with the *bad* admin that reaches the initial model quicker. For all admins, self-training keeps the workload lower. In summary, self-training makes the model more confident in its own decisions and should be used when it is assumed an admin is not able to consistently chose correct labels.

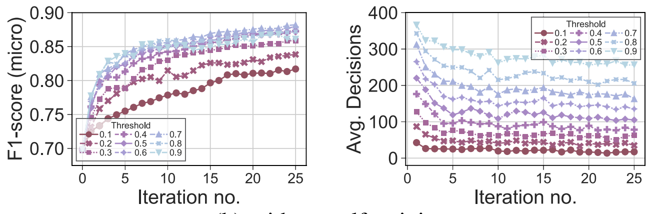
**Uncertainty Measures.** After investigating the various strategies in a more coarse-grained fashion earlier with Figure 3, Table IV actually counts the number of times each uncertainty measure has been part of the Pareto (anti-)front without self-training for the best admin. This allows us to analyze whether there is a measure that is more likely to outperform the others. Analogously, with the anti-front, we may also analyze if there is a measure that generally underperforms.

Most measures are contained more than once in both fronts, with exception of $KS$ and $DIFF$, where the former is only once Pareto optimal, and the latter only once a suboptimal option. Additionally, most of the elements in the Pareto front are stream-based approaches. Looking at the same analysis with self-training (not shown), these trends are more diluted.

*Discussion:* Pool-based approaches are less represented here since the strategies naturally all lie on a vertical line in the figure as they all rely on the same fixed amount of decisions to the admin. Hence, one measure will most likely dominate the



(a) with self-training



(b) without self-training

Fig. 5: Influence of different thresholds.

others, if not already dominated by a stream-based measure, which are more convoluted regarding the number of relayed decisions. The fronts with self-training enabled are generally bigger, since the self-training has a “smoothening” effect, as seen in the Pareto plots earlier. This even causes elements sometimes being contained in front and anti-front, because elements are more or less on a continuous line now, i. e., while there is no better option for a strategy (e.g., not dominated), there is also no worse option (e.g., nothing to dominate). Though, $KS$ stands out by being contained rarely contained in both fronts. $KS$ often has 0 relayed elements and thus does not perform well. Looking at the definition of the actual metric, it is the supremum of differences between the two cumulative distribution functions (CDFs), in this case of two probability vectors. Especially for smaller amounts of labels, this measure is less meaningful. In addition, the table depicts a trend towards simpler metrics. In summary, while most measures have appropriate configurations, it is preferred to stick to simpler ones, which are more intuitive/easier to explain.

**Thresholds/Query Sizes.** We now turn our attention to uncertainty thresholds in Figure 5. Each subfigure depicts the number of iterations on the x-axis, and either the F1-score or workload on the y-axis for stream-based $DIFF$ and the expert admin. The colored lines depict different thresholds. Figure 5a illustrates the changing F1-score and workload when self-training is enabled. The stricter the threshold, the more elements are relayed and the higher the F1-score. Figure 5b shows the results when self-training is disabled. Here, the F1-score is higher, while also increasing the workload.

*Discussion:* As mentioned, self-training dampens the influence of a good admin and there are diminishing returns with stricter thresholds. Looking at thresholds of 0.9 to 0.7, there is visibly more workload, but no major influence on the F1-score. With a stricter threshold, more uninteresting instances are relayed, which have the described dampening effect. For pool-based methods, the effect of bigger query sizes is similar to stricter thresholds. However, here the diminishing returns may actually become counterproductive, as the pool-based approach relays significantly more elements, which we have

TABLE IV: Pareto front (cyan) and anti-front counter[a] (red).

| | Stream-based | | | | Pool-based | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | TOR | VPN | IOT | IDS | TOR | VPN | IOT | IDS | ∑ |
| DIFF | 5 - | 5 - | 1 - | 2 - | 1 - | 2 - | - - | - 1 | 16 1 |
| MAX | 4 3 | 2 - | 5 1 | 2 - | - - | - - | - - | - 2 | 13 6 |
| EUC | 2 2 | 3 2 | 5 1 | 2 1 | 1 - | - - | - - | - - | 13 6 |
| CHE | 4 2 | 1 1 | 1 - | 2 - | - - | - - | - - | - 2 | 8 5 |
| ENT | 3 3 | 2 3 | 1 - | 2 - | 1 - | - 1 | - - | - 1 | 9 8 |
| KL | 3 3 | 2 3 | 1 - | 2 - | 1 1 | - - | - - | - 2 | 9 9 |
| WS | 1 3 | 2 5 | 3 - | 2 1 | - 2 | - - | - - | 1 1 | 9 12 |
| JS | 2 2 | 4 5 | 1 1 | - 1 | 1 2 | - 1 | - - | - 1 | 8 13 |
| KS | - 2 | - - | - - | - 2 | - 2 | 1 - | - 1 | - - | 1 7 |
| MAN | 1 3 | 2 5 | 3 - | 1 1 | - 1 | - - | - 2 | - 2 | 7 14 |

[a] only elements with rel. decisions of $> 0$ counted here.

also seen in the Pareto plots earlier. Lastly, while the expert admin generally has a bigger positive effect the more elements are forwarded, this effect is decreased the noisier the labeling is. For the worst admins, it actually has a negative impact with stricter thresholds. In summary, more manually labeled samples do not always equate to higher accuracies in addition to increased workloads. Thus, we suggest starting off with looser thresholds and tighten them when needed.

## V. Conclusion

We performed a rigorous parameter study for integrating ML uncertainty into network monitoring tasks via AL. Our study covered four datasets involving application, device, and intrusion detection. We compared stream- and pool-based AL, evaluated ten uncertainty measures, and examined the impact of admin competence levels and self-training.

Our key takeaways are that stream-based AL has a slight edge over pool-based AL, especially when self-training is enabled, although stream-based AL might not be feasible in practice. Self-training slightly dilutes expert knowledge but helps manage workload and guides less experienced admins. Even admins with imperfect knowledge can improve the model, making AL valuable for practical adoption. While most uncertainty measures provide appropriate configurations, choosing the simpler ones yielded better results here and is advisable anyway, especially w.r.t. explainability, which ultimately has to be given when including a human in the AI/ML loop. Thresholds should not be too strict or query sizes should not be too large, to not only avoid overloading the admin, but to ensure only interesting instances are selected.

Future work will focus on simulating more realistic admins, considering factors such as time budgets, as well as adaptive or dual thresholds. Exploring other ML models, especially Deep Learning (DL) models, which are often overly confident, can introduce another parameter. Conducting user studies with real-world admins can help evaluating the system in a practical context, e. g., via Mockup-UIs and explainable AI (XAI).

## Acknowledgment

This work has been funded by the Bavarian Ministry of Economics, Regional Development and Energy (StMWI) as part of the project VIPNANO (DIK-2307-0006) and by Deutsche Forschungsgemeinschaft (DFG) under grant SE 3163/3-1, project nr.: 500105691 (UserNet). Pedro Casas was partially funded by the Austrian Science Fund, FWF project GRAPHS4SEC – Graph Neural Networks for Robust AI/ML-driven Network Security Applications – grant number I-6653. The authors alone are responsible for the content.

## References

[1] K. Dietz *et al.*, “The missing link in network intrusion detection: Taking AI/ML research efforts to users,” *IEEE Access*, 2024.

[2] M. Vielberth, F. Böhm, I. Fichtinger, and G. Pernul, “Security operations center: A systematic study and open challenges,” *IEEE Access*, 2020.

[3] D. Han *et al.*, “DeepAID: Interpreting and improving deep learning-based anomaly detection in security applications,” in *ACM CCS*, 2021.

[4] P. Ren *et al.*, “A survey of deep active learning,” *ACM CSUR*, 2021.

[5] M. S. Hajmohammadi, R. Ibrahim, A. Selamat, and H. Fujita, “Combination of active learning and self-training for cross-lingual sentiment classification with density analysis of unlabelled samples,” *Inf. Sci.*, 2015.

[6] A. Shahraki, M. Abbasi, A. Taherkordi, and A. Jurcut, “Active learning for network traffic classification: A technical study,” *IEEE TCCN*, 2021.

[7] K. Dietz *et al.*, “Demystifying user-based active learning for network monitoring tasks,” in *MaLeNe*, 2023.

[8] B. Settles, “Active learning literature survey,” Tech. Rep., 2009.

[9] J. Pešek, D. Soukup, and T. Čejka, “Active learning framework for long-term network traffic classification,” in *IEEE CCWC*, 2023.

[10] T. N. Cardoso *et al.*, “Ranked batch-mode active learning,” *Inf. Sci.*, 2017.

[11] A. Guerra-Manzanares and H. Bahsi, “On the application of active learning for efficient and effective IoT botnet detection,” *FGCS*, 2023.

[12] A. Beaugnon, P. Chifflier, and F. Bach, “ILAB: An interactive labelling strategy for intrusion detection,” in *RAID*, 2017.

[13] M. Tavallaee, E. Bagheri, W. Lu, and A. Ghorbani, “A detailed analysis of the KDD CUP 99 data set,” in *IEEE CISDA*, 2009.

[14] M. Hajizadeh, S. Barua, and P. Golchin, “FSA-IDS: A flow-based self-active intrusion detection system,” in *IEEE/IFIP NOMS*, 2023.

[15] I. Sharafaldin, A. H. Lashkari, and A. Ghorbani, “Toward generating a new intrusion detection dataset and intrusion traffic characterization,” *ICISSP*, 2018.

[16] S. Garcia, M. Grill, J. Stiborek, and A. Zunino, “An empirical comparison of botnet detection methods,” *Comput. Secur.*, 2014.

[17] E. K. Viegas, A. O. Santin, and L. S. Oliveira, “Toward a reliable anomaly-based intrusion detection in real-world environments,” *Comput. Netw.*, 2017.

[18] G. D. Gil, A. H. Lashkari, M. Mamun, and A. Ghorbani, “Characterization of encrypted and VPN traffic using time-related features,” in *ICISSP*, 2016.

[19] A. H. Lashkari, G. D. Gil, M. Mamun, and A. Ghorbani, “Characterization of Tor traffic using time based features,” in *ICISSP*, 2017.

[20] J. Talpini, F. Sartori, and M. Savi, “Enhancing trustworthiness in ML-based network intrusion detection with uncertainty quantification,” *arXiv:2310.10655*, 2023.

[21] M. Sarhan, S. Layeghy, and M. Portmann, “Towards a standard feature set for network intrusion detection system datasets,” *MONET*, 2022.

[22] R. Vaarandi and A. Guerra-Manzanares, “Network IDS alert classification with active learning techniques,” *JISA*, 2024.

[23] A. Guerra-Manzanares, J. Medina-Galindo, H. Bahsi, and S. Nõmm, “MedBIoT: Generation of an IoT botnet dataset in a medium-sized IoT network,” in *ICISSP*, 2020.

[24] R. Plný, K. Hynek, and T. Čejka, “DeCrypto: Finding cryptocurrency miners on ISP networks,” in *NordSec*, 2022.

[25] S. Wassermann, T. Cuvelier, and P. Casas, “RAL – Improving stream-based active learning by reinforcement learning,” in *IAL*, 2019.

[26] R. Fontugne, P. Borgnat, P. Abry, and K. Fukuda, “MAWILab: Combining diverse anomaly detectors for automated anomaly labeling and performance benchmarking,” in *ACM CoNEXT*, 2010.

[27] Z. Niu *et al.*, “A novel anomaly detection approach based on ensemble semi-supervised active learning (ADESSA),” *Comput. Secur.*, 2023.

[28] J. Goh, S. Adepu, K. N. Junejo, and A. Mathur, “A dataset to support research in the design of secure water treatment systems,” in *CRITIS*, 2017.

[29] Y. Zhang *et al.*, “Network intrusion detection based on active semi-supervised learning,” in *IEEE/IFIP DSN-W*, 2021.

[30] Z. Niu *et al.*, “QARF: A novel malicious traffic detection approach via online active learning for evolving traffic streams,” *CJE*, 2024.

[31] J. L. G. Torres, C. A. Catania, and E. Veas, “Active learning approach to label network traffic datasets,” *JISA*, 2019.

[32] S. Dadkhah *et al.*, “Towards the development of a realistic multidimensional IoT profiling dataset,” in *IEEE PST*, 2022.

[33] B. Miller *et al.*, “Adversarial active learning,” in *ACM AISec*, 2014.

[34] P. Probst, M. N. Wright, and A.-L. Boulesteix, “Hyperparameters and tuning strategies for random forest,” *WIREs DMKD*, 2019.