Design, framework and benchmark of safety monitors for black-box classifiers

Scientific Reports

https://doi.org/10.1038/s41598-026-45091-2

Article in Press

Design, framework and benchmark of safety monitors for black-box classifiers

Received: 2 October 2025

Accepted: 17 March 2026

Published online: 02 April 2026

Cite this article as: Khokhar F.A., Zoppi T., Cennini L. *et al.* Design, framework and benchmark of safety monitors for black-box classifiers. *Sci Rep* (2026). https://doi.org/10.1038/s41598-026-45091-2

Fahad Ahmed Khokhar, Tommaso Zoppi, Luigi Cennini, Andrea Ceccarelli & Andrea Bondavalli

Design, Framework and Benchmark of Safety Monitors for Black-Box Classifiers

*Fahad Ahmed Khokhar[1] *, Tommaso Zoppi[1], Luigi Cennini[1], Andrea Ceccarelli[1], Andrea Bondavalli[1]*

[1]Department of Mathematics and Informatics, University of Florence, Viale Morgagni 67/A, 50134 Florence (Italy)

***Corresponding author:**
Fahad Ahmed Khokhar
Mathematics and Informatics,
University of Florence,
Viale Morgagni 67/A, 50134
Florence (Italy)
Email: fahadahmed.khokhar@unifi.it

ABSTRACT

The last decade has seen a surging opportunity for developing autonomous systems (e.g., fully autonomous driving, computer-guided robotic surgery, mobile robots for inspections and surveillance, and manufacturing robots) to relieve the human workforce from time-consuming, laborious, and often hazardous tasks. These cutting-edge innovations require trustworthy and autonomous classification processes to provide their intended functionality without harming the health of people, infrastructures, and the environment, or causing financial losses. Unfortunately, even human experts cannot always act as trustworthy classifiers, let alone computer-guided decision-makers based on Machine Learning (ML). We advocate that striving for correct software will just lead to a dead end: instead, we should focus on minimizing wrong outputs - misclassifications in case of classifiers - suspecting prediction errors and handling them via tailored software architectures. To such extent, this paper proposes a software architecture that allows for monitoring a black-box software component, triggering a rejection of its output whenever specific uncertainty conditions are detected trough an ensemble of uncertainty measures that are continuously computed. Such Safety wraPper thROugh ensembles of UncertainTy measures (SPROUT) is designed to adapt to any classification task, either binary or multiclass, processing tabular or image data. Experimental results using a wide range of datasets, classifiers, and parameter setups show how our approach can consistently reject a significant portion of misclassifications, even suspecting all incorrect predictions in specific cases. To maximize usage and ease of use, SPROUT is available as an open-source library with pre-trained models and files for effortless case study execution.

Index Terms — Safety Monitor, Trustworthy Software, Software Architecture, Classifier, SPROUT

INTRODUCTION

We are at the dawn of the fifth industrial revolution, where retail and business logic are meant to synergize with sustainability and inclusivity for building human-centred autonomous systems to relieve the human workforce from time-consuming, laborious, and often hazardous tasks for the social good. Examples include, but are not limited to: fully autonomous driving, computer-guided robotic surgery, mobile robots for inspections and surveillance, optimized power management and generation, and manufacturing robots. All these cutting-edge innovations require trustworthy decision-making processes to provide their intended functionality without harming the health of people, infrastructures, environment, or causing financial losses. Decisions are typically a matter of choosing where to go, what to do, what to buy, or if an action is safe to perform. This often defines an underlying classification problem, where an erroneous prediction (i.e., misclassification) may even have potentially devastating detrimental effects e.g., undetected security breach, wrong recognition of a “STOP” traffic sign of undetected pedestrian in a trajectory for autonomous driving [72].

When a decision requires the analysis of massive data or has to be taken within constrained and short time-frames, one of the possible solutions is to employ functional components that (partially) rely on Machine Learning (ML) algorithms that perform classification (*classifiers* in the paper). Classifiers can model one or more expected behaviors of a system or component and detect deviations that may be due to the occurrence of faults or attacks, and perform error detection, intrusion detection,

failure prediction, or out-of-distribution detection, [4], [24] to name a few. Straightforwardly, academia, industry, and also National governments hugely invested in methodologies, mechanisms, and tools to embed classifiers into Information and Communication Technology (ICT) systems, including safety-critical ones. However, classifiers may predict a wrong class for a given data point, which is typically called a *misclassification.* This is a well-known limitation to the adoption of classifiers to operate safety-critical functions, requiring countermeasures that avoid or mitigate the potential cascading effects of misclassifications to the encompassing system.

Within critical software and system engineering, a typical approach to guarantee safety [27] is to equip a functional component with a detector [21], [23], [9] , [25] so to trigger a fail-safe or fail-stop behavior whenever the correct functioning is not guaranteed. **The effect is not to correct wrong outputs, but to transform erratic content failures, which are unpredictable and not easy to handle, into omission failures.** At the system level, it is often desirable that critical functions would either i) deliver a correct result or ii) reject outputs i.e., the function should have fail-omission failures only. This makes it easy for the system to timely detect the absence of outputs and react accordingly. Through years, safety monitors or safety wrappers have been applied to different functions with beneficial effects on the non-functional (either safety or security [12]) behavior of the component and the encompassing systems, becoming proven-in-use solutions especially for industrial applications.

This paper discusses how safety monitors could be the enabling technology for trustworthy decision-making, classification tasks in particular. Craving for a correct classifier is a waste of time and resources: nowadays, critical decisions are mostly up to humans (e.g., medical personnel for surgeries, drivers or pilots of airplanes or trains) who are not perfect, but just… trustable, good enough. It does not make sense to strive for perfect components or systems: **trust does not imply correctness**. Our conjecture is that **classifiers should be allowed to reject predictions they are not confident with: this will translate in a reduced amount of misclassifications, which will be rejected – but not corrected – instead.** Ideally, a fail-omission classifier would either produce correct predictions or reject them [21]. Clearly, this approach is different from building a classifier that never outputs misclassifications, which is unrealistic to assess at the state of the art due to the dimension of the input space and the unpredictable behavior with inputs close to the decision boundaries [24].

After discussing and motivating this conjecture, this paper presents uncertainty measures that quantify the confidence in a prediction of a black-box classifier (i.e., internals of the classification process do not need to be disclosed), which can then be used to suspect misclassifications [2], [8], [10]. These measures aim at delivering significantly different uncertainty between i) the predictions that turn out to be correct, and ii) those that turn out to be misclassifications, allowing to trigger rejections when uncertainty is exceedingly high. All the observations above are then conveyed to design, implement, and evaluate a Safety wraPper through ensembles of UncertainTy measures (SPROUT). SPROUT is a safety monitor for a black-box classifier, which computes many uncertainty measures and produces a binary confidence score that can be used to decide whether the classifier prediction is trustable and can be safely propagated to the encompassing system, or if it should be rejected, triggering mitigation strategies instead. The paper provides the following key contributions:

- discusses how some domains give more relevance to trustworthiness rather than correctness, which is a de-facto performance metric in classifier evaluation.
- introduces a formal ensemble-based uncertainty aggregation and binary adjudication mechanism that enables to reject uncertain predictions, which is an enabling condition for trustworthiness of classifiers.
- proposes and publicly shares and evaluates SPROUT, a safety monitor for black-box classifiers that works with supervised and unsupervised classifiers without requiring access to internal model parameters or to disclose model insights. SPROUT is available at the GitHub library at [5].
- provides an experimental evaluation on a total of 35 datasets of tabular and image data, using more than 20 supervised and unsupervised classifiers, and multiple instances of uncertainty measures quantifying how SPROUT dramatically reduces misclassifications, to a point that all of them are rejected in specific cases.

The reader should note that this paper is a more complete, detailed and solid version of the conference paper [77], which described the main idea about using ensembles of uncertainty measures to suspect

misclassifications and provided a preliminary experimental analysis concerned with supervised classifiers only.

BASICS AND RELATED WORKS

A large portion of classifier research focuses on improving predictive performance, primarily by minimizing misclassification rates and maximizing accuracy, rather than explicitly addressing uncertainty management or rejection mechanisms. However, trusting each individual output of a software, or predictions of a classifier, to the extent that it can be propagated towards the encompassing system and used in a (safety-)critical task, is a different problem that is still open [10].

Basics on Classifiers

Decades of research and practice on ML provided us with plenty of classifiers that are meant to always output a prediction. Supervised classifiers and particularly those based on Deep Neural Networks (DNNs) were proven to achieve excellent classification performance in many domains. Additionally, the last couple of years provided evidence that some classifiers are more suitable to process structured rather than unstructured input data. This is especially the case of tabular data, for which it is beneficial to use tree-boosting ensembles [20], [57], despite alternatives based on DNNs exists [6], [7]. Conversely, image classification employs DNNs, which can learn strong features from pixel maps [62]. Unsupervised classification, which does not require labeled training data, mainly relies of clustering, density-based methods, statistical approaches, and neural network models such as self-organizing maps or autoencoders [54], [55].

Misclassifications are known to be triggered by specific events as fake input data (including adversarial attacks [65]), anomalies, out-of-distribution (OOD) data, and distribution shifts [8], which tend to deceive models into incorrect classifications. Data points lying close to decision boundaries are also problematic, leading to instability in the model's outputs.

Regardless of its specific algorithm, a classifier *clf* first devises a mathematical model from a training dataset [4], which contains a given amount of data points. Each input data point *dp* contains a set of *f* feature values, where each feature value is an image pixel / channel or a floating point number dp_j with $0 \leq j < f$ and describes a specific input of the classification problem. The dataset is usually partitioned into training and test sets. The training set is used to learn the model of the classifier; data in the test set is used to quantify the classification performance of the trained model. Training data requires labels only in case of supervised classifiers [7]. Once training is completed, the classifier makes its prediction on a data point *dp* as:

$$dp\_prob = clf.predict\_p(dp)$$

where *dp_prob* is an array of *c* floating point numbers that represent the likelihood that *dp* belongs to each of the *c* classes that characterize the classification problem. In most of the classifiers, *dp_prob* is a probability distribution, where the *c* items have positive values that sum up to 1. Specific classifiers or implementations may output likelihoods that do not directly match such constraint, requiring a normalization step. In all cases, the element of *dp_prob* with the maximum value corresponds to the class *dp_class* to be predicted for *dp*:

$$dp\_class = clf.predict(dp) = arg\ max(clf.predict\_p(dp))$$

where *arg_max* is the argument of the maxima, that returns the position of the maximum value in the *dp_prob* array [7]; this position corresponds to the class predicted for the data point. The classification performance is usually computed by applying *clf* to data points in a test dataset and computing metrics such as accuracy, i.e., the percentage of correct predictions of a classifier clf over all predictions. Noticeably, *1 - accuracy* quantifies the misclassification probability by difference.

Uncertainty Measures and their Limitations

Researchers and practitioners are actively investigating ways to maximize correct predictions, or at least suspect potential misclassifications through the quantification of uncertainty, or confidence. The most relevant research results on measures to compute uncertainty are very recent (the last 5 years), which demonstrates the recent emergence and timeliness of the topic. Uncertainty is often [26] referred to as a combination of aleatoric and epistemic uncertainty. The former refers to the notion of randomness, that is, the variability in the outcome of an experiment which is due to inherently random effects e.g., coin-flip. The latter describes uncertainty due to a lack of knowledge of any underlying random phenomenon. In other words, epistemic uncertainty refers to the reducible part of the (total) uncertainty, whereas aleatoric uncertainty refers to the irreducible part [26]. Uncertainty measures typically quantify overall predictive uncertainty, without explicitly distinguishing between epistemic

and aleatoric components. Uncertainty can be statistically estimated through compliance with confidence intervals [1] or using the Bayes theorem [2]. Works as [9] estimate uncertainty by using ensembles of neural networks: scores from the ensembles are combined in a unified measure that describes the agreement of predictions and quantifies uncertainty. In [8], [12], authors processed softmax probabilities of neural networks to identify misclassified data points. A new proposal came from [10] and [3], where authors paired a k-Nearest Neighbor classifier with a neural network to compute uncertainty. The work [24] computed the cross-entropy on the SoftMax probabilities of a neural network, and used it to detect out-of-distribution input data that likely misclassified.

Uncertainty measures above compute either classifier-specific or classifier-independent quantities. However, classifier-specific uncertainty may not always be a meaningful indicator of misclassifications since "*neural networks which yield a piecewise linear classifier function [...] produce almost always high confidence predictions far away from the training data*" [21].

Learning to Reject

Once uncertainty is computed, it can be used to decide if a prediction should be trusted, or if it should be rejected instead. As advocated in [51], "*human intelligence is usually modest and prudent, but, contrarily, machine intelligence is always omniscient and conceited, resulting in ridiculous and overconfident errors. To match this gap, learning to reject is actually an urgently needed skill for machines*". From the survey [51], there are three categories of predictions to reject:

- *Failure rejection*. This is the primary aim of prediction rejection strategies, which should suspect wrong predictions due to classifier errors. These are not necessarily triggered by unknown or malicious inputs, but mostly happen due to an imperfect training phase.
- *Unknown rejection*. Here, data is seen as either in-distribution or unknown, i.e., out-of-distribution. Unknown data fed into a classifier may result in unpredictable outcomes: despite not necessarily leading to misclassifications, sometimes it is better to be safe than sorry and discard predictions related to unknown data.
- *Fake rejection*, where unnatural, modified, or forged samples must be rejected to ensure correct system behavior. These inputs are typically crafted through adversarial attacks (e.g., FGSM [65], PGD [67]), generative models like GANs [74], or data poisoning techniques [75], all designed to produce realistic yet deceptive inputs that can mislead the model.

In the last decade, relevant research was conducted primarily within the community of ML, aiming at crafting rejection techniques that are suitable for rejecting failures, whereas others are tailored to reject overconfident predictions due to unknown (i.e., anomalous, out-of-distribution) or fake inputs.

Some rejection techniques were found to be beneficial for all three rejection tasks. Using the maximum a-posteriori probability, the gap between top-two class scores, or the entropy of the probability array, seemed to help for understanding when to reject failures. Other strategies exploit the concept of stability under perturbation, i.e., a correct decision should be invariant to small perturbations [52], which is easier to apply in image classification compared to dealing with tabular data. Specifically for unknown rejection, it has been proposed to compute a distance score as the Euclidean distance between the input and the k neighbours from the training set and reject if such a score is exceedingly high [53].

Nevertheless, rejecting overconfident predictions due to unknown inputs mostly revolves around strategies to compute confidence that are robust to unknown inputs. Whereas many uncertainty of confidence estimations were made available throughout the years, most of them were tested under the closed world (IID) assumption. This has a negative drawback on the representativeness of these metrics when dealing with unknowns, as there is no way to understand if these quantities lean towards overconfidence in specific situations. To the best of the author's knowledge, this is still an open problem that has suggestions but no best practices yet.

Related Works on Safety Monitors for Classifiers

Recently, there have been few studies that specifically aim at building safety monitors for classifiers [3], [23], [9], [25]. The paper [3] ran a k-nearest neighbor classifier in parallel to a Deep Neural Network (DNN) to detect misclassifications. The paper [23] conducted an active monitoring of the behavior and the operational context of the data-driven system based on distance measures of the Empirical Cumulative Distribution Function, and used them as triggers for the safety monitor. The work [9] used probabilistic neural networks to model predictive distributions and thus estimate misclassifications thanks to adversarial training. This technique performed well for image classifiers. Lastly, [25] proposed a lightweight monitoring architecture to enhance the model robustness against different unsafe inputs, especially those due to adversarial attacks to neural networks. The logic to detect misclassifications revolved around an analysis of activation patterns of neurons in the layers

of a specific neural network, which authors showed to be distinguishable in case of an adversarial input. It is worth mentioning that these safety monitors [3], [8], [25] assume a white-box classifier, often rely on extensive knowledge of the classifier (e.g., [25] requires the structure of the DNN to be disclosed), require the implementation of complex and multi-step processes [23], or apply only to specific types of input data (e.g., [10], [9]are specifically crafted for image classifiers). Instead, we seek for an approach which i) applies to any classifier, which is seen as a black-box; ii) is easy to automatize, adopt and exercise to novel datasets or systems; iii) applies to binary and multiclass classification problems, and iv) does not have any constraint on the input data and as such works with tabular and image datasets. Recent research has extensively explored methods to identify and reject inputs that are likely to cause misclassifications in machine learning models. For instance, a baseline approach for detecting out-of-distribution (OOD) and misclassified examples using softmax probabilities was proposed in [8], highlighting the importance of uncertainty estimation in model predictions. Similarly, [8] and [67] introduced techniques such as temperature scaling to improve OOD detection, demonstrating its effectiveness across various datasets. In the context of adversarial attacks, [65] pioneered the study of adversarial examples, revealing the vulnerability of neural networks to small, carefully crafted perturbations. Building on this, [67] formalized adversarial robustness through adversarial training, showing that models can be made more resilient to such attacks. Additionally, [65] investigated model calibration, emphasizing the challenges posed by inputs near decision boundaries and proposing methods to better quantify prediction confidence. Collectively, these works underscore the need for robust mechanisms to handle fake data, anomalies, distribution shifts, and other problematic inputs, ensuring the reliability and trustworthiness of machine learning systems in real-world applications.

Problem Formulation: From Correctness to Trustworthiness

The desideratum is a classifier with excellent classification performance, very confident when correct, and not confident about predictions that turn out to be misclassifications. However, classifiers typically "bet" on a prediction they are unsure of, instead of admitting they do not know. This best-effort behavior does not pair well with critical systems, which require correct component and system-level behavior guarantees. The probability of failure on demand of a critical component or system should be proven to be lower than specific thresholds.

Instead of increasing accuracy and correct predictions, we focus on changing the failure semantics of classifiers from uncontrolled content failures (i.e., misclassifications) to rejection failures. Ideally, we want to reject all and only the erroneous predictions: on the downside, the availability of said component may be negatively affected when correct predictions are rejected in the process. Ways to build fail-controlled software architectures [27] are well-known in the literature and often rely on safety monitors [12],[21], which are intended to complement an existing critical component or task by continuously checking invariants, or processing additional data to detect dangerous behaviors, and blocking the erroneous output of the component before it is propagated through the system. Finding trade-offs between safety and availability is of utmost importance when dealing with critical systems: this approach is not different.

But how can classifiers be dependable to an extent to which they can be trusted and deployed even in critical systems? Generally speaking, dependability is achieved when there is the possibility to "*avoid service failures that are more frequent or more severe than acceptable*" [27]. Trust is the accepted dependence of system A on a component B: the extent to which System A's dependability is (or would be) affected by that of B. In a nutshell, a critical system (e.g., autonomous car, infrastructures) relies on a trustworthy component or function because it can be confident that this reliance will not compromise its reliability or performance. This means that a (complex) classifier function should be designed, implemented, verified, and validated to be trustworthy in its operational environment and within the encompassing system. This has a massive impact on software architecture: trust does not imply correctness; thus a component may be trustable even if not always correct. Therefore, there is no need to strive for very specific solutions that aim at maximizing the accuracy [68], which are exceedingly common when dealing with ML-based software. Instead, **a classifier has to be considered as an unreliable component to be supported by additional (software) mechanisms to ensure trustworthiness.** As per the ISO/IEC TR24028 [69] - overview of trustworthiness in artificial intelligence - guidelines, ML-based software "*must incorporate diagnostic measures to safely manage abnormal behavior, ensuring a transition to a secure state*". That is why even companies and stake-

holders came up with the "safety first for automated driving" whitepaper [70], which is geared towards autonomous driving, but offers a high-level software architecture for (DNN) classifiers. The proposed architecture employs 4 blocks (preprocessing, DNN, uncertainty, and post-processing) whose behavior is continuously monitored through a safety envelope (bottom of Figure 1) The blocks ensure i) a careful input check (e.g., anomaly detection, out-of-distribution detection) and sanitization; ii) a robust training of the model via data augmentation (in case of white-box classifiers) or usage of ensemble learning; iii) estimation of uncertainty or confidence in prediction; and iv) generic post-processing, to minimize classification errors and eventually transitioning to an unconfident state.

This pairs very well with our paradigm of shifting from correctness – which is desirable indeed – to trust, or rather the ability to avoid misclassifications. Typical ML algorithms for classifications see these two quantities as opposed: the higher the accuracy, the lower the misclassifications. However, when rejections come into play, the bond between correct predictions and misclassifications is loosened, allowing for a wide variety of solutions that are not constrained only to "raising accuracy".

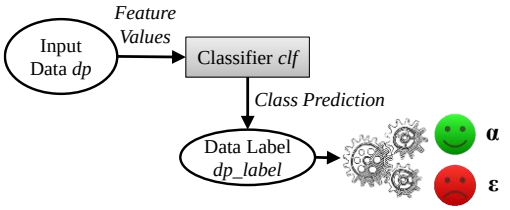
Takeaway 1: Correctness is not the only performance target: rejecting (all and only) misclassifications, will allow for predictions to be justifiably trusted even in absence of perfect correctness.

On the downside, handling rejections is not trivial: the encompassing system or software architecture should know how to promptly act to guarantee that the system will not be negatively affected by the rejection above (or notification of suspicious prediction). Intuitively, automatic or semi-automatic reaction and mitigation strategies are both domain-specific and system-specific. There are multiple examples in which rejecting potentially wrong predictions has clear benefits in the behaviour of a software or a system, even at the cost of rejecting a non-negligible amount of correct predictions.

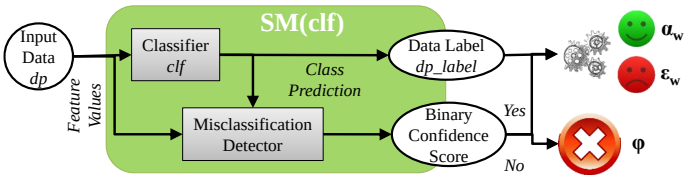
Within automated driving, tasks as semaphore or traffic sign recognition should avoid misclassifications of red/green semaphores or confusing a traffic sign with another, but can typically afford to occasionally reject uncertain predictions, provided that the correct recognition happens early enough for mitigations as emergency braking or evasive steering to take place. Other tasks as obstacle or pedestrian detectors may still prefer a rejection over a misclassification, with rejections that are likely triggering emergency braking to avoid hitting a potentially undetected pedestrian [72]. Traditional railway systems have cyclic interactions with sensors, actuators, and communication channels, where information is supposed to be continuously shared (i.e., request of data, or "I am alive" pings). When no information is exchanged across many subsequent cycles, the component is deemed as malfunctioning. This does not pair well with classifiers, which do not account for "rejections", limiting their usage despite the many possible applications e.g., automatic visual inspection, rail maintenance management [72]. Stopping is not an option in aerospace systems: therefore, the rejection of a prediction cannot trigger routines that completely stop or shutdown equipment, but that instead aim at handling or tolerating this potentially adverse situation [70].

PROPOSAL: SAFETY MONITORS FOR BLACK-BOX CLASSIFIERS

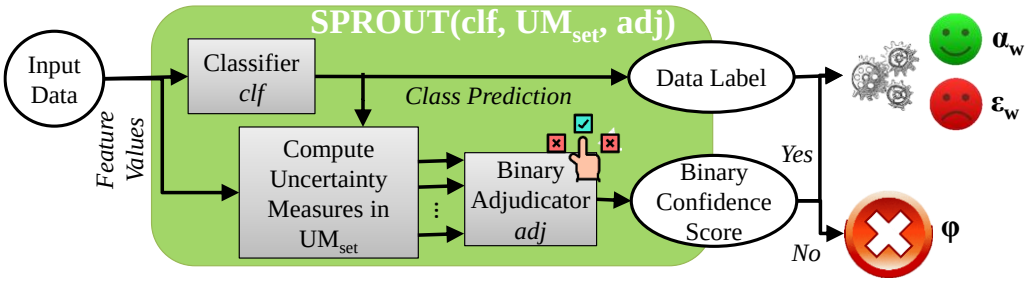
This section presents our proposal, which is supported by a publicly available Python framework.



a) Correctly predicts class with α accuracy and $\varepsilon = 1 - \alpha$ misclassification probability



b) Correctly predicts class with α_w accuracy, φ rejection probability and ε_w residual misclassification probability.



c) the SPROUT safety monitor, which uses an ensemble of uncertainty measures to detect misclassifications.

Figure 1. A classifier (top-left, Figure 1a), a classifier inside a SM (top-right, 1b), and within SPROUT (1c, down).

Evaluation of Safety Monitors

Regardless of how it is implemented, a safety monitor *SM(clf)* transforms a classifier *clf* which has $0 \leq \alpha \leq 1$ accuracy and a misclassification probability $0 \leq \varepsilon = (1 - \alpha) < 1$, into a software component that has:

- accuracy $\alpha_w \leq \alpha$;
- rejection probability $0 \leq \varphi \leq 1$. The *SM*(*clf*) may reject misclassifications φ_m or correct predictions φ_c. Overall, $\varphi = \varphi_m + \varphi_c$, and $\alpha_w + \varphi_c = \alpha$. Clearly, the desideratum is to minimize φ_c and maximize φ_m.
- residual misclassification probability ε_w, $0 < \varepsilon_w \leq \varepsilon \leq 1$; overall, $\varphi_m + \varepsilon_w = \varepsilon$.

Table 1. Probabilities α_w, ε_w, φ_c, φ_m for outputs of SM(clf) and compound probabilities.

SM(clf) behavior ↓	Correct Prediction	Misclassification	Sum
Not omitted	α_w	ε_w	$1 - \varphi$
Omitted	φ_c	φ_m	φ
clf behavior →	α	ε	1

All those probabilities are sketched in Table 1. Ideally, *SM(clf)* has almost the same accuracy as *clf* (i.e., $\alpha_w \approx \alpha$, or $\varphi_c \approx 0$), a substantially lower residual misclassification probability, $0 \approx \varepsilon_w << \varepsilon$, and an rejection probability close to ε, thus $\varphi \approx \varepsilon$. The following compound metrics may be calculated for a complete understanding of performance:

- φ_m *ratio* $= \varphi_m / \varphi$, the ratio of rejected misclassifications over all rejections of the *SM(clf)*, to be maximized. This is the precision of the rejection function implemented by the safety monitor.
- ε *drop* $= (\varepsilon - \varepsilon_w) / \varepsilon = \varphi_m / \varepsilon$, which is the drop in misclassifications due to SM, to be maximized. This is the coverage, recall or sensitivity of the rejection function implemented by the safety monitor.

A *SM(clf)* will never have better accuracy than *clf* (i.e., $\alpha_w \leq \alpha$), as it does not aim at improving correct classifications. It aims at transforming most of the erratic misclassifications, which are difficult to manage, into rejections i.e., have a high ε *drop*. Whereas at component-level this may seem a negligible improvement, at the system level it provides a way to prevent a misclassified prediction from propagating through the system, potentially causing (catastrophic) failures.

Black-Box Safety Monitors

Figure 1a depicts a simple classifier: the input data, and the features contained therein, is fed into a classifier *clf* that predicts a class label *dp_label* for that specific input data *dp* through its *predict* function. This classification process always outputs a class label that is then provided to the encompassing system, has α accuracy and ε misclassification probability. In this scenario, all the misclassifications are content failures. **Noticeably, insights of *clf* do not need to be disclosed for detecting misclassifications: as a result, *clf* is a black-box classifier**. The existence of a *predict* or *predict_p* function to generate a *dp_prob* and provide the output probabilities of the classifier to the misclassification detector is the **one and only assumption** we require for orchestrating any classifier in this study. Note that commonly used frameworks for machine learning (to name a few: *scikit-learn, xgboost, pyod, tensorflow*), expose this interface; therefore virtually any classifier can be deployed here. Another important observation is that this software architecture is general enough to suit image classifiers or classifiers for tabular data, rule or invariant-based algorithms, DNN, tree-based, statistical classifiers, for binary and multi-class classification.

Takeaway 2: Any classifier can be wrapped by a black-box safety monitor, provided that only a few interfaces are implemented for predicting labels for new data points.

Figure 1b still feeds the input data to the classifier, which predicts the class *dp_label* for an input data *dp*. However, the classifier is now paired with an additional component, a misclassification detector, which processes the input data *dp*, the classifier output *dp_prob* and *dp_label* and, if needed, can use the black-box classifier itself for additional predictions. The aim of the misclassification detector is to output a binary score [42] to decide if the class prediction is suspected to be a misclassification, enabling prediction rejection. This builds the software architecture of the safety monitor *SM(clf),* which rejects the *clf* output with probability φ; otherwise, the class prediction gets forwarded to the encompassing system, is correct with probability α_w and is a misclassification with probability ε_w. There is still a residual probability ε_w of content failure, while $\varepsilon - \varepsilon_w$ misclassifications are instead going to be rejected thanks to the safety monitor.

Uncertainty Measures for Classifiers

Our proposal needs uncertainty measures that are not classifier-specific, but instead have a generic formulation that pairs well with any classifier, which is seen as a black-box, and minimize the risk of poor significance of classifier-specific uncertainty. These measures are not limited to neural networks, but can be applied to any type of model, including both deep learning and classical machine

learning approaches. These are reported here with a reworked formulation that allows them to be used easily with shared interfaces.

Table 2 summarizes a total of 9 uncertainty measures UM1 to UM9, which process at least one of the following items: i) input data *dp*, ii) classifier *clf*, iii) class prediction *dp_prob*. Importantly,

- all measures but UM2, UM3 and UM8 require training data for set-up;
- all measures but UM2, UM3, UM4 are parametric, meaning that different values of parameters may create different instances of the same measure.

UM1: Confidence Range

Given a confidence level $0 \leq w \leq 1$ as input parameter, the confidence interval provides actionable information about the statistical distribution of the underlying feature values [1]. Let avg_i and std_i respectively be the average and standard deviation of feature *j* for a set of data points, and f the number of features. The confidence interval to be computed for each feature *j* is:

$$conf_j(w) = [avg_j - w*std_j;\ avg_j + w*std_j],\ 0 \leq j < f \qquad (1)$$

Instead, when labels belonging to *c* classes are provided in the training data, we compute confidence intervals for each feature and for each class, extending (1) to:

$$conf_j(w, i) = [avg_{ji} - w*std_{ji};\ avg_{ji} + w*std_{ji}], \qquad (2)$$
$$0 \leq j < f,\ 0 \leq i < c$$

A confidence level *w* very close to 1 defines a larger interval, reducing the likelihood of a feature value to fall outside of the confidence interval. Once confidence intervals $conf_j(w)$ or $conf_j(w,c)$ are computed for each feature, we can compute the uncertainty measure UM1 of the data point *dp*:

$$\text{from (1): UM1(dp, w)} = \frac{1}{f}\sum_{j=0}^{f} dp_j \notin conf_j(\text{w})$$

$$\text{from (2): UM1}(dp, w) = \min_{0 \leq i < c}\{\frac{1}{f}\sum_{j=0}^{f} dp_j \notin conf_j(\text{w,i})\}$$

UM1 ranges between 0 and 1. A value of 0 indicates that all feature values of *dp* lie within their corresponding empirical ranges, whereas a value of 1 indicates that all feature values lie outside them. Higher UM1 values therefore correspond to higher uncertainty, as a larger fraction of features deviates from the expected empirical range.

UM2: Highest Softmax Complement

Given *dp_prob* produced by a classifier for a given *dp*, we identify UM2 as the complement of the maximum in *dp_prob*:

$$\text{UM2}(dp\_prob) = 1 - \max(dp\_prob)$$

The higher the UM2, the more uncertain the output of the classifier [12].

UM3: Entropy of Probabilities

We retrieve the *dp_prob* produced by a classifier for a given *dp* and we compute UM3 using *db_prob* entropy [46]:

$$\text{UM3}(dp\_prob) = entropy(dp\_prob) =$$
$$-\sum_{i=0}^{c} dp\_prob_i * log(dp\_prob_i)$$

The higher the UM3, the more uncertain the classifier: a *dp_prob* array with constant values (i.e., all classes have the same probability) generates the highest UM3 of 1.

Table 2 Summary of Uncertainty Measures used in this study.

UM #	Name of the Uncertainty Measure	Needs Offline Setup	Uses Input Data	Uses clf Output	Parameters of the Uncertainty Measure	Tabular Data	Image Data	Supervised	Unsupervised
UM1	Confidence Range	✓	✓		*w*: confidence level	✓		✓	✓
UM2	Highest Softmax Complement			✓	-	✓	✓	✓	✓
UM3	Entropy of Probabilities			✓	-	✓	✓	✓	✓
UM4	Bayesian Uncertainty	✓	✓		-	✓	✓	✓	
UM5	Combined Uncertainty	✓	✓	✓	*chk_c*: classifier to check agreement with	✓	✓	✓	✓
UM6	Multi-Combined Uncertainty	✓	✓	✓	*CC*: classifiers to check agreement with	✓	✓	✓	✓
UM7	Feature Bagging	✓	✓		*bagC*: classifier to build bagger set	✓		✓	✓
UM8	Neighbourhood Agreement		✓	✓	*k*: number of relevant neighbors	✓	✓*	✓	✓
UM9	Reconstruction Loss	✓	✓		layers: structure of the AutoEncoder	✓	✓	✓	✓

UM4: Bayesian Uncertainty

This measure uses a Naïve Bayes process to estimate the probability that the input data point *dp* belongs to each of the possible *c* classes [2]. Noticeably, UM4 does not replace the prediction of the primary classifier. Instead, it provides an independent probabilistic assessment of the same data point *dp* using a model based on different assumptions and inductive bias. The goal is to capture additional uncertainty signals that may not be reflected in the primary classifier's confidence score. We define UM4 as:

$$\text{UM4}(dp) = \text{UM3}(\textit{NaïveBayes.predict\_p}(dp))$$

The assumption of independence between features of UM4 may not apply to many classification problems, especially those dealing with images, where a pixel clearly depends on its surrounding pixels.

UM5: Combined Uncertainty

We define UM5 as a generalization of UM4: instead of always using a Naïve Bayes classifier, we rely on a classifier *chk_c* that acts as a checker.

We define UM5 as follows: i) UM5 has positive sign if *clf* and *chk_c* agree on the predicted class, negative otherwise; ii) the absolute value of UM5 is quantified according to the entropy (UM3) in the results of *chk_c*.

$$\text{UM5}(\textit{dp\_prob}, \textit{chk\_c}, dp) = (-1)^k \, \text{UM3}(\textit{chk\_c.predict\_p}(dp))$$

where k=0 if the prediction of *clf* and *chk_c* is the same, i.e.,

$$\textit{arg max}(\textit{dp\_prob}) == \textit{chk\_c.predict}(dp), \text{ and } k=1 \text{ otherwise}$$

Results of UM5 range from -1 to 1. UM5 = 1 translates to high confidence that the prediction of *clf* is correct, UM5 = -1 means high confidence that the prediction is a misclassification, letting UM5 = 0 show maximum uncertainty.

UM6: Multi-Combined Uncertainty

UM6 computes uncertainty relying on more than one checker. UM6 uses a set CC of *ncc* checking classifiers, computes UM5 for each *chk_c* ∈ CC with respect to *clf*, and averages the results. The more checking classifiers in CC agree with *clf*, the higher the UM6.

$$\text{UM6}(\textit{dp\_prob}, CC, dp) = \frac{1}{ncc} \sum_{d_c \in CC} UM5(dp\_prob, d_c, dp)$$

UM7: Feature Bagging

UM7 exploits the concept of bagging [50], a method for generating multiple versions of a classifier *bagC* by making bootstrap replicates of the training set and using these as *bc* new instances of a classifier. Each instance of *bagC* is trained using different subsets of the original training set: it is only aware about a portion of the training set and decides using restricted knowledge. Should classifiers predict different classes for a given data point *dp*, this would mean that the prediction of *clf* should be treated with high uncertainty.

$$\text{UM7}(dp\_prob, dp, bagC) = \text{UM6}(dp\_prob, \{bagC_1, bagC_2, \ldots. bagC_{bc}\}, dp)$$

UM8: Neighbor Agreement

UM8 finds and uses the nearest neighbors [47] of a data point to understand if the input data point *dp* lies in a region of the input space where the majority of its neighbors is assigned (by *clf*) its same class. The usage of a prime *k* avoids ties in multi-class problems when the number of classes is smaller than *k*. The steps to execute UM8 are:

1. compute *dp_label* = arg max(*dp_prob*);
2. exercise a k-Nearest Neighbor [47] search, obtaining the *k* data points of the training set that are the closest to *dp*. We call this set of data points *neigh;*
3. for each data point *ne* in *neigh*, apply *clf.predict*(*ne*);
4. compute the final value as follows:

$$dp\_label = ag\ max(dp\_prob)$$

$$neigh = \text{Nearest-Neighbours}(Tr, dp, k)$$

$$nl = \text{for } ne \text{ in } neigh: clf.predict(ne)$$

$$\text{UM8}(dp\_prob, clf, dp, k) = \frac{1}{k}\sum_{i=0}^{k} 1\ if\ nl_i = dp\_label\ else\ 0$$

UM8 outputs a number between 0 and 1. The lower the value, the more disagreement in classifying neighboring data points to *dp*. This means that the input data point *dp* lies in an unstable region of the input space, which translates to high uncertainty (low UM8) in the prediction.

UM9 Reconstruction Loss

Reconstruction loss quantifies to what extent the input data point belongs to the distribution of past inputs or if it is an unseen, out-of-distribution data point [48]. The expectation is that misclassifications will happen frequently when dealing with out-of-distribution input data points. We compute UM9 through autoencoders, which are unsupervised neural networks that learn efficient encodings of the input data, which are validated and refined by attempting to reconstruct the input from the encoding [49]. Autoencoders are first trained using the training set and several layers, which SPROUT sets by default as composed of *f, f/2, f/4, f/2, f* neurons, with *f* being the amount of features of the input data. These layers can be customized whenever needed. UM9 is the L_2 norm (Euclidean distance) between the original data point and the reconstructed data point, and it quantifies the reconstruction loss.

$$\text{UM9}(dp) = \| dp - AutoEncoder.predict(dp) \|_2$$

UM9 is high when the data point cannot be efficiently reconstructed, and thus is likely out-of-distribution. A low UM9 value instead indicates that *dp* belongs to an expected distribution and as such is likely to be correctly classified.

Applicability of Measures

It is evident that some uncertainty measures require careful tuning, whereas others apply to any classification task as they are. It is the case of UM2 (Highest Softmax Complement) and UM3 (Entropy of Probabilities), which solely rely on probability distributions generated by classifiers. UM5 (Combined Uncertainty), UM6 (Multi-Combined Uncertainty), UM7 (Feature Bagging) leverage additional checker classifiers and compare their outputs with that of the main *clf*: this makes them applicable to any problem, but only after a specific parameter setup. Also, the parameter setup may have a relevant impact on their resource usage, making them flexible but computationally demanding. UM4 (Bayesian Uncertainty) can be applied only to supervised classification tasks, as it requires training a probabilistic classifier based on Bayesian inference, specifically a Naïve Bayes model, while UM1 (Confidence Range) and UM9 (Reconstruction Loss) focus on input data, providing an uncertainty quantification that is completely decoupled from the inference process.

In the context of image classification, specific uncertainty measures become less meaningful due to their reliance on statistical independence assumptions or feature-based proximity calculations, which do not usually apply to image data. UM1 (Confidence Range) assumes independent feature distributions, and, UM4 (Bayesian Uncertainty) struggles with images since the assumption of feature independence is unrealistic in visual data, where pixels are highly dependent on their neighboring pixels. UM8 (Neighbor Agreement) relies on k-Nearest Neighbors (k-NN) search, which is very ineffective even with all optimizations and may not effectively capture class separability in high-di-

mensional image spaces, making it impractical for most image classification tasks [64]. UM9 (Reconstruction Loss) has the potential to detect out-of-distribution samples but may struggle with complex image patterns, making it unreliable in many visual tasks. Other measures as UM2 (Highest Softmax Complement), UM3 (Entropy of Probabilities), UM5 (Combined Uncertainty) and UM6 (Multi-Combined Uncertainty) have the potential to work with images as well as with tabular data.

SPROUT: a Safety wraPper thROugh ensembles of UncertainTy measures

Our proposal of software architecture for a black-box safety monitor is in Figure 1c. Instead of relying on a unique mechanism to compute the binary confidence score and decide on rejection, we employ multiple uncertainty measures from Table 2, each processing different information, applying different rules and thus providing a different opinion on the input data (UM1, UM9), on the classifier output (UM2, UM3), on the behavior of the classifier against other checkers (UM4, UM5, UM6, UM7), or on the stability of a prediction (UM8). Each uncertainty measure outputs a floating point number, building an input array for a binary adjudicator, which delivers a binary score that decides on rejection. The resulting Safety wraPper thROugh ensembles of UncertainTy measures SPROUT(clf, UM_{set}, adj) is a software component where:

- *clf* is a black-box classifier,
- $UM_{set}=\{um_i, 0 \leq i \leq k \mid um_i \in \{UM1, UM2, \ldots, UM9\}\}$ is a set of $k = |UM_{set}|$ uncertainty measures
- *adj* is a binary classifier that is meant to handle tabular data (i.e., outputs of the measures in UM_{set})

Clearly, the choice of which uncertainty measures should be computed is of utmost importance [21]. Some uncertainty measures may make SPROUT detect most of the misclassifications (thus the residual misclassification probability ϵ_w would be very low) at a cost of many rejections, i.e., $\varphi >> 0$, $\alpha_w << \alpha$. Conversely, other measures may build a monitor with high accuracy ($\alpha_w \approx \alpha$, or $\varphi_c \approx 0$), but rarely rejects outputs ($\varphi \approx 0$) and fails in detecting many misclassifications ($\epsilon_w \approx \epsilon$, or $\varphi_m \approx 0$) making its behavior similar to the regular *clf*. The usage of a set of uncertainty measures also helps in smoothing this issue, balancing these corner cases. Moreover, several measures are parameter-dependent and as such can be instantiated multiple times and have a different behavior; this is the case of UM1, UM5, UM6, UM7, UM8 and UM9. The choice of parameters depends on the structure of the input data (e.g., tabular or image data), the type of the classification task (i.e., multi-class or binary).

Takeaway 3. Unlike existing works, SPROUT leverages multiple uncertainty measures to decide for rejection.

After uncertainty measures are exercised, a binary adjudicator processes the ensemble of floating point values computed using each uncertainty measure to output a binary confidence score. This binary adjudicator is completely decoupled from *clf* and can be implemented with thresholds, invariants, custom rules [11], or as a binary classifier, providing many degrees of freedom in finding the ideal function to combine ensembles of uncertainty measures into a unified output, even implementing non-linear decision functions. On a software architecture standpoint, the misclassification detector in SPROUT is a stacking meta-learner, with uncertainty measures at the base level, and a binary adjudicator at the meta-level [45]. The binary adjudicator is completely independent with respect to the input data of a specific problem: its features are entirely uncertainty measures: therefore, we could even deploy a unique adj for each and every classification task at hand, provided that the UM_{set} does not vary. This conjecture holds really well from an high-level architecture standpoint. However, the choice of UMs should be tailored to the classification task and, more importantly, to the input data type as already discussed in Table 2.

METHODS AND EXPERIMENTAL SETUP

This section details the experimental campaign to test SPROUT in detecting misclassifications of supervised and unsupervised classifiers for tabular and Image data. Experiments have been executed on a Dell Precision 5820 Tower with an Intel Xeon Gold 6250, GPU NVIDIA Quadro RTX6000 with 24GB VRAM, 192GB RAM and Ubuntu 18.04, NVIDIA driver 450.119.03 with CUDA 11.0, and required approximately three weeks of 24H execution with GPU support. The primary objective of these experiments is to address the following research questions:

- **RQ1**. To what extent can SPROUT monitors improve misclassification detection reliability compared to baseline classifiers without safety monitoring.

Takeaway 4. Once trained, a SPROUT monitor can be applied to a new case study (dataset, classifier) with minimal effort, reusing pre-trained models of the binary adjudicator.

- **RQ2.** How does SPROUT's misclassification detection performance vary across different data modalities, including supervised tabular, unsupervised tabular, and image classification tasks?
- **RQ3.** Which uncertainty measures most significantly contribute to SPROUT's ability to identify potential misclassifications?
- **RQ4.** What is the computational overhead introduced by SPROUT monitors, and how does it impact their applicability in resource-constrained or real-time systems?

Methodology

Our experimental methodology follows the steps below, and is depicted in Figure 2.

1. Gather 29 tabular and 4 image datasets, apply a 70/30 train–validation split, and divide them into two groups: monitor-train (MTR) for training SPROUT monitors and monitor-test (MTE) for evaluation.
2. Select uncertainty measures to build three SPROUT instances: *sup_tab* (supervised tabular), *uns_tab* (unsupervised tabular), and *img_sm* (image), as detailed in the SPROUT Setup with Image Classification (img_sm) and SPROUT Setup with Tabular Data Classification (sup_tab, uns_tab) sections.
3. Train supervised/unsupervised ML models for tabular data and DNNs for images on MTR train splits, and store the models for later use.
4. For each model, predict probabilities and classes on the validation set and compute the required UMs (*sup_tab, uns_tab, img_sm*). This generates one CSV per model–dataset with: classifier probabilities and label, UM values, and a binary *misc* flag (misclassification target for adjudicators)..

Combine the generated CSVs into three files (*sup_tab, uns_tab, img_sm*) to train and compare supervised binary adjudicators; after selecting the best, each monitor is ready for new datasets.

5. Apply sup_tab and uns_tab to tabular MTE datasets and img_sm to image MTE datasets (see bottom half of Figure 2), collect metrics for Results and Discussion and present UM importance in a . Importance of Uncertainty Measures for Safety Monitors section.

SPROUT Setup with Tabular Data Classification (sup_tab, uns_tab)

Here we detail the specific inputs for experiments with tabular data, which lead to deploying and using the *sup_tab* and *uns_tab* safety monitors.

Datasets

We consider the tabular datasets in Table 3: 11 datasets of network intrusion detection, 10 datasets of sensor spoofing attacks to biometric authentication, 7 datasets related to hardware monitoring for failure prediction, 3 datasets related to IoT systems. All of them will be used for supervised classification (train and evaluate the *sup_tab* monitor), but only those 9 containing two classes might be used for unsupervised classification (*uns_tab* monitor), which is constrained to be binary by design.

Network Intrusion Detection (NIDS). We select labelled datasets on network intrusions: NSL-KDD [29], ISCX12 [28], UNSW-NB15 [30], UGR16 [34], NGIDS-DS and ADFANet [33], AndMal17 [34], CIDDS-001 [29], CICIDS17 and CICIDS18 [32], SDN20 [36]. All datasets report normal data points and data points collected while the system is under attack. Features are mostly numeric features extracted by monitoring network traffic (e.g., bytes received per second, number of packets).

Spoofing Attacks to Biometric Authentication. We gather datasets related to biometric spoofing attacks from [14], including Fingerprint [37], Hand Gesture [41], Electrodermal Activity [39], Heart Rate [39], Human Gait [42], Keystroke [38], Voice [43], Face [44]. These datasets contain tabular features extracted from biometric samples provided by users. The dataset are enriched with the data augmentation process where the authors performed an attack injection campaign to generate the effects of 4 sensor spoofing attacks. Therefore, the 10 biometric datasets contain data belonging to 5 different classes: the normal class and 4 spoofing attack classes.

Hardware Failure Prediction. Classifiers may also be used to detect the failure of hardware components. We gather datasets of performance monitoring of hard disks, where data points are labeled as failure only if the monitored hard drive is in a fail state or going to fail thereafter. BackBlaze manufacturer [15] makes 5 years of hard drive data available to the public, reporting labeled data related to many SMART indicators of hard

Table 3. Name, number of classes, amount of features and # of data points of datasets used for experiments.

Domain	Dataset Name	# of Classes	# Features	Number of Data Points
Network Intr. Detection	ADFANet	6	11	132 002
	AndMal17	4	85	100 522
	CICIDS17	5	85	405 230
	CICIDS18	6	85	200 000
	CIDDS	5	16	400 000
	ISCX12	5	16	600 000
	NGDIS-DS	8	9	497 987
	NSLKDD	5	42	148 517
	SDN20	6	83	205 128
	UGR16	6	13	207 256
	UNSW-NB15	9	45	165 461
Biometric Authentication	Fingerprint	5	18	21 002
	Face	5	30	7 389
	Keystroke Tappy	5	8	24 247
	HRV(SWELL)	5	66	41 034
	HRV(WESAD)	5	62	14 213
	Human Gait	5	70	53 436
	EDA(SWELL)	5	54	10 345
	EDA(WESAD)	5	95	3 563
	Voice	5	21	3 324
	Hand Gesture Kinect	5	95	1 474
HW Monitoring	BackBlaze 2017	2	50	32 678
	BackBlaze 2018	2	50	29 003
	BackBlaze 2019	2	50	47 525
	BackBlaze 2020	2	44	22 955
	BackBlaze 17-20	2	90	132 155
	Mechanical Failure	2	18	7 906
	BAIDU	2	12	186 049
IoT	Scania Trucks	2	170	76 000
	HAI Pressure	2	59	200 000
	IoT Network	10	15	210 425

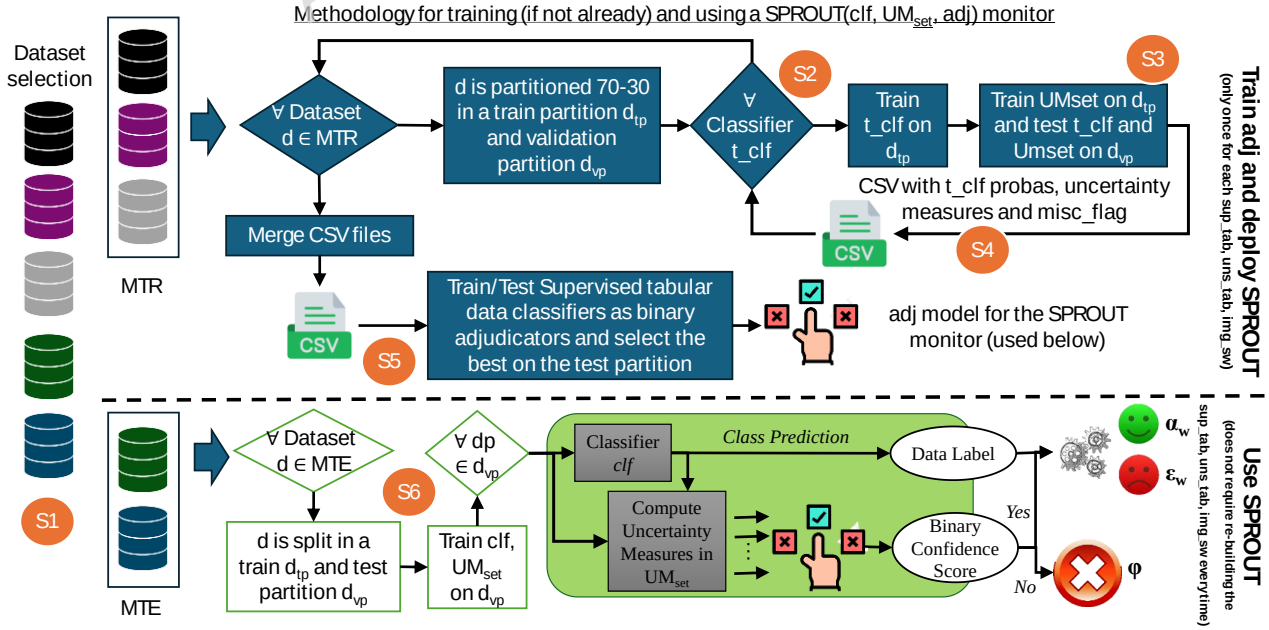


Figure 2. Picture of the experimental methodology for a generic SPROUT(clf, UM_{set}, adj). Steps from the methodology are depicted as orange circles.

drives, while another source of hard drive data came from the BAIDU [16] Kaggle competition whose input datasets are still available for download. Lastly, we used a dataset about the mechanical failure of electrical machinery in power plants [18].

IoT Systems Data (IoT). These datasets [17], [18], [76] are gathered by monitoring IoT systems: i) a network of sensors, ii) a distributed control systems of a power plant controlling a turbine, boilers, and water treatment systems, and iii) railroad trucks equipped for sensors to monitor brake pressure. Classifiers applied to the first dataset detect intrusions, while others act as error detectors and failure predictors.

Datasets Pre-processing. We transform the 31 tabular datasets into CSV files with the same tabular structure. We observe that all biometric authentication datasets but Hand-Gesture contain numeric ordinal features plus the label column. Instead, other datasets have heterogeneous structures: for instance, ISCX12, IoT-IDS, and UNSW-NB15 are available only as a collection of monitored network packets. Consequently, we first converted those packets into rows of CSV files to be merged at a later stage into a unique file for each dataset.

Then, we analyzed all datasets to remove features that are specific of the setup that was followed to gather data, namely: Timestamp, ID, experiment number, if any. Those features should be disregarded for classification purposes as they carry information about the experiments to build the dataset instead of being related to the system: classifiers using these features may learn how experiments were made instead of how the system behaves. Lastly, we removed duplicated columns (if any), and zero-filled all blank values in the BackBlaze datasets.

Definition of MTR and MTE sets. Many training (MTR) and test (MTE) sets were created by rotating datasets from Table 3. For *sup_tab*, we created 5 variants of MTR/MTE sets where 26 datasets were selected for MTR, leaving 5 for MTE. For example, one variant has MTE = {SDN20, UNSW, Fingerprint, BackBlaze 2020, HAI Pressure}, leaving the remaining 26 datasets as MTR. For *uns_tab*, only the 7 HW Monitoring, Scania Trucks and HAI Pressure datasets can be considered as they have a binary label. Similarly to sup_tab, we created 5 variants of MTR/MTE where 7 datasets were used as MTR, and 2 are left for MTE (e.g.,MTE = {Mechanical Failure, HAI Pressure}, leaving the other 7 into MTR). This creates 5 sup_tab and 5 uns_tab adjudicators, which will be exercised separately, presenting their performance and variability in the results section.

Classifier Selection

Amongst the many alternatives, we choose the following very well-known classifiers for tabular data. Classifiers' configuration was tuned through grid searches, using the HyperOpt [56] library whenever applicable.

Supervised Classifiers. Decision Trees (DT) and their ensembles as Random Forests (RF), Gradient Boosting (GB), eXtreme Gradient Boosting (XGB), are the de-facto standard [20], [57] for supervised classification of tabular data. For completeness, we exercised Logistic Regression (LR), Naïve Bayes (NB) and Linear Discriminant Analysis (LDA), alongside TabNet [6] and FastAI [7] neural networks, which are explicitly crafted to deal with tabular data. HyperOpt does not work with neural networks: in this case, we let the hyperparameter optimizer that is embedded in FastAI to automatically tune its parameters. For TabNet, we ran grid searches with 108 combinations of the following parameters and values: Learning rate $[e^{-5}, e^{-3}, e^{-1}]$, Batch size [512, 1024, 2048], Max Epochs [20, 50, 100], patience (for early stopping) [5, 8], target metric [mcc, accuracy].

Unsupervised classifiers. We select 8 algorithms belonging to different families [55] from the Python framework *PyOD* [54], namely: FastABOD (Angle-based), HBOS (Proximity), COPOD (Probabilistic), CBLOF (Clustering), PCA and MCD (Linear Model), IForest and SUOD (Ensembles). We do not consider autoencoders as unsupervised classifiers since they are already exercised within UM9. Note that unsupervised classifiers can only perform binary classification.

Uncertainty Measures and UM_{set} for sup_tab, uns_tab

The SPROUT monitors *sup_tab* and *uns_tab*, which apply to tabular data classifiers, will use the following uncertainty measures, which we instantiate with specific parameters when needed. Both monitors will use UM1) with w = 0.9, UM2) and UM3), which do not have parameters, UM8) with $k = 19$, which is prime to avoid ties in neighbor searches, UM9) using default parameter values.

Measures specific of Supervised Classification. In addition to that, the *sup_tab* monitor will use:

- UM4, which does not need parameters.
- UM5 with *chk_c* = XGB, which is a notoriously good classifier [20]. The measure is referred to as UM5_XGB.

- UM6 with three different groups of checking classifiers. We indicate the three UM6 configuration as *UM6_ST)* {NB, LDA, LR}, *UM6_TR*) {DT, RF, XGB}, and *UM6_NB)* {GaussianNB, BernoulliNB, MultinomialNB, ComplementNB}. The UM6_NB group includes different variants of the Naïve Bayes classifier.
- UM7 with *bagC* = DT, which has low computational complexity and has overall good classification performance (we call this UM7 configuration as UM7_DT). Since bagging creates multiple instances of *bagC*, choosing a complex *bagC* classifier may make UM7 take too much time.

The *sup_tab* relies on the UM_{sup} = {UM1, UM2, UM3, UM4, UM5_XGB, UM6_ST, UM6_TR, UM6_NB, UM7_DT, UM8, UM9}.

Measures specific of Unsupervised Classification. The *uns_tab* monitor will use:

- UM6 with all unsupervised classifiers {HBOS, COPOD, CBLOF, PCA, IForest, OCSVM}, and with classifiers that have linear computational complexity {HBOS, MCD, PCA}. Respectively, we refer to the two configurations as UM6_UNS and UM6_UF.
- UM7 with *bagC* = COPOD, which has the lowest computational complexity of our 8 unsupervised classifiers (we call this configuration UM7_CO).

The *uns_tab* relies on the UM_{uns} = {UM1, UM2, UM3, UM6_UNS, UM6_UF, UM7_CO, UM8, UM9}.

Binary Adjudicator

For supervised classification, we exercise each of the 9 supervised classifiers on each dataset in MTR, resulting in 9*26=234 experiments, each outputting a CSV file composed of *clf* probabilities and prediction, UM_{sup} values, and the *misc_flag*. These 234 CSV files are merged into a unique file composed of more than 13 million data points, which is used to train and validate the binary adjudicator. Supervised classifiers are tested as binary adjudicators using this dataset; results show that Random Forests (30 trees) are the preferred choice as binary adjudicator for the sup_tab monitor as they have better performance than other classifiers in suspecting misclassifications i.e., the binary adjudicator has to predict the *misc_flag* for a new data point. This completes the definition of the monitor:

$$sup\_tab(clf) = SPROUT(clf, UM_{sup}, RandomForest(n\_trees=30))$$

For unsupervised classification, we exercise each of the 8 unsupervised classifiers on each dataset in MTR, which is composed only by 7 datasets in case of unsupervised classification. This leads to creating a CSV file of roughly 2 million data points, which is used to train and validate the binary adjudicator. Supervised classifiers are tested as binary adjudicators using this CSV file; results show that Random Forests (50 trees) are the preferred choice as binary adjudicator for the *uns_tab* monitor as they have better performance than other classifiers in suspecting misclassifications of unsupervised classifiers. This completes the definition of the monitor:

$$uns\_tab(clf) = SPROUT(clf, UM_{uns}, RandomForest(n\_trees=50))$$

These safety monitors are depicted in Figure 3.

SPROUT Setup with Image Classification (img_sm)

Here we detail the specific inputs for experiments with images, which lead to deploying and using the *img_sm* safety monitor. Exercising each of the 6 classifiers on each of the 4 datasets and computing uncertainty measures provides a total of 24 csv files.

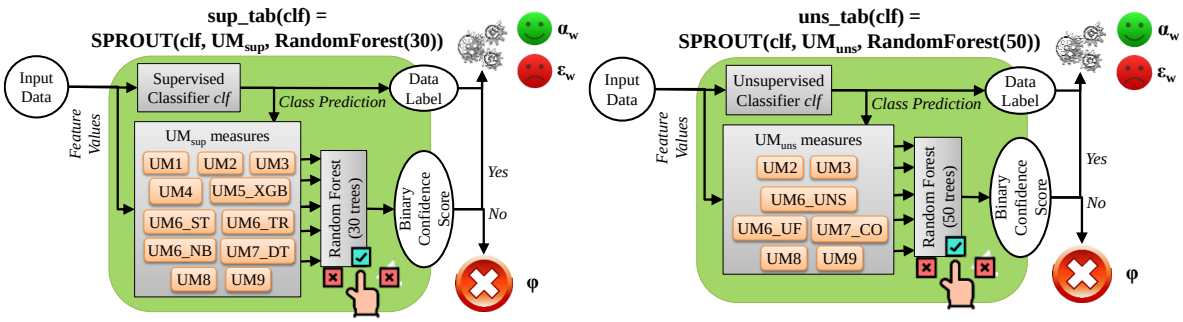


Figure 3. SPROUT monitors for supervised (*sup_tab*, left) and unsupervised (*uns_tab*, right) tabular classifiers.

Datasets and Preprocessing

We are using 4 different RGB (3 channel) datasets: CIFAR10 (32x32x3, 10 classes [58]), FER2013 (48x48x3, 7 classes [59]), FOOD-101 (512x512x3, 10 classes [60]) and Flower (variable size, 9 classes [59]) for training and testing the proposed technique. These were the datasets of choice as they are quite small, contain images in different RGB formats and are used a lot in the literature.

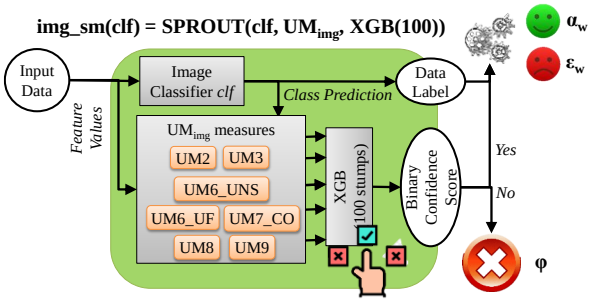


Figure 4. SPROUT for image classifiers *img_sm*.

After the train-test split, we augment training data through a composition of transformations: images are resized to a uniform size of 320x320 pixels, followed by random horizontal flipping and rotations up to 30 degrees to introduce spatial variability. Color adjustments such as brightness, contrast, saturation, and hue variations are applied to simulate diverse lighting conditions. The augmented images are then converted to tensors and normalized using the standard mean ([0.485, 0.456, 0.406]) and standard deviation ([0.229, 0.224, 0.225]) values derived from the ImageNet dataset i.e. in RGB format, ensuring consistency with the input distribution of pretrained models and enhancing compatibility with their feature extraction layers [62]. Resizing, normalization and conversion to tensors are applied to test data for consistency.

Given the limited amount (4) of image datasets, To ensure a comprehensive evaluation, we employ an alternative testing approach, where three datasets are used for training, and one dataset is reserved for testing. For instance, CIFAR-10 serves as the test set, while Flower, FER2013, and Food-101 are used for training. This process is repeated by rotating the datasets, ensuring a robust assessment of the adjudicator's performance across different scenarios.

Classifier Selection

Many DNN classifiers were proposed in the last decades, often with more and more demanding requirements concerning resource usage, posing challenges in scenarios where computational efficiency may be a limiting factor. Therefore, we selected DNNs for image classification that are relatively small and not excessively resource-hungry. We started from pretrained AlexNet, ResNet50, DenseNet, VGG11, GoogLeNet and InceptionV3 models [62], which are then used for transfer learning using a batch size of 32, ensuring efficient computation and stable gradient updates. A learning rate of $1e^{-4}$ was chosen to facilitate gradual learning, complemented by a weight decay of 0.001 to prevent overfitting. The Adam optimizer [63] was employed for its adaptive learning capabilities, while a step size of 1 and a gamma value of 0.5 were used to schedule learning rate decay, progressively reducing it to refine training. To prevent over-reliance on specific neurons, we apply a dropout layer with probability of 0.2. These hyperparameters were carefully selected to balance model performance, convergence speed, and regularization.

Uncertainty Measures and UM_{set} for img_sm

The SPROUT monitor *img_sm*, which applies to image classifiers, will use the following uncertainty measures, which we instantiate with specific parameters when needed.

- UM2 and UM3, which are not parameter-dependent.
- UM5 with chk_c = VGG11 as a parameter, referred to as UM5_VGG
- UM6, using as checking classifiers all classifiers from the previous section minus the classifier used as main classifier clf e.g., if clf =ResNet50, we define CC = {AlexNet, DenseNet, VGG11, GoogLeNet, InceptionV3}, which is all but ResNet50 DNN image classifiers. We refer to this measure as UM6_DNN.
- UM9 using the default parameter values considering images of size 320x320x3 i.e., 5 layers of size [320, 80, 40, 80, 320].

Overall, the *img_sm* relies on the following UMset = {UM2, UM3, UM5_VGG, UM6_DNN, UM9}.

Binary Adjudicator

The process for training the binary adjudicator is similar to that used for *sup_tab* and *uns_tab* monitors. We exercise each of the 6 DNN image classifiers on each dataset in MTR, collecting results and merging it in a unique CSV of roughly 0.26 million data points. The CSV is used to train and validate the binary adjudicator. Supervised classifiers are tested as binary adjudicators using this dataset; results show that XGB (100 stumps) is the preferred choice as binary adjudicator for the *img_sm* monitor as they have better performance than other classifiers in suspecting misclassifications. This completes the definition of the monitor, depicted in Figure 4:

$$img\_sm(clf) = SPROUT(clf, UM_{img}, XGB(n\_stumps=100))$$

A Python Library for Exercising SPROUT

SPROUT is available at [5] and as PIP Python package. The package implements all uncertainty measures discussed in this paper and makes SPROUT ready for deployment in any case study. Many already trained models for binary adjudication are already available in the library, and are accompanied by details about the uncertainty calculators they need, statistics on its binary classification performance and on the importance each uncertainty calculator had in learning that model. Those information are not needed to run SPROUT but provide interesting details for explaining why SPROUT works as intended.

Applying SPROUT to a brand new case study is very easy. Listing 1 reports a code snippet that shows a simple usage of SPROUT to wrap a supervised classifier from scikit-learn using a pre-defined (*ecai_sup*) binary adjudicator which uses the architecture in Figure 3. We assume to have a labeled dataset that we split in two parts. The first part will be provided as input to the *load_monitor* method

Takeaway 5. SPROUT monitors are extremely effective in suspecting misclassifications of supervised (tabular and image) classifiers, but have lower performance when applied to unsupervised classification tasks.

that prepares a SPROUT monitor according to the chosen model. Data is used to train the uncertainty measures, while the binary adjudicator is simply loaded from the repository. Then, we initialize and train a RF classifier, which we provide as input, alongside with unlabeled test data, to the *predict_misclassifications* function, which outputs

- a pandas *DataFrame* containing the values of uncertainty measures computed for all the test data points and the associated binary confidence score, and
- the model of the classifier used for binary adjudication. Binary confidence scores are extracted as *numpy* array and used to compute performance measures. If the test set is labeled, we can take advantage of labels to compute α_w and ϵ_w as shown in the last rows of Listing 1. Obviously, *y_test* labels will not available when deploying SPROUT in a real scenario.

Listing 1: applying SPROUT to supervised classifiers.

```
import sklearn as sk, numpy
from sprout.SPROUTObject import SPROUTObject
# We suppose having a dataset loaded as follows
#  x: a numpy array of tabular data points or images
#  y: a numpy array containing dataset labels
#  label_tags: unique labels in y
x_tr, x_te, y_tr, y_te =
       sk.model_selection.train_test_split(
              x, y, test_size=0.5)
# Initializes an empty SPROUT wrapper.
so = SPROUTObject()
# Loads a specific model for binary adjudication
so.load_monitor(model_tag='ecai_sup', x_train=x_tr,
       y_train=y_tr, la-bel_names=label_tags)
# Crafting classifier (can be any)
classifier  =  sk.ensemble.RandomForestClassifier()
classifier.fit(x_tr, y_tr)
# Suspects misclassifications of clf on test set
sp_df, bin_adj =
       so.predict_misclassifications(data_set=x_te,
              classifier=classifier)
# A numpy array of binary confidence scores
sprout_pred = sprout_df['pred'].to_numpy()
phi = 100*numpy.count_nonzero(sprout_pred == 0)
                       / len(sprout_pred)
# Computes alpha_w and eps_w
#           (only if y_test is available)
alpha_w = sum((1-sprout_pred)*(1-y_test)) /
           numpy.count_nonzero(sprout_pred == 0)
```

Results and Discussion

Once safety monitors are deployed, we can exercise them, collecting metric scores and comparing their behavior against traditional classifiers on different datasets.

Evaluation of Binary Adjudicators

Figure 5 presents a comparative analysis of a (supervised, unsupervised, image) classifier *clf* against a SPROUT monitor (respectively, *sup_tab, uns_tab, img_sm*) that uses *clf* as main classifier. The figure reports three different plots, where blue solid bars represent the misclassification probability (ε) of a *clf*, while the orange dotted line represents the residual misclassification probability (ε_w) after applying SPROUT. The green-striped bars depict the probability of rejections φ, ideally as low as possible. For *sup_tab* (Figure 5a) and *uns_tab* (Figure 5c), were multiple binary adjudicators were trained, the ε_w *and* φ values are averages, with error bars showing the standard deviation across the 5 variants. The ε value is always unchanged as it refers to the execution of the classifier without any binary adjudicator in place.

From Figure 5a, we observe how ε_w of the *sup_tab* SPROUT monitors is always far lower than ϵ (i.e., orange bars hover on the bottom of the plot and are always lower than 0.1, whereas the blue bars may even reach 0.4) of the supervised classifier *clf*, being extremely close to the optimum $\varepsilon_w \approx 0$ on SDN20 dataset. There are cases in which applying SPROUT to a *clf* that has a high misclassification probability ε may yield to the total absence of residual misclassifications $\varepsilon_w = 0$, which is an excellent result. For instance, LR on SDN20 has $\varepsilon = 0.2263$, meaning that more than 1 out of 5 predictions of

clf are misclassifications. Applying *sup_tab* leads to the total absence of residual misclassifications ($\varepsilon_w = 0$), which is an excellent result: the output of SPROUT is either a correct misclassification or a rejection, thus always trustworthy. This clear improvement in detecting and rejecting misclassifications over baseline classifiers directly addresses RQ1.

As a potential drawback, *sup_tab* rejects more than 1 out of 5 predictions of the classifier (φ = 0.2264, see green-striped bar), which is not desirable; however, this high rejection probability is a direct consequence of the high ε of LR classifier on SDN20. When the *clf* to be monitored has high ε (as it happens in the UNSW dataset), an high omission probability φ is unavoidable, even when omitting all ($\varepsilon_w = 0$) and only ($\varphi_c = 0$) misclassifications. On the downside, an high φ may also point to rejections of predictions that were going to be correct classifications. In the TabNet application to the UNSW dataset, the ε and φ bars are of the same height (as in the case we discussed previously), but the ε_w quantity is far higher than zero. This means that a consistent part of rejections are unnecessary, as they represent rejections of predictions that were going to be correct.

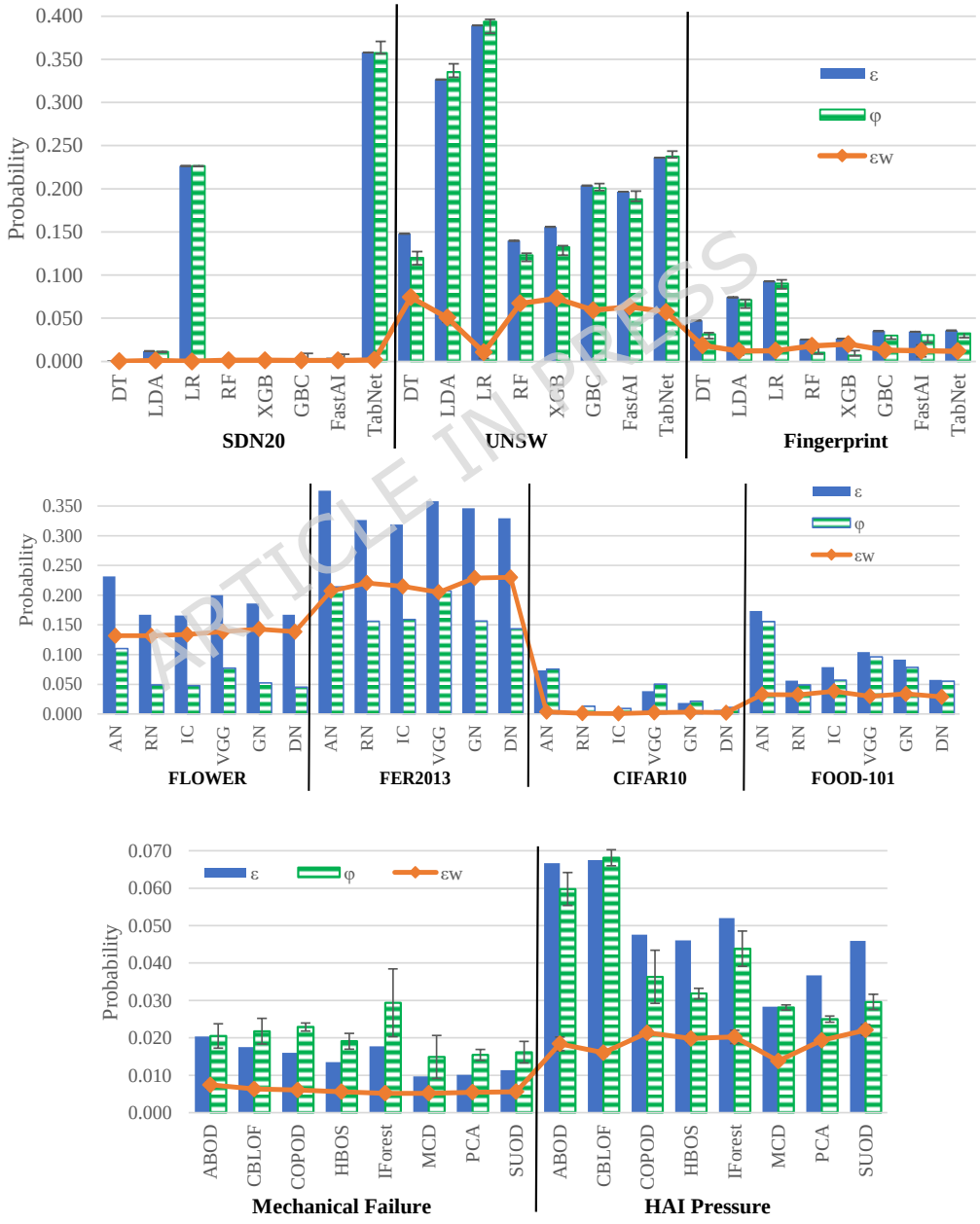


Figure 5. Misclassification probability (ε) of the classifier (solid bars), rejections (φ) (striped bars), and residual misclassification probability (εw) (line with diamonds) when applying sup_tab (up, 5a), img_sm (center, 5b), uns_tab (down, 5c) to test datasets.

Takeaway 6: Different uncertainty measures have different impact in suspecting misclassifications within SPROUT monitors. Additionally, some measures are more computationally (and time) intensive than others, making them applicable only in scenarios where computational resources are not tightly constrained.

Shifting to image classification in Figure 5b, we observe how the application of the *img_sm* monitor to the AN classifier on the Food-101 dataset allows for a substantial performance boost, with ε decreasing from 0.1669 to 0.0328 after applying SPROUT. Even better, almost all misclassifications were suspected when classifying images in the CIFAR-10 dataset, with a reasonably reduced amount of rejections (i.e., green bars are quite short). Results for the Flower and FER13 datasets show instead that misclassifications are reduced only by a small amount, with minor advantages from the SPROUT application to the classification of these images.

Lastly, we comment on the application of SPROUT for unsupervised classification (*uns_tab*) in Figure 5c. The trend is similar to the previous pictures: however, ε_w is always far from 0 in the HAI Pressure dataset (right of the plot), meaning that *uns_tab* is not able to suspect and reject a relevant portion of the misclassifications of unsupervised classifiers. We explain this result as follows. First, unsupervised classifiers always have lower accuracy and higher misclassification probability than supervised classifiers, thus $\varepsilon >> 0$. Secondly, and more importantly, the uncertainty measures UM6_UNS, UM6_UF, UM7_CO that we compute for unsupervised classifiers are trained without knowledge of labels, making them less accurate in suspecting misclassifications with respect to those of *sup_tab* and *img_sm*. Even in these sub-optimal case studies, applying the *uns_mon* monitor to unsupervised classifiers at least halves their residual misclassification probability ε_w, thus $\varepsilon_w < \varepsilon / 2$. These comparative evaluations across supervised tabular, image, and unsupervised tabular classification directly address RQ2, highlighting how SPROUT's performance varies across different data modalities and monitor configurations.

Importance of Uncertainty Measures for Safety Monitors

Ultimately, we quantify the impact each uncertainty measure has on learning the model for binary adjudication of each monitor *sup_tab*, *uns_tab* and *img_sm* using Table 4. The table presents the importance of each uncertainty measure (shown in the columns) has for each safety monitor, measured via the *feature_importances_* function from *sklearn* (for Random Forest binary adjudicators, *sup_tab* and *uns_tab*) and *XGBoost* (for the adjudicator of *img_sm*) Python packages, ensuring that the contributions of all uncertainty measures sum up to 1 within each safety monitor. Missing values in the table indicate that uncertainty measures that are not used by a particular SPROUT monitor i.e., they do not belong to their UM_{set}, while existing values are painted with a green gradient: the darker, the more a measure is relevant. In general, most of the variants of UM5, UM6, UM7 measures are very important for suspecting misclassifications, especially UM6_DNN for img_sm, UM6_UNS and UM6_UF for uns_tab, UM5_XGB and UM7_DT for *sup_tab*. Other uncertainty measures as UM1, UM8, UM9 have only a marginal role in suspecting misclassifications, having very low importance across the board on detecting misclassifications. These findings directly address RQ3 by identifying which uncertainty measures contribute most to SPROUT's misclassification detection capability across different monitor types.

Another important metric is the time overhead that each uncertainty measure adds to the inference process. It is expected that SPROUT monitors will be slower and more resource consuming than classifiers alone, but this impact has to be estimated. The last row of Table 4 reports a qualitative estimation for the time needed to compute all uncertainty measures used by SPROUT. Some measures as UM2 and UM3 can be computed in negligible time and do not add any overhead to the classification task. Measures as UM6_DNN, UM8 and UM6_TR require heavy computations which may significantly slowdown the execution of the classification task. We are aware that the overhead generated by the application of SPROUT may constitute an obstacle in systems which are resource-limited or that have tight real-time deadlines. Should the system be tightly constrained on resource usage, the user may decide to design a SPROUT monitor that uses time inexpensive uncertainty measures: this may lower the effectiveness of safety monitors, but will also guarantee their applicability in almost

Table 4. Importance of uncertainty measures for *sup_mon*, *uns_mon and img_mon* monitors to compute each measure.

SPROUT Monitor	UM1	UM2	UM3	UM4	UM5_XGB	UM5_VGG	UM6_DNN	UM6_ST	UM6_TR	UM6_NB	UM6_UNS	UM6_UF	UM7_DT	UM7_CO	UM8	UM9
sup_tab	0.032	0.027	0.036	0.01	0.189	n.a.	n.a	0.128	0.128	0.138	n.a.	n.a.	0.289	n.a.	0.004	0.017
uns_tab	0.054	0.07	0.083	n.a.	n.a.	n.a.	n.a	n.a.	n.a.	n.a.	0.236	0.214	n.a.	0.186	0.034	0.122
img_sm	n.a.	0.181	0.146	n.a.	n.a.	0.107	0.479	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.	0.086

every application scenario.

Back to the SPROUT monitors presented in this paper, with our experimental setup we found that applying *sup_tab* over a regular supervised classifier increases classification time by a factor of 4×, *uns_mon* triggers an even higher 5× overhead compared to only using an individual unsupervised classifier, while *img_sm* also results in a approx. 5× overhead, highlighting the computational trade-offs associated with each monitoring approach. These results directly address RQ4 by quantifying the computational cost of SPROUT and assessing its implications for deployment in constrained environments.

Threats to Validity and Reproducibility

Findings of this paper are supported by an experimental evaluation which intrinsically has (a few) threats to its validity. ML algorithms and classifiers have hyperparameters whose tuning critically affects results or may lead to a wide variety of problems when learning a model for each dataset during training (e.g., under/overfitting, poor quality of features, feature selection to leave out noisy features). Our experimental evaluation aims to compare the performance of classifiers safety monitors that wrap these classifiers, without optimizing performance on a specific scenario. Therefore, whereas the choice of hyperparameters of ML classifier is bound to change their behavior, it is unlikely to alter the conclusions we drew when experimenting with SPROUT monitors. Additionally, our experiments are directed to different application domains and classification tasks (supervised, unsupervised, tabular, image, rarely found in similar studies), meaning that even if a single result is skewed by an imperfect parameter tuning of issues in learning the model, this issue is smoothed down thanks to the wide range of experiments. This is important because this paper presents safety monitors to be applied to black-box classifiers: therefore, the conclusions should not be impacted by classifier's internals, which are not disclosed in such a scenario.

Concerning external validity: we cannot claim the validity of this study for classifiers other than those that we used in this study, but we believe that our findings are strong enough to generalize well to other classifiers. Regarding the application domain of our results beyond classification, they have the potential to apply to regression problems as well but will require a different set of uncertainty measures tailored for numerical labels.

Last, the usage of public data and public tools to run classifiers was a prerequisite of our analysis to allow reproducibility and to rely on proven-in-use data. All datasets are publicly available and referenced in the paper for download at publisher's sites; code is available at [5].

Conclusions and Future Works

Deploying autonomous decision-makers is one of the most challenging research issues these days, especially when they are used in critical tasks, whose failures may harm the health of people, infrastructures or the environment, e.g., fully autonomous driving, computer-guided robotic surgery, mobile robots for inspections and surveillance, optimized power management and generation, and manufacturing robots. Researchers and practitioners strive to build software components that are correct, meaning no misclassifications are provided as output of a classifier. Instead, this paper showed how misclassifications could be reduced not only by increasing correct predictions (i.e., having more accurate classifiers), but also by suspecting misclassifications and rejecting corresponding predictions, triggering alternative strategies instead. This shifts the paradigm from correct to a software that can justifiably be trusted thanks to its ability to reject predictions they are not confident with.

To deal with this problem, we presented the design, the implementation and experimental evaluation of SPROUT (Safety wraPper thROugh ensembles of UncertainTy measures), a safety monitor that applies to any black box classifier software, computes an ensemble of uncertainty measures and use them to compute a confidence score that can be used to decide if a prediction could be trusted. Experimental results using a wide range (35) of tabular and image datasets, 20+ supervised and unsupervised classifiers, showed how is it possible to consistently reject a significant portion of misclassifications, even suspecting all misclassifications of some classifiers (e.g., Logistic Regression on the SDN20 dataset). More in general, the main findings of the paper were highlighted as takeaways 1–6 in dedicated boxes throughout the paper.

Future work will focus on further reducing computational overhead, expanding support for additional uncertainty measures, and improving adaptability to dynamic, real-time environments. With its open-source implementation [5], SPROUT is readily available for integration into autonomous decision-making pipelines, paving the way for trustworthy software-guided decisions.

Acknowledgments

The authors have no acknowledgments to declare.

Funding

This work was supported in part by the project SERICS (PE00000014) under the MUR National Recovery and Resilience Plan funded by the European Union – NextGenerationEU, by the Cognitive Safety with Point Clouds (CogniSafe3D) Eurostars 3, Call 6 by the European Union, and by the HORIZON-JU-Chips-2024-1-IA-T3 Shift2SDV (101194245) project.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data is not owned by authors of this paper, but is publicly available and referenced in the paper. Code is available at [5].

Author contributions

F.A.K. and T.Z. provided the problem statement and perform Experimentations. L.C. and A.C. contributed to technical writing and analytical writing. A.B. performed technical proofreading and result gathering.

References

[1] Meeker, W. Q., Hahn, G. J., & Escobar, L. A. (2017). Statistical intervals: a guide for practitioners and researchers (Vol. 541). John Wiley & Sons.

[2] Krzanowski, W. J., et. Al. (2006). Confidence in classification: a bayesian approach. Journal of Classification, 23(2), 199-220.

[3] Bilgin, Z., & Gunestas, M. (2021). Explaining Inaccurate Predictions of Models through k-Nearest Neighbors. In ICAART (2) (pp. 228-236).

[4] Bishop, C. M. (2006). Pattern recognition. Machine learning, 128(9).

[5] SPROUT Repository on GitHub (online), https://github.com/tommyippoz/SPROUT

[6] Arik, S. Ö., & Pfister, T. (2021, May). Tabnet: Attentive interpretable tabular learning. In Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 35 (8), pp. 6679-6687.

[7] Howard, J. et. Al. (2020). Fastai: a layered API for deep learning. Information, 11(2), 108.

[8] Dan Hendrycks and Kevin Gimpel. A baseline for detecting misclassified and out-of-distribution examples in neural networks. arXiv preprint arXiv:1610.02136, 2016.

[9] Lakshminarayanan, et. Al. Safety and scalable predictive uncertainty estimation using deep ensembles. In Advances in Neural Information Processing Systems, pp 6405–6416, 2017.

[10] Jiang, H., Kim, B., Guan, M., & Gupta, M. (2018). To trust or not to trust a classifier. Advances in neural information processing systems, 31.

[11] Di Giandomenico, F., & Strigini, L. (1990, October). Adjudicators for diverse-redundant components. Proc. 9th Symposium on Reliable Distributed Systems (pp. 114-123). IEEE.

[12] A. Tiwari, B. Dutertre, D. Jovanović, T. de Candia, P. D. Lincoln, J. Rushby, and S. Seshia. 2014. Safety wrapper for security. In *HiCoNS 2014*, 85–94.

[13] Fonseca, J. R., et. al. (2005). Uncertainty identification by the maximum likelihood method. Journal of Sound and Vibration, 288(3), 587-599.

[14] Zoppi, T., Gharib, M., Atif, M., & Bondavalli, A. (2021). MetaLearning to Improve Unsupervised Intrusion Detection in CyberPhysical Systems. ACM Transactions on Cyber-Physical Systems (TCPS), 5(4), 1-27.

[15] BackBlaze: BackBlaze Hard Drive Data (online) https:// www.backblaze.com/b2/hard-drive-test-data.html

[16] BAIDU: Baidu Smart HDD Competition (online) https:// www.kaggle.com/drtycoon/baidu-hdds-dataset-2017/version/1

[17] MechFailure: Machine Failure Prediction Competition (online), https://www.kaggle.com/c/machine-failure-prediction

[18] C. Gondek et al. 2016. Prediction of failures in the air pressure system of Scania trucks using a random forest and feature engineering. In *IDA 2016*, LNCS 9897, Springer, Cham.

[19] IoT-IDS: IoT Intrusion (online) https://ieee-dataport.org/openaccess/iot-network-intrusion-dataset#files
[20] Shwartz-Ziv, R., & Armon, A. (2022). Tabular data: Deep learning is not all you need. Information Fusion, 81, 84-90.
[21] J. Guérin, R. S. Ferreira, K. Delmas, and J. Guiochet. 2022. Unifying evaluation of machine learning safety monitors. In *ISSRE 2022*, IEEE, 414–422.
[22] Hein, M., Andriushchenko, M., & Bitterwolf, J. (2019). Why relu networks yield high-confidence predictions far away from the training data and how to mitigate the problem. In Proc. of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (pp. 41-50).
[23] Aslansefat, K., et. al. (2020, September). SafeML: safety monitoring of machine learning classifiers through statistical difference measures. In International Symposium on Model-Based Safety and Assessment (pp. 197-211). Springer, Cham.
[24] Wang, M., Shao, Y., Lin, H., Hu, W., & Liu, B. (2022). Cmg: A class-mixed generation approach to out-of-distribution detection. Proceedings of ECML/PKDD-2022.
[25] Rossolini, G., Biondi, A., & Buttazzo, G. (2022). Increasing the Confidence of Deep Neural Networks by Coverage Analysis. IEEE Transactions on Software Engineering.
[26] Hüllermeier, E., & Waegeman, W. (2021). Aleatoric and epistemic uncertainty in machine learning: An introduction to concepts and methods. Machine Learning, 110(3), 457-506.
[27] Avizienis, A., Laprie, J. C., Randell, B., & Landwehr, C. (2004). Basic concepts and taxonomy of dependable and secure computing. IEEE transactions on dependable and secure computing, 1(1), 11-33.
[28] Ali Shiravi, Hadi Shiravi, Mahbod Tavallaee, and Ali A Ghorbani. 2012. Toward developing a systematic approach to generate benchmark datasets for intrusion detection. Computers & Security 31, 3 (2012), 357–374.
[29] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani. 2009. A detailed analysis of the KDD CUP 99 dataset. In *CISDA 2009*, IEEE, 1–6.
[30] M. Ring et al. 2017. Flow-based benchmark datasets for intrusion detection. In *Proc. 16th European Conf. on Cyber Warfare and Security*, ACPI, 361–369
[31] Nour Moustafa, Jill Slay. 2015. "UNSW-NB15: a comprehensive data set for network intrusion detection systems". In Military Communications and Information Systems Conference (MilCIS), 2015. IEEE, 1–6.
[32] Sharafaldin, I., Lashkari, A. H., & Ghorbani, A. A. (2018, January). Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization. In ICISSP (pp. 108-116).
[33] Haider, W., Hu, J., Slay, J., Turnbull, B. P., & Xie, Y. (2017). Generating realistic intrusion detection system dataset based on fuzzy qualitative modeling. Journal of Network and Computer Applications, 87, 185-192.
[34] A. H. Lashkari et al. 2018. Toward developing a systematic approach to generate benchmark Android malware datasets and classification. In *ICCST 2018*, IEEE, 1–7.
[35] Maciá-Fernández, G., Camacho, J., Magán-Carrión, R., GarcíaTeodoro, P., & Theron, R. (2018). UGR '16: A new dataset for the evaluation of cyclostationarity-based network IDSs. Computers & Security, 73, 411-424.
[36] Elsayed, M. S., Le-Khac, N. A., & Jurcut, A. D. (2020). InSDN: A Novel SDN Intrusion Dataset. IEEE Access, 8, 165263-165284.
[37] BIT – Biometrics Ideal Test, CASIA-FingerprintV5, http://biometrics.idealtest.org/
[38] Adams, Warwick R. "High-accuracy detection of early Parkinson's Disease using multiple characteristics of finger movement while typing." PloS one 12.11 (2017): e0188226.
[39] S. Koldijk, M. Sappelli, S. Verberne, M. A. Neerincx, and W. Kraaij. 2014. The SWELL knowledge work dataset for stress and user modeling research. In *ICMI 2014*, 291–298.
[40] Philip Schmidt, Attila Reiss, Robert Duerichen, Claus Marberger, Kristof Van Laerhoven, "Introducing WESAD, a multimodal dataset for Wearable Stress and Affect Detection", ICMI 2018, Boulder, USA, 2018

[41] A. Memo, L. Minto, P. Zanuttigh, "Exploiting Silhouette Descriptors and Synthetic Data for Hand Gesture Recognition", STAG. Smart Tools & Apps for Graphics, 2015

[42] Vajdi, A., Zaghian, M. R., Farahmand, S., Rastegar, E., Maroofi, K., Jia, S., ... & Bayat, A. (2019). Human Gait Database for Normal Walk Collected by Smart Phone Accelerometer. arXiv preprint arXiv:1905.03109.

[43] Kaggle – Voice Recognition, Jeganathan Kolappan. https://www.kaggle.com/jeganathan/voice-recognition (online), accessed: 2022-11-20

[44] Kaggle – Face Images with Marked Landmark Points, Omri Goldstein. https://www.kaggle.com/drgilermo/face-images-withmarked-landmark-points (online), accessed: 2022-11-20

[45] Wolpert, D. H. (1992). Stacked generalization. Neural networks, 5(2), 241-259.

[46] Dan Hendrycks and Kevin Gimpel. A baseline for detecting misclassified and out-of-distribution examples in neural networks. arXiv preprint arXiv:1610.02136, 2016.

[47] Cheung, K. L., & Fu, A. W. C. (1998). Enhanced nearest neighbor search on the R-tree. AUM SIGMOD Record, 27(3), 16-21.

[48] Xiao, Z., Yan, Q., & Amit, Y. (2020). Likelihood regret: An out-of-distribution detection score for variational auto-encoder. Advances in neural information processing systems, 33, 20685-20696.

[49] Kramer, M. A. (1991). Nonlinear principal component analysis using autoassociative neural networks. AIChE journal, 37(2), 233-243.

[50] Breiman, L. (1996). Bagging predictors. Machine learning, 24(2), 123-140.

[51] X.-Y. Zhang, G.-S. Xie, X. Li, T. Mei, and C.-L. Liu, "A Survey on Learning to Reject," Proceedings of the IEEE, vol. 111, no. 2, pp. 185–215, Feb. 2023, doi: 10.1109/JPROC.2023.3238024.

[52] Y. Bahat and G. Shakhnarovich, "Confidence from Invariance to Image Transformations," Apr. 2018, [Online]. Available: http://arxiv.org/abs/1804.00657

[53] A. Mandelbaum and D. Weinshall, "Distance-based Confidence Score for Neural Network Classifiers," Sep. 2017, [Online]. Available: http://arxiv.org/abs/1709.09844

[54] Zhao, Y., Nasrullah, Z. and Li, Z., 2019. PyOD: A Python Toolbox for Scalable Outlier Detection. Journal of machine learning research (JMLR), 20(96), pp.1-7.

[55] Zoppi, T., Ceccarelli, A., & Bondavalli, A. (2021). Unsupervised algorithms to detect zero-day attacks: Strategy and application. Ieee Access, 9, 90603-90615.

[56] Bergstra, J., Komer, B., Eliasmith, C., Yamins, D., & Cox, D. D. (2015). Hyperopt: a python library for model selection and hyperparameter optimization. Computational Science & Discovery, 8(1), 014008.

[57] Zoppi, T., Gazzini, S., & Ceccarelli, A. (2024). Anomaly-based error and intrusion detection in tabular data: No DNN outperforms tree-based classifiers. Future Generation Computer Systems, 160, 951-965.

[58] "CIFAR-10 and CIFAR-100 datasets." Accessed: Nov. 06, 2024. [Online]. Available: https://www.cs.toronto.edu/~kriz/cifar.html

[59] Dumitru, I. et al. (2013). *Challenges in representation learning: Facial expression recognition challenge*. Kaggle. https://www.kaggle.com/competitions/facial-expression-recognition-challenge

[60] L. Bossard et al. 2014. Food-101: Mining discriminative components with random forests. In *ECCV 2014*, LNCS 8694, Springer, 446–461.

[61] "National Flowers." Accessed: Mar. 26, 2024. [Online]. Available: https://www.kaggle.com/datasets/shahidulugvcse/national-flowers

[62] "Models and pre-trained weights — Torchvision 0.20 documentation." Accessed: Nov. 25, 2024. [Online]. Available: https://pytorch.org/vision/stable/models.html

[63] Tato, A., & Nkambou, R. (2018). Improving adam optimizer.

[64] Koutenský, F., Šimánek, P., Čepek, M. et al. Overcoming Long Inference Time of Nearest Neighbors Analysis in Regression and Uncertainty Prediction. SN COMPUT. SCI. 5, 486 (2024).

[65] Goodfellow, I. J., Shlens, J., & Szegedy, C. (2014). Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*.

[66] Guo, C., Pleiss, G., Sun, Y., & Weinberger, K. Q. (2017). On calibration of modern neural networks. Proceedings of the 34th International Conference on Machine Learning (ICML), 1321–1330. https://arxiv.org/abs/1706.04599

[67] Madry, A., Makelov, A., Schmidt, L., Tsipras, D., & Vladu, A. (2018). Towards deep learning models resistant to adversarial attacks. International Conference on Learning Representations (ICLR). https://arxiv.org/abs/1706.06083

[68] Li, B., Qi, P., Liu, B., Di, S., Liu, J., Pei, J., ... & Zhou, B. (2023). Trustworthy AI: From principles to practices. ACM Computing Surveys, 55(9), 1-46.

[69] ISO/IEC. ISO/IEC TR 24028:2020: Information Technology — Artificial Intelligence (AI) — Overview of Trustworthiness in Artificial Intelligence. International Organization for Standardization, 2020.

[70] A. Aptiv, B. Apollo, D. Continental, H. FCA, and I. V. Infineon. 2019. *Safety First for Automated Driving*. Continental, Daimler, FCA, HERE, Infineon, Intel, and Volkswagen, pp. 1–157.

[71] Carmichael, C. (2001). Triple module redundancy design techniques for Virtex FPGAs. Xilinx Application Note XAPP197, 1.

[72] A. S. Ami et al. 2023. 'False negative—that one is going to kill you': Industry perspectives of static analysis–based security testing. In *IEEE S&P 2024*, IEEE, 19.

[73] Alawad, H., Kaewunruen, S., & An, M. (2019). Learning from accidents: Machine learning for safety at railway stations. IEEE Access, 8, 633-648

[74] Goodfellow, I., Pouget-Abadie, J., Mirza, M., et al. (2014). Generative adversarial nets. NeurIPS.

[75] Steinhardt, J., Koh, P. W., & Liang, P. (2017). Certified defenses for data poisoning attacks. NeurIPS.

[76] H.-K. Shin, W. Lee, J.-H. Yun, and H. Kim. 2020. HAI 1.0: HIL-based Augmented ICS Security Dataset. In *CSET '20*, USENIX, Santa Clara, CA.

[77] Zoppi, T., Ceccarelli, A., & Bondavalli, A. (2023). Ensembling Uncertainty Measures to Improve Safety of Black-Box Classifiers. In ECAI 2023 (pp. 3156-3164). IOS Press.

[78]