Fail-Controlled Classifiers: Do they Know when they don't Know?

Tommaso Zoppi[1,*], Fahad Ahmed Khokhar[1], Andrea Ceccarelli[1], Leonardo Montecchi[2], Andrea Bondavalli[1]
[1] Dept. of Mathematics and Informatics, University of Florence, Viale Morgagni 65 - 50142 Florence, Italy
[2] Dept. of Computer Science, Norwegian University of Science and Tech, SemSælands vei 7–9, 7034, Trondheim, Norway
{tommaso.zoppi, fahadahmed.khokhar, andrea.ceccarelli}@unifi.it, leonardo.montecchi@ntnu.no, bondavalli@unifi.it

***Abstract* — Domain experts are desperately looking to solve decision-making problems by designing and training Machine Learning algorithms that can perform classification with the highest possible accuracy. No matter how hard they try, classifiers will always be prone to misclassifications due to a variety of reasons that make the decision boundary unclear. This complicates the integration of classifiers into critical systems, where misclassifications could directly impact people, infrastructures, or the environment. The paper proposes to consider a classifier as a structural part of the system instead of an individual component to be tested in isolation and included in the system afterward. This allows for omitting those predictions that are suspected to be misclassifications, triggering system-level mitigation strategies. The resulting fail-controlled classifiers (FCCs) are software components that can correctly classify, misclassify, or omit outputs: ideally, they would omit all and only outputs that correspond to misclassifications. After presenting the theoretical foundations of FCCs, the paper proposes metrics to quantify their performance, 5 software architectures for FCCs, and an experimental analysis involving tabular data and image classifiers. Overall, this paper advocates the need for a system and software design in which ML classifiers are not separate components, but should rather be considered building blocks that interact with other components for improved performance.**

Keywords - fail safe, software architectures, classifiers, confidence, prediction rejection, critical systems

I. INTRODUCTION

"If you can't say something nice, don't say nothing at all" tells *Thumper* the rabbit to *Bambi* in the famous, 80-years-old Disney cartoon. This small rabbit teaches us an important lesson. There are cases in which omitting an answer that you are not confident about may be more beneficial than trying the "most likely" answer. This is true also when answering questions in tests, where a correct answer provides you a positive score, but a wrong answer may provide a non-neutral, negative score. Obviously, the rate of omissions has to be reasonably low: a decision-making entity that always omits outputs will never be wrong but will also never be useful for any purpose. These two aspects have to be carefully balanced, aiming at an ideal trade-off that omits all and only wrong answers. This is substantially different from crafting decision-making entities that are always correct, which is usually an unrealistic expectation.

Within the Information and Communication Technology (ICT) domain, many software components act as decision-makers, with the desideratum of being correct. In critical systems engineering, a typical approach to guarantee safety [5] is to equip a functional component with another component that aims at triggering a fail-controlled, fail-safe, or fail-stop behaviour whenever the correct functioning is not guaranteed [1], [6], [41], [60]. System-wise, a critical function would either i) deliver a correct result or ii) omit those results that would have been incorrect i.e., the function should have fail-omission failures only. Omissions should be handled by the encompassing system, e.g., exercising diagnostic routines, protecting key assets, and implementing automatized commands [36], [37]. This is a solid approach in systems engineering, which allows for the integration of Commercial Off-The-Shelf (COTS) components or non-trustable components in general into critical systems e.g., railway, avionics, autonomous driving, provided that individual components are orchestrated to achieve the desired non-functional properties at system-level.

Nowadays, the real challenge system architects are dealing with is integrating Machine Learning (ML)-based components that perform classification (referred to as "classifiers" in the paper) into critical systems such that their wrong predictions do not trigger catastrophic failures. Classifiers can effectively serve a wide variety of purposes: in critical systems, they are usually used for detecting deviations that may be due to the occurrence of faults or attacks, and perform error detection, intrusion detection, or failure prediction [6], [8], [20], [24], [50]. Moreover, classifiers can perform high-quality classification of images, which is of paramount importance for obstacle detection [32], [36] and traffic sign recognition [51] for autonomous driving, or to equip webcams (edge computing) and related components (cloud/fog computing, and other standalone or centralized architectures) with image quality checks, accurate access control, or even classifying diseases in the medical domain [39]. In the last decade, academia, industry, and also National governments hugely invested in methodologies, mechanisms, and tools to embed classifiers into ICT systems, especially critical ones. Regardless of how much effort we put into building classifiers that are more and more accurate, they could still end up predicting a wrong class for a given input data point, i.e., a *misclassification*.

That's where the paper advocates for a paradigm change, leaning towards system thinking rather than component engineering. We should consider the classifier as a component to be deployed into a system rather than chasing the holy grail of perfect accuracy. This provides more flexibility as it does not require the classifier to be infallible "in isolation", but allows for multi-component or system-level mechanisms or protocols to handle uncertain predictions that are suspected to be misclassifications. For example, in autonomous vehicles, a classifier detecting road signs doesn't need to be 100%

accurate; if uncertainty arises, the system can trigger additional verification methods such as querying GPS data or alerting the driver to take control. This flexibility reduces reliance on perfect predictions and enables multi-component systems to manage misclassifications. When there is not enough confidence in the prediction, we shift the responsibility from the classifier - which would have output its "best guess" - to the encompassing system, which runs more appropriate diagnostic or mitigation routines. The aim is not to reduce the amount of misclassifications of the classifier: instead, we aim at suspecting and omitting most (if not all) of them and trigger alternative strategies instead. Straightforwardly, finding a solid way to suspect misclassifications is of paramount importance for designing what we call a Fail-Controlled Classifier (FCC).

After detailing the related works, we dive into FCCs, motivating their importance, formalizing their basic mathematical notions and the concept of confidence or uncertainty in predictions of classifiers. Then, we present and discuss Self-Checking Classifiers (SCC), Watchdog Timers (WT), Input Processor (IP), Output processor (OP), Safety Wrapper (SW) software architectures to build Fail-Controlled Classifiers (FCCs), discussing possible variants due to implementation and design choices. Further, we conduct a preliminary experimental analysis in which we apply FCCs to tabular data and image classification, comparing their performance with those of traditional classifiers, discussing differences and highlighting the main takeovers. As a last contribution of the paper, we discuss how more complex safety and reliability engineering paradigms as recovery blocks, N-Version Programming, voting [41], [42], [44], could be used for building FCCs that potentially have both low misclassifications and low probability of omissions.

The rest of the paper is structured as follows: Section II reports background on ML classifiers, while Section III introduces the theoretical foundations of FCCs, whose software architectures are presented in Section IV. Section V presents our preliminary experimental analysis, letting Section VI discuss other potential approaches for FCCs. Section VIII concludes the paper.

II. Background on ML Classifiers

A. Classification of Structured and Unstructured Data

Decades of research and practice on ML provided us with plenty of classifiers that are meant to always output a prediction. Supervised classifiers [18] 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 [46], [48], [75], despite alternatives based on DNNs exists [8]. Conversely, image classification employs DNNs, which can learn strong features from pixel maps [7], [11].

Regardless of their complexity and structure, a classifier *clf* first devises a mathematical model from a training dataset [7], which contains a given amount of data points. Each data point *dp* contains a set of *f* feature values and describes a specific input of the classification problem. Once the model is learned, it can be used to predict the label of a new data point, different from those in the training dataset. The classification performance is usually computed by applying *clf* to data points in a test dataset and computing metrics such as *accuracy* [17], i.e., the percentage of correct predictions of a classifier *clf* over all predictions. Noticeably, *1 – accuracy* quantifies the misclassification probability by difference.

B. Confidence of Classifiers

Trusting each prediction of a classifier, to the extent that the prediction can be propagated toward the encompassing system and safely used in a critical task, is very challenging [13]. Researchers and practitioners are actively investigating ways to quantify uncertainty and learning to reject [71] misclassifications. Some approaches use confidence intervals [9] or the Bayes theorem [10]. Works as [12] 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 [11], authors processed softmax (i.e., a probability distribution over all possible classes obtained from raw outputs of the ML algorithm) probabilities of neural networks to identify misclassified data points. A new proposal came from [13] and [14], where authors paired a k-Nearest Neighbor classifier with a neural network to compute uncertainty. The work [34] computed the cross-entropy on the softmax probabilities of a neural network and used it to detect out-of-distribution input data that is likely to be misclassified.

Ideally, we want classifiers to be highly confident about predictions that turn out to be correct, and show low confidence for all and only the predictions that will instead be misclassifications. However, classifiers may have high confidence even when misclassifying data points e.g., "*neural networks which yield a piecewise linear classifier function […] produce almost always high confidence predictions far away from the training data*" [15].

C. Towards Trustworthy Classifiers

Tackling very complex problems naturally exposes classifiers to a high probability of misclassifications, which can be reduced but not avoided at all. The sup-optimal choice of a suitable ML algorithm(s), the poor availability or quality of training data, and biased pre-processing and analyses may all constitute additional causes of misclassifications that instead should be avoided. On top of that, there may be other problems due to the operational environment in which the classifier is expected to operate [74], once deployed in a real-world or simulated scenario.

1) Out-of-Distribution Data and Outliers

Systems and software components may encounter anomalous inputs or operating conditions [24], [11], [26], [74] even with semi-static systems and in the absence of security threats that may be intentionally willing to damage our system. For tabular data, these are known as point or contextual anomalies (global or local outliers), whereas for recent image-based applications, those events are usually

referred to as out-of-distribution (OOD) data. Overall, those inputs do not belong to the distribution of training data: thus, the behaviour of the classifier may become unpredictable [11] and prone to misclassifications. Conversely, what makes OOD data and outliers tricky to classify also makes them detectable, provided that we can precisely characterize the "in-distribution" data [11], [26], [28].

2) Adversarial Attacks

Second, classifiers may operate in situations in which malicious entities may be willing to actively disturb the behaviour of classifiers, triggering misclassifications with targeted attacks. For image classification, this is the case of adversarial attacks, whose popularity saw an outstanding growth in the last decade after the first findings on data poisoning [29], adversarial patches [30], and gradient-based attacks [31]. As it happens with security-related issues, the likelihood of occurrence of adversarial attacks is a compound quantity that depends on the attacker's intent, the attack surface of the system, the knowledge of the attacker (i.e., white-box or black-box attacks) and many other attributes. Conversely to OOD and anomaly detection, ways to deal with adversarial attacks are still being actively researched as the topic is rather new. Many solutions already exist [23], [26], but nothing that can be considered proven-in-use yet.

3) Distribution Shifts, Emerging and Unexpected Events

Third, Machine Learning often works under the *Independent and Identically Distributed* (IID) or "closed world" assumption [27]. In a closed world, train, validation and test data are independently and randomly sampled from the same underlying distribution. However, most (if not all) the operational environments are dynamic, evolving, or complex enough to make this assumption very restrictive and valid only in a very small subset of static standalone systems. As a result, research moved to deploying classifiers that go beyond this assumption and are meant to operate in an open-world [27] where test data may be distributed (slightly) differently from training and validation data. These classifiers have to be robust to environmental changes, distribution shifts, emerging and unexpected behaviours, and even changes in the threat landscape [25], [32].

III. Fail-Controlled Classifiers: an Opportunity?

The desideratum is a classifier that has excellent classification performance when dealing with in-distribution data, but that is also pretty robust to the above events [33].

A. The Rationale and Applicable Domains

These characteristics are quite difficult to achieve, and typically make classifiers "bet" on a prediction they are unsure of. This best-effort behaviour does not pair well with critical systems, which require guarantees of correct component and system-level behaviour. Practically speaking, the probability of failure on demand of a critical component or system should be proven to be lower than specific thresholds.

It would be beneficial to change the failure semantics of classifiers from uncontrolled content failures (i.e., misclassifications) to omission failures. Ideally, we want to omit all and only erroneous predictions: on the downside, the availability of said component may be negatively affected when correct predictions are omitted in the process. Ways to build fail-controlled components [5] are well-known in the literature and often rely on safety wrappers or monitors [1], [6]. Safety wrappers are intended to complement an existing critical component or task by continuously checking invariants, or processing additional data to detect dangerous behaviors and block 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 [60]: this approach is not different.

Table 1: α_w, ε_w, φ_c, φ_m and compound probabilities.

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

Fail-Controlled Classifiers (FCCs) should perform runtime monitoring for suspecting misclassifications of the classifier itself, building on top of studies as [2], [3], [4]. Authors of [2] use probabilistic neural networks to model predictive distributions and, as a result, quantify predictive uncertainty using methods such as adversarial training. In [3], authors use distance measurements of the Empirical Cumulative Distribution Function as a trigger for the failure detector to actively track the behavior and operational context of the data-driven system. The study [4] suggests a simple monitoring architecture to improve the model's robustness to different harmful inputs, particularly those resulting from adversarial attacks on neural networks. Finally, authors of [20] combine a voting strategy with a safety monitor to build a safe and secure classifier for application in embedded systems.

Regardless of how it is implemented, a fail-controlled classifier *FCC(clf)* transforms a classifier *clf* which has $0 \leq \alpha \leq 1$ accuracy and a misclassification probability ε, $0 \leq \varepsilon = (1 - \alpha) \leq 1$, into a component that has:

- accuracy $\alpha_w \leq \alpha$;
- omission probability $0 \leq \varphi \leq 1$. The *FCC*(*clf*) may omit misclassifications (φ_m, desirable and to be maximized), or correct predictions (φ_c, unnecessary omissions to be minimized). Overall, $\varphi = \varphi_m + \varphi_c$, and $\alpha_w + \varphi_c = \alpha$;
- residual misclassification probability ε_w, $0 < \varepsilon_w \leq \varepsilon \leq 1$; overall, $\varphi_m + \varepsilon_w = \varepsilon$.

All those probabilities are sketched in Table 1. Ideally, *FCC(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 omission 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 omitted misclassifications over all omissions of the *FCC(clf)*, to be maximised;
- ε *drop* $= (\varepsilon - \varepsilon_w) / \varepsilon = \varphi_m / \varepsilon$, which is the drop in misclassifications due to FCC, to be maximized.

A *FCC(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 omissions 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.

B. Suspecting Misclassifications

Ways to suspect misclassifications of classifiers, and thus trigger omissions, can be partitioned in two groups: black-box and white-box approaches.

Black-box approaches do not assume any knowledge about the classifier, thus observe its inputs and outputs without any access to internals. They allow for building statistical machinery that conveys input pre-processing [9], output analysis [11] and even ensembles of them [19] for complex and quite effective techniques for suspecting misclassifications. They mostly check if inputs and outputs belong to specific statistical distributions, and deem the prediction as non-trustable otherwise. Other approaches aim at identifying unstable regions of the input space in which the classifier may be likely to output misclassifications [57], [59], or use external classifiers (e.g., nearest neighbours [13]) to validate the output of the target classifier.

When insights of the classifiers are at least partially disclosed, it is possible to apply white-box approaches. Those take advantage of specific features of the algorithm or the resulting model and use them to suspect misclassifications. For neural networks, a common approach is to check the activation patterns of neurons [4], [11] – which vary from a DNN model to another. Classifiers that orchestrate ensembles may use the degree of agreement or the diversity of predictions of the classifiers in the ensemble [2], [34] as a way to estimate the confidence in a given prediction: the looser the agreement, the more likely the misclassification. Tree-based classifiers have their own unique features that may be exploited for building custom trust measures [21]. Last but not least, knowing the structure of the classifier allows for a more careful interpretation of the computed confidence score, with the potential of limiting the problem of high-confidence, erroneous, predictions [15].

C. Motivation

The encompassing system should know how to promptly act to guarantee that the system will not be negatively affected in case of omission of the output of the FCC (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 omitting potentially wrong predictions has clear benefits in the behaviour of software or a system, even at the cost of rejecting a non-negligible amount of correct predictions.

FCCs could find wide application in the control system of semi-autonomous vehicles. 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 [36] to take place. Other tasks as obstacle or pedestrian detectors may still prefer an omission over a misclassification, with omissions that are likely triggering emergency braking to avoid hitting a potentially unseen pedestrian [37].

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 [34]. This does not pair well with classifiers, which do not account for "omissions", limiting their usage despite the many possible applications e.g., automatic visual inspection, rail maintenance management [73]. Differently, FCCs pair extremely well with this paradigm as they can minimize misclassifications, knowing that subsequent omissions will likely trigger safe states where the component will be stopped.

Stopping is not an option in aerospace systems: therefore, the omission of a prediction cannot trigger routines that completely stop or shutdown equipment, but that instead aim at handling or tolerating this potentially adverse situation [35].

IV. Software Architectures for FCCs

This section reviews and adapts existing architectures for critical systems engineering that focus on pre-processing /input validation (IP), post-processing /output validation (OP), component monitoring (WT, SW), or use built-in functionalities (SCC), for crafting FCCs in Figure 1. The numbering of sections matches the alphabetic numbering of subfigures of Figure 1, e.g., Section IV.A details what it is shown in Figure 1a.

A. Self-Checking Classifier SCC

Self-checking or self-testing hardware or software components embed built-in and custom strategies to check for the quality of their execution. This approach is required by many standards for deploying transportation systems and usually involves crafting hardware with redundancy and seeking for an agreement of the outputs of the replicas [55], or employing testing libraries that are periodically exercised on both hardware and software equipment [56]. Whenever one of these checks fail, the target component was deemed as failed and in need to be replaced or fixed. Whereas replicas refer to multiple redundant systems or components that perform the same task independently. The system compares the outputs of these replicas, and if they agree, the result is considered reliable. If there is a disagreement, further checks or fail-safe mechanisms are triggered to ensure safety.

For classifiers, this approach translates into looking for measures, indexes or other variables that may be generated during the inference process and that provide a quantitative confidence or trust score to assign to each prediction. In case the classifier computes a score and then applies a threshold to decide on the class probabilities, the distance of such score from the decision boundary can be used as a confidence measure: the closer to the decision boundary, the less confident the prediction. Applications of this way of computing confidence can be found for unsupervised (binary)

classifiers, DNNs, and also for improving classification of noisy data [57], but are not available by default in standard libraries used for supervised learning e.g., Python's *scikit-learn, PyTorch, Keras*.

B. Watchdog Timer WT

Watchdog timers (WTs) aim at measuring the length of the execution of a target function to understand if the elapsed time conforms with expectations [53]. In case the function completes too early, the watchdog timer generates an alert that we can use to trigger omissions. If the function completes too late, it indirectly delivers an omission as well.

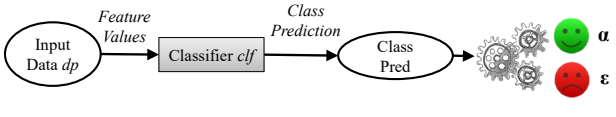
The reader may see this as a trivial check; however, it has been and currently is being used in many embedded or cyber-physical systems as a runtime check of the state of IT machinery. Decades ago, watchdog timers were meant to check electronic or mechanical-related functions [54], but transitioned to check the execution of software [53] and thus constitute an additional way to build FCCs. The clear advantage is that they add negligible overhead and work with any black-box classifier. On the negative side, they will be able to spot only a limited subset of issues (e.g., several anomalous activation pattern of neurons in DNNs, long paths in decision trees that may be due to an overfitted model, problems due to underlying hardware, operating system or virtualized middleware, or slow-downs due to malicious or malfunctioning software acting in the host system), resulting in a low omission probability but high residual misclassifications. Importantly, tuning timers is a system-specific process: a WT may work well with a specific hardware, but requiring re-tuning when the same hardware gets updated i.e., the notion of normal prediction time varies.

C. Input Processing IP

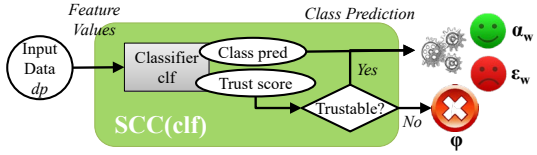
This FCC performs a pre-processing to seek for anomalies, suspicious values, low quality of such input, and the like. The pre-processing is implemented by an input checker, which could exercise adversarial attack detectors [26], out-of-distribution detectors [28], image corruption detectors [39], statistical distributions [9], or unknown data detectors in general [24]. Detecting one of the cases above could trigger an omission of the output of the FCC, without exercising the classifier at all. Should these events be quite frequent, the IP will show a fairly high amount of omissions. Importantly, some classifiers are "robust enough" to successfully deal with minor issues in the input data: in this situation, the FCC would want to omit the output only when the issue or corruption will not be tolerated by the robust classifier, reducing omissions. Some strategies pre-process the input to seek for issues but also provide a "cleared up" version of the same input at the end of the process. This is especially common for image classifiers, where autoencoders are often used to remove background noise, small alterations or minor damage to the image [50]. The reconstruction error is used as a symptom of corruptions, but the process also generates the "clean" image that can therefore be fed to the classifier instead of the initial, potentially noisy, image. In this case, even a "non-robust" classifier may still be able to correctly classify the "clean" image.

D. Output Processing OP

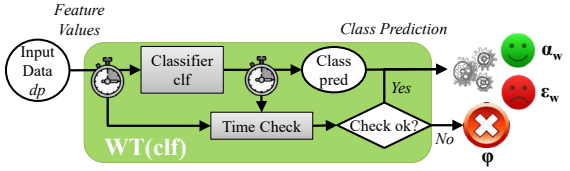
This is the simplest FCC out of the ones that we present in this paper, as this directly acts on the output probabilities of a prediction [11], computing the entropy of the probabilities, or use the absolute value of the highest probability as indicators



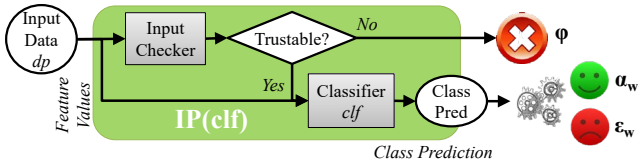
x) A simple (reference) classifier, that provides correct or incorrect output against input.



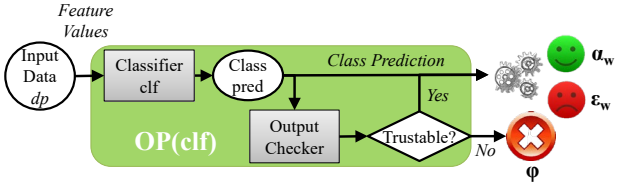
a) Self-Checking Classifier (SCC), which already has built-in and non-trivial methods for calculating trust in a prediction.



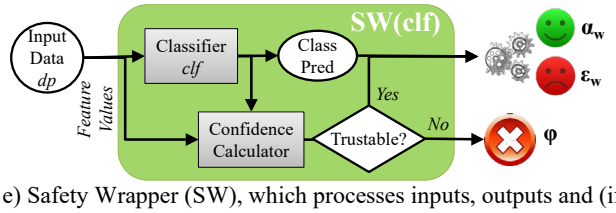
b) Watchdog Timer (WT), which measures inference time seeking for abnormal (too long or too short) executions.



c) Input Processor (IP), which checks for integrity issues, anomalies or legitimacy of inputs.



d) Output Processor (OP), which checks if the output of the classifier *clf* should be trusted or discarded.



e) Safety Wrapper (SW), which processes inputs, outputs and (if white box) the internals of *clf* to compute confidence and use it to decide on trustworthiness.

Figure 1. Software Architectures for FCCs with accuracy α_w, misclassification probability ε_w, and omission probability φ.

of trust in the prediction. Entropy, in this case, quantifies the uncertainty inherent in the prediction, with lower entropy indicating a more confident prediction where one class probability dominates, and higher entropy suggesting greater uncertainty due to more evenly distributed probabilities across multiple classes. In case the entropy is too high or the highest probability is below a given threshold, the prediction would be rejected, triggering an omission.

E. Safety Wrapper SW

IP and OP FCCs act before or after the classifier. However, we may think of a monitor or wrapper [6] that acts before, during and after inference. This is the case of the SW, which builds an envelope around the classifier to extract the most information to quantify the uncertainty of a prediction.

SW FCCs are partitioned into two big groups depending on their knowledge of the classifier. If the internals of the classifier are not disclosed (i.e., the classifier is a black-box), the SW can only act on interfaces and can eventually use the classifier for additional predictions. Implementations of black-box SW may rely on Bayesian approaches [10], ensembles of confidence measures [19], relative positioning of input data with respect to the prediction [13] or run other classifiers (e.g., kNN [14], probabilistic DNNs [12]) and check for agreement with the main classifier.

Conversely, when internals of the classifiers are fully available, it is possible to craft very specific mechanisms that are going to be run in parallel with the classifier (thus speeding up the execution) and that seek for very specific information throughout the inference process. The activation patters of neurons in a DNN or the length of a path in a decision tree can provide information on the whole prediction process and thus on its uncertainty [4].

F. Discussion

SCC, WT, IP, OP, and SW FCCs all have their limitations and advantages. Ideally, we want low omission probability φ, low residual misclassifications ε_w with low overhead. Since none of the solutions above (and none at all, according to the knowledge of authors) guarantees these properties by design, the software architect or engineer will need to choose the approach that brings the most convenient trade-off depending on the specific use case. A full white-box knowledge of the classifier and access to its internals pave the way for classifier-specific FCCs (i.e., safety wrappers) that can be very accurate and have the potential to add minimal overhead at runtime as they can be run in parallel while the classifier is performing inference. On the downside, they may require complex conceptualization, design, and implementation plus expensive sensitive analyses to fine-tune the overall mechanism. Another potential issue comes from the fact that a very fine-tuned approach may be very effective in dealing with known events and system states but may struggle when the behaviour of the system changes, even slightly.

V. Experimental Evaluation

This section describes the experimental campaign to quantify how the behaviour of tabular and image classifiers changes when FCCs are applied instead of typical classifiers for images (neural networks) or tabular data (ensembles of decision trees for the most part).

A. Experimental Methodology, Setup and Code

Our experiments are structured as follows. First, we choose a subset of FCCs to be used in our experimental evaluation: IP, OP, SW, see Section V.B. Then, we gather datasets for exercising tabular and image classifiers, spanning over a wide variety of classification tasks and simulating unknown data (Section V.C). The classifiers to be used for classification are described in Section V.D: once classifiers are defined and trained, Section V.E and Section V.F report results for tabular data and image classification, respectively, Section V.G conclude the experimental evaluation and highlight takeovers, and finally Section V.H highlights threats to validity. The performance of classifiers and FCCs is quantified via the metrics from Section III. Experiments have been performed on a server with Intel(R) Core (TM) i5-8350U CPU@1.7 GHz 1.9 GHz, using an NVIDIA Quadro RTX 5000 GPU. The code for repeating experiments is available in the anonymous GitHub at [68].

B. Selection of FCCs and Parameters

Some of the FCCs that are presented in Section IV cannot be instantiated in general settings. This is the case of the WT which, as a timer, depends on the typical inference time a classifier has on a specific software-hardware platform and with a specific workload. Results we get using this FCC may wildly change when repeating experiments in a different setup, thus we avoid it. Also, widely used ML algorithms for classification do not typically provide dedicated and custom ways for computing confidence in predictions, and cannot be used as SCCs.

Consequently, we instantiate IP, OP, SW FCCs as follows for image and tabular classifiers:

- Input processor (IP) uses a binary classifier to understand if the input belongs to the training data distribution or if it is unknown i.e., out of distribution or anomalous. When the input is predicted unknown, an omission is triggered. We use two input checkers for images and two input checkers for tabular data, as shown in Section V.D.
- Output processor (OP), where uncertainty is quantified by the *softmax* probability associated to the predicted class. If the probability does not exceed a threshold p_{thr}, the prediction is deemed uncertain. When p_{thr} is arbitrarily defined by the user instead of being classifier-specific, it is possible to end up having very different omission rates when using different classifiers. Thus, our experiments use a static $p_{thr} = 0.8$ threshold, but also find a dynamic p_{thr} (dp_{thr}) value corresponding to the 15^{th} percentile of the distribution of output probabilities of a classifier obtained on the validation set. This derives a classifier-dependent threshold value that makes the OP omit roughly 15% of the predictions across all classifiers. Since the threshold is computed on the validation set and not on the test set, the likelihood of omissions may still vary a bit.
- In our experiments, the confidence calculator of the Safety wrapper (SW) uses input checkers (as in IP) and probability checks (as in OP), omitting the prediction if

any of the two strategies trigger an omission. This leads to 4 different safety wrappers for each experiment, using p_{thr}=0.8 or dp_{thr} with any of the two input processors.

Overall, we will experiment with a total of 8 FCCs that can be created upon each classifier exercised in each dataset.

C. Datasets

1) Tabular Datasets

We select three tabular datasets belonging to different domains in which classifiers are typically willing to be applied: intrusion detection (CICIDS18 [65]), error detection (ARANCINO [67]), and control systems (MetroPT [66]). These datasets contain hundreds of thousands of data points corresponding to the behaviour of the system under normal operating conditions or due to: attack (six attack classes in CICIDS18), the manifestation of errors (nine errors in ARANCINO), control system failures (air and oil leak in MetroPT). For these datasets, we target a binary classification problem, aiming at distinguishing normal operating conditions against anomalies due to attacks, errors, or component failures. Similarly to image classifiers, we use a 50-20-30 train-validation-test split.

2) Image Datasets

Then, we select three image datasets: Flower (9 classes [62]), Fruit (24 classes [63]) and STL-10 (10 classes [64]). Out of the many alternatives for image classification, we favoured those since they are publicly available, belong to different domains, have a varying number of classes, and allow for fast experimentation times since they are composed of many small images (96x96 rgb at most) in the order of thousands or tens of thousands of images per dataset. We split each dataset using a 50-20-30 train-validation-test split.

3) Generation of Out-Of-Distribution Data

Public datasets are useful for experimentations, but may not generalize well to real scenarios that are prone to encounter unknown operating conditions, resulting in out-of-distribution inputs, different from those used for training the classifier. This typically makes the likelihood of misclassifications skyrocket, thus it is of utmost interest to simulate them for the purpose of our experiments.

We generate out-of-distribution images by applying three different alterations i.e. Rotation, Color Space and Gaussian Noise [61] to 30% of images from the test set i.e., unseen by the classifier in the training set. We create the Gaussian noise image by generating a noise map using *mean 0* and *st.d 25*, and then overlapping it to images. Colour space anomalies result from chaining operations as Brightness 0.5, contrast 1.5, saturation 1.5, and Hue adjustment at scale of 20. For rotation, we rotate the image of 90° left. These alterations were injected using the OpenCV library, and the parameters above are amongst the ones suggested in the handbook of such library.

The generation of out-of-distribution tabular data is not as straightforward as it happens with images. Fuzzing or adding random noise generate a new data point that may belong to the same distribution of the original data point, but may also fall into a different class. To overcome this problem, we remove specific classes of anomalous behaviors from the training set, letting them appear only in the test set, being unknown to the

Figure 2. Unknown image generation, from left to right a) original, b) gaussian noise, c) color space, d) rotation.

tabular classifier. For CICIDS18, *SSH-Bruteforce, FTP-BruteForce* and *Infiltration* attacks only appear in the test set (see Figure 3). For ARANCINO, errors in *the NodeRed, Redis* and *Arancino-manager* services only appear in the test set, whereas in MetroPT data of the *OilLeak* failure is removed from the train set as well.

This allows for building test sets that are composed of in-distribution (those from the original dataset) and out-of-distribution data, simulating the occurrence of unexpected operational conditions in real scenarios. The rate of out-of-distribution data ranges from 30% of the test set in the Fruit, Flower, STL-10 image datasets, to 20%, 12%, 8% for CICIDS18, MetroPT and ARANCINO tabular datasets, respectively.

D. Classifiers and Input Checkers

Tabular data classifiers are preferably built over ensembles of decision trees or statistical ML algorithms [46], [48], [75]. Thus, the binary tabular classifiers used in this study are *Random Forests (RF), XGBoost (XGB) and Logistic Regression (LR)*. As input checkers, we rely on two very fast tabular classifiers in a *Decision Tree* (DT) and *Linear Discriminant Analysis (LDA)*. All tabular classifiers are exercised using their default parameters from *scikit-learn*, and *xgboost* Python libraries [70]. Setting up input checkers allows to define 8 FCCs to be used for tabular data classification: IP_lda and IP_dt are input processors using LDA and DT as input checkers, OP_08 and OP_% are output processors using a static threshold p_{thr}=0.8 and using dp_{thr}=15%, while SW_lda_08, SW_lda_%, SW_dt_08, SW_dt_% are safety wrappers that combine IPs and OPs above.

For image classification, the literature acknowledges how neural networks are the preferred choice when dealing with unstructured data [32], [36] [51]. We chose AlexNet, ResNet50, and InceptionV3 as image classifiers given their wide usage in the last decade. The final models are obtained by transfer learning with learning rate = 0.001 and batch size = 16 from pre-trained DNNs using ImageNet weights stored in PyTorch. Input checkers for images (to be used in the IP

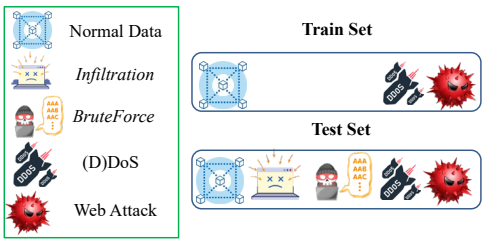


Figure 3. Unknown tabular data: the example of CICIDS18, where some attack classes are removed from the training set occurring only in the test set, being unknown to the classifier.

Table 2: Results of tabular clf and FCCs across tabular datasets used in the paper. For each dataset, clf and FCC we report the misclassifications ε (for clf) or ε_w (for FCC), omissions φ, φ_m *ratio* and ε *drop*. All data is in percentage.

Row	Datasets			CICIDS18					MetroPT					ARANCINO				
#	FCC	FCC tag	Clf	ε	ε_w	φ	ε drop	φ_m ratio	ε	ε_w	φ	ε drop	φ_m ratio	ε	ε_w	φ	ε drop	φ_m ratio
1	-	-	LR	24.47					15.02					13.90				
2	IP	IP_lda	LR		19.05	23.22	22.13	23.32		2.50	18.34	83.38	68.29		10.25	6.06	26.27	60.23
3	IP	IP_dt	LR		19.09	23.19	21.97	23.18		2.56	12.53	82.93	99.39		9.10	7.99	34.51	60.01
4	OP	OP_%	LR		18.80	11.32	23.16	50.08		4.41	21.01	70.66	50.52		9.64	15.82	30.63	26.91
5	OP	OP_08	LR		1.70	62.12	93.06	36.65		13.24	5.61	11.87	31.76		9.65	10.02	30.60	42.46
6	SW	SW_lda_%	LR		13.40	34.48	45.21	32.08		0.13	27.77	99.16	53.62		7.61	18.27	45.26	34.44
7	SW	SW_dt_%	LR		13.48	34.38	44.90	31.95		0.16	25.28	98.97	58.80		5.94	21.36	57.30	37.28
8	SW	SW_lda_08	LR		1.08	76.10	95.59	30.73		1.24	23.09	91.74	59.67		7.65	13.93	44.93	44.84
9	SW	SW_dt_08	LR		1.42	72.64	94.20	31.73		1.34	17.58	91.06	77.81		5.77	16.63	58.46	48.86
10	-	-	RF	12.33					12.57					6.34				
11	IP	IP_lda	RF		3.47	23.22	71.88	38.19		0.20	18.34	98.38	67.44		4.55	6.06	28.23	29.53
12	IP	IP_dt	RF		2.38	23.14	80.73	43.03		0.12	12.53	99.01	99.37		2.54	7.92	59.97	48.04
13	OP	OP_%	RF		3.90	9.71	68.36	86.88		11.44	3.83	9.03	29.64		4.58	10.09	27.75	17.44
14	OP	OP_08	RF		12.31	0.12	0.23	24.29		12.57	0.05	0.03	7.69		5.62	2.62	11.33	27.42
15	SW	SW_lda_%	RF		3.18	24.70	74.19	37.04		0.11	20.77	99.16	60.03		3.37	15.10	46.86	19.68
16	SW	SW_dt_%	RF		2.25	24.45	81.80	41.26		0.10	15.25	99.22	81.81		1.64	16.44	74.20	28.62
17	SW	SW_lda_08	RF		3.45	23.33	72.07	38.11		0.20	18.39	98.41	67.29		3.91	8.47	38.40	28.74
18	SW	SW_dt_08	RF		2.36	23.22	80.84	42.94		0.12	12.58	99.02	98.99		1.97	10.31	68.87	42.35
19	-	-	XGB	12.32					12.57					6.99				
20	IP	IP_lda	XGB		3.46	23.22	71.95	38.19		0.20	18.34	98.38	67.44		5.20	6.06	25.66	29.60
21	IP	IP_dt	XGB		2.36	23.15	80.83	43.02		0.13	12.51	98.95	99.42		3.19	7.98	54.41	47.71
22	OP	OP_%	XGB		7.60	15.19	38.36	31.11		11.43	12.79	9.06	8.91		4.90	14.18	29.90	14.74
23	OP	OP_08	XGB		12.31	0.02	0.11	57.14		12.57	0.01	0.02	16.67		5.97	3.06	14.61	33.36
24	SW	SW_lda_%	XGB		3.05	33.81	75.27	27.44		0.05	28.57	99.59	43.82		3.16	20.03	54.87	19.16
25	SW	SW_dt_%	XGB		2.13	32.50	82.73	31.36		0.12	24.17	99.04	51.52		1.61	21.02	76.94	25.59
26	SW	SW_lda_08	XGB		3.44	23.24	72.05	38.21		0.20	18.35	98.40	67.41		4.19	9.10	40.08	30.82
27	SW	SW_dt_08	XGB		2.36	23.15	80.85	43.02		0.13	12.52	98.96	99.34		2.30	10.83	67.07	43.32

FCC) could use DNNs as well, but Support Vector Machines (SVMs) were successfully applied as well [68]: thus, input checkers for image datasets are either SVMs or a ResNet50 different from the one used for image classification. Similarly to tabular classifiers, we end up with the following 8 FCCs for image classification: IP_svm, IP_rn, OP_08, OP_%, SW_svm_08, SW_svm_%, SW_rn_08, SW_rn_%.

E. Results: Tabular Data Classification

We start commenting results of classifiers and FCCs using tabular datasets with the aid of Table 2, which reports the percentage of misclassifications ε (for clf) or ε_w (for FCC), omissions φ, ε *drop* and φ_m *ratio* for different datasets and different classifiers. The table has 27 lines, 9 (base *clf* plus 8 FCCs) for each of the three classifiers LR, RF, XGB. The columns for φ_m *ratio* and ε *drop* are painted with a gradient of green that gets darker the more these two metrics have optimal result (the higher, the better).

Reading the table for dataset blocks (i.e., groups of 4 columns), we can observe the following. In the CICIDS18 dataset, the misclassifications of LR, RF, and XGB are respectively at 24.47 (1st row), 12.33 (10th row), and 12.32 (19th row). Applying FCCs always lowers misclassifications, at a cost of a specific amount of omissions φ. For example, misclassifications of the RF may drop to 2.25 using the safety wrapper SW_dt_% (16th row of Table 2), at a cost of 24.45% of omissions. Roughly, we are reducing misclassification by a factor of 5, but 25% of the predictions of the FCC will be rejected, omitting the output. This is because only 41.26% (φ_m *ratio*) of omissions correspond to misclassifications, or the remaining 58.74% of omissions would have been correct predictions by the classifier. This is far from optimal, as it means that the price for lowering misclassifications may be too high in terms of accuracy degradation.

Results related to the MetroPT datasets, reported in the columns in the middle of the table, offer a different example. In this case, and for all *clf* LR, RF, XGB, employing an IP that uses DT as input checker (i.e., IP_dt) allows for omitting almost all (high ε *drop*) and only (high φ_m *ratio*) prediction that would have been misclassifications. This is the optimal scenario in which the application of the FCC brings misclassifications of XGB from 12.57% to 0.13%, with an ε *drop* of almost 99% (see 21st row of the table, 8th to 11th column). In other words, the residual misclassifications are lowered by a factor of 100, and there are just a few omissions of correct predictions i.e., φ_m *ratio* is 99.37%, very close to the optimum 100. On the extreme right of the table, we see results for the ARANCINO dataset. Here, we see that FCCs can significantly lower the number of misclassifications, but typically show non-optimal performance as they either omit an exceedingly high amount of predictions (high φ and low φ_m *ratio*) in the process.

Other important information could be obtained by reading the table horizontally. First, the OP using a static threshold for probabilities (OP_08) delivers the worst result overall: the threshold is either too low (thus omissions are almost non-existent $\varphi \approx 0$) or too high, delivering an obnoxious omission probability as for LR in CICIDS18, see 5th row 5th column of Table 2. Using an OP that has a dynamic threshold delivers a more balanced result as omissions are usually of a reasonable amount. The benefits of using IP are situational: there are cases in which it is game-changing as in the MetroPT dataset,

but there are also cases in which it does not reduce misclassifications by much (i.e., ARANCINO dataset), or where it omits many correct predictions in the process, as quantified by the low φ_m *ratio* in the CICIDS18 dataset. Safety wrappers SW are meant to omit predictions if either the input or the output check highlight issues, thus they always have high omission rates, at the benefit of having very low residual misclassifications.

F. Results: Image Classification

Results similar to those presented above can be obtained also for image classifiers. To avoid being tedious, this section follows a different structure, focusing on three cases: the ResNet50 *clf* and FCCs on the Flower dataset (Figure 4), AlexNet on STL-10 (Figure 5) and InceptionV3 on Fruits (Figure 6). These scenarios offer interesting discussion items that we explore as follows.

Figure 4 shows a 10-bar chart: a bar for the ResNet50 *clf*, a bar for each FCC, and a bar showing the theoretical optimum FCC that omits all and only misclassifications, paired with *ε drop* and φ_m ratio quantities on the right. From a visual standpoint, the aim is to reduce the red bar (ε if *clf* or ε_w if FCC) as much as possible, keeping the blue bar (α or α_w) untouched, replacing the red area with the yellow-striped omissions φ. Whereas some FCCs succeed in reducing the red bar i.e., OP_%, SW_svm_% and SW_rn_%, they also have a shorter blue bar: this is due to a sub-optimal φ_m ratio, that is 57.84, 36.06 and 21.76, respectively. In this case, there is no FCC that has both an high φ_m ratio and ε drop: all reduce misclassifications, but with a major price to pay in terms of unnecessary omissions.

Figure 5 has the same structure of Figure 4, but refers to the application of AlexNet as image classifier in the STL-10 dataset. This is a case in which the AlexNet *clf* already has low (59.23%) accuracy, generating a whopping 40.77% of misclassifications. FCCs are able to reduce this amount of misclassifications by a fair amount but typically omit many correct classifications in the process. The *SW_rn_%* FCC has $\varepsilon_w = 13.9\%$, roughly a third of those of AlexNet alone, but also has an accuracy of 33.68%, which is almost half than those of *clf*. A more balanced performance is offered by OP_%, which roughly halves misclassifications and reduces accuracy from 59.23 to 50.84%, limiting the amount of omissions: it has better φ_m ratio than *SW_rn_%*.

Lastly, Figure 6 shows the application of InceptionV3 on the Fruit dataset. Similarly to Figure 4, the OP_% FCC is the solution that offers the best tradeoff between reducing misclassifications (see and ε drop) and having a reasonably low amount of unnecessary omissions (φ_m ratio). IPs have very bad performance in this case, making also SWs lean towards poor performance as they can reduce up to ε drop = 85.55% of misclassifications, but scoring an exceedingly low φ_m ratio of 25.04, with only 1 in 4 omissions corresponding to misclassifications.

G. Takeovers

Our experimental analysis is only preliminary: however, it provides some key information that we summarize as follows:

- The OP adds virtually no overhead to the process as it just computes basic thresholding on probabilities of predictions and is typically helpful in suspecting most of the misclassifications.
- The IP proved to be more or less useful depending on the scenario, as it may also end up having quite low *φ_m ratio* (i.e., many unnecessary omissions) when unknown data is not easily distinguishable from in-distribution data or when the classifier would have been robust enough to correctly classify even unknown data. Differently from OP, it requires training the input checker and exercising it before or alongside the main classifier.
- In our experiments, the SW is the solution that suspects the most misclassifications (i.e., has the best *ε drop)* due to the implementation of the confidence calculator, which combines input and output checks.
- The performance of FCCs depends on several factors, including the nature of the system, the characteristics of the input data, and their specific implementation. Only a careful choice of uncertainty calculators (e.g., input checkers for IP, probability thresholds for OP) could pave the way for a successful application.
- Researchers interested in lowering misclassifications should primarily focus on maximising the *ε drop*, but should also make sure that the *φ_m ratio* is high enough.

H. Threats to Validity and Reprobucibility

Classifiers have hyperparameters whose tuning critically affects results, or may encounter 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). However, this experimental evaluation aims to compare the performance of classifiers

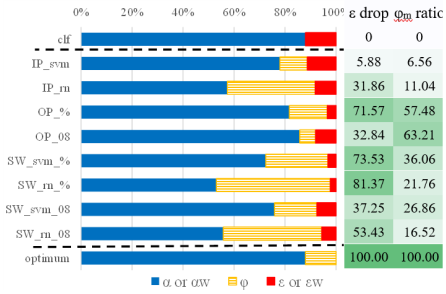


Figure 4. ResNet50 clf, FCCs and the optimum FCC for Flower dataset.

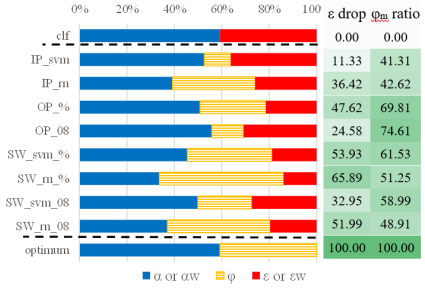


Figure 5. AlexNet clf, FCCs and the optimum FCC for STL-10 dataset.

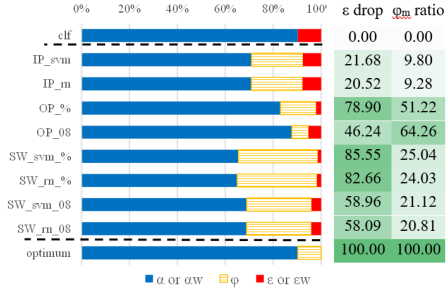


Figure 6. InceptionV3 clf, FCCs and the optimum FCC for Fruit dataset.

against FCCs, thus even a sub-optimal choice of hyperparameters is more than acceptable provided that the same setup is used throughout all experiments.

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. We publicly shared scripts, methodologies, and all metric scores, allowing any researcher or practitioner to repeat the experiments. We do not use any private dataset: all datasets are referenced in the papers, and all code is available at [68].

VI. What about Availability?

These FCCs allow for reducing misclassifications thanks to the omission mechanisms, which has an obvious downside: whenever FCC omits many correct predictions alongside with misclassifications, it becomes almost unusable as it hardly provides a beneficial behaviour for the encompassing system. To address this problem, we report here other approaches that may be used to reduce misclassifications and at the same time keep unnecessary omissions as low as possible.

A. Recovery Blocks

Recovery blocks are known since many decades as one of the strategies for reliable software design [41]. Practically speaking, we foresee the usage of a set of *m* alternative implementations of a function that are called sequentially whenever the output of the main function is deemed as non-trustable. For classifiers, this translates into calling a sequence of at most *m+1* classifiers before obtaining the prediction, or omitting the result if none of the recovery blocks is confident enough in their output. This concept is not entirely new in the machine learning domain: delegating classifiers [43] define a group of classifiers, each specialized to be confident in the analysis of a subset of the input space, choosing the classifier to use for inference depending on the input alone.

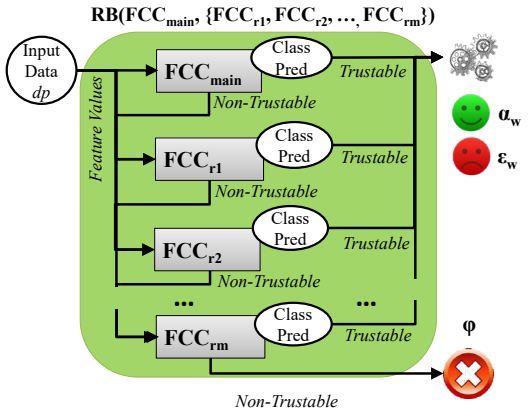
A FCC based on recovery blocks does not use a single classifier for inference: in fact, it delegates the decision to the first replica whose output is trustable (see Figure 7a). The amount and the diversity of replicas has a direct impact on the likelihood of omissions: it will be easier to find a "trustable" replica if the set of replicas is wide and diverse instead of relying on a few replicas. Note that such FCC may add a major overhead to the inference process as – in the worst case – it may require exercising *m+1* classifiers in sequence.

B. N-version Programmung

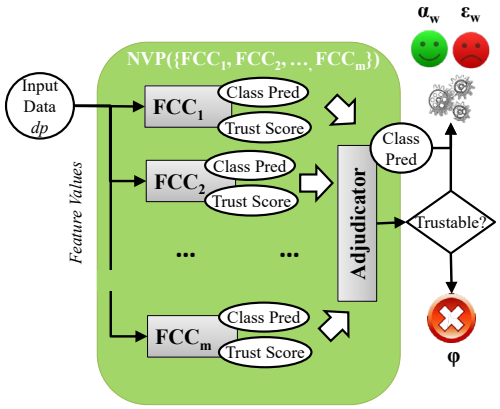
Another approach relies on N-Version Programming (NVP), or exercising different classifiers in parallel [41], [44] each acting independently but processing the same inputs. Their predictions, alongside with trust scores, are sent to the adjudicator, which is a function that takes as input the predictions and the trust scores of the m classifiers, generates a prediction and a trust score to decide if the prediction has to be omitted or if it is trustworthy. For a problem of *k*-class classification and *m* replicas, the adjudicator is a function that has *(k+1)m* floating point inputs and outputs *k* probabilities plus a floating point trust score. This adjudicator [41] can be implemented with thresholds, invariants, custom rules [44], or can be a classifier itself, providing many degrees of freedom in finding the ideal function to combine outputs and trust scores of the *m* replicas. A FCC using NVP builds a two-layer architecture which is often referred to as stacking, with *m* classifiers at the base level, and the adjudicator at the meta-level [45], see Figure 7b. Those two steps are necessarily sequential, cannot be parallelized and may, again, add a relevant overhead to the inference process.

C. Other Notable Approaches

Decades of critical systems engineering and system-level thinking originated more architectures [41], [44] than those shown above. Voting was explored in different formulations (i.e., hard, soft, weighted), and can be even used to build a hierarchical agreement structure that is known as n-self-checking-programming [41]. Boosting techniques were proven to be very effective for classifying known [18], [48] and unknown tabular data points, but boosting ensembles of DNNs for image classification are not yet a thing and are still in their early stages [49]. Interestingly, different architectures



a) Recovery Blocks (RB) or delegating FCC, which pairs the main FCC with other FCCs that are executed in sequence, seeking for the first that outputs a confident prediction.



b) NVP or FCC ensembles, which exercise many FCCs in parallel and then use a final adjudication function to decide on the confidence in the prediction.

Figure 7. Software Architectures for Complex FCCs with accuracy α_w, misclassification probability ε_w, and omissions φ.

may be combined into unified architectures. For example, there are works that pair classifiers based on ensembles with a monitor to check for trustworthiness [20], but this is still quite an uncharted territory in which there are no widespread and solid proposals (yet).

VII. Conclusions, Viewpoints and Future Works

This paper motivated the need for a paradigm shift when applying classifiers based on Machine Learning (ML) in critical systems, where misclassifications may result in catastrophic consequences. Instead of chasing the holy grail of perfect accuracy, we argue that misclassification may be acceptable if i) they could be suspected and are of a reasonably low amount, and ii) the system can promptly react when predictions are rejected due to suspected misclassifications. This is the baseline of Fail-Controlled Classifiers (FCCs), or rather classifiers that may omit predictions they are not confident with. Ideally, a FCC either answers correctly, or does not answer at all, thus having no misclassifications even without perfect accuracy. FCCs build on the following pillars:

- **Better Safe than Sorry**. There are domains in which it is beneficial to answer questions or provide service only when the output could justifiably be trusted. In any other case, it is better to omit answers: betting on the "most likely" answer is not an option.
- **Confidence, not over-confidence**. Self-awareness is always a desirable property: without it, people, equipment, or controllers are hardly trustable. For ML classifiers, this translates into computing confidence in a prediction, and using this quantity to decide if the prediction could be propagated to the encompassing critical system.
- **Old but gold?** Critical system and software architectures that are known and applied since decades for safety engineering may be successfully reworked to design software components that (at least partially) rely on ML classifiers for a wide variety of tasks.

We believe that this paper will bring awareness to the problem of deploying ML classifiers in critical systems. Also, the ideas, the architectures, and the discussions presented herein will pave the way for further experiments aimed at a quantitative assessment of FCCs in real or simulated systems, to precisely understand their applicability and focus on the gap that still must be bridged to safely put ML classifiers into operation.

Acknowledgments

This work was supported in part by the 202297YF75 PRIN 2022 project S2, by the project SERICS (PE00000014) under the MUR National Recovery and Resilience Plan funded by the European Union – NextGenerationEU, and by the RDS - PTR22-24 P2.1 Cybersecurity project funded within the Ricerca Sistema Elettrico, Piano Triennale 22-24.

References

[1] Tiwari, A., et. al. (2014, April). Safety wrapper for security. In Proceedings of the 3rd international conference on High confidence networked systems (pp. 85-94).

[2] Lakshminarayanan, B., Pritzel, A., & Blundell, C. (2017). Simple and scalable predictive uncertainty estimation using deep ensembles. Advances in neural information processing systems, 30.

[3] 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.

[4] Rossolini, G., Biondi, A., & Buttazzo, G. (2022). Increasing the confidence of deep neural networks by coverage analysis. IEEE Transactions on Software Engineering, 49(2), 802-815.

[5] 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.

[6] Guérin, J., Ferreira, R. S., Delmas, K., & Guiochet, J. (2022, October). Unifying evaluation of machine learning safety monitors. In 2022 IEEE 33rd International Symposium on Software Reliability Engineering (ISSRE) (pp. 414-422). IEEE.

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

[8] 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.

[9] Meeker, W. Q., et. al. (2017). Statistical intervals: a guide for practitioners and researchers (Vol. 541). John Wiley & Sons.

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

[11] Hendrycks, D., & Gimpel, K. (2016, November). A Baseline for Detecting Misclassified and Out-of-Distribution Examples in Neural Networks. In International Conference on Learning Representations.

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

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

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

[15] 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).

[16] Lever, J. (2016). Classification evaluation: It is important to understand both what a classification metric expresses and what it hides. Nature methods, 13(8), 603-605.

[17] Chen, T., & Guestrin, C. (2016, August). Xgboost: A scalable tree boosting system. In Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining (pp. 785-794).

[18] Zoppi, T., Ceccarelli, A., & Bondavalli, A. (2023). Ensembling Uncertainty Measures to Improve Safety of Black-Box Classifiers. In proceedings at the 26th European Conference on Artificial Intelligence (October, 2023).

[19] Biondi, A., et. al. (2019). A safe, secure, and predictable software architecture for deep learning in safety-critical systems. IEEE Embedded Systems Letters, 12(3), 78-82.

[20] Calzavara, S., Cazzaro, L., Lucchese, C., Marcuzzi, F., & Orlando, S. (2022). Beyond robustness: Resilience verification of tree-based classifiers. Computers & Security, 121, 102843.

[21] Su, J., Zhang, Z., Wu, P., Li, X., & Zhang, J. (2022, October). Adversarial input detection based on critical transformation robustness. In 2022 IEEE 33rd International Symposium on Software Reliability Engineering (ISSRE) (pp. 390-401). IEEE.

[22] Chandola, V., Banerjee, A., & Kumar, V. (2009). Anomaly detection: A survey. ACM computing surveys (CSUR), 41(3), 1-58.

[23] Ma, M., Zhang, S., Pei, D., Huang, X., & Dai, H. (2018, October). Robust and rapid adaption for concept drift in software system anomaly detection. In 2018 IEEE 29th International Symposium on Software Reliability Engineering (ISSRE) (pp. 13-24). IEEE.

[24] Lee, K., Lee, K., Lee, H., & Shin, J. (2018). A simple unified framework for detecting out-of-distribution samples and adversarial attacks. Advances in neural information processing systems, 31.

[25] Sommer, R., & Paxson, V. (2010, May). Outside the closed world: On using machine learning for network intrusion detection. In 2010 IEEE symposium on security and privacy (pp. 305-316). IEEE.

[26] Zhou, Y. (2022). Rethinking reconstruction autoencoder-based out-of-distribution detection. In Proc. of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (pp. 7379-7387).

[27] Biggio, B., Nelson, B., & Laskov, P. (2012). Poisoning attacks against support vector machines. arXiv preprint arXiv:1206.6389.

[28] Brown, T. B., Mané, D., Roy, A., Abadi, M., & Gilmer, J. (2017). Adversarial patch. arXiv preprint arXiv:1712.09665.

[29] Carlini, N., & Wagner, D. (2017, May). Towards evaluating the robustness of neural networks. In 2017 ieee symposium on security and privacy (sp) (pp. 39-57). Ieee.

[30] Mirza, M. J., Buerkle, C., Jarquin, J., Opitz, M., Oboril, F., Scholl, K. U., & Bischof, H. (2021, September). Robustness of object detectors in degrading weather conditions. In 2021 International Intelligent Transportation Systems Conference (ITSC) (pp. 2719-2724). IEEE.

[31] Koh, P. W., et. al. (2021, July). Wilds: A benchmark of in-the-wild distribution shifts. In International Conference on Machine Learning (pp. 5637-5664). PMLR.

[32] Lopez, I., et. al. (2016, August). Exploiting redundancy and path diversity for railway signalling resiliency. In IEEE Int. conference on intelligent rail transportation (ICIRT) (pp. 432-439). IEEE.

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

[34] Keller, C. G., et. al. (2011). Active pedestrian safety by automatic braking and evasive steering. IEEE Transactions on Intelligent Transportation Systems, 12(4), 1292-1304

[35] Ami, A. S., et. al. (2023, October). "False negative-that one is going to kill you."-Understanding Industry Perspectives of Static Analysis based Security Testing. In 2024 IEEE Symposium on Security and Privacy (SP) (pp. 19-19). IEEE Computer Society.

[36] Ceccarelli, A., & Secci, F. (2022). RGB cameras failures and their effects in autonomous driving applications. IEEE Transactions on Dependable and Secure Computing.

[37] McAllister, D. F., & Vouk, M. A. (1996). Fault-tolerant software reliability engineering. Handbook of Software Reliability Engineering, 567-614.

[38] Randell, B., & Xu, J. (1995). The evolution of the recovery block concept. Software fault tolerance, 3, 1-22.

[39] Ferri, C., Flach, P., & Hernández-Orallo, J. (2004, July). Delegating classifiers. In Proceedings of the twenty-first international conference on Machine learning (p. 37).

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

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

[42] Shwartz-Ziv, R., & Armon, A. (2022). Tabular data: Deep learning is not all you need. Information Fusion, 81, 84-90.

[43] Grinsztajn, L., Oyallon, E., & Varoquaux, G. (2022). Why do tree-based models still outperform deep learning on tabular data? arXiv preprint arXiv:2207.08815.

[44] Ganaie, M. A., Hu, M., Malik, A. K., Tanveer, M., & Suganthan, P. N. (2022). Ensemble deep learning: A review. Engineering Applications of Artificial Intelligence, 115, 105151.

[45] Sarıkaya, A., Kılıç, B. G., & Demirci, M. (2023). RAIDS: Robust Autoencoder-Based Intrusion Detection System Model Against Adversarial Attacks. Computers & Security, 103483.

[46] Mathias, M., et. al. (2013, August). Traffic sign recognition—How far are we from the solution?. In The 2013 international joint conference on Neural networks (IJCNN) (pp. 1-8). IEEE.

[47] Miremadi, S. G., et. al. (1992, July). Two Software Techniques for On-line Error Detection. In FTCS (pp. 328-335).

[48] David, I., Ginosar, R., & Yoeli, M. (1995). Self-timed is self-checking. Journal of Electronic Testing, 6, 219-228.

[49] Ocheretny, V. (2010, July). Self-checking arithmetic logic unit with duplicated outputs. In 2010 IEEE 16th International On-Line Testing Symposium (pp. 202-203). IEEE.

[50] Psarakis, M., Gizopoulos, D., Sanchez, E., & Reorda, M. S. (2010). Microprocessor software-based self-testing. IEEE Design & Test of Computers, 27(3), 4-19.

[51] Brau, F., Rossolini, G., Biondi, A., & Buttazzo, G. (2023, June). Robust-by-design classification via unitary-gradient neural networks. In Proceedings of the AAAI Conference on Artificial Intelligence (Vol. 37, No. 12, pp. 14729-14737).

[52] Gharib, M., et. al. (2022). On the properness of incorporating binary classification machine learning algorithms into safety-critical systems. IEEE Transactions on Emerging Topics in Computing, 10(4), 1671-1686.

[53] Powell, D., Chérèque, M., & Drackley, D. (1991). Fault-tolerance in Delta-4. ACM SIGOPS Operating Systems Review, 25(2), 122-125.

[54] Shorten, C., & Khoshgoftaar, T. M. (2019). A survey on image data augmentation for deep learning. Journal of big data, 6(1), 1-48.

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

[56] "Fruits-360." Accessed: Mar. 26, 2024. [Online]. Available: https://www.kaggle.com/datasets/moltean/fruits

[57] "STL-10 dataset." Accessed: Mar. 26, 2024. [Online]. Available: https://cs.stanford.edu/~acoates/stl10/

[58] W. Haider, J. Hu, J. Slay, B. P. Turnbull, and Y. Xie, "Generating realistic intrusion detection system dataset based on fuzzy qualitative modeling," Journal of Network and Computer Applications, vol. 87, pp. 185–192, Jun. 2017.

[59] N. Davari, B. Veloso, R. P. Ribeiro, P. M. Pereira, and J. Gama, "Predictive maintenance based on anomaly detection using deep learning for air production unit in the railway industry," in 2021 IEEE 8th International Conference on Data Science and Advanced Analytics (DSAA), IEEE, Oct. 2021, pp. 1–10.

[60] Zoppi, T., Merlino, G., Ceccarelli, A., Puliafito, A., & Bondavalli, A. (2023, October). Anomaly Detectors for Self-Aware Edge and IoT Devices. In 2023 IEEE 23rd International Conference on Software Quality, Reliability, and Security (QRS) (pp. 24-35). IEEE.

[61] 'ciperlab-813E', Accessed: May, 03, 2024. [Online]. Available: https://anonymous.4open.science/r/BetterSafeThanSorry-190C/README.md

[62] Scikit-Learn Python Library – Classifiers Accessed: Apr. 30, 2024. [Online] https://scikit-learn.org/stable/supervised_learning.html

[63] Zhang, X. Y., Xie, G. S., Li, X., Mei, T., & Liu, C. L. (2023). A survey on learning to reject. Proceedings of the IEEE, 111(2), 185-215.

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

[65] Baye, G., Silva, P., Broggi, A., Fiondella, L., Bastian, N. D., & Kul, G. (2023, May). Performance analysis of deep-learning based open set recognition algorithms for network intrusion detection systems. In NOMS 2023-2023 IEEE/IFIP Network Operations and Management Symposium (pp. 1-6). IEEE.

[66] 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.