Software: Practice and Experience

WILEY

SPECIAL ISSUE PAPER OPEN ACCESS

Fail-Controlled Classifiers: A Swiss-Army Knife Toward Trustworthy Systems

Fahad Ahmed Khokhar[1] | Tommaso Zoppi[1] | Andrea Ceccarelli[1] | Leonardo Montecchi[2] | Andrea Bondavalli[1]

[1]Department of Mathematics and Informatics, University of Florence, Florence, Italy | [2]Department of Computer Science, Norwegian University of Science and Tech, Trondheim, Norway

Correspondence: Tommaso Zoppi (tommaso.zoppi@unifi.it)

Received: 15 January 2025 | **Revised:** 2 September 2025 | **Accepted:** 26 October 2025

Funding: This work was supported by Ministero dell'Istruzione, dell'Università e della Ricerca and European Commission.

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

ABSTRACT

Background: Modern critical systems often require to take decisions and classify data and scenarios autonomously without having detrimental effects on people, infrastructures or the environment, ensuring desired dependability attributes. Researchers typically strive to craft classifiers with perfect accuracy, which should be always correct and as such never threaten the encompassing system. Unfortunately, this is a very unrealistic goal, as classification tasks are typically complex and may encounter a wide variety of unexpected operating conditions and unknown inputs.

Methods: Classifiers should be considered as building blocks that interact with other components that help rejecting those predictions that are suspected to be misclassifications, triggering system-level mitigation strategies instead. Fail-Controlled Classifiers (FCCs) are software components that can either correctly classify, misclassify, or reject outputs: ideally, they would reject all and only outputs that correspond to misclassifications. Nine different FCCs are presented: Self-Checking Classifiers (SCC), Watchdog Timers (WT), Input Processor (IP), Output processor (OP), Safety Wrapper (SW), Recovery Blocks (RB), weighted and non-weighted Voting (VT, WVT) and Stacking (STK).

Results: These 9 FCCs are instantiated in experiments with tabular and image classifiers, showing their potential in rejecting most misclassifications and paving the ways for trustworthy decisions to be deployed in critical systems. If the system can tolerate more omissions, the IP FCC is a good choice. On the other hand, if achieving the highest accuracy is the priority, RB FCC performs better.

Conclusions: Findings show that FCCs do not primarily aim at improving correct classifications, but allow for transforming many misclassifications into rejections, which may be easily handled by the encompassing system and paving the way for trustworthy decisions to be deployed in critical systems.

1 | Introduction

"If you can't say something nice, don't say nothing at all" tells *Thumper* the rabbit to *Bambi* in the famous, 80-year-old Disney cartoon. This small rabbit teaches us an important lesson. There are cases in which rejecting 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

school 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 rejections has to be reasonably low: a decision-making entity that always rejects 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 rejects all and only wrong answers. This is substantially different from crafting decision-making entities that are always correct, which is usually an unrealistic expectation.

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 [1–5]. Moreover, classifiers can perform high-quality classification of images, which is of paramount importance for obstacle detection [6, 7] and traffic sign recognition [8] for autonomous driving, or even for webcams (edge computing) and related components (cloud/fog computing, and other standalone or centralized architectures) with image quality checks, accurate access control, or even for classifying diseases in the medical domain [9]. In the last decade, academia, industry, and national governments have 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, that is, a *misclassification*.

This 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 and protocols to handle uncertain predictions that are suspected to be misclassifications [10]. For example, in autonomous vehicles, a classifier detecting road signs does not 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 rejecting 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 reviewing the existing literature, we motivate the need for FCCs, motivating their importance with respect to key concepts such as dependability and trust. This allows formulating 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), Recovery Blocks (RB), weighted and non-weighted Voting (VT, WVT) and Stacking (STK) software architectures to build Fail-Controlled Classifiers (FCCs), discussing possible variants due to implementation and design choices. These are evaluated for tabular data and image classification, computing metrics and aiming at both low misclassifications and low probability of rejections. Findings show that FCCs do primarily aim at improving correct classifications, but allow for transforming many misclassifications into rejections, which may be easily handled by the encompassing system. The code for repeating experiments is available in GitHub at [11].

The paper is structured as follows. Section 2 reports the background of the study, while Section 3 motivates FCCs, which are then presented in Sections 4 and 5. Sections 6 and 7 present and discuss the experimental results of FCCs, letting Section 8 list threats to validity and conclude the work, by summarizing the main takeaways.

2 | Background and Related Works

2.1 | 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 [12] 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 [13–16], despite alternatives based on DNNs existing [2]. Conversely, image classification employs DNNs, which can learn strong features from pixel maps [17, 18].

2.2 | Confidence of Classifiers

Classifiers may exhibit high confidence even when misclassifying data points; for example, "neural networks which yield a piecewise linear classifier function [. . .] produce almost always high confidence predictions far away from the training data" [19]. This issue highlights the central challenge: ideally, classifiers should be highly confident about predictions that are correct and show low confidence only for those that are misclassifications. 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 [20]. Researchers and practitioners are actively investigating ways to quantify uncertainty and learning to reject [21] misclassifications. Some approaches rely on statistical measures such as confidence intervals [22] or the Bayes' theorem [23]. Works such as [24] 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 [18], the 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 [20, 25], where authors paired a k-Nearest Neighbor classifier with a neural network to compute uncertainty. The work [7] 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. Authors of [26] use probabilistic neural networks to model predictive distributions and, as a result, quantify predictive uncertainty using methods such as adversarial training. In [27], the 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 [28] 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, the authors of [3] combine a voting strategy with a safety monitor to build a safe and secure classifier for application in embedded systems. While these methods provide valuable advances, none fully solve the problem of misplaced confidence, underscoring the need for Fail-Controlled Classifiers (FCCs) that explicitly incorporate mechanisms to detect and handle untrustworthy predictions.

2.3 | Unknown Inputs: Threats to Classification

Tackling very complex problems naturally exposes classifiers to a high probability of misclassifications, which can be reduced but not avoided at all. The sub-optimal choice of 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 [29], which may expose classifiers to unknown, unexpected inputs. There is no universally accepted taxonomy of threats [30]. In this paper, we structure the discussion around three widely recognized categories—out-of-distribution data, adversarial attacks, and distributional shifts—while treating the simulation of unknown inputs as a mitigation strategy rather than as a new threat category.

2.3.1 | Out-of-Distribution Data, Anomalies, and Outliers

Systems and software components may encounter anomalous inputs or operating conditions [4, 18, 29, 31] 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 behavior of the classifier may become unpredictable [18] 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 [18, 31, 32].

2.3.2 | Adversarial Attacks

Second, classifiers may operate in situations in which malicious entities may be willing to actively disturb the behavior 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 [33], adversarial patches [34], and gradient-based attacks [35]. 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 [31, 36], but nothing that can be considered proven-in-use yet.

2.3.3 | Distribution Shifts, Emerging and Unexpected Events

Third, ML often works under the *Independent and Identically Distributed* (IID) or "closed world" assumption [37]. In a closed world, train, validation and test data are independently and randomly sampled from the same underlying distribution. However, most (if not all) of 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 [37] 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 behaviors, and even changes in the threat landscape [6, 38].

2.4 | On Simulating Unknown Inputs

Unknown inputs are intrinsically unpredictable in their entirety: however, there is a growing need to design and evaluate techniques whose behavior is stable even when processing unknowns. To do that, unknown inputs have to be simulated and used when testing research proposals and artifacts. When a testbed of the case study is available, these unknown inputs can be simulated or obtained after monitoring campaigns in which the operational environment is different from that used for training the classifier. Examples include, but are not limited to: collecting images during specific weather events, monitoring hardware in untested temperature and humidity setups, crafting and exercising novel attacks, and simulating software bugs.

When only a dataset is available, unknown inputs may be simulated or generated. One approach is to remove specific classes of anomalous behaviors from the training set, letting them appear only in the test set, being unknown to the classifier. Then, if the classifier is a binary intrusion detector trained with normal data and with data about three attacks (say *a1*, *a2*, *a3*), an unknown input can be simulated by providing data about a novel attack (say

a4) only during testing. Here, *a4* is not a real zero-day attack (it is logged in the existing dataset), but it is a zero-day to the classifier.

The generation of in-distribution and out-of-distribution data is typically up to Generative Adversarial Networks (GANs), which can learn how to generate samples on the tail of a (known) data distribution. Often, the approach includes an autoencoder, which aims to reconstruct the original input, with the reconstruction error used as a way to understand if the generated data complies with the user needs, that is, if it belongs or not to a target distribution [39]. Another approach is applying gradient-descend techniques as Fast Gradient Signed Method (FGSM) [40], which introduces adversarial perturbations by adding a scaled sign of the gradient to the input. Other options rely on statistical methods: a Soft Brownian Offset [41] defines an iterative approach to translate a point $x \in X$ by a most likely distance d away from the distribution X, and as such can be used for generating out-of-distribution inputs as follows. Suitable representations of an in-distribution dataset are found through an encoder; then, those are transformed via the Soft Brownian Offset and then decoded into the original data shape.

2.5 | Replicas, Redundancy, Diversity, and Adjudication

Redundancy is a key principle in fault-tolerant systems, ensuring continuous operation by adding extra components or systems as backups in case of failure [42]. N-version Programming (NVP), or software redundancy, runs multiple replicas of software on diverse hardware, operating systems or diverse versions of the software itself. Diversity plays a major role also in ML-based software and is a staple for ensemble learning. Diversity in ML can focus on three main areas: *data diversification*, *model diversification*, and *inference diversification* [43]. Employing different training data potentially makes for learning diverse ML models. Model diversity, on the other hand, promotes variety in how models are structured or combined, which reduces repetition and enhances their ability to represent complex systems. Inference diversification allows models to consider multiple plausible outcomes, and is often used in object detection scenarios.

An important part of NVP or ensembles is to combine outputs from replicas or versions, which is typically done via an adjudicator [44], or meta-learner. A meta-learner uses knowledge from base classifiers—along with their outputs, called meta-features—to compute a unified prediction [14, 16, 45]. Techniques like voting, stacking, cascading, and arbitrating have been developed for this purpose. Majority voting is common in bagging and boosting, while STK is particularly effective for combining heterogeneous classifiers. Noticeably, combining classifiers is not always beneficial: misleading models can make the ensemble lean towards misclassifications if not carefully managed [46].

2.6 | Diversity Metrics for Ensemble Classifiers

The paper [47], despite being 20 years old, provides an excellent overview of metrics to quantify the diversity of classifiers, from different viewpoints. Authors list pair-wise metrics, which allow for estimating the diversity of pairs of classifiers, but also provide metrics that apply to sets of $n > 2$ classifiers. In the vast majority of cases, pair-wise metrics can be extended to a formulation that works also for sets of classifiers. Two metrics introduced in [47] are particularly relevant for our work, which we report below using a test set TS and a set of classifiers $\mathrm{CLF}_{\mathrm{set}}$.

- The disagreement (DIS), defined as "the ratio between the number of observations where two classifiers predict different results to the total number of observations."

$$\mathrm{DIS}(\mathrm{CLF}_1, \mathrm{CLF}_2, \mathrm{TS}) = \sum_{\mathrm{dp} \in \mathrm{TS}} \begin{cases} 1 \text{ if } \mathrm{CLF}_1.\mathrm{predict(dp)} \neq \mathrm{CLF}_2.\mathrm{predict(dp)} \\ 0 \text{ otherwise} \end{cases}$$

  and scales as follows with $n \geq 2$ classifiers:

$$\mathrm{DIS}(\mathrm{CLF}_{\mathrm{set}}, \mathrm{TS}) = \sum_{i=0 \text{ to } n-1} \sum_{j=i+1 \text{ to } n} \mathrm{DIS}(\mathrm{CLF}_i, \mathrm{CLF}_j, \mathrm{TS})$$

- The Double Fault (DF), defined as the "proportion of the cases that have been misclassified by both classifiers," which scales as the "proportion of the cases that have been misclassified by all classifiers" for more than two classifiers. This metric quantifies the probability of having all classifiers misclassifying the same data point, which is the same as having replicas with a *common mode failure* on a specific input, which is typically recognized as one of the most—if not the most—critical situations to avoid in critical systems engineering [48].

$$\mathrm{DF}(\mathrm{CLF}_{\mathrm{set}}, \mathrm{TS}) = \sum_{\mathrm{dp} \in \mathrm{TS}} \begin{cases} 1 & \bigwedge_{\mathrm{fc} \in \mathrm{CLF}_{\mathrm{set}}} \mathrm{fc\ misclassifies\ dp} \\ 0 \text{ otherwise} \end{cases}$$

Ideally, classifiers in an ensemble will always agree on the correct prediction, which is not the case in real applications. Thus, ensembles should be built to minimize DF and maximize DIS, to ensure diversity of opinions: there is no benefit in orchestrating multiple classifiers if all of them never disagree, or always predict the same output.

3 | 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 unknown events [49].

3.1 | The Rationale and Motivation

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

However, not all the failures have the same impact, and it would be beneficial to change the failure semantics of classifiers from uncontrolled content failures (i.e., misclassifications) to omission, or 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 components [48] are well known in the literature and often rely on safety wrappers or monitors [1, 10, 50]. 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 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 [51]: this approach is not different.

3.2 | FCCs: Building Trust

How can FCCs 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 of more severe than acceptable" [48]. 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 [52, 53], 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 [54]—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."

This pairs very well with the general formulation of FCCs we introduced at the beginning of this section: the focus shifts 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. FCCs do not aim at improving classification performance; they aim at rejecting (all and only) misclassifications, allowing for predictions to be justifiably trusted even if there is misclassification.

3.3 | Handling Rejections at the System Level

For an adequate deployment of FCCs, the encompassing system should know how to promptly act to guarantee that the system will not be negatively affected in case of rejections 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 rejecting potentially wrong predictions has clear benefits in the behavior of a 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 such 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 such as emergency braking or evasive steering [7] to take place. Other tasks such 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 [55].

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 [56]. This does not pair well with classifiers, which do not account for "rejections," limiting their usage despite the many possible applications, for example, automatic visual inspection, rail maintenance management [57]. Differently, FCCs pair extremely well with this paradigm as they can minimize misclassifications, knowing that subsequent rejections will likely trigger safe states where the component will be stopped.

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 [58].

4 | Basics of FCCs

As depicted in Figure 1, Fail-Controlled Classifiers (FCCs) should perform runtime monitoring for suspecting misclassifications of the classifier itself.

4.1 | Evaluation Metrics

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 $\alpha_w \leq \alpha$;
- rejection probability $0 \leq \varphi \leq 1$. The $FCC(clf)$ may reject misclassifications (φ_m, desirable and to be maximized), or correct predictions (φ_c, unnecessary rejections 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, $\varphi_m + \varepsilon_w = \varepsilon$.

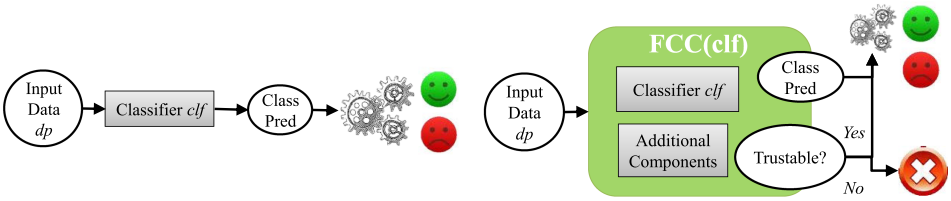

FIGURE 1 | Transitioning from traditional classifiers (left) to fail-controlled classifiers (right), which may reject predictions.

TABLE 1 | α_w, ε_w, φ_c, φ_m, and compound probabilities.

clf behavior→ FCC(clf) behavior↓	Correct prediction	Mis-classification	Sum
Not rejected	α_w	ε_w	$1-\varphi$
Rejected	φ_c	φ_m	φ
Sum	α	ε	1

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 a rejection probability close to ε, thus $\varphi \approx \varepsilon$. The following compound metrics may be calculated for a complete understanding of its performance:

- φ_m ratio $= \varphi_m/\varphi$, the ratio of rejected misclassifications over all rejections of the *FCC*(*clf*), to be maximized;
- 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 typically have lower accuracy than *clf* (i.e., $\alpha_w \leq \alpha$ aside from corner cases), as it does not primarily aim at improving correct classifications. It aims at transforming most of the erratic misclassifications, which are difficult to manage, into rejections, that is, having a high ε drop. Whereas at the 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.

Takeaway 2. Researchers interested in lowering misclassifications should primarily focus on maximizing the ε *drop*, but should also make sure that the φ_m *ratio* is high enough.

Another important aspect is related to comparing predictions of different FCCs. Generally speaking, any measure that works for comparing predictions of multi-class classifiers works even with FCCs, as the "reject" option is simply considered an additional class to the problem in this formulation. Specifically, for the diversity metrics we already reviewed in Section 2.5, some of them already apply to FCCs as they are, but they cannot catch the behavior of FCCs with respect to rejections. It is important to quantify the probability of having different FCCs that both reject their prediction on the same inputs. Using a formulation similar to that of the DF metric from [47], we define the *double reject* DR measure between FCC_i and FCC_j as

$$\mathrm{DR}(\mathrm{FCC}_i, \mathrm{FCC}_j, \mathrm{dp}) = \begin{cases} 1 \text{ if } \mathrm{FCC}_i \text{ rejects dp prediction} \\ \wedge \mathrm{FCC}_j \text{ rejects dp prediction} \\ 0 \text{ otherwise} \end{cases}$$

Notably, this pair-wise metric can be extended to group FCCs into FCC_{set} as follows.

$$\mathrm{DR}(\mathrm{FCC}_{set}, \mathrm{dp}) = \begin{cases} 1 \bigwedge_{\mathrm{fc} \in \mathrm{FCC}_{set}} \text{fc rejects dp prediction} \\ 0 \text{ otherwise} \end{cases}$$

4.2 | Confidence of FCCs

FCCs run additional components alongside the main classifier to complement its execution and potentially trigger rejections whenever desired conditions are not met. This has an obvious impact on its output, but it may also affect the confidence in the prediction, even when the output remains unchanged.

Assume that a main classifier *clf* is a binary classifier predicting a tabular data point to belong to normal or anomalous network data of a specific target machine. On top of the *clf*, we instantiate three different FCCs: Alice(*clf*), Bob(*clf*) and Carl(*clf*). Alice rejects all predictions for which the confidence is lower than 0.99, Bob rejects those where confidence is lower than 0.9, and Carl rejects those under 0.7. Let us suppose that a data point is fed into *clf*, which classifies it as normal with a probability of 0.91, and as being collected when the network was under attack with the remaining 0.09 probability. To keep things simple, this example considers the (softmax) probability assigned to the most likely class as the confidence of the classifier prediction, which is 0.91 in this case. Alice(*clf*) rejects the prediction, resulting in a different output with respect to the main *clf*. Conversely, Bob(*clf*) and Carl(*clf*) do not reject it, leading to the same output of *clf*; however, the confidence in the output may change a lot due to the additional components run by Bob(*clf*) and Carl(*clf*). The confidence of 0.91 is very close to Bob's threshold, but is very far from Carl's; thus, Carl is rejecting the prediction with a higher confidence than Bob.

A precise quantification of the confidence of FCCs depends on the specific design of the FCC itself; thus, we will elaborate more on this aspect in Section V.

4.3 | Detecting and Suspecting Misclassifications

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

Black-box approaches do not assume any knowledge about the classifier, thus observing its inputs and outputs without any

access to internals. They allow for building statistical machinery that conveys input pre-processing [22], output analysis [18] and even ensembles of them [59] 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-trustworthy otherwise. Other approaches aim at identifying unstable regions of the input space in which the classifier may be likely to output misclassifications [60, 61], or use external classifiers (e.g., nearest neighbors [20]) to validate the output of the target classifier.

When insights on 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 [18, 28]—which vary from one 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 [26, 56] 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 [62]. 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 [19].

4.4 | Learning to Reject

A recent survey summarizes possible approaches for rejecting predictions of classifiers [21]. As advocated there, "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 [21], 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*: Data are seen as either in-distribution or unknown, that is, 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, and forged samples need to be rejected in order to guarantee the correct behavior of a system. This is typically due to adversarial attacks or other mechanisms to forge realistic (but fake) data.

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 [63, 64]. Other strategies exploit the concept of stability under perturbation, that is, a correct decision should be invariant to small perturbations [65], 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 neighbors from the training set and reject if such a score is exceedingly high [66].

Nevertheless, rejecting overconfident predictions due to unknown inputs mostly revolves around strategies to compute confidence values that remain well-calibrated when encountering out-of-distribution (OOD) data or inputs from shifted distributions. Whereas many uncertainty and confidence estimation methodswere 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 authors' knowledge, this is still an open problem that has suggestions but no best practices yet.

5 | 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 the FCCs in Figure 2. These FCCs allow for reducing misclassifications thanks to the rejection mechanisms, which has an obvious downside: whenever the FCC rejects many correct predictions alongside misclassifications, it becomes almost unusable as it hardly provides a beneficial behavior for the encompassing system. To address this problem, we present additional approaches based on ensembles of FCCs: Voting (VT), Weighted Voting (WVT), Recovery Blocks (RB), and STK, which aim to reduce misclassifications and at the same time keep unnecessary rejections as low as possible. The numbering of sections matches the alphabetic numbering of subfigures of Figure 2, for example, Section 5.1 details what is shown in Figure 2a.

5.1 | 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 an agreement of the outputs of the replicas [67], or employing testing libraries that are periodically exercised on both hardware and software equipment [68]. Whenever one of these checks fails, the target component

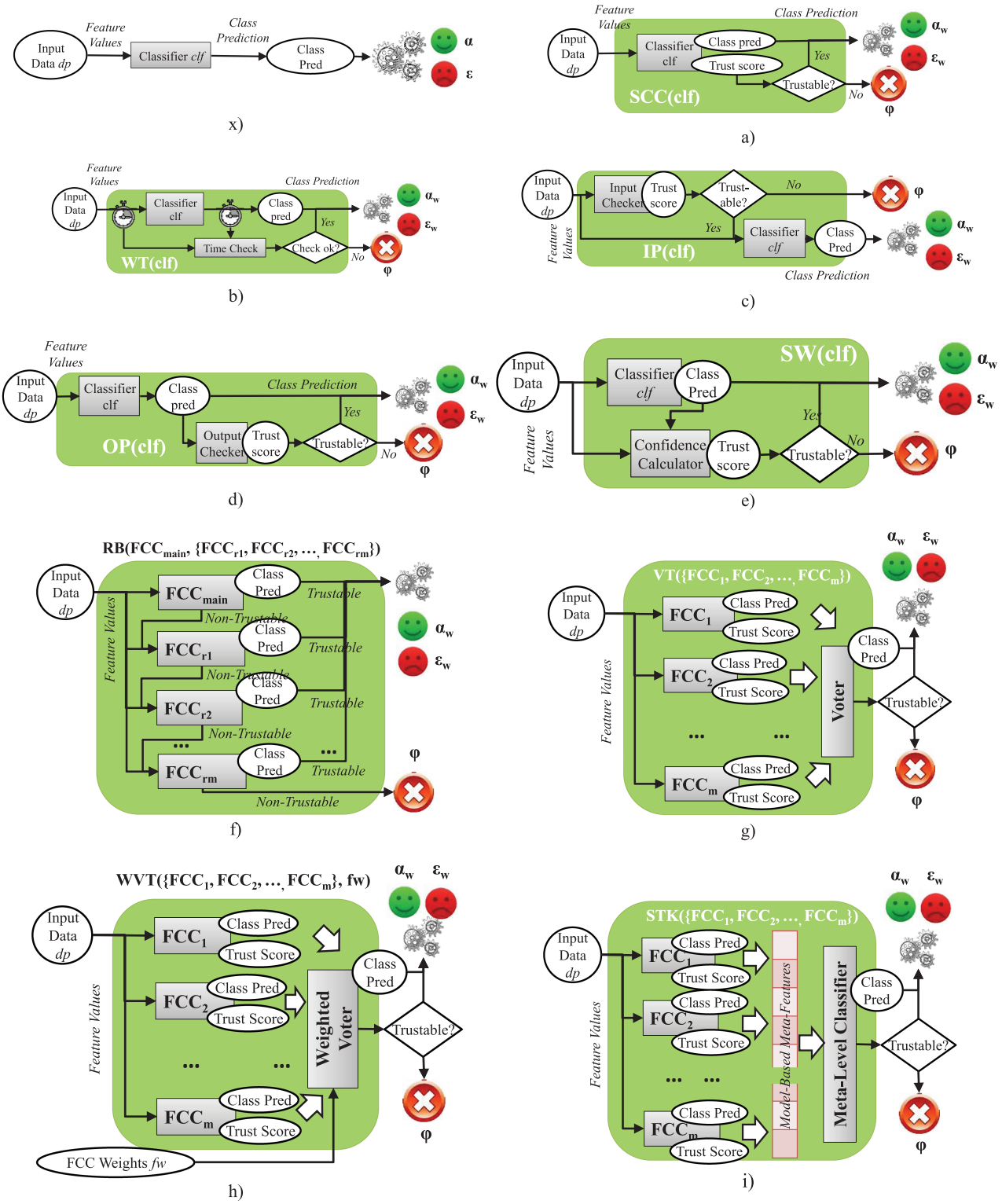

FIGURE 2 | Software architectures for FCCs with accuracy α_w, misclassification probability ϵ_w, and rejection probability φ. (x) A simple (reference) classifier, that provides correct or incorrect output against input. (a) Self-Checking Classifier (SCC), which already has built-in and non-trivial methods 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 inference to compute confidence and decide on trustworthiness. (f) Recovery Blocks (RB) pair the main FCC with other FCCs ran in sequence, seeking for the first that outputs a confident prediction. (g) Voting VT. Different FCCs are run in parallel and their results used for adjudication by means of k out of n voting. (h) Weighted Voting (WVT). Weights *fw* can be provided as a parameter or be derived during inference. (i) Stacking STK: FCCs are run in parallel as base-learners, creating model-based meta-features to perform adjudication.

is deemed failed and in need of being replaced or fixed. Replicas refer instead 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 [60], but are not available by default in standard libraries used for supervised learning, for example, Python's *scikit-learn*, *PyTorch, Keras*.

5.2 | WT

WTs aim at measuring the length of the execution of a target function to understand if the elapsed time conforms with expectations [69]. In case the function completes too early, the WT generates an alert that we can use to trigger rejections. If the function completes too late, it indirectly delivers a rejection 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, WTs were meant to check electronic or mechanical-related functions [70], but transitioned to check the execution of software [69] 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 patterns 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 slowdowns due to malicious or malfunctioning software acting in the host system), resulting in a low rejection probability but high residual misclassifications. Importantly, tuning timers is a system-specific process: a WT may work well with specific hardware, but require re-tuning when the same hardware gets updated; that is, the notion of normal prediction time varies.

5.3 | IP

This FCC performs a pre-processing to seek 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 [31], out-of-distribution detectors [32], image corruption detectors [9], statistical distributions [22], or unknown data detectors in general [4]. Detecting one of the cases above could trigger a rejection 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 rejections. Importantly, some classifiers are "robust enough" to successfully deal with minor issues in the input data: in this situation, the FCC should reject the output only when the issue or corruption will not be tolerated by the robust classifier, reducing rejections. Some strategies pre-process the input to identify issues, while also providing 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 [5]. 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.

5.4 | OP

This is the simplest FCC out of the ones that we present in this paper, as it directly acts on the output probabilities of a prediction [18], computing the entropy of the probabilities, or using the absolute value of the highest probability as indicators of trust in the prediction. Entropy, in this case, quantifies the inherent uncertainty 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 a rejection. The threshold can be arbitrarily defined by the user or be classifier-specific; in the latter case, it is possible to end up having very different rejection rates when using different classifiers.

5.5 | SW

IP and OP FCCs act before or after the classifier. However, we may think of a monitor or wrapper [1] that acts before, during and after inference. This is the case of the SW, which builds an envelope around the classifier to extract as much information as possible, 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 possibly use the classifier for additional predictions. Implementations of black-box SW may rely on Bayesian approaches [23], ensembles of confidence measures [59], relative positioning of input data with respect to the prediction [20] or run other classifiers (e.g., kNN [25], probabilistic DNNs [24]) and check for agreement with the main classifier.

Conversely, when the internals of the classifiers are fully available, it is possible to craft very specific mechanisms that seek very specific information throughout the inference process. The activation patterns 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 [28].

5.6 | RBs

Recovery blocks are known since many decades as one of the strategies for reliable software design [71]. Practically speaking, they consist of a set of m alternative implementations of a function that are called sequentially whenever the output of the main function is deemed non-trustable. For classifiers, this translates into calling a sequence of at most $m+1$ classifiers before obtaining the prediction, or rejecting the result if none of the recovery blocks is confident enough in its output. This concept is not entirely new in the ML domain: delegating classifiers [72] 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 2f). The amount and the diversity of replicas have a direct impact on the likelihood of rejections: 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.

5.7 | Majority and k-o-o-n Voting VT

Other approaches rely on N-Version Programming, or exercising different classifiers in parallel [44, 71] each acting independently but processing the same inputs. Their predictions, along 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 an aggregate prediction and a trust score, and decides if the prediction has to be rejected or if it is trustworthy. For a problem of m-class classification and n replicas, the adjudicator is a function that has $(m+1)n$ floating point inputs and outputs m probabilities plus a floating point trust score. Voting (VT) is a simple adjudication function that applies a rule called *k-out-of-n*: when at least k out of n replicas agree on the same output, the "agreed upon" output becomes the output of the VT, with a trust score that is higher the more replicas reach the agreement. Conversely, the output is rejected.

5.8 | WVT

A variant of voting is the WVT, where replicas are assumed to have different reputations: opinions of some replicas should count more towards the final adjudication than others. Therefore, WVT requires an array of weights *fw* of n floating point values. These could be provided as input, as shown in Figure 2h, or could depend on the inference process. In the former case, WVT requires information about the behavior of each FCC to be used as weight, for example, their accuracy a_w, the non-misclassification probability $1-\epsilon_w$, or an arbitrary value assigned by domain experts. When the weights depend on the inference of a specific input, the reputation of replicas changes from prediction to prediction. A straightforward way to define weights in this way is to use the trust score of FCC replicas as weight: the more confident an FCC is, the more it contributes towards the output of the WVT.

5.9 | STK

The adjudicator [71] of an NVP system can be implemented with thresholds as in (weighted) voting, or can be a classifier itself, providing many degrees of freedom in finding the ideal function to combine the n replicas. This builds a two-layer software architecture that is often referred to as STK, with n FCCs at the base level, and a different, independent, classifier at the meta-level [73]. The meta-level classifier expects $2\times n$ inputs, or rather the outputs and trust scores of FCCs that are run as the base-level: these are called model-based meta-features (see the red vertical array in Figure 2i). By design, the meta-level classifier can process its inputs only after all FCCs have been executed at the first stage: those two steps are necessarily sequential (cannot be parallelized), and as such they add a relevant overhead to the inference process.

5.10 | Other Notable Approaches

Decades of critical systems engineering and system-level thinking originated more architectures [44, 71] than those shown above. Voting was explored in different formulations (i.e., hard, soft, weighted), and can even be used to build a hierarchical agreement structure that is known as n-self-checking programming [71]. Boosting techniques were proven to be very effective for classifying known [12, 14] 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 [45]. Interestingly, different architectures may be combined into unified architectures. For example, there are works that pair classifiers based on ensembles with a monitor to check for trustworthiness [3], but this is still quite uncharted territory in which there are no widespread and solid proposals (yet).

5.11 | Discussion

The SCC, WT, IP, OP, and SW FCCs all have their limitations and advantages. Ideally, we want low rejection probability φ, low residual misclassifications ϵ_w, and low overhead. Since none of the solutions above (and none at all, according to the knowledge of the authors) guarantee 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 sensitivity analyses to fine-tune the overall mechanism.

Takeaway 3. Different FCCs may be more suitable than others under specific circumstances. The choice and combination of FCCs is up to the system architect.

Complexity may be high also for RB, (W)VT and STK, which exercise ensembles of FCCs. RB exercises FCCs in sequence: if the i-th FCC rejects the output, the $i+1$-th FCC is invoked, until the

last FCC in the RB is exercised. The fact that FCCs are exercised in a given order and are not necessarily exercised at each prediction makes the choice and the ordering of FCCs a major concern in RBs. Generally speaking, it is beneficial to exercise fast FCCs first, for promoting a quick inference process in the majority of cases, using slow FCCs only later in the sequence. This guarantees that the average time needed for inference is kept reasonably low. Another concern is about the rejection probability φ of FCCs used in RBs. If the first FCC in the RB ensemble rarely omits, the behavior of the RB is going to be the same as that of the first FCC with few exceptions. Thus, FCCs have to be ordered starting from those with low residual misclassifications ε_w and potentially high rejections φ to FCCs with low rejections φ at the end.

Takeaway 4. RB may be beneficial only if the first FCCs to be executed have low residual misclassifications ε_w, and potentially high rejections φ, transitioning to FCCs with low rejections φ at the end of the sequence.

(W)VT approaches exercise many FCCs in parallel and decide on a (weighted) voting of their outputs, accounting for rejections. This is a very straightforward approach; however, the WVT formulation requires a careful choice of weights, which may be assigned to favor accurate classifiers (high α or α_w values) or, more interestingly for limiting rejections of those that have low residual misclassifications ε. STK may increase the computational complexity even more since the adjudication is performed by yet another FCC, and thus has to be used only when enough resources are available.

6 | Experimental Setup

This section describes the experimental campaign to quantify how the behavior 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).

6.1 | 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 and the main classifiers: IP, OP, SW, see Section 6.2. In Section 6.3, we gather datasets for exercising tabular and image classifiers, spanning over a wide variety of classification tasks and simulating unknown data (Section 6.4). Sections 6.5 and 6.6 report results for tabular data and image classification, respectively. The performance of classifiers and FCCs is quantified via the metrics from Section 4.1. 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 GitHub at [11].

6.2 | Selection of FCCs and Classifiers

Some of the FCCs that are presented in Section 5 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, in this first experimental analysis, we instantiate the IP, OP, SW FCCs as follows.

6.2.1 | Tabular FCCs and Main Classifiers

Tabular data classifiers are preferably built over ensembles of decision trees or statistical ML algorithms [13–15]. Thus, the five binary tabular classifiers used as base classifiers in this study are *Decision Trees* (*DT*), *Random Forests* (*RF*), *XGBoost* (*XGB*) *Gaussian Naïve Bayes* (*GNB*), *and Logistic Regression* (*LR*). To implement the input checkers in the IP and SW FCCs, we rely on *Extra Trees* (*ET*) and *Linear Discriminant Analysis* (*LDA*), which are different from those that we use as main classifiers of FCCs. Output checkers can rely either on static or dynamic thresholds for deciding on rejections: for the sake of our study, either of the two approaches forces us to choose an arbitrary value. Looking at the results of the experiments and at the probabilities of classifiers, we found that a threshold of 0.8 on probabilities (i.e., predictions with probability lower than 80% are rejected) allows for a reasonable amount of rejections. All tabular classifiers are exercised using their default parameters from the *scikit-learn*, and *xgboost* Python libraries [74]. This experimental setup created 10 IPs (i.e., five classifiers, each checked via ET or LDA), 5 OPs (one per classifier), and 10 SWs, obtained by combining each of the 10 IPs with the OP. In this configuration, a SW is a combination of an IP and an OP.

6.2.2 | Image FCCs and Main Classifiers

For image classification, the literature acknowledges how neural networks are the preferred choice when dealing with unstructured data [6–8]. We chose AlexNet (AN), DenseNet121 (DN), VGG11 (VGG), and InceptionV3 (IC) as image classifiers given their wide usage in the last decade. Input checkers for images use DNNs as well: ResNet50 (RN) and GoogLeNet (GN). OP and SW are configured similarly to tabular classifiers: this results in 8 IPs, 4 OPs, and 8 SWs. Each of the six DNNs above brings unique features: VGG is known for its simplicity, DN improves the information flow with densely connected layers, and GN introduces inception modules for computational efficiency. IC builds on GN with further optimization, while RN's residual connections enable the training of deeper networks. AN, a pioneer in CNN architectures, laid the groundwork for modern image classification. The final models are obtained by transfer learning with learning rate = 0.001 and batch size = 32 from pre-trained DNNs using ImageNet weights stored in PyTorch Lightning [75].

Naming is as follows: IP, OP, and SW FCCs are tagged as IP_<x>_<y>, OP_<x>, SW_<x>_<y>, where x is the name of the main classifier, and y is the name of the classifier used as input checker.

6.3 | Datasets

This section lists image and tabular datasets used for experiments. We split each dataset using a 50-20-30 train-validation-test split.

6.3.1 | Tabular Datasets

We select three tabular datasets belonging to different domains in which classifiers are typically willing to be applied: intrusion detection (CICIDS18 [76]), error detection (ARANCINO [77]), and control systems (MetroPT [78]). These datasets contain hundreds of thousands of data points corresponding to the behavior 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 from anomalies due to attacks, errors, or component failures.

6.3.2 | Image Datasets

We select three image datasets: Flower (9 classes [79]), FER2013 (7 classes [80]), and FOOD-101 (10 classes [76]). Out of the many alternatives for image classification, we favored those since they are publicly available, belong to different domains, have a varying number of classes, and allow for fast experimentation times. Since the images vary in size, we apply a transformation to resize them (at most 96×96 RGB). The datasets contain on the order of thousands or tens of thousands of images per dataset.

6.4 | Generation of Unknown Inputs

Public datasets are useful for experimentations, but may not generalize well to real scenarios that are prone to encountering 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 these conditions as part of our experiments.

6.4.1 | Unknown Image Data

We generate a first batch of unknown (out-of-distribution) images by applying four different alterations, that is, Rotation, Color Space, and Gaussian Noise [81] and GANs to 40% of images from the test set, that is, 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. Color space anomalies result from chaining operations such as Brightness 0.5, Contrast 1.5, Saturation 1.5, and Hue adjustment at the scale of 20. For rotation, we rotate the image of 90° left.

The second batch of unknown images was generated via the Fast Gradient Signed Method (FGSM), which leverages gradient information to create perturbations that maximize the classifier's error (Figure 3). FGSM's parameters are as follows: (i) input image size

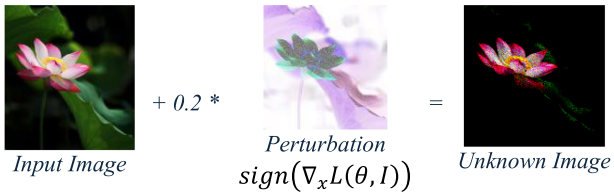

FIGURE 3 | Example of generating unknown images in the Flower Dataset, where input images are altered with the perturbation with a multiplier to form an unknown image.

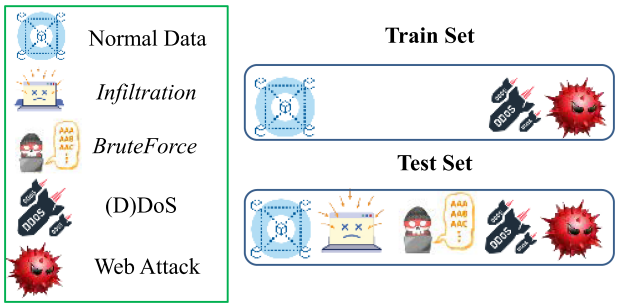

FIGURE 4 | Unknown tabular data: The example of CICIDS18, where some attack classes are removed from the training set occurring only in the test set, being unknown to the classifier.

(x, y) as in the dataset, $\gamma = 0.2$, θ: AN, DN, IC, VGG model parameters, L: AN, DN, IC, VGG loss. These alterations were injected using the OpenCV and PyTorch library, and the parameters above are amongst the ones suggested in the documentation.

6.4.2 | Unknown Tabular Data

The generation of unknown tabular data is not as straightforward as it happens with images. Fuzzing or adding random noise generates 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 tabular classifier. For CICIDS18, *SSH-Bruteforce*, *FTP-BruteForce*, and *Infiltration* attacks only appear in the test set (see Figure 4). 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 20% of the test set in the Food-101, Flower, FER2013 image datasets, to 20%, 12%, 8% for CICIDS18, MetroPT, and ARANCINO tabular datasets, respectively.

6.5 | Results: Tabular Data Classification

We start commenting on the results of classifiers and FCCs using tabular datasets with the aid of Table 2, which reports the percentage of misclassifications ϵ (for the main classifier), ϵ_w (for FCC),

TABLE 2 | Results of tabular clf and FCCs across tabular datasets used in the paper.

				NIDS—CICIDS18					Metro—MetroPT					Error detection—ARANCINO				
Row #	**FCC**	**Tag**	**Main 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	IP	IP_DT_ET	DT	0.123	0.026	0.222	0.793	0.441	0.126	0.000	0.126	0.997	0.995	0.063	0.021	0.066	0.664	0.636
2	IP	IP_DT_LDA	DT	0.123	0.035	0.232	0.719	0.382	0.126	0.002	0.183	0.983	0.673	0.063	0.046	0.061	0.269	0.281
3	OP	OP_DT	DT	0.123	0.123	0.000	0.000	0.000	0.126	0.126	0.000	0.000	0.000	0.063	0.063	0.000	0.000	0.000
4	SW	SW_DT_ET	DT	0.123	0.026	0.222	0.793	0.441	0.126	0.000	0.126	0.997	0.995	0.063	0.021	0.066	0.664	0.636
5	SW	SW_DT_LDA	DT	0.123	0.035	0.232	0.719	0.382	0.126	0.002	0.183	0.983	0.673	0.063	0.046	0.061	0.269	0.281
6	IP	IP_GNB_ET	GNB	0.269	0.253	0.222	0.058	0.070	0.151	0.051	0.126	0.665	0.799	0.167	0.124	0.066	0.261	0.661
7	IP	IP_GNB_LDA	GNB	0.269	0.253	0.232	0.060	0.070	0.151	0.039	0.183	0.740	0.610	0.167	0.146	0.061	0.125	0.343
8	OP	OP_GNB	GNB	0.269	0.268	0.001	0.002	0.516	0.151	0.123	0.050	0.186	0.567	0.167	0.142	0.058	0.153	0.437
9	SW	SW_GNB_ET	GNB	0.269	0.253	0.223	0.060	0.072	0.151	0.037	0.154	0.758	0.743	0.167	0.099	0.122	0.406	0.555
10	SW	SW_GNB_LDA	GNB	0.269	0.252	0.233	0.062	0.072	0.151	0.029	0.199	0.807	0.614	0.167	0.122	0.117	0.271	0.388
11	IP	IP_LR_ET	LR	0.245	0.193	0.222	0.209	0.231	0.150	0.025	0.126	0.835	0.996	0.138	0.086	0.066	0.375	0.788
12	IP	IP_LR_LDA	LR	0.245	0.191	0.232	0.221	0.233	0.150	0.025	0.183	0.834	0.683	0.138	0.102	0.061	0.261	0.597
13	OP	OP_LR	LR	0.245	0.017	0.621	0.931	0.367	0.150	0.132	0.056	0.119	0.318	0.138	0.104	0.109	0.250	0.318
14	SW	SW_LR_ET	LR	0.245	0.015	0.723	0.939	0.318	0.150	0.013	0.176	0.915	0.780	0.138	0.060	0.156	0.564	0.499
15	SW	SW_LR_LDA	LR	0.245	0.011	0.761	0.956	0.307	0.150	0.012	0.231	0.918	0.597	0.138	0.082	0.137	0.409	0.414
16	IP	IP_RF_ET	RF	0.123	0.026	0.222	0.793	0.441	0.126	0.000	0.126	0.998	0.996	0.062	0.018	0.066	0.703	0.661
17	IP	IP_RF_LDA	RF	0.123	0.035	0.232	0.719	0.382	0.126	0.002	0.183	0.984	0.674	0.062	0.044	0.061	0.288	0.295
18	OP	OP_RF	RF	0.123	0.123	0.001	0.003	0.297	0.126	0.126	0.001	0.000	0.065	0.062	0.056	0.029	0.098	0.212
19	SW	SW_RF_ET	RF	0.123	0.025	0.223	0.793	0.439	0.126	0.000	0.126	0.998	0.992	0.062	0.014	0.092	0.777	0.523
20	SW	SW_RF_LDA	RF	0.123	0.034	0.233	0.721	0.381	0.126	0.002	0.184	0.984	0.673	0.062	0.039	0.087	0.373	0.266
21	IP	IP_XGB_ET	XGB	0.123	0.026	0.222	0.793	0.440	0.126	0.000	0.126	0.998	0.996	0.069	0.026	0.066	0.624	0.656
22	IP	IP_XGB_LDA	XGB	0.123	0.035	0.232	0.719	0.382	0.126	0.002	0.183	0.984	0.674	0.069	0.051	0.061	0.259	0.295
23	OP	OP_XGB	XGB	0.123	0.123	0.000	0.001	0.571	0.126	0.126	0.000	0.000	0.167	0.069	0.060	0.031	0.137	0.310
24	SW	SW_XGB_ET	XGB	0.123	0.025	0.222	0.793	0.440	0.126	0.000	0.126	0.998	0.996	0.069	0.018	0.095	0.745	0.544
25	SW	SW_XGB_LDA	XGB	0.123	0.034	0.232	0.721	0.382	0.126	0.002	0.184	0.984	0.674	0.069	0.042	0.091	0.395	0.300

Note: For each dataset, main classifier and FCC we report the misclassifications ε and ε_w, rejections φ, φ_m ratio and ε drop.

rejections φ, ε drop, and φ_m ratio for different datasets, FCCs, and main classifiers. The table has 25 lines, 5 for each of the five main classifiers DT, GNB, 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 approach optimal results (the higher, the better).

Reading the table by dataset blocks (i.e., 3 groups of 5 columns), we can observe the following. In the NIDS—CICIDS18 dataset, the misclassifications ε of DT, GNB, LR, RF, and XGB are respectively at 0.123 (1st–5th row), 0.269 (6th–10th row), 0.245 (11th–15th row), 0.123 (16th–20th row), and 0.123 (21st–25th row). FCCs always lower misclassifications as ε_w values are lower than ε, at a cost of a nonzero amount of rejections φ. Misclassifications of the RF may drop from 0.123 to 0.025 using the SW_RF_ET (19th row of Table 2), at a cost of 22.3% of rejections, or $\varphi = 0.223$. Roughly, we are reducing misclassification by a factor of 5, but 22% of the predictions of the FCC will be rejected. This is because only 43.9% (φ_m ratio) of rejections correspond to misclassifications, meaning that the remaining 56.1% of rejections would have been correctly predicted by the main 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 FCCs employing an IP or SW that uses ET as an input checker (i.e., IP_x_ET, SW_x_ET, where x is any main classifier), it allows for rejecting almost all (high ε drop) and only (high φ_m ratio) predictions that would have been misclassifications. This is the optimal scenario in which the application of the FCC brings misclassifications ε of XGB and RF from 12.3% to an ε_w of almost zero, with an ε drop of almost 99.8% (see 16th, 19th, 21st, 24th rows of the table, 8th to 11th column). In other words, the residual misclassifications are lowered by a factor of almost 1000, and there are just a few rejections of correct predictions; that is, φ_m ratio is 0.998, very close to the optimum 1. 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 they typically show non-optimal performance as they either reject an exceedingly high amount of predictions (high φ and low φ_m ratio) in the process.

Other important information can be obtained by reading the table horizontally. First, the OPs have very different results depending on the main classifier, as the 0.8 confidence threshold is either too low (thus rejections are almost non-existent $\varphi \approx 0$ as for the DT, which always predicts with probability 1, or maximum confidence) or too high, delivering an obnoxious rejection probability as for LR in CICIDS18, see OP_LR, 13th row, 5th column of Table 2.

Takeaway 5. 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, or achieving high ε drop, but it has to be calibrated well for each main classifier.

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., Error Detection—ARANCINO dataset), or where it rejects many correct predictions in the process, as quantified by the low φ_m ratio in the NIDS—CICIDS18 dataset. Safety wrappers SW are meant to reject predictions if either the input or the output check highlights issues; thus they always have high rejection rates, at the benefit of having very low residual misclassifications.

6.6 | Results: Image Classification

Results similar to those presented above can be obtained also for image classifiers. To avoid being tedious, this section focuses on a subset of FCCs for each dataset: using AN and DN main classifiers on the Flower dataset (Figure 5a), IC and VGG on FER13 (Figure 5b), DN and IC on Food (Figure 5c). These scenarios offer interesting discussion items that we explore as follows.

Figure 5a shows a 12-bar chart, 6 bars for AN (up in the figure) and 6 bars for DN used as main classifiers (down in the figure). For each classifier, the 6 bars show 2 IPs, 1 OP and 2 SW FCCs, plus the theoretical optimal performance assuming rejections of all and only misclassifications. For each result, the bar is partitioned into three blocks: blue solid for α_w, striped for φ, and red solid for ε_w; also, ε drop and φ_m ratio are reported in the table on the right. From a visual standpoint, the aim of a FCC is to reduce

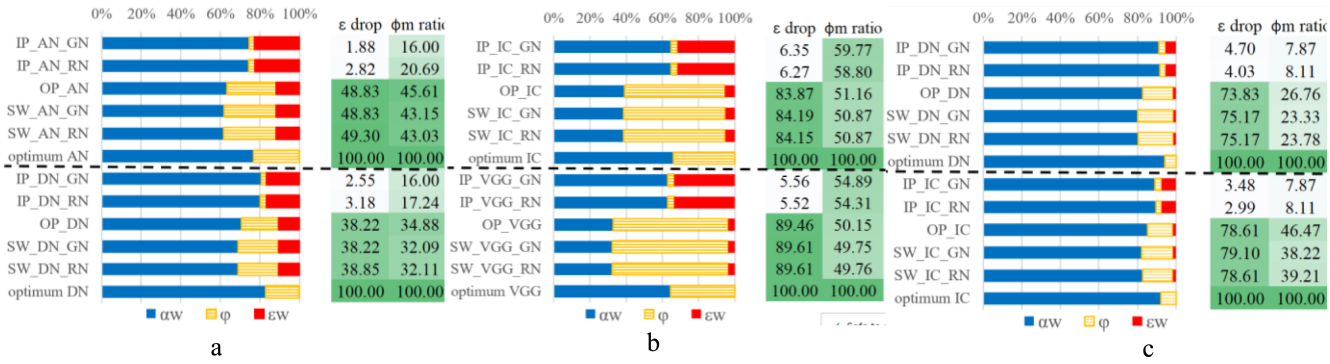

FIGURE 5 | Bar charts showing the performance of image FCCs on different datasets. (a) FCCs using AN and DN as main classifiers on the Flower dataset. (b) FCCs using IC and VGG as main classifiers on the FER13 dataset. (c) FCCs using DN and IC as main classifiers on the Food dataset.

the red bar (ε_w) as much as possible, keeping the blue bar (α_w) untouched, and replacing the red area with the yellow-striped rejections φ. Whereas the OP and SW FCCs succeed in reducing the red bar, they also have a shorter blue bar than the optimum: this is due to a low φ_m ratio, which is around 40% for the OP and SW with AN as the main classifier, and around 30% for OP and SW built using DN as the main classifier. In this case, there is no FCC that has both high φ_m ratio and high ε drop: all reduce misclassifications, but with a major price to pay in terms of unnecessary rejections.

Figure 5b has the same structure as Figure 5a, but refers to the application of IC and VGG as main (image) classifiers in the FER13 dataset. The trend is very similar to those of Figure 5a, with OP and SW FCCs succeeding in rejecting misclassifications. Importantly, ε drop is quite high here (e.g., ε drop = 89.61% for SW_VGG_GN and SW_VGG_RN), meaning that these FCCs are able to reject almost 90% of misclassifications, which is an important achievement.

Last, Figure 5c shows the application of FCCs using DN and IC main classifiers on the Food dataset, which confirms the trend from the previous figures. IPs have low ε *drop* and low rejection rate φ (the yellow-striped bar is always very short), whereas OP and SW trigger more rejections, allowing for an increase in the ε drop. FCCs not shown in the figures followed a similar trend to those shown here.

Takeaway 6. 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 rejections) 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.

7 | Diversity Analysis and FCC Ensembles

This section discusses how to design and experiment with FCCs that use ensembles of FCC algorithms, as RB, VT, WVT, and STK. First, we will discuss how to evaluate the diversity of FCCs and the results in our experiments (Section 7.1). Then, we will proceed to instantiate different RB, VT, WVT, and STK FCCs for tabular (Section 7.2) and image data (Section 7.3), discussing their results and comparing them against FCCs from the previous section.

7.1 | Computing Diversity

As already discussed in Section 2.5 and widely acknowledged in the literature, diversity plays a major role when crafting ensemble classifiers that depend on the opinions of multiple individual classifiers. This is no different when crafting FCCs ensembles, which orchestrate multiple FCCs as part of the inference process. Results discussed in the previous section are useful to understand the individual performance of FCCs, but they can also be used to quantify how diverse they are, and thus how beneficial it is to build ensembles. Using non-diverse FCCs in an FCCs ensemble often results in weaker performance, because similar components tend to make the same predictions and errors, which reduces the system's ability to adapt and correct itself.

We compute three different diversity measures for pairs of FCCs: disagreement ($0 \leq \text{DIS} \leq 1$, higher is better), double fault ($0 \leq \text{DF} \leq 1$, lower is better), and double reject ($0 \leq \text{DR} \leq 1$, lower is better) using the formulas introduced earlier in the paper. This allows for quantifying the diversity of FCCs and for maximizing the performance of FCC ensembles. We use the same datasets and ML algorithms as in the previous section: complete data about diversity is provided in [12], whereas in the followings we will only report the subset of diversity data that is most relevant for the discussion.

7.2 | Tabular Data

Here, we discuss the diversity of IP, OP, SW FCCs for tabular data, and how to exploit this information for creating diverse RB, VT, WVT, STK ensembles.

7.2.1 | Diversity Analysis

7.2.1.1 | General Comments. Data about the performance of IP, SW, OP tabular classifiers was already commented on in the previous section and fully reported in Appendix A in Table A.1 of [12]. Table A.2–Table A.4 of [12] report all diversity metrics DIS, DF, DR for tabular classifiers. OPs for tabular classifiers have high accuracy and very few rejections, but tend to fail on the same inputs, having high DF (see the middle of Table A.3 of [12]). IPs and SWs are more prone to rejections, with a low residual misclassification rate, low DF but also a pretty high DR (see the corners of Table A.4 of [12]) due to the many rejections they output. Overall, the maximum DIS is reached when comparing IPs against OPs, especially when using GNB and LR as main classifiers. This is due to the fact that FCCs using GNB and LR as main classifiers have overall poor classification performance, resulting in many misclassifications that tree-based classifiers such as DT, RF, XGB do not have. The most relevant combinations of FCCs are summarized in Table 3: note that these report a very low DF, which is typically important for avoiding common-mode failures.

7.2.1.2 | FCCs for RB. First, we aim at devising a group of FCCs that have very low residual misclassifications (thus low DF) and high DIS, to be used as building blocks of the RB FCC. From Table 3, it makes sense to leave OP_DT and OP_LR out of the picture as they have high ε_w. However, OP_DT has no rejections with $\varphi = 0$: using this as the last item of an RB will drop rejections of the RB as well, which may be beneficial in specific scenarios. Thus, we plan for two versions of RBs, where FCCs are ordered by decreasing rejection probability φ: (RB1) SW_LR_ET, SW_XGB_LDA, IP_RF_LDA, IP_DT_ET, and (RB2) SW_LR_ET, SW_XGB_LDA, IP_RF_LDA, IP_DT_ET, OP_DT. The only difference is that RB2 employs OP_DT as the last FCC, making it have no rejections and maximizing accuracy compared to RB1.

7.2.1.3 | FCCs for (W)VT, STK. Voting approaches demand maximum disagreement, thus requiring a more careful analysis of DIS values in Table 3. OP_LR and SW_LR_ET exhibit

TABLE 3 | Classification performance, DIS, DF, DR of FCCs used for building RB, VT, WVT, STK tabular classifiers.

FCC	φ	α_w	ε_w	DIS (best if high)						DF (best if low)						DR (best if low)					
				IP_DT_ET	IP_RF_LDA	OP_DT	OP_LR	SW_LR_ET	SW_XGB_LDA	IP_DT_ET	IP_RF_LDA	OP_DT	OP_LR	SW_LR_ET	SW_XGB_LDA	IP_DT_ET	IP_RF_LDA	OP_DT	OP_LR	SW_LR_ET	SW_XGB_LDA
IP_DT_ET	0.138	0.846	0.016	—	0.05	0.05	0.27	0.23	0.06	—	0.01	0.02	0.01	0.01	0.01	—	0.11	0.00	0.05	0.14	0.12
IP_RF_LDA	0.159	0.814	0.027	0.05	—	0.09	0.29	0.26	0.01	0.01	—	0.03	0.02	0.01	0.03	0.11	—	0.00	0.04	0.12	0.16
OP_DT	0.000	0.896	0.104	0.05	0.09	—	0.25	0.28	0.09	0.02	0.03	—	0.06	0.01	0.02	0.00	0.00	—	0.00	0.00	0.00
OP_LR	0.262	0.654	0.084	0.27	0.29	0.25	—	0.03	0.29	0.01	0.02	0.06	—	0.03	0.02	0.05	0.04	0.00	—	0.26	0.05
SW_LR_ET	0.352	0.619	0.029	0.23	0.26	0.28	0.03	—	0.26	0.01	0.01	0.01	0.03	—	0.01	0.14	0.12	0.00	0.26	—	0.12
SW_XGB_LDA	0.169	0.805	0.026	0.06	0.01	0.09	0.29	0.26	—	0.01	0.03	0.02	0.02	0.01	—	0.12	0.16	0.00	0.05	0.12	—

Note: Results are averaged across the three tabular datasets used in experiments. Red denote bad and green denote good results.

low disagreement of 0.03, see the 4th–5th row, 8th–9th column of Table 3. These two have high disagreement with others, up to almost 0.3, which is very high. Therefore, we choose IP_DT_ET, IP_RF_LDA, OP_DT, SW_LR_ET, and SW_XGB_LDA as FCCs to build VT FCCs. WVT requires weights to be assigned to each of the three classifiers. In our case, we use the "non-misclassification probability," which is $1-\varepsilon_w$: this gives more weight to FCCs that are rarely wrong. STK requires a meta-level classifier, which is RF in our case. We tried other (faster) classifiers such as LR, GNB, LDA in its place, but they delivered very unsatisfactory performance.

7.2.2 | Experiments With RB, VT, WVT, STK

Figure 6 reports the comparison of the performance of RB1, RB2, STK, VT, WVT against a regular (RF) classifier and IP, OP, SW FCCs from the previous section. We report results related to Error Detection (Figure 6a) and MetroPT (Figure 6b), avoiding NIDS as it was showing a trend very similar to that of Error Detection. Bar charts in Figure 6 follow the same color scheme as Figure 5: blue solid bar for α_w, orange-striped bars for φ and red bars for ε_w. On top of the bar charts, we find the performance of RF used as a traditional classifier, thus with no rejections. Then, we report on IP, OP, SW and the ensemble FCCs using RF as the main classifier in both datasets. Noticeably, the OP_RF has the same behavior as the RF alone, as the RF always answers with a confidence higher than 0.8, which is the threshold we chose for OPs in our experiments. SW and VT have short red bars in both cases, showing a very low number of residual misclassifications; however, the blue part tends to be shorter than that of the RF main classifier, meaning that this result is achieved at a cost of lowering correct classifications. RB1 and RB2 bring a similar amount of correct classifications as the regular RF classifier. RB1 transforms many misclassifications into rejections, but is less effective than VT overall. WVT allows for rejecting misclassifications, but has a detrimental impact on accuracy, or the fraction of correct classifications.

Noticeably, STK is able to improve α_w, which rarely happens in FCCs. This is because the meta-level STK classifier learns that when base learners reject a prediction, this is likely to be an anomaly. This allows not only for rejecting some predictions, but also to correct others. In the case of the MetroPT dataset, where IPs are able to reject almost all misclassifications, this means that the STK FCC achieves a near perfect accuracy, which is a rare but indeed beneficial case.

Takeaway 7. The STK FCC is the only FCC that allows for improving correct classifications of binary classifiers, as the meta-level classifier may even learn to interpret rejections as anomalies. Does not apply for multi-class.

7.3 | Image Data

Here we report the diversity analysis for devising RB, VT, WVT, STK FCCs for image data, discussing results of these classifiers.

7.3.1 | Diversity Analysis

7.3.1.1 | General Comments. Data about the performance of IP, SW, OP image classifiers was already commented on in the

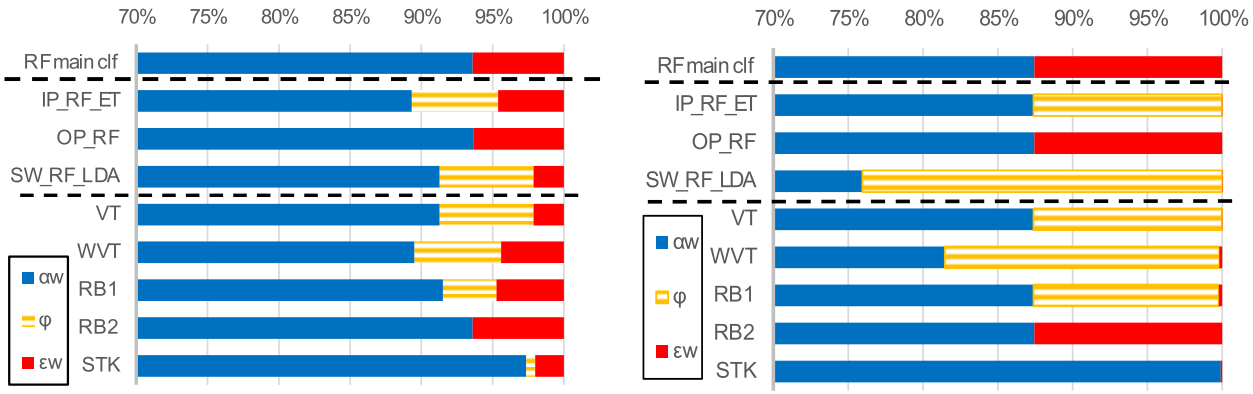

FIGURE 6 | Comparison of FCCs using RF as the main classifier on the Error Detection (a, left) and MetroPT (b, left) tabular datasets. The color scheme complies with that of Figure 5.

previous section and fully reported in [12] in Table A.5. Table A.6–Table A.8 of [12] report all diversity metrics DIS, DF, DR for image classifiers. IPs for image classifiers have high accuracy and very few rejections, but tend to fail on the same inputs, having high DF (see Table A.7 of [12]). OPs and SWs are more prone to rejections, with a low residual misclassification rate, low DF but also a pretty high DR (Table A.8 of [12]), due to the many rejections they output. Overall, the maximum DIS is reached when comparing IPs against OPs or SWs, as in the top-right and bottom-left of Table A.6 of [12]. That said, the most relevant combinations of FCCs are summarized in Table 4: note that these report a very low DF, which is typically important to avoid common-mode failures.

FCCs for RB. First, we aim at devising a group of FCCs that have very low residual misclassifications (thus low DF) and high DIS, to be used as building blocks of the RB FCC. From Table 4, it makes sense to leave IP_DN_RN out of the picture as it has high ε_w. However, it is the only FCC that has reasonably low rejections with $\varphi = 0.032$: using this in a RB will dramatically lower rejections of the RB as well, which may be beneficial in specific scenarios. Thus, we plan for two versions of RBs, where FCCs are ordered by decreasing rejection probability φ: (RB1) SW_AN_GN, SW_VGG_RN, OP_DN, SW_IC_GN, OP_IC and (RB2) SW_AN_GN, SW_VGG_RN, OP_DN, SW_IC_GN, OP_IC, IP_DN_RN. The only difference is that RB2 employs IP_DN_RN as the last FCC, making it have fewer rejections and higher accuracy (but more misclassifications) than RB1.

7.3.1.2 | FCCs for (W)VT, STK. Focusing on DIS, OPs exhibit relatively low disagreement among themselves; see the intersection of OP_DN and OP_IC (0.12) in the 2nd–3rd row, 5th–6th column of Table 4. The IP has a high disagreement with others, and also OP_IC and SW_AN_GN show DIS > 0.20, which is pretty high compared to other scores. Therefore, we choose IP_DN_RN, OP_IC, and SW_AN_GN as FCCs to build VT FCCs. We followed the approach used for tabula data for crafting WVT and STK.

7.3.2 | Discussion: Performance of RB, VT, WVT, STK

Figure 7 reports image data in the same format as what Figure 6 did with tabular data, but is related to FCCs using DN as the main classifier on the FER13 (Figure 7a) and Food (Figure 7b) image datasets. Results of the Flower dataset are omitted as they had a trend very similar to that of Food and provided no additional information. The IP and the RB2 FCCs tend to behave similarly to the DN classifier, with few rejections and many misclassifications. Other FCCs are instead rejecting a noticeable amount of predictions (i.e., the orange-striped bars are longer than those of IP and RB2 in both Figure 7a,b): VT is the one that allows for minimal misclassifications. STK here is not performing particularly well as it has lower accuracy than others, and the misclassifications are still pretty high.

Takeaway 8. The VT FCC allows for the biggest reduction of misclassifications overall, showing the lowest ε_w across tabular and image datasets.

7.4 | Rejections of Unknown Inputs

The last result we present concerns the ability of FCCs to reject unknown inputs in particular. Results we saw up to this point are related to a test set composed of both known and unknown inputs and do not suffice for quantifying the ability of rejecting unknown inputs specifically. Thus, Table 5 reports the rejection probability φ computed for different FCCs in tabular and image datasets, using a test set composed of unknown inputs alone. Ideally, FCCs should have $\varphi = 1$, or 100% rejections of these items.

As shown in Table 5, the results highlight significant performance differences among various FCCs, overall confirming what was already discussed in the previous sections. IP, SW, WVT and, to a lesser extent, RB1 FCCs are more likely to trigger rejections, and end up having a very high rejection probability also in case of unknowns, reaching even perfect rejections in some cases (IP and VT on MetroPT, SW and WVT on FER13). Conversely, RB2 and STK try to balance accuracy and rejections; as such, they result in a very low likelihood of omitting predictions due to unknown inputs (even flat 0 for the three tabular datasets), potentially triggering unhandled misclassifications.

8 | Concluding Remarks

This paper called for a paradigm shift towards the conceptualization, design and deployment of ML classifiers in critical systems,

TABLE 4 | Classification performance, DIS, DF, DR of FCCs used for building RB, VT, WVT, STK image classifiers.

FCC	φ	α_w	ε_w	DIS (best if high)					DF (best if low)					DR (best if low)				
				IP_DN_RN	OP_DN	OP_IC	SW_AN_GN	SW_IC_GN	SW_VGG_RN	IP_DN_RN	OP_DN	OP_IC	SW_AN_GN	SW_IC_GN	SW_VGG_RN	IP_DN_RN	OP_DN	OP_IC	SW_AN_GN	SW_IC_GN	SW_VGG_RN
IP_DN_RN	0.032	0.787	0.181	—	0.19	0.18	0.30	0.17	0.20	—	0.05	0.06	0.05	0.06	0.06	—	0.02	0.01	0.03	0.03	0.03
OP_DN	0.327	0.622	0.051	0.19	—	0.12	0.20	0.13	0.13	0.05	—	0.04	0.04	0.04	0.04	0.02	—	0.23	0.28	0.23	0.25
OP_IC	0.279	0.656	0.065	0.18	0.12	—	0.23	0.02	0.15	0.06	0.04	—	0.04	0.06	0.05	0.01	0.23	—	0.24	0.28	0.21
SW_AN_GN	0.449	0.494	0.056	0.30	0.20	0.23	—	0.21	0.15	0.05	0.04	0.04	—	0.04	0.04	0.03	0.28	0.24	—	0.25	0.30
SW_IC_GN	0.297	0.638	0.064	0.17	0.13	0.02	0.21	—	0.14	0.06	0.04	0.06	0.04	—	0.05	0.03	0.23	0.28	0.25	—	0.23
SW_VGG_RN	0.336	0.594	0.070	0.20	0.13	0.15	0.15	0.14	—	0.06	0.04	0.05	0.04	0.05	—	0.03	0.25	0.21	0.30	0.23	—

Note: Results are averaged across the three tabular datasets used in experiments.

where misclassifications may have catastrophic consequences. Researchers typically strive to craft classifiers with perfect accuracy, which should be always correct and as such never threaten the encompassing system. Unfortunately, this is a very unrealistic goal, as classification tasks are typically complex and may encounter a wide variety of unexpected operating conditions and unknown inputs, for which there are no guarantees of correct functioning.

We advocate that classifiers should not be conceptualized in isolation and only at a later stage deployed into their final operational environment. Instead, they should be conceptualized, designed and evaluated as a building block of their encompassing (critical) system, actively cooperating with additional components to achieve dependability-related properties. Fail-Controlled Classifiers (FCCs) are classifiers that complement a ML algorithm with additional components that primarily aim at suspecting misclassifications, and rejecting the corresponding predictions of the ML classifiers. This changes the failure semantics from unhandled content failures to omissions, which can be managed by system-level mechanisms (e.g., triggering recovery strategies, or bringing the system into a safe state, if possible).

Ideally, all and only misclassifications should be rejected: exceedingly likely rejections may turn FCCs into a liability, making them unpractical in real scenarios. Nine different FCCs are presented, discussed and evaluated in the paper: SCC, WTs, IP, OP, SW, RB, VT, WVT and STK. Each has its own advantages and disadvantages, which are highlighted as Takeaways 1–8 through the paper. Overall, the selection of FCCs for a system depends on factors like availability and the system's criticality. If the system can tolerate more omissions, the IP FCC is a good choice. On the other hand, if achieving the highest accuracy is the priority, the RB FCC performs better. However, these decisions aren't limited to just accuracy or omissions; other factors, such as the environment in which the system operates, also play a crucial role in determining the most suitable FCC.

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 (main classifiers in the paper) against FCCs built on top of these classifiers. The comparison and the discussion that is carried out throughout the paper compare the performance of the classifier against FCCs and do not aim at optimizing performance in a specific scenario. Unless changing the hyperparameters turns it into a perfect classifier, which is unlikely, then FCCs will be able to provide a useful contribution anyway. Also, this is one of the few studies that uses many datasets from both tabular and image domains, ensuring solid and general findings.

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; code is available at [11].

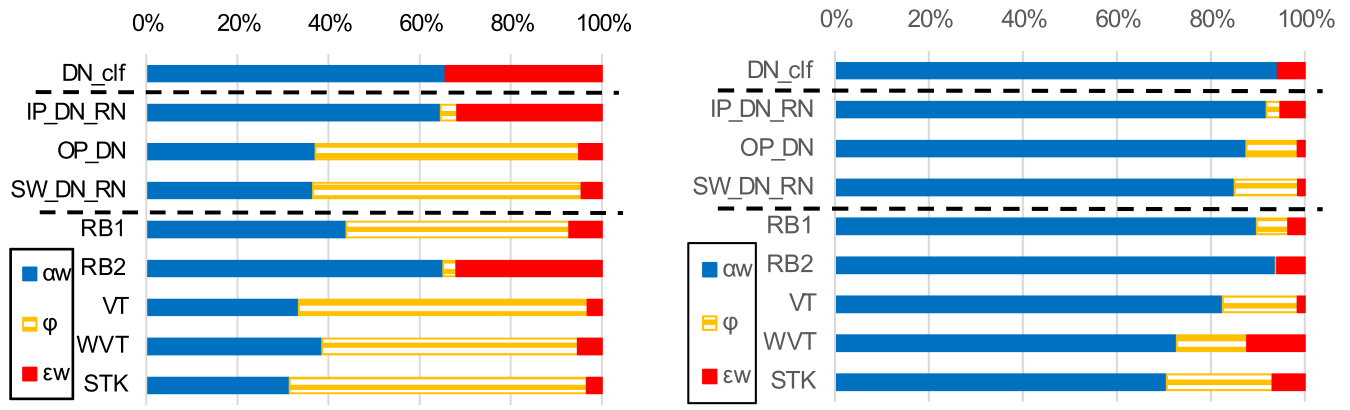

FIGURE 7 | Comparison of FCCs using DN as the main classifier on the FER13 (a, left) and Food (b, left) image datasets. The color scheme complies with that of Figures 5 and 6.

TABLE 5 | Rejection probability φ of unknown inputs for different FCCs.

Tabular datasets	NIDS	Error detection	MetroPT	Image datasets	FER13	Flower	Food
IP_RF_ET	0.83	0.82	1.00	IP_DN_RN	0.99	0.83	0.86
OP_RF	0.00	0.04	0.00	OP_DN	0.90	0.32	0.26
SW_RF_LDA	0.78	0.50	0.98	SW_DN_RN	1.00	0.87	0.89
RB1	0.73	0.48	0.98	RB1	0.82	0.19	0.12
RB2	0.00	0.00	0.00	RB2	0.82	0.17	0.11
VT	0.83	0.82	1.00	VT	0.90	0.37	0.25
WVT	0.78	0.49	0.98	WVT	1.00	0.80	0.86
STK	0.00	0.00	0.00	STK	0.90	0.22	0.20

Author Contributions

Fahad Ahmed Khokhar: and **Tommaso Zoppi:** provided the problem statement and perform Experimentations. **Leonardo Montecchi:** and **Andrea Ceccarelli:** contributed to technical writing and analytical writing. **Andrea Bondavalli:** performed technical proofreading and result gathering.

Acknowledgments

This work was supported in part by the 202297YF75 PRIN 2022 project S2: Safe and Secure Industrial IoT, by the B53D23012930006 PRIN PNRR 2022 project FLEGREA - Federated Learning for Generative Emulation of Advanced Persistent Threats, 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 (Eureka E6085) by the European Union and by the 101194245 Shift2SDV HORIZON-JU-IA HORIZON JU 2024 Innovation Actions. Open access publishing facilitated by Universita degli Studi di Firenze, as part of the Wiley - CRUI-CARE agreement.

Conflicts of Interest

The authors declare no conflicts of interest.

Data Availability Statement

Data sharing not applicable to this article as no datasets were generated or analyzed during the current study.

References

1. J. Guérin, R. S. Ferreira, K. Delmas, and J. Guiochet, "Unifying Evaluation of Machine Learning Safety Monitors," in *2022 IEEE 33rd International Symposium on Software Reliability Engineering (ISSRE)* (IEEE, 2022), 414–422.

2. S. Ö. Arik and T. Pfister, "Tabnet: Attentive Interpretable Tabular Learning," *Proceedings of the AAAI Conference on Artificial Intelligence* 35, no. 8 (2021): 6679–6687.

3. A. Biondi, F. Nesti, G. Cicero, D. Casini, and G. Buttazzo, "A Safe, Secure, and Predictable Software Architecture for Deep Learning in Safety-Critical Systems," *IEEE Embedded Systems Letters* 12, no. 3 (2019): 78–82.

4. V. Chandola, A. Banerjee, and V. Kumar, "Anomaly Detection: A Survey," *ACM Computing Surveys* 41, no. 3 (2009): 1–58.

5. A. Sarıkaya, B. G. Kılıç, and M. Demirci, "RAIDS: Robust Autoencoder-Based Intrusion Detection System Model Against Adversarial Attacks," *Computers & Security* 135 (2023): 103483.

6. M. J. Mirza, C. Buerkle, J. Jarquin, et al., "Robustness of Object Detectors in Degrading Weather Conditions," in *2021 International Intelligent Transportation Systems Conference (ITSC)* (IEEE, 2021), 2719–2724.

7. C. G. Keller, T. Dang, H. Fritz, A. Joos, C. Rabe, and D. M. Gavrila, "Active Pedestrian Safety by Automatic Braking and Evasive Steering," *IEEE Transactions on Intelligent Transportation Systems* 12, no. 4 (2011): 1292–1304.

8. M. Mathias, R. Timofte, R. Benenson, and L. Van Gool, "Traffic Sign Recognition — How Far Are We From the Solution?," in *The 2013 International Joint Conference on Neural Networks (IJCNN)* (IEEE, 2013), 1–8.

9. A. Ceccarelli and F. Secci, "RGB Cameras Failures and Their Effects in Autonomous Driving Applications," *IEEE Transactions on Dependable and Secure Computing* 20 (2022): 2731–2745.

10. B. Sayin, T. Zoppi, N. Marchini, F. A. Khokhar, and A. Passerini, "Bringing Machine Learning Classifiers Into Critical Cyber-Physical Systems: A Matter of Design," *IEEE Access* 13 (2025): 94858–94877.

11. "FCC," accessed January 09, 2024, https://github.com/fahadahmedkhokhar/FCC.git.

12. F. A. Khokhar, T. Zoppi, A. Ceccarelli, L. Montecchi, and A. Bondavalli, "Fail-Controlled Classifiers: A Swiss-Army Knife Towards Trustworthy Systems (Appendices)," 2025, https://doi.org/10.17605/OSF.IO/YU75K.

13. R. Shwartz-Ziv and A. Armon, "Tabular Data: Deep Learning Is Not All You Need," *Information Fusion* 81 (2022): 84–90.

14. L. Grinsztajn, E. Oyallon, and G. Varoquaux, "Why Do Tree-Based Models Still Outperform Deep Learning on Typical Tabular Data?," *Advances in Neural Information Processing Systems* 35 (2022): 507–520.

15. T. Zoppi, S. Gazzini, and A. Ceccarelli, "Anomaly-Based Error and Intrusion Detection in Tabular Data: No DNN Outperforms Tree-Based Classifiers," *Future Generation Computer Systems* 160 (2024): 951–965.

16. T. Zoppi and P. Popov, "Confidence Ensembles: Tabular Data Classifiers on Steroids," *Information Fusion* 120 (2025): 103126.

17. C. M. Bishop, "Pattern Recognition," *Machine Learning* 128, no. 9 (2006).

18. D. Hendrycks and K. Gimpel, "A Baseline for Detecting Misclassified and out-Of-Distribution Examples in Neural Networks," in *International Conference on Learning Representations* (2016).

19. M. Hein, M. Andriushchenko, and J. Bitterwolf, "Why Relu Networks Yield High-Confidence Predictions Far Away From the Training Data and How to Mitigate the Problem," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (IEEE, 2019), 41–50.

20. H. Jiang, B. Kim, M. Guan, and M. Gupta, "To Trust or Not to Trust a Classifier," in *Advances in Neural Information Processing Systems* (Neurips foundation, 2018), 31.

21. X. Y. Zhang, G. S. Xie, X. Li, T. Mei, and C. L. Liu, "A Survey on Learning to Reject," *Proceedings of the IEEE* 111, no. 2 (2023): 185–215.

22. G. J. Hahn and W. Q. Meeker, *Statistical Intervals: A Guide for Practitioners and Researchers*, vol. 541 (John Wiley & Sons, 2017).

23. W. J. Krzanowski, T. C. Bailey, D. Partridge, J. E. Fieldsend, R. M. Everson, and V. Schetinin, "Confidence in Classification a Bayesian Approach," *Journal of Classification* 23, no. 2 (2006): 199–220.

24. B. Lakshminarayanan, A. Pritzel, and C. Blundell, "Safety and Scalable Predictive Uncertainty Estimation Using Deep Ensembles," *Advances in Neural Information Processing Systems* (2017): 6405–6416.

25. Z. Bilgin and M. Gunestas, "Explaining Inaccurate Predictions of Models Through k-Nearest Neighbors," in *ICAART* (ICAART, 2021), 228–236.

26. B. Lakshminarayanan, A. Pritzel, and C. Blundell, "Simple and Scalable Predictive Uncertainty Estimation Using Deep Ensembles," *Advances in Neural Information Processing Systems* 30 (2017): 6405–6416.

27. K. Aslansefat, I. Sorokos, D. Whiting, R. T. Kolagari, and Y. Papadopoulos, "SafeML: Safety Monitoring of Machine Learning Classifiers Through Statistical Difference Measures," in *International Symposium on Model-Based Safety and Assessment* (Springer, 2020), 197–211.

28. G. Rossolini, A. Biondi, and G. Buttazzo, "Increasing the Confidence of Deep Neural Networks by Coverage Analysis," *IEEE Transactions on Software Engineering* 49, no. 2 (2022): 802–815.

29. G. Baye, P. Silva, A. Broggi, L. Fiondella, N. D. Bastian, and G. Kul, "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* (IEEE, 2023), 1–6.

30. R. S. Ferreira, J. Guérin, K. Delmas, J. Guiochet, and H. Waeselynck, "Safety Monitoring of Machine Learning Perception Functions: A Survey," *Computational Intelligence* 41, no. 2 (2025): e70032.

31. K. Lee, K. Lee, H. Lee, and J. Shin, "A Simple Unified Framework for Detecting Out-of-Distribution Samples and Adversarial Attacks," *Advances in Neural Information Processing Systems* 31 (2018): 7167–7177.

32. Y. Zhou, "Rethinking Reconstruction Autoencoder-Based Out-of-Distribution Detection," in *Proc. of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (IEEE, 2022), 7379–7387.

33. B. Biggio, B. Nelson, and P. Laskov, "Poisoning Attacks Against Support Vector Machines," in *Proceedings of the 29th International Conference on Machine Learning* (ICML, 2012), 1807–1814.

34. T. B. Brown, D. Mané, A. Roy, M. Abadi, and J. Gilmer, "Adversarial patch," *arXiv Preprint* (2017): 1712.09665.

35. N. Carlini and D. Wagner, "Towards Evaluating the Robustness of Neural Networks," in *2017 IEEE Symposium on Security and Privacy (sp)* (Ieee, 2017), 39–57.

36. J. Su, Z. Zhang, P. Wu, X. Li, and J. Zhang, "Adversarial Input Detection Based on Critical Transformation Robustness," in *2022 IEEE 33rd International Symposium on Software Reliability Engineering (ISSRE)* (IEEE, 2022), 390–401.

37. R. Sommer and V. Paxson, "Outside the Closed World: On Using Machine Learning for Network Intrusion Detection," in *2010 IEEE Symposium on Security and Privacy* (IEEE, 2010), 305–316.

38. M. Ma, S. Zhang, D. Pei, X. Huang, and H. Dai, "Robust and Rapid Adaption for Concept Drift in Software System Anomaly Detection," in *2018 IEEE 29th International Symposium on Software Reliability Engineering (ISSRE)* (IEEE, 2018), 13–24.

39. X. Ran, M. Xu, L. Mei, Q. Xu, and Q. Liu, "Detecting Out-of-Distribution Samples via Variational Auto-Encoder With Reliable Uncertainty Estimation," *Neural Networks* 145 (2022): 199–208.

40. I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and Harnessing Adversarial Examples," *arXiv Preprint* (2014): 1412.6572.

41. F. Moller, D. Botache, D. Huseljic, F. Heidecker, M. Bieshaar, and B. Sick, "Out-of-Distribution Detection and Generation Using Soft Brownian Offset Sampling and Autoencoders," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (IEEE, 2021), 46–55.

42. V. Bandari, "Proactive Fault Tolerance Through Cloud Failure Prediction Using Machine Learning," *ResearchBerg Review of Science and Technology* 3, no. 1 (2020): 51–65.

43. Z. Gong, P. Zhong, and W. Hu, "Diversity in Machine Learning," *IEEE Access* 7 (2019): 64323–64350.

44. F. Di Giandomenico and L. Strigini, "Adjudicators for Diverse-Redundant Components," in *Proceedings 9th Symposium on Reliable Distributed Systems* (IEEE, 1990), 114–123.

45. M. A. Ganaie, M. Hu, A. K. Malik, M. Tanveer, and P. N. Suganthan, "Ensemble Deep Learning: A Review," *Engineering Applications of Artificial Intelligence* 115 (2022): 105151.

46. S. Džeroski and B. Ženko, "Is Combining Classifiers With Stacking Better Than Selecting the Best One?," *Machine Learning* 54 (2004): 255–273.

47. L. I. Kuncheva and C. J. Whitaker, "Measures of Diversity in Classifier Ensembles and Their Relationship With the Ensemble Accuracy," *Machine Learning* 51 (2003): 181–207.

48. A. Avizienis, J. C. Laprie, B. Randell, and C. Landwehr, "Basic Concepts and Taxonomy of Dependable and Secure Computing," *IEEE Transactions on Dependable and Secure Computing* 1, no. 1 (2004): 11–33.

49. P. W. Koh, S. Sagawa, H. Marklund, et al., "Wilds: A Benchmark of In-the-Wild Distribution Shifts," in *International Conference on Machine Learning* (PMLR, 2021), 5637–5664.

50. A. Tiwari, B. Dutertre, D. Jovanović, et al., "Safety Envelope for Security," in *Proceedings of the 3rd International Conference on High Confidence Networked Systems* (ACM, 2014), 85–94.

51. D. Powell, M. Chérèque, and D. Drackley, "Fault-Tolerance in Delta-4," *ACM SIGOPS Operating Systems Review* 25, no. 2 (1991): 122–125.

52. B. Li, P. Qi, B. Liu, et al., "Trustworthy AI: From Principles to Practices," *ACM Computing Surveys* 55, no. 9 (2023): 1–46.

53. R. Dwivedi, D. Dave, H. Naik, et al., "Explainable AI (XAI): Core Ideas, Techniques, and Solutions," *ACM Computing Surveys* 55, no. 9 (2023): 1–33.

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

55. A. S. Ami, K. Moran, D. Poshyvanyk, and A. Nadkarni, ""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)* (IEEE Computer Society, 2023), 19.

56. I. Lopez, "Exploiting Redundancy and Path Diversity for Railway Signalling Resiliency," in *IEEE International Conference on Intelligent Rail Transportation (ICIRT)* (IEEE, 2016), 432–439.

57. H. Alawad, S. Kaewunruen, and M. An, "Learning From Accidents: Machine Learning for Safety at Railway Stations," *IEEE Access* 8 (2019): 633–648.

58. C. Carmichael, "Triple Module Redundancy Design Techniques for Virtex FPGAs," 2001 Xilinx Application Note XAPP197, 1.

59. T. Zoppi, A. Ceccarelli, and A. Bondavalli, "Ensembling Uncertainty Measures to Improve Safety of Black-Box Classifiers," in *Frontiers in Artificial Intelligence and Applications*, vol. 372 (IOS Press, 2023), 3156–3164.

60. F. Brau, G. Rossolini, A. Biondi, and G. Buttazzo, "Robust-By-Design Classification via Unitary-Gradient Neural Networks," *Proceedings of the AAAI Conference on Artificial Intelligence* 37, no. 12 (2023): 14729–14737.

61. M. Gharib, T. Zoppi, and A. Bondavalli, "On the Properness of Incorporating Binary Classification Machine Learning Algorithms Into Safety-Critical Systems," *IEEE Transactions on Emerging Topics in Computing* 10, no. 4 (2022): 1671–1686.

62. S. Calzavara, L. Cazzaro, C. Lucchese, F. Marcuzzi, and S. Orlando, "Beyond Robustness: Resilience Verification of Tree-Based Classifiers," *Computers & Security* 121 (2022): 102843.

63. C. De Stefano, C. Sansone, and M. Vento, "To Reject or Not to Reject: That Is the Question-An Answer in Case of Neural Classifiers," *IEEE Transactions on Systems, Man and Cybernetics, Part C (Applications and Reviews)* 30, no. 1 (2000): 84–94, https://doi.org/10.1109/5326.827457.

64. C. Chow, "On Optimum Recognition Error and Reject Tradeoff," *IEEE Transactions on Information Theory* 16, no. 1 (1970): 41–46, https://doi.org/10.1109/TIT.1970.1054406.

65. Y. Bahat and G. Shakhnarovich, "Confidence From Invariance to Image Transformations," 2018, http://arxiv.org/abs/1804.00657.

66. A. Mandelbaum and D. Weinshall, "Distance-Based Confidence Score for Neural Network Classifiers," 2017, http://arxiv.org/abs/1709.09844.

67. V. Ocheretny, "Self-Checking Arithmetic Logic Unit With Duplicated Outputs," in *2010 IEEE 16th International on-Line Testing Symposium* (IEEE, 2010), 202–203.

68. M. Psarakis, D. Gizopoulos, E. Sanchez, and M. S. Reorda, "Microprocessor Software-Based Self-Testing," *IEEE Design and Test of Computers* 27, no. 3 (2010): 4–19.

69. S. G. Miremadi, "Two Software Techniques for On-Line Error Detection," in *FTCS* (IEEE, 1992), 328–335.

70. I. David, R. Ginosar, and M. Yoeli, "Self-Timed Is Self-Checking," *Journal of Electronic Testing* 6 (1995): 219–228.

71. D. F. McAllister and M. A. Vouk, "Fault-Tolerant Software Reliability Engineering," in *Handbook of Software Reliability Engineering* (IEEE, 1996), 567–614.

72. C. Ferri, P. Flach, and J. Hernández-Orallo, "Delegating Classifiers," in *Proceedings of the Twenty-First International Conference on Machine Learning* (ACM, 2004), 37.

73. D. H. Wolpert, "Stacked Generalization," *Neural Networks* 5, no. 2 (1992): 241–259.

74. "Scikit-Learn Python Library — Classifiers," accessed April 30, 2024, https://scikit-learn.org/stable/supervised_learning.html.

75. W. A. Falcon, "Pytorch Lightning," 2019 GitHub, 3.

76. L. Bossard, M. Guillaumin, and L. van Gool, "Food-101 — Mining Discriminative Components With Random Forests," in *Computer Vision — ECCV 2014*, vol. 8694, ed. D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars (Springer International Publishing, 2014), 446–461, https://doi.org/10.1007/978-3-319-10599-4_29.

77. T. Zoppi, G. Merlino, A. Ceccarelli, A. Puliafito, and A. Bondavalli, "Anomaly Detectors for Self-Aware Edge and IoT Devices," in *2023 IEEE 23rd International Conference on Software Quality, Reliability, and Security (QRS)* (IEEE, 2023), 24–35.

78. 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, 2021), 1–10.

79. "National Flowers," accessed March 26, 2024, https://www.kaggle.com/datasets/shahidulugvcse/national-flowers.

80. "Challenges in Representation Learning: Facial Expression Recognition Challenge," accessed November 06, 2024, https://kaggle.com/competitions/challenges-in-representation-learning-facial-expression-recognition-challenge.

81. C. Shorten and T. M. Khoshgoftaar, "A Survey on Image Data Augmentation for Deep Learning," *Journal of Big Data* 6, no. 1 (2019): 1–48.