UNIVERSITÀ DEGLI STUDI FIRENZE

UNIVERSITÀ DEGLI STUDI DI PERUGIA

[iNδAM] Istituto Nazionale di Alta Matematica

Università di Firenze, Università di Perugia, INdAM consorziate nel CIAFM

DOTTORATO DI RICERCA
IN MATEMATICA, INFORMATICA, STATISTICA

CURRICULUM IN INFORMATICA
CICLO XXXVIII

Sede amministrativa Università degli Studi di Firenze
Coordinatrice Prof.ssa Alessandra Sestini

Software Architectures for Trustworthy Classifiers

Settore Scientifico Disciplinare INF/01

Dottorando:
Fahad Ahmed Khokhar

Supervisore
Prof. Tommaso Zoppi
Co-Supervisore
Prof. Andrea Ceccarelli

Coordinatrice
Prof.ssa Alessandra Sestini

Anni 2022/2025

ABSTRACT

Modern critical systems increasingly rely on autonomous decision-making and data classification, requiring that such operations be performed safely i.e., without negative consequences for people, infrastructure, or the environment. These tasks are often designed for high accuracy, yet remain vulnerable to misclassifications caused by noise, ambiguity, distribution shifts, or unknown inputs. This thesis proposes a paradigm shift: rather than treating classifiers as isolated components to be perfected individually, they should be complemented with mechanism that enable runtime monitoring triggering coordinated responses and mitigations to uncertain or potentially harmful predictions. This means enabling the rejection of outputs suspected to be incorrect, thereby triggering appropriate mitigation strategies.

To address these challenges, we introduce the concept of Fail-Controlled Classifiers (FCCs), Software Architectures that are capable of correctly classifying, misclassifying, or explicitly rejecting uncertain predictions. FCCs are designed to reject only those predictions that are likely to be erroneous, minimizing the system's exposure to incorrect decisions. This approach is particularly relevant in safety-critical applications such as autonomous driving, robotic surgery, and industrial inspection, where rejection or omitting an output is a proven-in-use design pattern for many existing applications.

After discussing background, we introduce FCCs, relevant evaluation metrics, and several architectural patterns built on the FCC concept, including the Input Processor (IP), Output Processor (OP), and Safety Wrapper (SW). To further strengthen robustness, these single architectures are extended into ensemble solutions through Voting, Stacking, and Recovery Blocks, enabling richer diversity and broader coverage of the input space. Metrics such as Double Fault, Double Reject, and Disagreement guide the selection and combination of FCCs by quantifying how components differ in their predictions. Building on these ideas, we introduce SPROUT (Safety wraPper thROugh ensembles of UncertainTy measures), a safety monitor that continuously computes multiple uncertainty indicators to identify and reject classifier outputs that cannot be trusted. SPROUT is model-agnostic, fully black-box, and applicable to binary and multiclass tasks over tabular or image data.

To ensure that findings of this thesis extend beyond standalone systems, we embed FCCs and SPROUT within a collaborative learning paradigm, specifically Federated Learning (FL). While traditional FL requires clients to adopt the same model architecture and share gradients or weights which impose

limitation in the privacy, and industrial deployment. Our Trustworthy Black-Box Federated Learning (TBB-FL) allows clients to train any model they choose, sharing only executable versions treated as black-box components.

Extensive experimentation across a wide range of datasets, classifiers, and parameter configurations confirms that the proposed architectures consistently reject a significant proportion of misclassifications and, in some scenarios, detect all incorrect predictions. TBB-FL further demonstrates substantial reductions in misclassifications compared to traditional FL, while retaining strong predictive performance and enhanced privacy. To promote reproducibility and real-world applicability, we provide an open-source library equipped with pre-trained models and ready-to-run case studies, offering a practical pathway for trustworthy ML integration in both centralized and federated safety-critical systems.

ACKNOWLEDGEMENTS

My participation in this Ph.D. has been an extraordinary life-changing experience, and I would not have been able to do it without the support and guidance of many individuals. First and foremost, with great gratitude, I extend my sincere appreciation to my supervisor, Tommaso Zoppi, Department of Mathematics and Informatics for his advice, encouragement, constant support, continuous guidance, and valuable suggestion during my Ph.D.

I would also wish to express my gratitude to Prof. Andrea Bondavalli, Andrea Ceccarelli, and Muhammad Atif for extended discussions, valuable suggestions, and their continuous support from the first day that contributed greatly to the improvement of the thesis. I would also like to thank Paolo Lollini, who was a great help from the moment I received my doctoral offer until I arrived in Italy. Moreover, Prof. Andrea Bondavalli and Tommaso Zoppi gave me a course on "Safe Machine Learning: Are we there yet?", which was very interesting and provided me with a lot of new information. Andrea Ceccarelli taught me an important course " Architecting Cyber-Physical Systems of Systems" to understand many new aspects of Safety Critical Systems and System of Systems that gave me the ability to understand and analyze the behavior of different real time systems. I have always found them to be very friendly and helpful and their doors were always open to me when I needed advice and support.

I am also thankful to the RCL members Tommaso Puccetti, Marzieh Kordi, Francesco Mariotti, and Nawaz Abdullah with whom I shared the lab during these three years of my Ph.D. and always found them very welcoming and cooperative.

Finally, I would like to thank my parents for their love, support, and continuous sacrifices from my childhood till now to always see me better than yesterday. Without them, this day would not have been possible. I would also like to thank my siblings and Ms. Kainat Naeem for their support and good times in my life

LIST OF RELEVANT PUBLICATIONS

This section highlights the publication produced in the context of the research work done and awards in this thesis.

ACCEPTED

1. **KhoKhar, F. A.**, Zoppi, T., & Shah, J. H. (2025, November). Orchestrating Fail-Safe, Black-Box Models Within Federated Learning Scenarios. In *2025 IEEE 30th Pacific Rim International Symposium on Dependable Computing (PRDC)* (pp. 45-57). IEEE.

2. **Khokhar, F. A.**, T. Zoppi, A. Ceccarelli, L. Montecchi, and A. Bondavalli, " Fail-Controlled Classifiers: A Swiss-Army Knife Toward Trustworthy Systems," *Software: Practice and Experience* (2025): 1–21, https://doi.org/10.1002/spe.70033.

3. Zoppi, T., **Khokhar, F. A.**, Ceccarelli, A., & Bondavalli, A. (2024). Position paper—Bringing classifiers into critical systems: Are we barking up the wrong tree? In International Conference on Computer Safety, Reliability, and Security (pp. 351–357). Springer. https://doi.org/10.1007/978-3-031-44696-2_28

4. Zoppi, T., **Khokhar, F. A.**, Montecchi, L., Ceccarelli, A., & Bondavalli, A. (2024). Fail-controlled classifiers: Do they know when they don't know? In 29th IEEE Pacific Rim International Symposium on Dependable Computing (PRDC 2024). IEEE. https://doi.org/10.1109/PRDC60859.2024.00014

5. Sayin, B., Zoppi, T., Marchini, N., **Khokhar, F. A.**, & Passerini, A. (2025). Bringing machine learning classifiers into critical cyber-physical systems: A matter of design. IEEE Access. IEEE. https://doi.org/10.1109/ACCESS.2025.00000

6. Zoppi, T., Atif, M., **Khokhar, F. A**. & Bondavalli, A. (2025). Back to the future: Refurbishing critical software engineering to enable trustworthy classification. In *Proceedings of the 41th ACM/SIGAPP Symposium on Applied Computing*

SUBMITTED

1. **Khokhar, F. A.**, Zoppi, T., Cennini, L., Ceccarelli, A. & Bondavalli, A. (2025). Safety monitors for black-box classifiers: Design, framework and benchmark. Scientific Reports.

2. **Khokhar, F. A.**,Zoppi, T.,Shah, J. H. (2025). Towards Trustworthy Black-Box Federated Learning for Critical Applications

AWARDS

- **Best Paper Award** for the paper titled *"Orchestrating Fail-Safe, Black-Box Models within Federated Learning Scenarios"* at the 30th IEEE Pacific Rim International Symposium on Dependable Computing (PRDC 2025).

CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ACRONYMS

Acronym	Abbreviation
FCC	Fail Controlled Classifier
UM	Uncertainty Measures
ML	Machine Learning
DNN	Deep Neural Network
Clf	Classifier
RB	Recovery Block
IP	Input Processor
OP	Output Processor
SW	Safety Wrapper
VT	Voting
WVT	Weighted Voting
STK	Stacking
SPROUT	Safety wraPper thROugh ensembles of UncertainTy measures
FL	Federated Learning
TBB-FL	Trustworthy Black Box Federated Learning
Chk_c	Checking Classifier
Sup	Supervised
Uns	Un-Supervised
Img	Image
Tab	Tabular
MTR	Monitor Train
MTE	Monitor Test
Misc	Misclassification
Adj	Adjudicator
CC	Confidence Calculator
BBM	BlackBox Model
SCM	Self-Checking Model (combination of BBM and ConfC)

1 INTRODUCTION

We are at the dawn of the fifth industrial revolution [1], where retail and business logic are meant to synergize with sustainability and inclusivity for building human-centered autonomous systems, relieving the human workforce from time-consuming, laborious, and often hazardous tasks for the social good. Examples include, but are not limited to: fully autonomous driving, computer-guided robotic surgery, mobile robots for inspections and surveillance, optimized power management and generation, and manufacturing robots. All these cutting-edge innovations require trustworthy decision-making processes to provide their intended functionality without harming the health of people, infrastructures, environment, or causing financial losses. Decisions are typically a matter of choosing where to go, what to do, what to buy, or if an action is safe to perform. This often defines an underlying classification problem, where an erroneous prediction (i.e., misclassification) may even have potentially devastating detrimental effects e.g., undetected security breach [2], wrong recognition of a "STOP" traffic sign of undetected pedestrian in a trajectory for autonomous driving [3].

Nowadays, the real challenge system and software architects are dealing with is integrating components that perform classification (referred to as "classifiers" in the thesis) into critical systems while ensuringthat 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 [4], [5], [6], [7], [8]. Moreover, classifiers can perform high-quality classification of images, which is of paramount importance for obstacle detection [9] and traffic sign recognition [10] 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 [11]. In the last decade, academia, industry, and National governments hugely invested in methodologies, mechanisms, and tools to embed classifiers into ICT systems, especially critical ones. Regardless of how much effort we put into building classifiers that are more and more accurate, they could still end up predicting a wrong class for a given input data point, i.e., a *misclassification*.

1.1 Classifiers with a Reject Option

Decades of research and practice on classifiers with reject option provided us with plenty of classifiers that are meant to always output a prediction. Supervised Machine Learning (ML) classifiers and particularly those based on Deep Neural Networks (DNNs) were proven to achieve benchmark-levelclassification 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 [12], [13], despite alternatives based on DNNs exists [8], [14]. Conversely, image classification employs DNNs, which can learn strong features from pixel maps [15]. For unsupervised classification, which applies even in the presence of unlabeled training data, options are clustering, density-based, statistical, neural networks (self-organizing maps or autoencoders) algorithms, recently benchmarked in [16], [17].

Typically classifiers "bet" on a prediction they are unsure of. This best-effort behaviour does not pair well with critical systems, which require guarantees of correct component and system-level behaviour e.g., the probability of failure on demand of a critical component or system should be proven to be lower than specific thresholds.

It would be beneficial to change the failure semantics of classifiers from uncontrolled content failures (i.e., misclassifications) to omission, or rejection, failures [18]. 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 erroneously rejected in the process. Ways to build fail-controlled components [19] are well-known in the literature and often rely on safety wrappers or monitors [4]. 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[20]. Finding trade-offs between safety and availability is of utmost importance when dealing with critical systems [21]: this approach is not different.

1.2 Main Contribution of Thesis

This section describes the main contributions of the thesis.

- In the literature, many solutions aim to enhance trust in safety-critical systems by reducing misclassifications, since trust does not imply correctness of system outputs. However, most

techniques rely on a single classifier or confidence metric and struggle with unknown, anomalous, out-of-distribution, or adversarial inputs. To address this gap, we introduce Fail-Controlled Classifiers (FCCs), building upon existing classifiers with abstain or reject options, which allow predictions to be correct, incorrect, or explicitly rejected, transforming uncontrolled failures into controlled omissions.

- We proposed a suite of software architectures for building FCCs**,** including Self-Checking Classifiers (SCC), Watchdog Timers (WT), Input Processors (IP), Output Processors (OP), Safety Wrappers (SW), Recovery Blocks (RB), weighted and non-weighted voting (VT, WVT), and Stacking (STK), evaluated on both tabular and image datasets. Unlike existing approaches, these architectures are classifier-agnostic, black-box compliant, and applicable to binary or multiclass settings, providing a unified pathway to trustworthy ML in critical systems.

- Existing works often address uncertainty or failure detection only in limited or white-box scenarios. In contrast, we introduce SPROUT, a black-box Safety Wrapper based on ensembles of uncertainty measures and adjudication rules, capable of detecting risky predictions without requiring any internal access to the classifier.

- To extend trustworthiness to collaborative environments, we introduce Trustworthy Black-Box Federated Learning (TBB-FL), enabling clients with heterogeneous black-box models to contribute to a safe global model. Unlike traditional FL, which aggregates predictions without considering uncertainty or failure semantics, TBB-FL incorporates uncertainty-aware aggregation and rejection strategies, ensuring reliable global behavior despite data heterogeneity and client diversity.

- The thesis includes extensive empirical validation across 31 tabular and 7 image datasets, showing that FCCs, SPROUT, and TBB-FL significantly reduce misclassifications, detect harmful predictions, and maintain robustness under unknown inputs, anomalies, OOD data, and distribution shifts. In some cases, the proposed architectures detect all erroneous predictions. To support reproducibility and adoption, an open-source library with pre-trained models, datasets, and code is released**,** providing a practical toolbox for deploying trustworthy ML in real-world safety-critical applications.

1.3 Structure of Thesis

This section provides an overview of how the thesis is organized, outlining the ideas, methodologies, and contributions presented across the chapters. Each chapter builds upon the previous ones to form a pathway toward designing trustworthy, fail-controlled, and uncertainty-aware machine learning systems for safety-critical applications.

Chapter 2 introduces the foundational concepts of dependable and trustworthy systems, covering classifiers, uncertainty estimation, out-of-distribution inputs, adversarial threats, and safety monitoring principles. It establishes the theoretical basis required for the proposed architectures.

Chapter 3 describes the tabular and image datasets used throughout the thesis and presents the supervised, unsupervised, and deep learning classifiers employed. This chapter provides the experimental foundation for all later analyses.

The core contributions begin in Chapter 4, which motivates the shift from traditional correctness-oriented classifiers to trust-oriented designs suitable for critical systems. It explains why rejection-based behavior enhances safety by converting uncontrolled failures into controlled omissions.

Chapter 5 presents the main architectural contribution of the thesis i.e., Fail-Controlled Classifiers (FCCs) and presents several architectural patterns for safe prediction rejection. Chapter 6 complements this by presenting the experimental evaluation of FCCs on diverse datasets, demonstrating their effectiveness in reducing misclassifications.

Building upon FCCs, Chapter 7 introduces SPROUT, a black-box Safety Wrapper based on ensembles of uncertainty measures and adjudication strategies. It benchmarks black-box safety monitoring across multiple classifiers and modalities, demonstrating robustness in detecting harmful inputs before they propagate.

Expanding the scope to collaborative learning, Chapter 8 proposes Trustworthy Black-Box Federated Learning (TBB-FL) which embedded FCC principles and uncertainty-aware adjudication. Experiments show improved safety and reliability compared to traditional FL.

Finally, Chapter 9 concludes the thesis by summarizing the findings, outlining the implications of fail-controlled and uncertainty-aware ML architectures for real-world critical systems.

2 BACKGROUND AND RELATED WORKS

This chapter provides a comprehensive overview of the foundational concepts of *dependability* and *trust* in software systems, outlining key design patterns used in critical and reliable system engineering. It further discusses the fundamentals of machine learning classifiers, their confidence estimation, and challenges arising from unknown inputs. Finally, the chapter introduces ensemble diversity principles and metrics that guide the construction of trustworthy, fault-tolerant decision-making systems.

2.1 Basics on Dependability and Trust

The thesis focuses on building trustworthy software systems tasked of making decisions, composed of different components that are treated as individually unreliable components to be integrated into a broader, reliable, and trustworthy decision-making system. Thus, it makes sense to start reviewing literature by exploring common design patterns for reliable, critical systems engineering [19], [22], [23]. The biggest body of work in this discipline was produced in the 1980s and 1990s, when systems were transitioning from being hardware-intensive to software-intensive: design guidelines needed to adapt to this technological shift. Particularly, the following concepts [19] were and are of utmost importance:

- Dependability: avoiding service failures that are more frequent or severe than acceptable.
- Reliability/Availability: continuity/readiness for correct service.
- Robustness: dependability with respect to external faults, or the ability of a computer system to cope with erroneous or unexpected inputs.
- Safety: the absence of catastrophic failures.
- Trust is the accepted dependence of system A on system B: the extent to which System A's dependability is (or would be) affected by that of System B.

In a nutshell, an encompassing system is happily dependent on a trustworthy system, as they have guarantees that this dependence will not trigger service failures that are more frequent or severe than those that were already happening in the encompassing system. A trustworthy system should be designed, implemented, verified, and validated to achieve this property. Achieving trust is not easy and may require building the system in a way that is robust to unknown inputs and ends up having safe behavior even in the presence of prediction errors. **Trust does not imply correctness**: a component may reject a prediction that

is likely to be wrong (i.e., better safe than sorry), making the output not correct, not wrong, but still trustable. This changes the failure semantics of components or functions from uncontrolled content failures to omission failures. Ideally, we want to omit all and only erroneous output: on the downside, the availability of said component may be negatively affected in case the rate of omissions gets extremely high.

2.2 Design Patterns for Critical Software/Systems Engineering

This is the baseline for many design patterns initially intended for critical and reliable system design that are relevant within this thesis. Ways to build fail-controlled components and systems are well-known in the literature and often rely on safety wrappers or monitors [4], [5], [22]. Redundancy is a key principle in fault-tolerant systems, ensuring continuous operation by adding extra components or systems as backups in case of failure [24].

Safety wrappers are intended to complement an existing critical component or task by continuously checking invariants, or processing additional data to detect dangerous behaviors and block the erroneous output of the component before it is propagated through the encompassing system.

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 on the outputs of the replicas [25], or employing testing libraries that are periodically exercised on both hardware and software equipment [26]. Whenever one of these checks fails, the target component is deemed as failed and in needs to be replaced or fixed. Sometimes, this check is implemented via redundant systems or components that perform the same task independently: if replicas agree on the same output, the result is accepted. If there is a disagreement, further checks or fail-safe mechanisms are triggered to ensure safety.

Watchdog timers measure the length of the execution of a target function to understand if the elapsed time conforms with expectations [27]. In case the function completes too early, the watchdog timer generates an alert that we can use to trigger omissions. If the function completes too late, it indirectly delivers an omission as well.

N-Version Programming (NVP, [28]) exercises different replicas of the target function in parallel, with each replica acting independently on the same inputs. Their outputs are sent to an adjudicator [29] which delivers the unified output by implementing thresholds (e.g., voting), invariants, custom rules, or a complex

function, even an ML algorithm. A variant of NVP is recovery blocks [30], which orchestrate replicas sequentially whenever the output of the main function, or that of the previous replicas, cannot be trusted. In modern ensemble learning, the same principle appears through meta-learners that combine outputs (meta-features) from diverse base classifiers using methods such as voting, stacking, cascading, or arbitrating [31]. Voting is widely used in bagging and boosting, while stacking is effective for heterogeneous models. However, combining replicas or classifiers must be done carefully, as misleading members of the ensemble can bias the final output toward misclassification if not properly managed [32]. However, voting was explored in different formulations (i.e., hard, soft, weighted), and can be even used to build a hierarchical agreement structure that is known as n-self-checking-programming [22].

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* [33]. 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 recognition scenarios. N-version Programming (NVP), or software redundancy, runs multiple replicas of a software on diverse hardware, operating systems or diverse versions of the software itself.

The idea of achieving fault tolerance through design diversity emerged as a foundational concept in dependable computing, where independently developed software versions are used to mitigate the risk of shared design faults. In [34] authors introduced the principle that multiple, diverse implementations of the same specification could enhance system reliability, provided that their failures remain statistically independent. This approach matured into NVP, later formalized by [28], as a structured design paradigm emphasizing independent design, verification, and adjudication across versions. However, authors of [35] empirically demonstrated that true independence among software versions is difficult to achieve, as human developers may introduce correlated faults due to similar interpretations or misunderstandings of the same requirements. Their findings marked a turning point, highlighting that while design diversity can reduce failure probability, it does not guarantee failure independence, thus requiring careful validation of version behavior and assumptions about independence.

Subsequent research focused on modeling and quantifying diversity as a measurable property rather than a design assumption. Moreover, [36] proposed probabilistic models to evaluate how different forms of design diversity contribute to overall system reliability, emphasizing that diversity effectiveness depends on the likelihood of correlated failures across versions. In parallel, a broader perspective by classifying diversity into *design*, *data*, and *execution* dimensions, arguing that diversity can be achieved not only through independent development but also through runtime variation and input manipulation [37]. These studies collectively underline that dependability and trust in critical software systems arise not from redundancy alone but from well-structured diversity that minimizes common-mode failures. In modern trustworthy AI, these classical principles remain central to ensuring robust, fail-controlled behavior in ensemble models and safety-aware machine learning architectures.

2.3 Basics on Classifiers

Classifiers exist in many forms, from traditional rule-based systems to modern machine learning models, and they are designed to always produce an output for a given input. In supervised learning, tree-based ensembles tend to perform best on structured tabular data, while deep neural networks are highly effective for image data. When labels are unavailable, unsupervised methods such as clustering, density estimators, and neural reconstruction models are used to separate or group data based on inherent patterns. Misclassifications are known to be triggered by specific events as fake input data (including adversarial attacks [38]), anomalies, out-of-distribution (OOD) data, and distribution shifts [39], which tend to deceive models into incorrect classifications. Data points lying close to decision boundaries are also problematic, leading to instability in the model's outputs.

Regardless of its specific algorithm, a classifier *clf* first devises a mathematical model from a training dataset [40], which contains a given amount of data points. Each input data point *dp* contains a set of *f* feature values, where each feature value is an image pixel / channel or a floating point number dp_j with $0 \leq j < f$ and describes a specific input of the classification problem. The dataset is usually partitioned into training and test sets. The training set is used to learn the model of the classifier;cross-validation is used for model selection and hyperparameter tuning; data in the test set is used to quantify the classification performance of the trained model. Training data requires labels only in case of supervised classifiers [14]. Once training is completed, the classifier makes its prediction on a data point *dp* as:

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

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

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

where *arg_max* is the argument of the maxima, that returns the position of the maximum value in the *dp_prob* array [14]; this position corresponds to the class predicted for the data point. The classification performance is usually computed by applying *clf* to data points in a test dataset and computing metrics such as accuracy, i.e., the percentage of correct predictions of a classifier clf over all predictions. Noticeably, *1 - accuracy* quantifies the misclassification probability by difference.

2.4 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 completely. The sup-optimal choice of a suitable ML algorithm(s), the poor availability or quality of training data, and biased pre-processing and analyses may all constitute additional causes of misclassifications that instead should be avoided. On top of that, there may be other problems due to the operational environment in which the classifier is expected to operate [41], which may expose classifiers to unknown, unexpected inputs.

2.4.1 Out-of-Distribution Data, Anomalies and Outliers

Systems and software components may encounter anomalous inputs or operating conditions [6], [39], [41], [42] 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 empirical distribution of training data: thus, the behaviour of the classifier may become unpredictable [39] 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 [39], [42], [43].

2.4.2 Adversarial Attacks

Second, classifiers may operate in situations in which malicious entities may be willing to actively disturb the behaviour of classifiers, triggering misclassifications with targeted attacks. For image classification, this is the case of adversarial attacks, whose popularity saw an outstanding growth in the last decade after the first findings on data poisoning [44], adversarial patches [45], and gradient-based attacks [46]. 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 [42], [47], but nothing that can be considered proven-in-use yet.

2.4.3 Distribution Shifts, Emerging and Unexpected Events

Third, Machine Learning often works under the *Independent and Identically Distributed* (IID) or "closed world" assumption [48]. In a closed world, train, validation and test data are independently and randomly sampled from the same underlying distribution. However, most (if not all) the operational environments are dynamic, evolving, or complex enough to make this assumption very restrictive and valid only in a very small subset of static standalone systems. As a result, research moved to deploying classifiers that go beyond this assumption and are meant to operate in an open-world [48] where test data may be distributed (slightly) differently from training and validation data. These classifiers have to be robust to environmental changes, distribution shifts, emerging and unexpected behaviours, and even changes in the threat landscape [49].

2.4.4 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 scenarios and artefacts. 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 than 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 test. Here, *a4* is not a real zero-day attack (it is logged in the existing dataset), but it is a zero-day [50] 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 i.e., if it belongs or not to a target distribution [51]. Another approach is applying gradient-descent techniques as Fast Gradient Signed Method (FGSM) [38], which introduces adversarial perturbations to the dataset. The process adds a scaled sign of the gradient to the input, as shown below.

$$Unknown\ Input = I_{(x,y)} + \gamma * sign(\nabla_x L(\theta, I))$$

Where $I_{(x,y)}$: input with size x and y, γ: multiplier of the perturbation, θ: model parameters, and L: loss obtained by the model. Other options rely on statistical methods: a Soft Brownian Offset [52] 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 Learning to Reject

A recent survey summarizes possible approaches for rejecting predictions of classifiers [53]. 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 [53], 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 is seen as either in-distribution or unknown, i.e., out-of-distribution. Unknown data fed into a classifier may result in unpredictable outcomes: despite not necessarily leading to misclassifications, sometimes it is better to be safe than sorry and discard predictions related to unknown data.
- *Fake rejection*, where unnatural, modified, and forged samples need to be rejected in order to guarantee the correct behaviour 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 [54]. Other strategies exploit the concept of stability under perturbation, i.e., a correct decision should be invariant to small perturbations [55], which is easier to apply in image classification compared to dealing with tabular data. Specifically for unknown rejection, it has been proposed to compute a distance score as the Euclidean distance between the input and the k neighbours from the training set and reject if such a score is exceedingly high [56].

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 [57]. Researchers and practitioners are actively investigating ways to quantify uncertainty and learning to reject [53] misclassifications. Some approaches use confidence intervals [58] or the Bayes' theorem [59]. Works as [60] 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 [39], 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 [57] and [61], where authors paired a k-Nearest Neighbor classifier with a neural network to compute uncertainty. The work [34] computed the cross-entropy on the SoftMax probabilities of a neural network and used it to detect out-of-distribution input data that is likely to be misclassified. Authors of [60] use probabilistic neural networks to model predictive distributions and, as a result, quantify predictive uncertainty using methods such as adversarial training. In [62], 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 [63] suggests a

simple monitoring architecture to improve the model's robustness to different harmful inputs, particularly those resulting from adversarial attacks on neural networks and the authors of [5] combine a voting strategy with a safety monitor to build a safe and secure classifier for application in embedded systems.

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

Ideally, we want classifiers to be highly confident about predictions that turn out to be correct, and show low confidence for all and only the predictions that will instead be misclassifications. However, classifiers may have high confidence even when misclassifying data points e.g., "*neural networks which yield a piecewise linear classifier function […] produce almost always high confidence predictions far away from the training data*" [64].

2.6 Existing Safety Monitors for Classifiers

Recently, there have been few studies that specifically aim at building safety monitors for classifiers [60], [61], [62], [63]. The paper [61] ran a k-nearest neighbor classifier in parallel to a Deep Neural Network (DNN) to detect misclassifications. The paper [62] conducted an active monitoring of the behavior and the operational context of the data-driven system based on distance measures of the Empirical Cumulative Distribution Function, and used them as triggers for the safety monitor. The work [60] used probabilistic neural networks to model predictive distributions and thus estimate misclassifications thanks to adversarial training. This technique performed well for image classifiers. Lastly, [63] proposed a lightweight monitoring architecture to enhance the model robustness against different unsafe inputs, especially those due to adversarial attacks to neural networks. The logic to detect misclassifications revolved around an analysis of activation patterns of neurons in the layers of a specific neural network, which authors showed to be distinguishable in case of an adversarial input. It is worth mentioning that these safety monitors [39], [61], [63] assume a white-box classifier, often rely on extensive knowledge of the classifier (e.g., [63] requires the structure of the DNN to be disclosed), require the implementation of complex and multi-step

processes [62], or apply only to specific types of input data (e.g., [60] are specifically crafted for image classifiers). Instead, we seek for an approach which i) applies to any classifier, which is seen as a black-box; ii) is easy to automatize, adopt and exercise to novel datasets or systems; iii) applies to binary and multiclass classification problems, and iv) does not have any constraint on the input data and as such works with tabular and image datasets. Recent research has extensively explored methods to identify and reject inputs that are likely to cause misclassifications in machine learning models. For instance, a baseline approach for detecting out-of-distribution (OOD) and misclassified examples using softmax probabilities was proposed in [65], highlighting the importance of uncertainty estimation in model predictions. Similarly, [39] and [66] introduced techniques such as temperature scaling to improve OOD detection, demonstrating its effectiveness across various datasets. In the context of adversarial attacks, [38] pioneered the study of adversarial examples, revealing the vulnerability of neural networks to small, carefully crafted perturbations. Building on this, [66] formalized adversarial robustness through adversarial training, showing that models can be made more resilient to such attacks. Additionally, [67] investigated model calibration, emphasizing the challenges posed by inputs near decision boundaries and proposing methods to better quantify prediction confidence. Collectively, these works underscore the need for robust mechanisms to handle fake data, anomalies, distribution shifts, and other problematic inputs, ensuring the reliability and trustworthiness of machine learning systems in real-world applications.

3 MATERIAL AND METHODS

In this chapter, we describe the datasets and classifiers used throughout our study, covering both tabular and image data. The tabular datasets span multiple application domains, while the image datasets include diverse RGB image collections commonly used for classification research. All datasets are preprocessed and split into training, validation, and test sets. We also discussed the classifiers adopted in our experiments, including supervised and unsupervised methods for tabular data, as well as a set of deep neural networks used for image classification. Together, these datasets and models form the experimental foundation for all analyses conducted in this thesis.

3.1 Datasets

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

3.1.1 Tabular Datasets

We consider the tabular datasets in Table 1: 11 datasets of network intrusion detection, 10 datasets of sensor spoofing attacks to biometric authentication, 7 datasets related to hardware monitoring for failure prediction, 3 datasets related to IoT systems. All of them will be used for supervised classification, but only those 9 containing two classes might be used for unsupervised classification, which is constrained to be binary by design.

Network Intrusion Detection (NIDS). We select labelled datasets on network intrusions: NSL-KDD [68], ISCX12 [69], UNSW-NB15 [70], UGR16 [71], NGIDS-DS and ADFANet [72], AndMal17 [69], CIDDS-001 [73], CICIDS17 and CICIDS18 [74], SDN20 [75]. All datasets report normal data points and data points collected while the system is under attack. Features are mostly numeric features extracted by monitoring network traffic (e.g., bytes received per second, number of packets).

Spoofing Attacks to Biometric Authentication. We gather datasets related to biometric spoofing attacks from [31], including Fingerprint [76], Hand Gesture [77], Electrodermal Activity [78], Heart Rate [79], Human Gait [80], Keystroke [81], Voice [82], Face [83]. These datasets contain tabular features extracted from biometric samples provided by users. The dataset are enriched with the data augmentation process where the authors performed an attack injection campaign to generate the effects of 4 sensor

spoofing attacks. Therefore, the 10 biometric datasets contain data belonging to 5 different classes: the normal class and 4 spoofing attack classes.

Hardware Failure Prediction. Classifiers may also be used to detect the failure of hardware components. We gather datasets of performance monitoring of hard disks, where data points are labeled as failure only if the monitored hard drive is in a fail state or going to fail thereafter. BackBlaze manufacturer [84] makes 5 years of hard drive data available to the public, reporting labeled data related to many SMART indicators of hard drives, while another source of hard drive data came from the BAIDU [85] Kaggle competition whose input datasets are still available for download. Lastly, we used a dataset about the mechanical failure of electrical machinery in power plants [86].

IoT Systems Data (IoT). These datasets [87], [88], [89] are gathered by monitoring IoT systems: i) a network of sensors, ii) a distributed control systems of a power plant controlling a turbine, boilers, and water treatment systems, and iii) railroad trucks equipped for sensors to monitor brake pressure. Classifiers applied to the first dataset detect intrusions, while others act as error detectors and failure predictors.

Table 1. Name, number of classes, amount of features and # of data points of datasets used for experiments.

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

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

Then, we analyzed all datasets to remove features that are specific of the setup that was followed to gather data, namely: Timestamp, ID, experiment number, if any. Those features should be disregarded for

classification purposes as they carry information about the experiments to build the dataset instead of being related to the system: classifiers using these features may learn how experiments were made instead of how the system behaves. Lastly, we removed duplicated columns (if any), and zero-filled all blank values in the BackBlaze datasets.

3.1.2 Image Datasets

We are using 7 different RGB (3 channel) datasets: CIFAR10 (32x32x3, 10 classes [90]), FER2013 (48x48x3, 7 classes [91]), FOOD-101 (512x512x3, 10 classes [92]), Flower (variable size, 9 classes [93]), Tencent 100K (variable size, 100 classes [94]), CCTSDB2021 (variable size, 3 classes [95]), and NCT-CRC-HE-100K (variable size, 9 classes [96]) for training and testing the proposed technique. These were the datasets of choice as they are quite small, contain images in different RGB formats and are used a lot in the literature.

After the train-test split, we augment training data through a composition of transformations: images are resized to a uniform size of 320x320 pixels, followed by random horizontal flipping and rotations up to 30 degrees to introduce spatial variability. Color adjustments such as brightness, contrast, saturation, and hue variations are applied to simulate diverse lighting conditions. The augmented images are then converted to tensors and normalized using the standard mean ([0.485, 0.456, 0.406]) and standard deviation ([0.229, 0.224, 0.225]) values derived from the ImageNet dataset. This normalization is performed in RGB format, ensuring consistency with the input distribution of pretrained models and enhancing compatibility with their feature extraction layers [15]. Resizing, normalization and conversion to tensors are applied to test data for consistency.

3.2 Classifiers

There are several classifiers used in machine learning; some are integrated within neural networks, while others are based on different approaches. In this section we describe the different types of classifiers that are used in whole study.

3.2.1 Tabular Classifiers

Amongst the many alternatives, we choose the following very well-known classifiers for tabular data. Tabular data classifiers are preferably built over ensembles of decision trees or statistical ML algorithms

[13]. All tabular classifiers are exercised using their default parameters from the *scikit-learn*, and *xgboost* Python libraries [97].

Supervised Classifiers. Decision Trees (DT) and their ensembles as Random Forests (RF), Gradient Boosting (GB), eXtreme Gradient Boosting (XGB), are the de-facto standard [13] for supervised classification of tabular data. For completeness, we exercised Logistic Regression (LR), Naïve Bayes (NB), *Gaussian Naïve Bayes (GNB), Extra Trees (ET)* and Linear Discriminant Analysis (LDA), alongside TabNet [8] and FastAI [14] neural networks, which are explicitly crafted to deal with tabular data. HyperOpt does not work with neural networks: in this case, we let the hyperparameter optimizer that is embedded in FastAI to automatically tune its parameters. For TabNet, we ran grid searches with 108 combinations of the following parameters and values: Learning rate [e^{-5}, e^{-3}, e^{-1}], Batch size [512, 1024, 2048], Max Epochs [20, 50, 100], patience (for early stopping) [5, 8], target metric [mcc, accuracy].

Unsupervised classifiers. We select 8 algorithms belonging to different families [17] from the Python framework *PyOD* [16], namely: FastABOD (Angle-based), HBOS (Proximity), COPOD (Probabilistic), CBLOF (Clustering), PCA and MCD (Linear Model), IForest and SUOD (Ensembles). Note that unsupervised classifiers can only perform binary classification.

3.2.2 Image Classifiers

Many DNN classifiers were proposed in the last decades, often with more and more demanding requirements concerning resource usage, posing challenges in scenarios where computational efficiency may be a limiting factor. Therefore, we selected DNNs for image classification that are relatively small and not excessively resource-hungry. We started from pretrained AlexNet, ResNet50, DenseNet, VGG11, GoogLeNet and InceptionV3 models [15], which are then used for transfer learning using a batch size of 32, ensuring efficient computation and stable gradient updates. A learning rate of $1e^{-4}$ was chosen to facilitate gradual learning, complemented by a weight decay of 0.001 to prevent overfitting. The Adam optimizer [98] was employed for its adaptive learning capabilities, while a step size of 1 and a gamma value of 0.5 were used to schedule learning rate decay, progressively reducing it to refine training. To prevent over-reliance on specific neurons, we apply a dropout layer with probability of 0.2. These hyperparameters were carefully selected to balance model performance, convergence speed, and regularization.

4 FROM CORRECTNESS TO TRUSTWORTHINESS

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

Instead of increasing accuracy and correct predictions, we focus on changing the failure semantics of classifiers from uncontrolled content failures (i.e., misclassifications) to rejection failures. Ideally, we want to reject all and only the erroneous predictions: on the downside, the availability of said component may be negatively affected when correct predictions are rejected in the process. Ways to build fail-controlled software architectures [19] are well-known in the literature and often rely on safety monitors [4], which are intended to complement an existing critical component or task by continuously checking invariants, or processing additional data to detect dangerous behaviors, and blocking the erroneous output of the component before it is propagated through the system. Finding trade-offs between safety and availability is of utmost importance when dealing with critical systems: this approach is not different.

4.1 Building Trust

How can classifiers be made dependable enough to 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*" [19]. 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 [99], which are exceedingly common when dealing with ML-based software. Instead, **a**

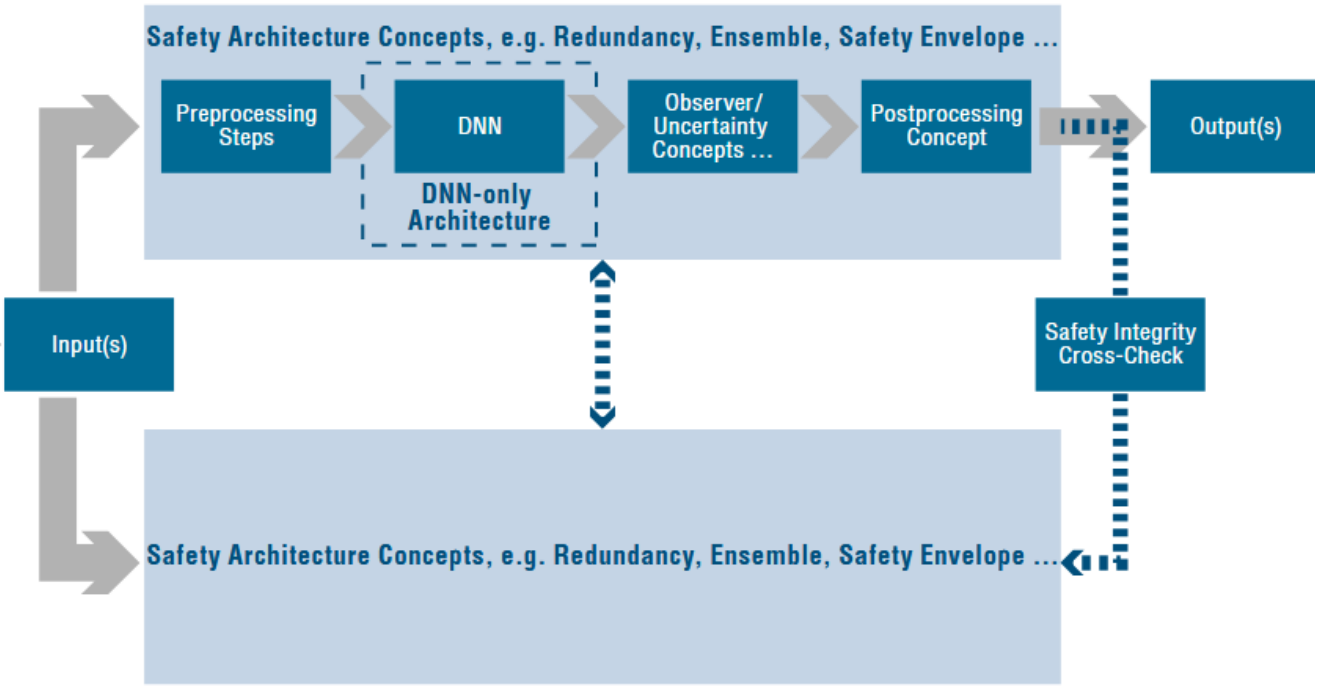


Figure 1: DNN architecture design from Annex B of [96].

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 [100] - overview of trustworthiness in artificial intelligence - guidelines, ML-based software "*must incorporate diagnostic measures to safely manage abnormal behavior, ensuring a transition to a secure state*". That is why even companies and stakeholders came up with the "safety first for automated driving" whitepaper [101], which is geared towards autonomous driving, but offers a high-level software architecture for (DNN) classifiers depicted in Figure 1. The proposed architecture employs 4 blocks (preprocessing, DNN, uncertainty, and post-processing) whose behavior is continuously monitored through a safety envelope (bottom of the figure) The blocks ensure i) a careful input check (e.g., anomaly detection, out-of-distribution detection) and sanitization; ii) a robust training of the model via data augmentation (in case of white-box classifiers) or usage of ensemble learning; iii) estimation of uncertainty or confidence in prediction; and iv) generic post-processing, to minimize classification errors and eventually transitioning to an unconfident state.

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

4.2 An Enabling Condition for Certification

The deployment of software for critical applications and infrastructures is constrained and regulated by safety methodologies, designs, verification, and validation activities to prevent the occurrence of catastrophic failures. Standards such as the ISO 26262 and 21448 (SOTIF – Safety of the Intended Functionality) for automotive, the CENELEC EN50xxx series for railways, and the DO178x documents for avionics all stem from the general IEC61508 for generic hardware-software systems. Compliance with standards ensures traceability across requirements, architectural and unit design, code, and verification. This allows for the system to be certified as having a specific Safety Integrity Level (SIL), Acceptable Level of Risk (ALR), or Tolerable Hazard Rate (THR) which quantifies the likelihood of the system having critical failures. SILs, ALRs and THRs are system-level properties but depend on the behavior of individual software components, their interaction, and the interaction with the underlying hardware [19]. Thus, these guidelines are extremely relevant also for any software sub-systems or components – including ML-based classifiers – that are willing to be deployed into a critical system, and consequently need to be trustworthy.

The SOTIF standard recognizes [102] performance limitations of software (including and especially for ML-based components) and characterizes its behavior as a combination of safe/unsafe states due to known/unknown inputs (see Figure 2a). The desideratum is the safe state due to processing known inputs; however, the system may be safe even when processing unknown inputs due, for example, to robust design practices. On the other hand, misclassifications may trigger scenarios that belong to unsafe-unknown (e.g., due to inputs out of training distribution) and unsafe-known (e.g., due to inputs out of operational design domain) situations. According to SOTIF, these have to be reduced to the extent that the frequency of system failures due to these events is considered acceptable [103]. Dealing with unsafe-known situations typically

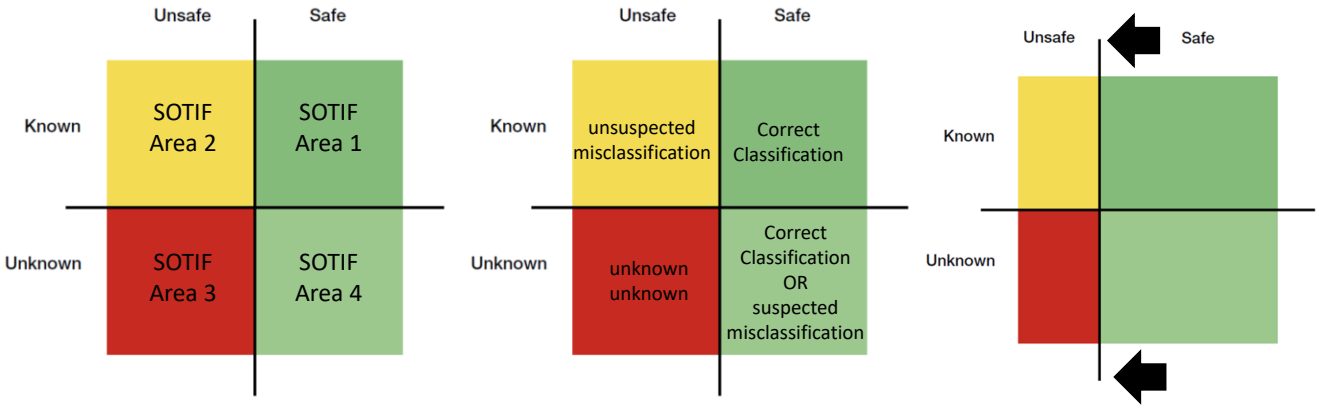


Figure 2: a) Classifier predictions according to the ISO/PAS 21448 SOTIF, with Area 2 and Area 3 to be minimized. B) shows the contents of each area, while c) depicts how a trustworthy software helps towards SOTIF.

requires defining the operating conditions (e.g., weather, time of the day, traffic congestion, etc.) for which a safety claim would apply, and thus the performance expected from a classifier within a given Operational Design Domain (ODD, more information in the PAS 1883:2020). Unfortunately, dealing with unknown-unknowns is intrinsically challenging, as preparing against something we do not expect or even imagine is very difficult. Noticeably, this approach is not geared towards correctness, but it is orientated towards safety. The system may be safe even in the presence of wrong software outputs, or misclassifications, if these do not trigger catastrophic failures at the encompassing system level. Here, **the safety claim is not just an alias of "correct classification": the behavior could still be considered safe in the presence of misclassifications** (e.g., when a misclassification is suspected and then rejected, see Figure 2b). When aiming at a specific ALR, SIL or THR (e.g., 0.00001, or 10^{-5}, which is the maximum probability of failure on demand for a SIL1 software in the railway domain), shifting the architecture from a software that has to be correct with probability 1-ALR (e.g., 99.999 % of correct outputs) to a software that is trustworthy under the same threshold (e.g., wrong outputs < 0.001 %) is far more flexible and easy to implement. Within this requirement, a trustworthy software here may even be a software that is correct 20% of the times, and rejects other outputs, thus delivers no wrong outputs (i.e., fail-omission software component). The result is depicted in Figure 2c: when wrong outputs are rejected, the behavior of the system becomes safe (even if not correct), provided that the rejection is handled adequately, making the "safe" Area 1 and Area 4 SOTIF areas bigger.

4.3 Handling Rejections at the System Level

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

Within automated driving, tasks as semaphore or traffic sign recognition should avoid misclassifications of red/green semaphores or confusing a traffic sign with another, but can typically afford to occasionally reject uncertain predictions, provided that the correct recognition happens early enough for mitigations as emergency braking or evasive steering [3] to take place. Other tasks as obstacle or pedestrian detectors may still prefer a rejection over a misclassification, with rejections that are likely triggering emergency braking

to avoid hitting a potentially undetected pedestrian [2]. 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 [104]. This does not pair well with classifiers, which do not account for “rejections”, limiting their usage despite the many possible applications e.g., automatic visual inspection, rail maintenance management [105]. 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 [106].

5 FAIL-CONTROLLED CLASSIFIERS: A SWISS-ARMY KNIFE TOWARDS TRUSTWORTHY SYSTEMS

In this chapter, we present Fail Controlled Classifiers (FCC), which aim at suspecting and rejecting most (if not all) of misclassifications and triggering alternative strategies instead.

5.1 SOFTWARE ARCHITECTURES FOR FCCS

As depicted in Figure 3, Fail-Controlled Classifiers (FCCs) should perform runtime monitoring for suspecting misclassifications of the classifier itself. 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 4. These FCCs allow for reducing misclassifications thanks to the rejection mechanisms, which has an obvious downside: whenever FCC rejects many correct predictions alongside with misclassifications, it becomes almost unusable as it hardly provides a beneficial behaviour for the encompassing system. To address this problem, we present additional approaches based on ensembles of FCCs: Voting (VT), Weighted Voting (WVT), Recovery Blocks (RB), and Stacking (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 4, e.g., Section 5.1.1 details what it is shown in Figure 4a.

5.1.1 Self-Checking Classifier SCC

Self-checking or self-testing hardware or software components embed built-in and custom strategies to check for the quality of their execution. This approach is required by many standards for deploying transportation systems and usually involves crafting hardware with redundancy and seeking for an agreement of the outputs of the replicas [25], or employing testing libraries that are periodically exercised

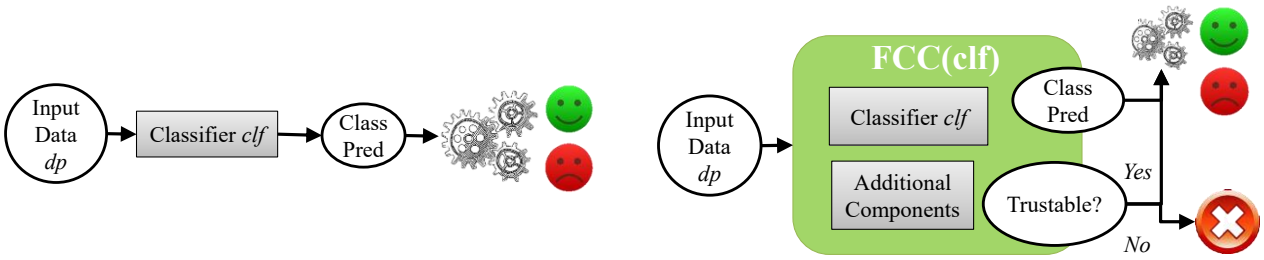
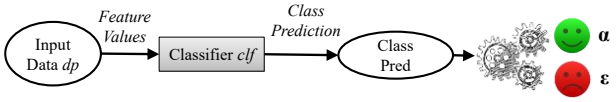
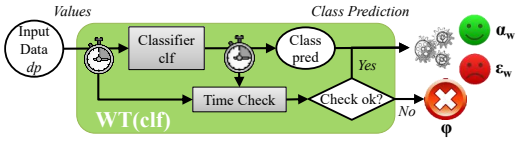


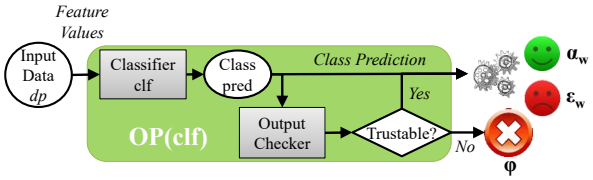
Figure 3: Transitioning from traditional classifiers (left) to Fail-Controlled Classifiers (right), which may reject predictions.



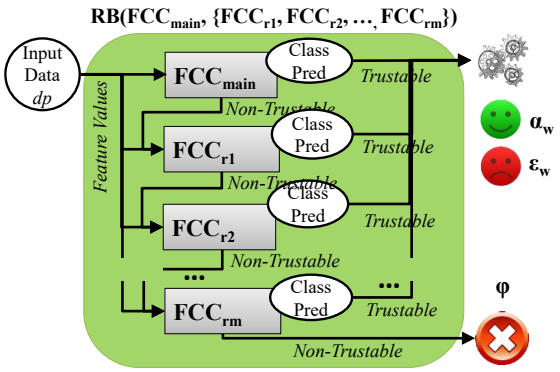
x) A simple (reference) classifier, that provides correct or incorrect output against input.



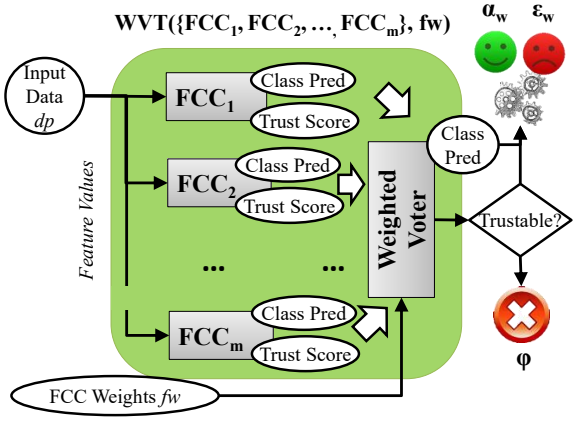
b) Watchdog Timer (WT), which measures inference time seeking for abnormal (too long or too short) executions.



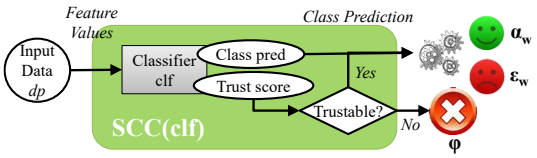
d) Output Processor (OP), which checks if the output of the classifier *clf* should be trusted or discarded.



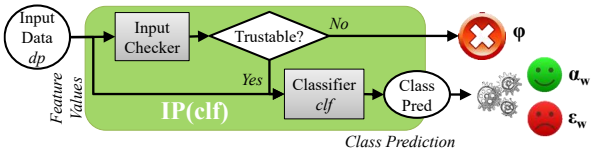
f) Recovery Blocks (RB) pair the main FCC with other FCCs ran in sequence, seeking for the first that outputs a confident prediction.



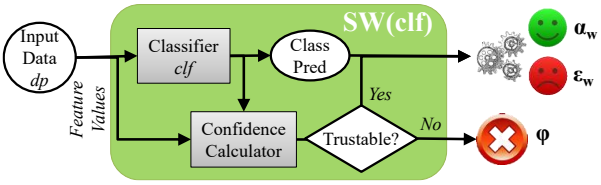
h) Weighted Voting (WVT). Weights *fw* can be provided as a parameter or be derived during inference.



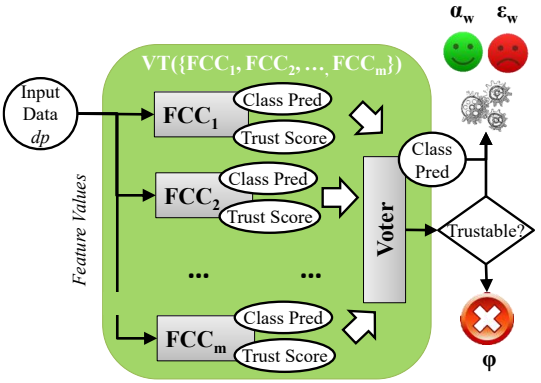
a) Self-Checking Classifier (SCC), which already has built-in and non-trivial methods for calculating trust in a prediction.



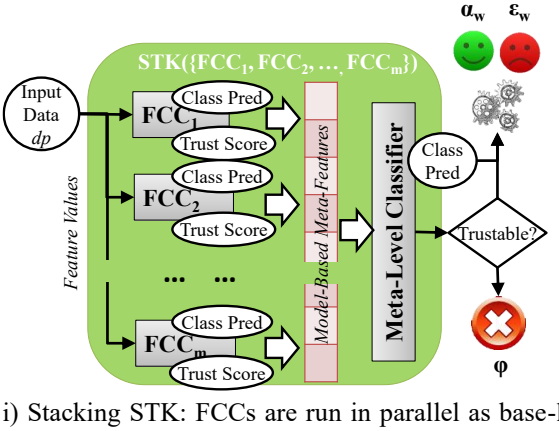
c) Input Processor (IP), which checks for integrity issues, anomalies or legitimacy of inputs.



e) Safety Wrapper (SW), which processes inputs, outputs and inference to compute confidence and decide on trustworthiness.



g) Voting VT. Different FCCs are run in parallel and their results used for adjudication by means of k out of n voting.



i) Stacking STK: FCCs are run in parallel as base-learners, creating model-based meta-features to perform adjudication.

Figure 4: Software Architectures for FCCs with accuracy αw, misclassification probability εw, and rejection probability φ.

on both hardware and software equipment [26]. Whenever one of these checks fails, the target component is deemed failed and in need to be 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 [107], but are not available by default in standard libraries used for supervised learning e.g., Python's *scikit-learn, PyTorch, Keras*.

5.1.2 Watchdog Timer WT

Watchdog timers (WTs) aim at measuring the length of the execution of a target function to understand if the elapsed time conforms with expectations [108]. In case the function completes too early, the watchdog timer 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, watchdog timers were meant to check electronic or mechanical-related functions [27], but transitioned to check the execution of software [108] and thus constitute an additional way to build FCCs. The clear advantage is that they add negligible overhead and work with any black-box classifier. On the negative side, they will be able to spot only a limited subset of issues (e.g., several anomalous activation pattern of neurons in DNNs, long paths in decision trees that may be due to an overfitted model, problems due to underlying hardware, operating system or virtualized middleware, or slow-downs due to malicious or malfunctioning software acting in the host system) [109], [110], resulting in a low rejection probability but high residual misclassifications. Importantly, tuning timers is a system-specific process: a WT may work

well with a specific hardware, but requiring re-tuning when the same hardware gets updated i.e., the notion of normal prediction time varies.

5.1.3 Input Processing IP

This FCC performs a pre-processing to seek for anomalies, suspicious values, low quality of such input, and the like. The pre-processing is implemented by an input checker, which could exercise adversarial attack detectors [42], out-of-distribution detectors [43], image corruption detectors [11], statistical distributions [58], or unknown data detectors in general [6]. 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 [7]. 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.

5.1.4 Output Processing 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 [39], 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 being classifier-specific; in the latter case, it is possible to end up having very different rejection rates when using different classifiers.

5.1.5 Safety Wrapper SW

IP and OP FCCs act before or after the classifier. However, we may think of a monitor or wrapper [4] that acts before, during and after inference. This is the case of the SW, which builds an envelope around the classifier to extract the 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 [59], ensembles of confidence measures [111], relative positioning of input data with respect to the prediction [57] or run other classifiers (e.g., kNN [61], probabilistic DNNs [60]) and check for agreement with the main classifier.

Conversely, when internals of the classifiers are fully available, it is possible to craft very specific mechanisms that seek for very specific information throughout the inference process. The activation patters of neurons in a DNN or the length of a path in a decision tree can provide information on the whole prediction process and thus on its uncertainty [63].

5.1.6 Recovery Blocks RB

Recovery blocks are known since many decades as one of the strategies for reliable software design [22]. 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 as 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 their output. This concept is not entirely new in the machine learning domain: delegating classifiers [112] 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 4f). 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.

5.1.7 Majority and k-o-o-n Voting VT

Other approaches rely on N-Version Programming, or exercising different classifiers in parallel [22], [29] each acting independently but processing the same inputs. Their predictions, alongside with trust scores, are sent to the adjudicator, which is a function that takes as input the predictions and the trust scores of the m classifiers, generates 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.

5.1.8 Weighted Voting WVT

A variant of voting is the Weighted Voting (WVT), where replicas are assumed to have different reputation: 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 e.g., their accuracy a_w, the non-misclassification probability $1-\varepsilon_w$, 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 a FCC is, the more it contributes towards the output of the WVT.

5.1.9 Stacking STK

The adjudicator [22] 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 which is often referred to as Stacking (STK), with *n* FCCs at the base level, and a different, independent, classifier at the meta-level [113]. The meta-level classifier expects *2*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 4i). 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.

5.1.10 Other Notable Approaches

Decades of critical systems engineering and system-level thinking originated more architectures [22], [29] than those shown above. Voting was explored in different formulations (i.e., hard, soft, weighted), and can be even used to build a hierarchical agreement structure that is known as n-self-checking-programming [22]. Boosting techniques were proven to be very effective for classifying known [114] 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. 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 [5], but this is still quite an uncharted territory in which there are no widespread and solid proposals (yet).

5.1.11 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, with low overhead. Since none of the solutions above (and none at all, according to the knowledge of 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 sensitive analyses to fine-tune the overall mechanism.

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 of the first FCC with few exceptions. Thus, FCCs have to be ordered staring from those with low residual misclassifications ε_w and potentially high rejections φ to FCCs with low rejections φ at the end.

Table 2: 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

(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

5.2 Evaluation Metrics

As depicted in Figure 3, Fail-Controlled Classifiers (FCCs) should perform runtime monitoring for suspecting misclassifications of the classifier itself.. 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 (φm, desirable and to be maximized), or correct predictions (φc, unnecessary rejections to be minimized). Overall, $\varphi = \varphi m + \varphi c$, and $\alpha w + \varphi c = \alpha$;
- residual misclassification probability εw, $0 < \varepsilon w \leq \varepsilon \leq 1$; overall, $\varphi m + \varepsilon w = \varepsilon$.

All those probabilities are sketched in Table 2. 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 maximised;
- ε *drop* $= (\varepsilon - \varepsilon_w) / \varepsilon = \varphi_m / \varepsilon$, which is the drop in misclassifications due to FCC, to be maximized.

A *FCC(clf)* will 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 i.e., having a high ε *drop*. Whereas at component-level this may seem a negligible improvement, at the system level it provides a way to prevent a misclassified prediction from propagating through the system, potentially causing (catastrophic) failures.

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 as an additional class to the problem in this formulation. Specifically, for the diversity metrics, 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 [115], [116], [117], we define the *double reject* DR measure between FCC_i and FCC_j as

$$DR\big(FCC_i, FCC_j, dp\big) = \begin{cases} 1, & if\ FCC_i\ rejects\ dp\ prediction\ \wedge FCC_j\ rejects\ dp\ prediction \\ 0, & otherwise \end{cases}$$

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

$$DR(FCC_{set}, dp) = \begin{cases} 1, & \bigwedge_{fc \in FCC_{set}} fc\ rejects\ dp\ prediction \\ 0, & otherwise \end{cases}$$

Another diversity metrics disagreement (DIS) is used which defined as "the ratio between the number of observations where two classifiers predict different results to the total number of observations", which we report below using a test set TS and a set of FCC_{set}.

$$DIS(FCC_1, FCC_2, TS) = \frac{1}{|TS|} \sum_{dp \in TS} \begin{cases} 1, & if\ FCC_1.predict(dp) \neq FCC_2.predict(dp) \\ 0, & otherwise \end{cases}$$

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

$$DIS(FCC_{set}, TS) = \frac{2}{n(n-1)} \sum_{i=0}^{n-1} \sum_{j=i+1}^{n} DIS\big(FCC_i, FCC_j, TS\big)$$

The Double Fault (DF), defined as the "proportion of the cases that have been misclassified by both classifiers", which scales as "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 [19].

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

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.

5.3 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, we may assume that the probability assigned to the most likely class is based on the confidence of the classifier, 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 Carls: 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 elaborated more on this aspect in Section 5.1.

5.4 Suspecting Misclassifications

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

Black-box approaches do not assume any knowledge about the classifier, thus observe its inputs and outputs without any access to internals. They allow for building statistical machinery that conveys input pre-processing [58], output analysis [39] and even ensembles of them [111] for complex and quite effective techniques for suspecting misclassifications. They mostly check if inputs and outputs belong to specific statistical distributions, and deem the prediction as non-trustable otherwise. Other approaches aim at identifying unstable regions of the input space in which the classifier may be likely to output misclassifications [107], or use external classifiers (e.g., nearest neighbours [57]) 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 [39] – which vary from a DNN model to another. Classifiers that orchestrate ensembles may use the degree of agreement or the diversity of predictions of the classifiers in the ensemble [60] 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 [118]. 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 [64].

6 EXPERIMENTATION WITH FCCS

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

6.1.1 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 6.1.2. In Section 6.1.3 we gather datasets for exercising tabular and image classifiers, spanning over a wide variety of classification tasks and simulating unknown data (Section 6.1.4). Section 6.1.5 and Section 6.1.6 report results for tabular data and image classification, respectively. The performance of classifiers and FCCs is quantified via the metrics from Section 5.2. 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 [119].

6.1.2 Selection of FCCs and Classifiers

Some of the FCCs that are presented in Section 5.1 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.

6.1.2.1 Tabular FCCs and Main Classifiers

We use 5 binary tabular classifiers used as base classifier in this study are *Decision Trees (DT)*, *Random Forests (RF), XGBoost (XGB) Gaussian Naïve Bayes (GNB) and Logistic Regression (LR)* as detailed in Section 3.2.1. 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 probability of classifiers, the rejection threshold is selected empirically. A preliminary sensitivity analysis over threshold values in the range [0.6, 0.9] indicated that higher thresholds tend to cause excessive rejections, while lower thresholds yield fewer rejections., A probability threshold of 0.8 (i.e., predictions with probability lower than 80% are rejected) is therefore adopted, as it provided a reasonable amount of rejections.This experimental setup created 10 IPs (i.e., 5 classifiers, each checked via ET or LDA), 5 OPs (one per classifier), and 10 SWs, obtained combining each of the 10 IPs with the OP. In this configuration, a SW is a combination of an IP and an OP.

6.1.2.2 Image FCCs and Main Classifiers

For image classification, 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 6 DNNs above brings unique features that are elaborated in Section 3.2.

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.

6.1.3 Datasets

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

6.1.3.1 Tabular Datasets

We select three tabular datasets belonging to different domains in which classifiers are typically willing to be applied: intrusion detection (CICIDS18 [72]), error detection (ARANCINO [13]), and control systems (MetroPT [120]) detailed shown in Section 3.1.1.For these datasets, we target a binary classification problem, aiming at distinguishing normal operating conditions against anomalies due to attacks, errors, or component failures.

6.1.3.2 Image Datasets

We select three image datasets: Flower (9 classes [93]), FER2013 (7 classes [91]) and FOOD-101 (10 classes [92]). Since the images vary in size, we apply a transformation to resize them (at most 96x96 RGB).

The datasets contain in the order of thousands or tens of thousands of images per dataset detailed in Section 3.1.2.

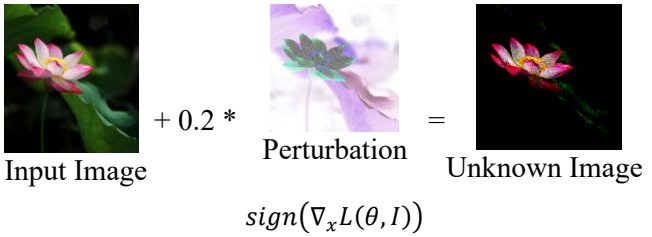


Figure 5: Example of generating unknown images in the Flower Dataset, where input image are altered with the perturbation with multiplier to form an unknown image.

6.1.4 Generation of Unknown Inputs

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

6.1.4.1 Unknown Image Data

We generate a first batch of unknown (out-of-distribution) images by applying four different alterations i.e. Rotation, Color Space and Gaussian Noise [121] and GANs to 40% of images from the test set, i.e., unseen by the classifier in the training set. We create the Gaussian noise image by generating a noise map using *mean 0* and *st.d 25*, and then overlapping it to images. Color space anomalies result from chaining operations 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 90º 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 5). FGSM's parameters are as follows: i) input image size (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.

6.1.4.2 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 6). 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.

6.1.5 Results: Tabular Data Classification

We start commenting the results of classifiers and FCCs using tabular datasets with the aid of Table 3, which reports the percentage of misclassifications ε (for the main classifier), ε_w (for FCC), 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).

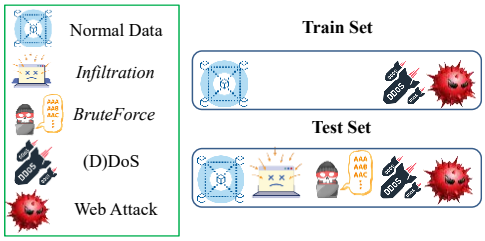


Figure 6: Unknown tabular data: the example of CICIDS18, where some attack classes are removed from the training set occurring only in the test set, being unknown to the classifier.

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 to 5th row), 0.269 (6th to 10th row), 0.245 (11th to 15th row), 0.123 (16th to 20th row), and 0.123 (21st to 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 3), 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 input checker (i.e.,

IP_x_ET, SW_x_ET, where x is any main classifier) 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 i.e., 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

Table 3: Results of tabular clf and FCCs across tabular datasets used in the paper. For each dataset, main classifier and FCC we report the misclassifications ε and ε_w, rejections φ, φ_m *ratio* and ε *drop*.

				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	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	0	0	0.126	0.126	0	0	0	0.063	0.063	0	0	0
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	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	0.001	0.571	0.126	0.126	0	0	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	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

maximum confidence) or too high, delivering an obnoxious rejection probability as for LR in CICIDS18, see OP_LR, 13th row, 5th column of Table 3.

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.

6.1.6 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 7a), IC and VGG on FER13 (Figure 7b), DN and IC on Food (Figure 7c). These scenarios offer interesting discussion items that we explore as follows.

Figure 7a 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 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, that is around 40% for the OP and SW with AN as main classifier, and around 30% for OP and SW built using DN as 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 7b has the same structure of Figure 7a, 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.

Lastly, Figure 7c 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 to increase the *ε drop*. FCCs not shown in the figures followed a similar trend to those showed here.

6.2 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 6.2.1). Then, we will proceed to instantiate different RB, VT, WVT, SKT FCCs for tabular (Section 6.2.2) and image data (Section 6.2.3), discussing their results and comparing them against FCCs from the previous section.

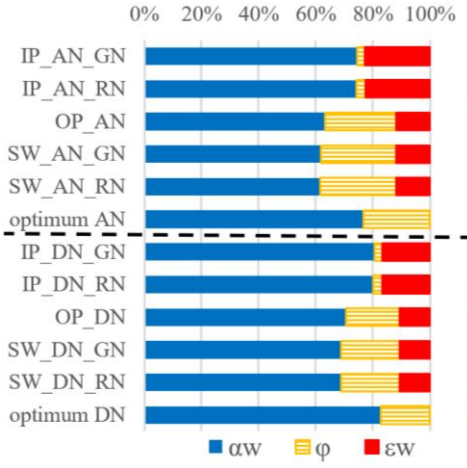


Figure 7 a.FCCs using AN and DN as main classifiers on the Flower dataset.

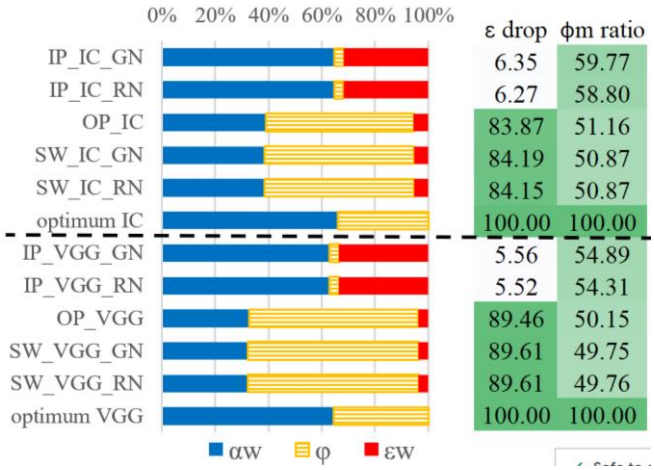


Figure 7b. FCCs using IC and VGG as main classifiers on the FER13 dataset.

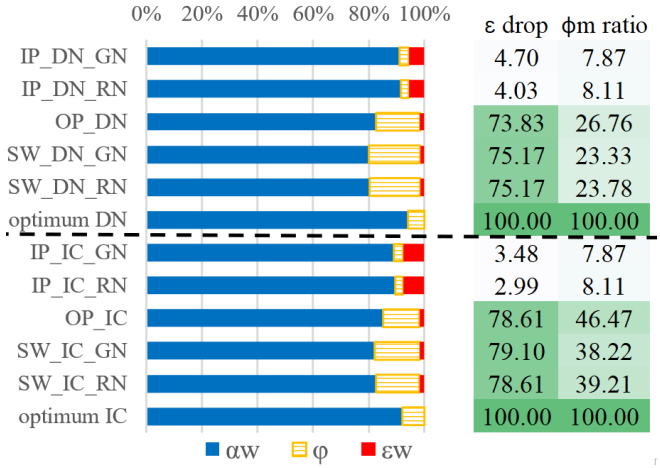


Figure 7c. FCCs using DN and IC as main classifiers on the Food dataset.

Figure 7: Bar charts showing the performance of FCCs on different Image datasets.

6.2.1 Computing Diversity

As already discussed in Section 2.2 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 a 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 ($0 \leq DIS \leq 1$, higher is better), double fault ($0 \leq DF \leq 1$, lower is better), and double reject ($0 \leq DR \leq 1$, lower is better) using the formulas introduced earlier in the chapter. 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 sectionfull detail on the experiments about diversity is provided in [122], whereas in the followings we will only report the subset of diversity data that is most relevant for the discussion.

6.2.2 Tabular Data

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

6.2.2.1 Diversity Analysis

General Comments. Data about the performance of IP, SW, OP tabular classifiers was already commented in the previous section and fully reported in Appendix A in Table A.1 of [122]. Table A.2 to Table A.4 of [122] 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 middle of Table A.3 of [122]). IPs and SWs are more prone to rejections, with low residual misclassification rate, low DF but also a pretty high DR (see corners of Table A.4 of [122]) 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 as DT, RF,

XGB do not have. The most relevant combinations of FCCs are summarized in Table 4: note that these report a very low DF, which is typically important for avoiding 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 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 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 maximising accuracy compared to RB1.

FCCs for (W)VT, STK. Voting approaches demand for maximum disagreement, thus requiring a more careful analysis of DIS values in Table 4. OP_LR and SW_LR_ET exhibit low disagreement of 0.03, see the 4th-5th row, 8th-9th column of Table 4. 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 as LR, GNB, LDA in its place, but they delivered very unsatisfactorily performance.

Table 4: Classification performance, DIS, DF, DR of FCCs used for building RB, VT, WVT, STK tabular classifiers. Results are averaged across the three tabular datasets used in experiments.

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	-

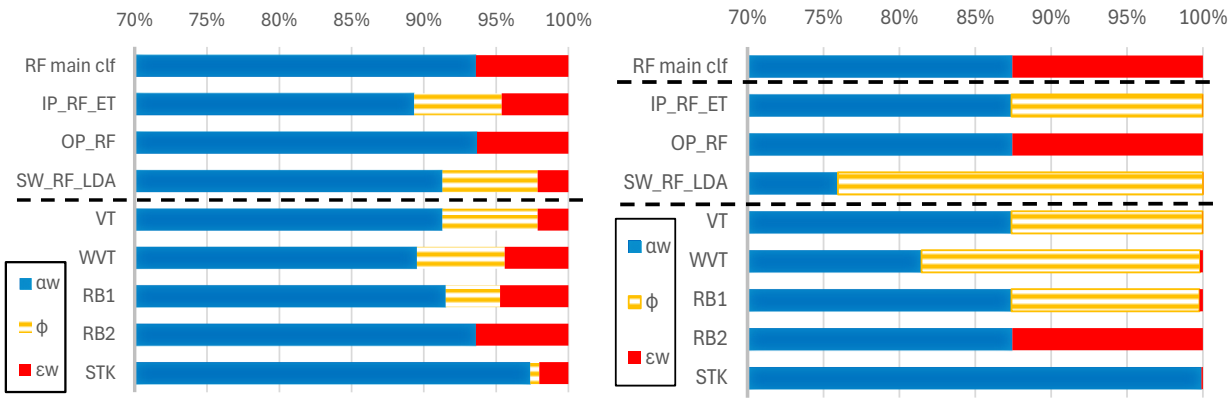


Figure 8: Comparison of FCCs using RF as the main classifier on the Error Detection (Figure 8a, left) and MetroPT (Figure 8b, left) tabular datasets. The color scheme complies with that of Figure 7.

6.2.2.2 Experiments with RB, VT, WVT, STK

Figure 8 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 8a) and MetroPT (Figure 8b), avoiding NIDS as it was showing a trend very similar to that of Error Detection. Bar charts in Figure 8 follow the same color scheme of Figure 7: blue solid bar for α_w, orange-striped bars for φ and red bars for ε_w. On top of 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 main classifier in both datasets. Noticeably, the OP_RF has the same behavior of 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 classification 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 stacking 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 and only misclassifications, this means that the STK FCC achieves a near perfect accuracy, which is a rare but indeed beneficial case.

6.2.3 Image Data

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

6.2.3.1 Diversity Analysis

General Comments. Data about the performance of IP, SW, OP image classifiers was already commented in the previous section and fully reported in [122] in Table A.5. Table A.6 to Table A.8 of [122] 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 [122]). OPs and SWs are more prone to rejections, with low residual misclassification rate, low DF but also a pretty high DR (Table A.8 of [122]), due to the many rejection 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 [122]. That said, the most relevant combinations of FCCs are summarized in Table 5: 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 5, 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 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

Table 5: Classification performance, DIS, DF, DR of FCCs used for building RB, VT, WVT, STK image classifiers. Results are averaged across the three image datasets used in experiments.

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	-

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.

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 5. 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.

6.2.3.2 Discussion: Performance of RB, VT, WVT, STK

Figure 9 reports image data in the same format as what Figure 6 did with tabular data, but is related to FCCs using DN as main classifier on the FER13 (Figure 9a) and Food (Figure 9b) image datasets. Results of 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 9a and Figure 9b): 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.

6.2.4 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

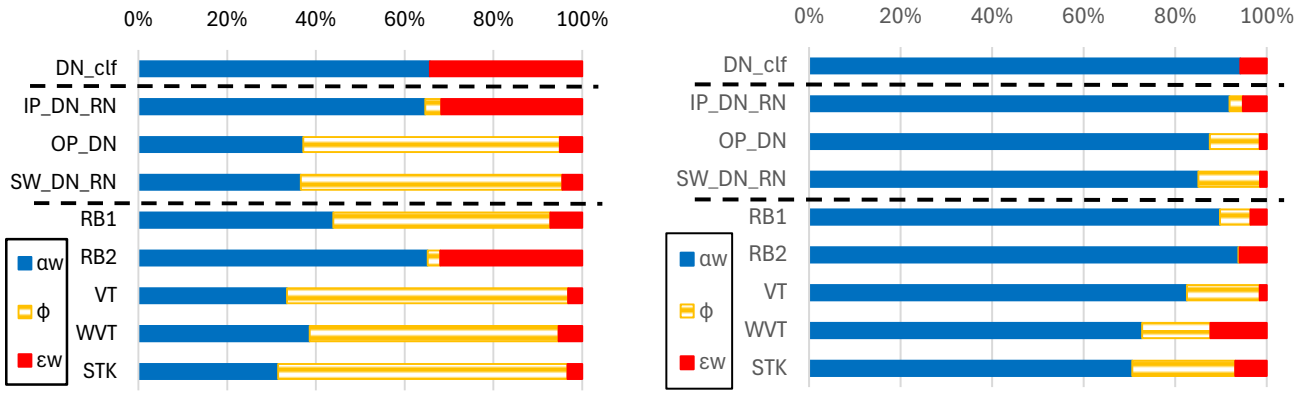


Figure 9: Comparison of FCCs using DN as the main classifier on the FER13 (Figure 9a, left) and Food (Figure 9b, left) image datasets. The color scheme complies with that of Figure 7 and Figure 8.

suffice for quantifying the ability of rejecting unknown inputs specifically. Thus, Table 6 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 φ = 1, or 100% rejections of these items.

As shown in the Table 6, 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.

6.3 Lessons Learned

Table 6: 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

Nine different FCCs are presented, discussed and evaluated in the chapter: 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). Overall, 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, 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.

6.4 Threats to Validity

Findings of this chapter 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 load 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 thesis) against FCCs built on top of these classifiers. The comparison and the discussion that is carried out throughout the chapter compares the performance of the classifier against FCCs and does not aim at optimizing performance on a specific scenario. Unless changing the hyperparameters turns it in a perfect classifier, which is unlikely, then FCCs will be able to provide a useful contribution anyways.

The reported results are primarily based on point estimates of performance metrics. While these allow direct comparison across methods and datasets, statistical confidence intervals are not explicitly reported, and therefore conclusions regarding relative superiority should be interpreted with appropriate caution.

In addition, the perturbation-based approach used to simulate unknown inputs represents a controlled experimental approximation. Although such perturbations are commonly used in robustness studies, the perturbation process itself is not externally validated; hence, the results should be interpreted as illustrative rather than exhaustive representations of all possible unknown inputs.

Also, this is one of the few studies that uses many datasets from both tabular and image domains, ensuring solid and general findings.

7 SAFETY MONITORS FOR BLACK-BOX CLASSIFIERS: DESIGN, FRAMEWORK AND BENCHMARK

All the observations from previous chapters are then conveyed to design, implement, and evaluate a Safety wraPper through ensembles of UncertainTy measures (SPROUT). SPROUT is a safety monitor for a black-box classifier, which computes many uncertainty measures and produces a binary confidence score that can be used to decide whether the classifier prediction is trustable and can be safely propagated to the encompassing system, or if it should be rejected, triggering mitigation strategies instead.

7.1 Uncertainty Measures for Classifiers

This section introduces the uncertainty measures used to quantify how confident a classifier is in its predictions. These measures operate under a black-box assumption and capture different aspects of uncertainty, ranging from probability-based indicators to data-driven reconstruction and neighborhood analyses to detect unreliable or incorrect outputs.

7.1.1 Quantitative Measures to Compute Uncertainty

This work focuses on uncertainty measures that are not classifier-specific, but instead have a generic formulation that pairs well with any classifier, which is seen as a black-box, and minimize the risk of poor significance of classifier-specific uncertainty. Table 7 summarizes a total of 9 uncertainty measures UM1 to UM9, which process at least one of the following items: i) input data *dp*, ii) classifier *clf*, iii) class prediction *dp_prob*. Importantly,

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

7.1.1.1 UM1: ***Confidence Interval***

Given a confidence level $0 \leq w \leq 1$ as input parameter, the confidence interval provides actionable information about the statistical distribution of the underlying feature values [58]. Let avg_i and std_i

Table 7 Summary of Uncertainty Measures used in this study.

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

respectively be the average and standard deviation of feature *j* for a set of data points, and f the number of features. The confidence interval to be computed for each feature *j* is:

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

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

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

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

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

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

UM1 takes values in the range [0,1]. A value of UM1 = 0 indicates that all feature values of a dp fall within their corresponding confidence intervals, while UM1 = 1 indicates that all feature values lie outside the confidence intervals. Higher UM1 values therefore correspond to higher uncertainty, as an increasing fraction of the dp features deviate from the expected statistical ranges.

7.1.1.2 UM2: Least Confidence Score

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

$$UM2(dp\_prob) = 1 - \max(dp\_prob)$$

The higher the UM2, the more uncertain the output of the classifier [123].

7.1.1.3 UM3: Entropy of Probabilities

We retrieve the *dp_prob* produced by a classifier for a given *dp* and we compute UM3 using *db_prob* entropy [39]:

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

To obtain values in the range [0,1], entropy is normalized by its maximum value $log_2\ (c)$.The higher the UM3, the more uncertain the classifier: a *dp_prob* array with constant values (i.e., all classes have the same probability) generates the highest UM3 of 1.

7.1.1.4 UM4: Naïve Bayes predictive entropy

This measure uses a Naïve Bayes process to estimate the probability that the input data point *dp* belongs to each of the possible *c* classes [59]. Briefly, this process applies Bayes' theorem assuming strong (i.e., naive) independence between the features. This assumption makes the process scalable, with linear time complexity with respect to the input size. Noticeably, the Naïve Bayes itself is a classifier, which needs to be trained before predicting the class of a data point *dp*. We define UM4 as:

$$UM4(dp) = UM3(NaïveBayes.predict\_p(dp))$$

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

7.1.1.5 UM5: Combined Uncertainty

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

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

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

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

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

Results of UM5 lies in the range $[-1,1]$under the assumption that UM3 is normalized.. UM5 = 1 translates to high confidence that the prediction of *clf* is correct, UM5 = -1 means high confidence that the prediction is a misclassification, letting UM5 = 0 show maximum uncertainty.

7.1.1.6 UM6: Multi-Combined Uncertainty

UM6 computes uncertainty relying on more than one checker. The goal is to aggregate the agreement and confidence signals provided by multiple checking classifiers. UM6 uses a set CC of *ncc* checking classifiers, computes UM5 for each *chk_c* ∈ CC with respect to *clf*, and averages the results. Each checker contributes independently to the final uncertainty score through its corresponding UM5 value. If more checking classifiers in CC agree with *clf*, the higher the UM6.

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

However, UM5 ∈ [-1,1], UM6 also lies in the range [-1,1].

7.1.1.7 UM7: Feature Bagging

UM7 exploits the concept of bagging [124], a method for generating multiple versions of a classifier *bagC* by making bootstrap replicates of the training set and using these as *bc* new instances of a classifier. Each instance of *bagC* is trained using different subsets of the original training set: it is only aware about a portion of the training set and decides using restricted knowledge. Should classifiers predict different classes for a given data point *dp*, this would mean that the prediction of *clf* should be treated with high uncertainty. This disagreement is quantified by aggregating the outputs of the *bagC* through UM6, yielding a bounded uncertainty score.

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

7.1.1.8 UM8: Neighbor Agreement

UM8 finds and uses the nearest neighbors [125] of a data point to understand if the input data point *dp* lies in a region of the input space where the majority of its neighbors is assigned (by *clf*) its same class. The usage of a prime *k* avoids ties in multi-class problems when the number of classes is smaller than *k*. The steps to execute UM8 are:

1. compute dp_label = arg max(dp_prob);
2. exercise a k-Nearest Neighbor [125] search, obtaining the k data points of the training set that are the closest to dp. We call this set of data points neigh;
3. for each data point ne in neigh, apply clf.predict(ne);
4. compute the final value as follows:

$$dp\_label = ag\ max(dp\_prob)$$
$$neigh = \text{Nearest-Neighbours}(Tr, dp, k)$$
$$nl = \text{for } ne \text{ in } neigh: clf.predict(ne)$$
$$\text{UM8}(dp\_prob, clf, dp, k) = \frac{1}{k}\sum_{i=0}^{k} 1\ if\ nl_i = dp\_label\ else\ 0$$

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

7.1.1.9 UM9 Reconstruction Loss

Reconstruction loss quantifies to what extent the input data point belongs to the distribution of past inputs or if it is an unseen, out-of-distribution data point [126]. The expectation is that misclassifications will happen frequently when dealing with out-of-distribution input data points. We compute UM9 through autoencoders, which are unsupervised neural networks that learn efficient encodings of the input data, which are validated and refined by attempting to reconstruct the input from the encoding [127]. Autoencoders are first trained using the training set and several layers, which SPROUT sets by default as composed of *f, f/2, f/4, f/2, f* neurons, with *f* being the amount of features of the input data. These layers can be customized whenever needed. UM9 is the L_2 norm (Euclidean distance) between the original data point and the reconstructed data point, and it quantifies the reconstruction loss.

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

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

7.1.2 Applicability of Measures

It is evident that some uncertainty measures require careful tuning, whereas others apply to any classification task as they are. It is the case of UM2 (Maximum Likelihood) and UM3 (Entropy of Probabilities), which solely rely on probability distributions generated by classifiers. UM5 (Combined Uncertainty), UM6 (Multi-Combined Uncertainty), UM7 (Feature Bagging) leverage additional checker classifiers and compare their outputs with that of the main *clf*: this makes them applicable to any problem, but only after a specific parameter setup. Also, the parameter setup may have a relevant impact on their resource usage, making them flexible but computationally demanding. UM4 (Bayesian Uncertainty) can

be applied only to supervised classification tasks, as it requires to train a probabilistic Bayesian network, while UM1 (Confidence Interval) and UM9 (Reconstruction Loss) focus on input data, providing an uncertainty quantification that is completely decoupled from the inference process.

In the context of image classification, specific uncertainty measures become less meaningful due to their reliance on statistical independence assumptions or feature-based proximity calculations, which do not usually apply to image data. UM1 (Confidence Interval) assumes independent feature distributions, and, UM4 (Bayesian Uncertainty) struggles with images since the assumption of feature independence is unrealistic in visual data, where pixels are highly dependent on their neighboring pixels. UM8 (Neighbor Agreement) relies on k-Nearest Neighbors (k-NN) search, which is very ineffective even with all optimizations and may not effectively capture class separability in high-dimensional image spaces, making it impractical for most image classification tasks [128]. UM9 (Reconstruction Loss) has the potential to detect out-of-distribution samples but may struggle with complex image patterns, making it unreliable in many visual tasks. Other measures as UM2 (Maximum Likelihood), UM3 (Entropy of Probabilities), UM5 (Combined Uncertainty) and UM6 (Multi-Combined Uncertainty) have the potential to work with images as well as with tabular data.

7.2 Safety Monitors for Black-Box Classifiers

This section presents our approach on safety monitors for black box classifiers, which is supported by a publicly available Python framework.

7.2.1 Black-Box Safety Monitors

Figure 3(left) depicts a simple classifier: the input data, and the features contained therein, is fed into a classifier *clf* that predicts a class label *dp_label* for that specific input data *dp* through its *predict* function. This classification process always outputs a class label that is then provided to the encompassing system, has α accuracy and ε misclassification probability. In this scenario, all the misclassifications are content failures. **Noticeably, insights of *clf* do not need to be disclosed for detecting misclassifications: as a result, *clf* is a black-box classifier**. The existence of a *predict* or *predict_p* function to generate a *dp_prob* and provide the output probabilities of the classifier to the misclassification detector is the **one and only assumption** we require for orchestrating any classifier in this study. Note that commonly used frameworks for machine learning (to name a few: *scikit-learn, xgboost, pyod, tensorflow*), expose this interface; therefore virtually any classifier can be deployed here. Another important observation is that this software

architecture is general enough to suit image classifiers or classifiers for tabular data, rule or invariant-based algorithms, DNN, tree-based, statistical classifiers, for binary and multi-class classification.

Figure 3(right) still feeds the input data to the classifier, which predicts the class *dp_label* for an input data *dp*. However, the classifier is now paired with an additional component, a misclassification detector, which processes the input data *dp*, the classifier output *dp_prob* and *dp_label* and, if needed, can use the black-box classifier itself for additional predictions. The aim of the misclassification detector is to output a binary score [80] to decide if the class prediction is suspected to be a misclassification, enabling prediction rejection. This builds the software architecture of the safety monitor *SM(clf),* which rejects the *clf* output with probability φ; otherwise, the class prediction gets forwarded to the encompassing system, is correct with probability α_w and is a misclassification with probability ε_w. There is still a residual probability ε_w of content failure, while $\varepsilon - \varepsilon_w$ misclassifications are instead going to be rejected thanks to the safety monitor.

7.2.2 SPROUT: a Safety wraPper thROugh ensembles of UncertainTy measures

Our proposal of software architecture for a black-box safety monitor is in Figure 10. Instead of relying on a unique mechanism to compute the binary confidence score and decide on rejection, we employ multiple uncertainty measures from Section 7.1, each processing different information, applying different rules and thus providing a different opinion on the input data (UM1, UM9), on the classifier output (UM2, UM3), on the behavior of the classifier against other checkers (UM4, UM5, UM6, UM7), or on the stability of a prediction (UM8). Each uncertainty measure outputs a floating point number, building an input array for a binary adjudicator, which delivers a binary score that decides on rejection. The resulting Safety wraPper thROugh ensembles of UncertainTy measures SPROUT(clf, UM_{set}, adj) is a software component where:

- *clf* is a black-box classifier,
- $UM_{set}=\{um_i, 0 \leq i \leq k \mid um_i \in \{UM1, UM2, \ldots, UM9\}\}$ is a set of $k = |UM_{set}|$ uncertainty measures

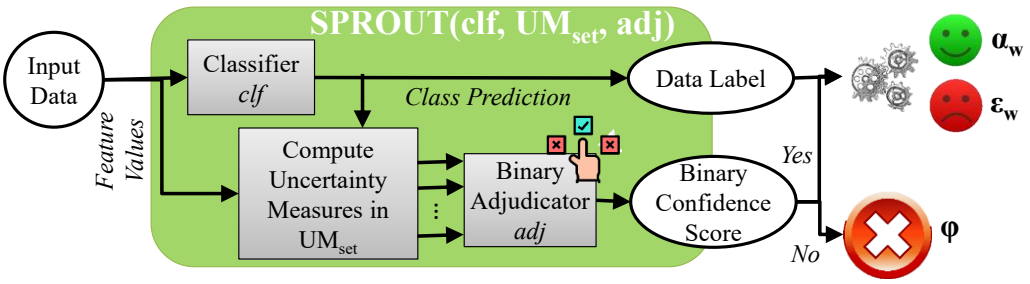


Figure 10: SPROUT safety monitor, which uses an ensemble of uncertainty measures to detect misclassifications.

- *adj* is a binary classifier that is meant to handle tabular data (i.e., outputs of the measures in UM_{set})

Clearly, the choice of which uncertainty measures should be computed is of utmost importance [64]. Some uncertainty measures may make SPROUT detect most of the misclassifications (thus the residual misclassification probability ϵw would be very low) at a cost of many rejections, i.e., $\varphi >> 0$, $\alpha_w << \alpha$. Conversely, other measures may build a monitor with high accuracy ($\alpha_w \approx \alpha$, or $\varphi_c \approx 0$), but rarely rejects outputs ($\varphi \approx 0$) and fails in detecting many misclassifications ($\epsilon_w \approx \epsilon$, or $\varphi_m \approx 0$) making its behavior similar to the regular *clf*. The usage of a set of uncertainty measures also helps in smoothing this issue, balancing these corner cases. Moreover, several measures are parameter-dependent and as such can be instantiated multiple times and have a different behavior; this is the case of UM1, UM5, UM6, UM7, UM8 and UM9. The choice of parameters depends on the structure of the input data (e.g., tabular or image data), the type of the classification task (i.e., multi-class or binary).

After uncertainty measures are exercised, a binary adjudicator processes the ensemble of floating point values computed using each uncertainty measure to output a binary confidence score. This binary adjudicator is completely decoupled from *clf* and can be implemented with thresholds, invariants, custom rules [29], or as a binary classifier, providing many degrees of freedom in finding the ideal function to combine ensembles of uncertainty measures into a unified output, even implementing non-linear decision functions. On a software architecture standpoint, the misclassification detector in SPROUT is a stacking meta-learner, with uncertainty measures at the base level, and a binary adjudicator at the meta-level [113]. The binary adjudicator is completely independent with respect to the input data of a specific problem: its features are entirely uncertainty measures: therefore, we could even deploy a unique adj for each and every classification task at hand, provided that the UM_{set} does not vary. This conjecture holds really well from an high-level architecture standpoint. However, the choice of UMs should be tailored to the classification task and, more importantly, to the input data type as already discussed in Section 7.1.

7.2.3 A Python Library for Exercising SPROUT

SPROUT is available at [129] and as PIP Python package. The package implements all uncertainty measures discussed in this thesis and makes SPROUT ready for deployment in any case study. Many already trained models for binary adjudication are already available in the library, and are accompanied by details about the uncertainty calculators they need, statistics on its binary classification performance and

on the importance each uncertainty calculator had in learning that model. Those information are not needed to run SPROUT but provide interesting details for explaining why SPROUT works as intended.

Listing 1: applying SPROUT to supervised classifiers.

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

Applying SPROUT to a brand new case study is very easy. Listing 1 reports a code snippet that shows a simple usage of SPROUT to wrap a supervised classifier from scikit-learn using a pre-defined (*ecai_sup*) binary adjudicator which uses the architecture in Figure 10. We assume to have a labeled dataset that we split in two parts. The first part will be provided as input to the *load_monitor* method that prepares a SPROUT monitor according to the chosen model. Data is used to train the uncertainty measures, while the binary adjudicator is simply loaded from the repository. Then, we initialize and train a RF classifier, which we provide as input, alongside with unlabeled test data, to the *predict_misclassifications* function, which outputs

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

7.3 Experimental Evaluation

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

7.3.1 Methodology

Our experimental methodology follows the steps below, and is depicted in Figure 11.

Gather datasets related to tabular data (29 datasets, see Section 7.3.2.1) and image classification (4, see Section 7.3.3.1), and split them into two partitions following a 70-30 train-validation split. Partition datasets into two groups: those that will be used to train the SPROUT monitors (monitor train MTR), and those that will only be used for evaluation (monitor test, MTE).

1. Select the uncertainty measures to be used for SPROUT monitors. This results in creating 3 different SPROUT instances, which use different UM_{set}: supervised tabular data classification (*sup_tab*), unsupervised tabular data binary classification (*uns_tab*), image classification (*img_sm*). These safety monitors are defined in Section 7.3.3.3 (*img_sm*) and in Section 7.3.2.3 (*sup_tab*, *uns_tab*).
2. Select ML algorithms capable of supervised or unsupervised tabular data classification, and DNNs for image classification. These will be trained on the train partition of all datasets in the MTR sets, and models will be stored for further usage.
3. For each model obtained at the previous stage, predict probabilities and classes on the validation partition of the respective dataset and compute the uncertainty measures needed by the applicable *sup_tab, uns_tab, img_sm*. This generates a CSV output file per model per dataset, one line per data point in the validation set, where each line includes:
   - The output probabilities *dp_prob* of the classifier, and the predicted label.
   - The value of different uncertainty measures, needed by the applicable *sup_tab, uns_tab, img_sm*
   - A binary *misc flag*, which specifies if the model misclassified the given validation data point. This is the target to be predicted by binary adjudicators.
4. Combine the CSV files generated previously into 3 CSV files, each containing data related to *sup_tab, uns_tab, img_sm monitors*. These are the datasets for training and comparing binary adjudicators for each safety monitors. Since the binary adjudicator is always a supervised tabular data classifier,

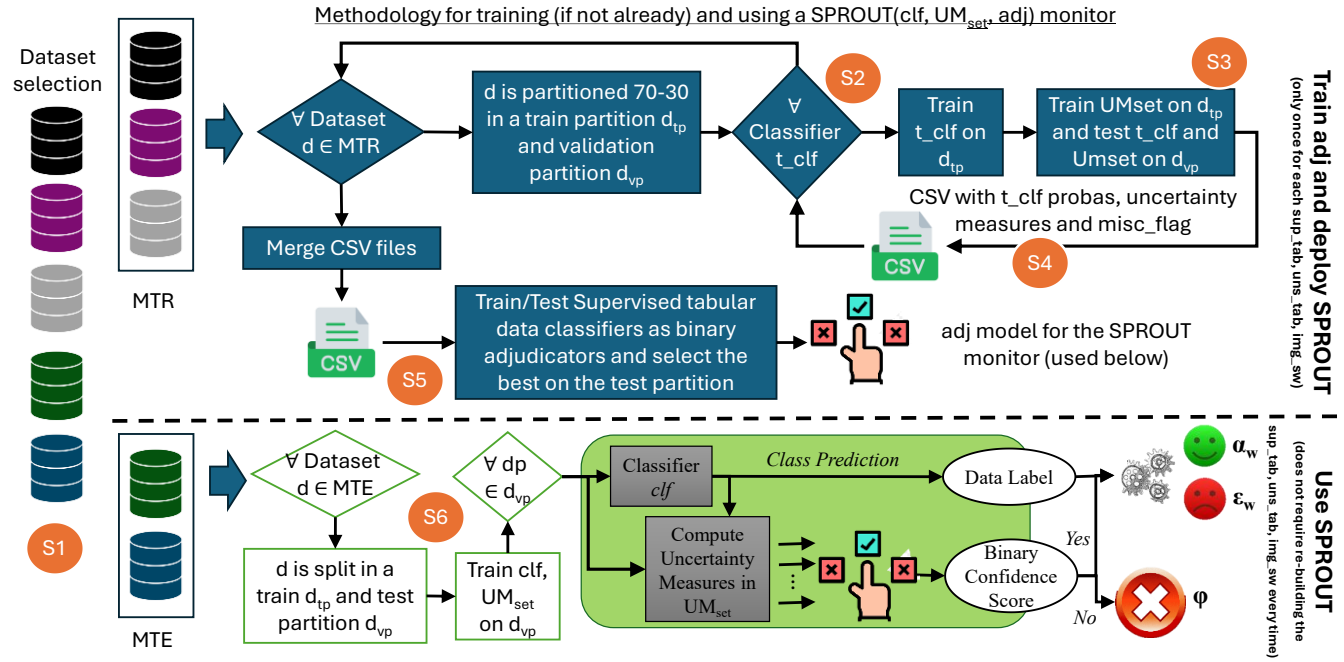


Figure 11: Picture of the experimental methodology for a generic SPROUT(clf, UM_{set}, adj). Steps are depicted as orange circles.

candidate classifiers are the supervised classifiers. Once the binary adjudicator is trained, compared and selected, each sup_tab, uns_tab, img_sm safety monitor is ready to be applied to a new dataset.

5. Apply sup_tab and uns_tab to the tabular datasets in the MTE partition, and img_sm to image datasets in the MTE partition (see bottom half of Figure 11), collect metric scores and discuss them in Section 7.3.4. The importance of uncertainty measures in suspecting misclassifications is discussed in a separate Section 7.3.5.

7.3.2 Setup with Tabular Data Classification (sup_tab, uns_tab)

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

7.3.2.1 Datasets

We consider the tabular datasets detailed in Section 3.1.1: 11 datasets of network intrusion detection, 10 datasets of sensor spoofing attacks to biometric authentication, 7 datasets related to hardware monitoring for failure prediction, 3 datasets related to IoT systems. All of them will be used for supervised classification (train and evaluate the *sup_tab* monitor), but only those 9 containing two classes might be used for unsupervised classification (*uns_tab* monitor), which is constrained to be binary by design.

Definition of MTR and MTE sets. For *sup_tab*, we craft a MTE = {SDN20, UNSW, Fingerprint, BackBlaze 2020, HAI Pressure}, leaving the remaining 26 datasets as MTR. For *uns_tab*, only the 7 HW Monitoring, Scania Trucks and HAI Pressure datasets can be considered as they have a binary label. In this case, MTE = {Mechanical Failure, HAI Pressure}, leaving the other 7 into MTR.

7.3.2.2 Classifier Selection

Amongst the many alternatives, we choose the following very well-known classifiers for tabular data. It includes Supervised and Unsupervised Classifiers and their configuration was tuned through grid searches, using the HyperOpt [130] library whenever applicable. Detail is given in Section 3.2.1

7.3.2.3 Uncertainty Measures and UM_{set} for sup_tab, uns_tab

According to the discussion in Section 7.1.2, the SPROUT monitors *sup_tab* and *uns_tab*, which apply to tabular data classifiers, will use the following uncertainty measures, which we instantiate with specific parameters when needed. Both monitors will use UM1) with w = 0.9, UM2) and UM3), which do not have parameters, UM8) with k = 19, which is prime to avoid ties in neighbor searches, UM9) using default parameter values.

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

- UM4, which does not need parameters.
- UM5 with *chk_c* = XGB, which is a notoriously good classifier [12]. The measure is referred to as UM5_XGB.
- UM6 with three different groups of checking classifiers. We indicate the three UM6 configuration as *UM6_ST)* {NB, LDA, LR}, *UM6_TR*) {DT, RF, XGB}, and *UM6_NB)* {GaussianNB, BernoulliNB, MultinomialNB, ComplementNB}. The UM6_NB group includes different variants of the Naïve Bayes classifier.
- UM7 with *bagC* = DT, which has low computational complexity and has overall good classification performance (we call this UM7 configuration as UM7_DT). Since bagging creates multiple instances of *bagC*, choosing a complex *bagC* classifier may make UM7 take too much time.

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

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

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

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

7.3.2.4 Binary Adjudicator

For supervised classification, we exercise each of the 9 supervised classifiers on each dataset in MTR, resulting in 9*26=234 experiments, each outputting a CSV file composed of *clf* probabilities and prediction, UM_{sup} values, and the *misc_flag*. These 234 CSV files are merged into a unique file composed of more than 13 million data points, which is used to train and validate the binary adjudicator. Supervised classifiers are tested as binary adjudicators using this dataset; results show that Random Forests (30 trees) are the

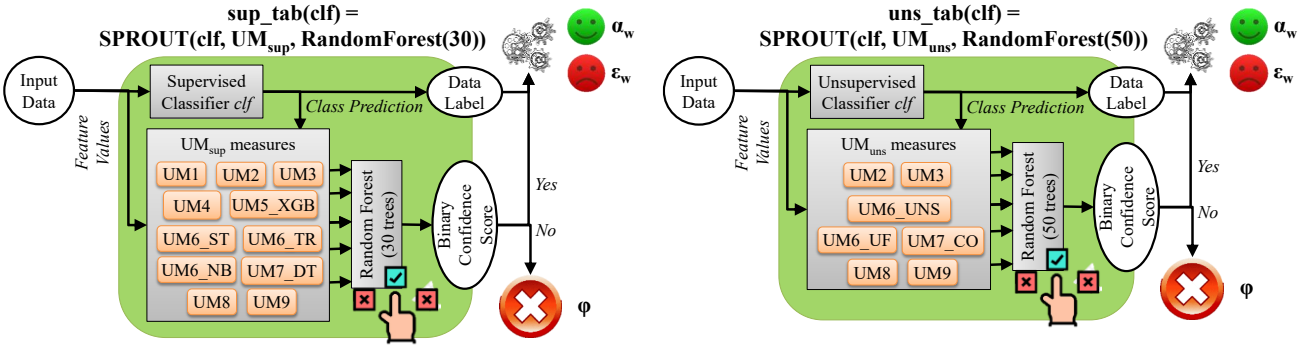


Figure 12: SPROUT monitors for supervised (sup_tab, left) and unsupervised (uns_tab, right) tabular classifiers.

preferred choice as binary adjudicator for the sup_tab monitor as they have better performance than other classifiers in suspecting misclassifications i.e., the binary adjudicator has to predict the *misc_flag* for a new data point. This completes the definition of the monitor:

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

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

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

These safety monitors are depicted in Figure 12.

7.3.3 SPROUT Setup with Image Classification (img_sm)

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

7.3.3.1 Datasets and Preprocessing

We are using 4 different RGB (3 channel) datasets: CIFAR10, FER2013, FOOD-101 and Flower for training and testing the proposed technique. These were the datasets of choice as they are quite small, contain images in different RGB formats and are used a lot in the literature, as detailed in section 3.1.2

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

7.3.3.2 *Classifier Selection*

Many DNN classifiers were proposed in the last decades, often with more and more demanding requirements concerning resource usage, posing challenges in scenarios where computational efficiency may be a limiting factor. Therefore, we selected DNNs for image classification that are relatively small and not excessively resource-hungry. We started from pretrained AlexNet, ResNet50, DenseNet, VGG11, GoogLeNet and InceptionV3 models, Detail of each classifier and their hyperparameter selection is given in section 3.2.2.

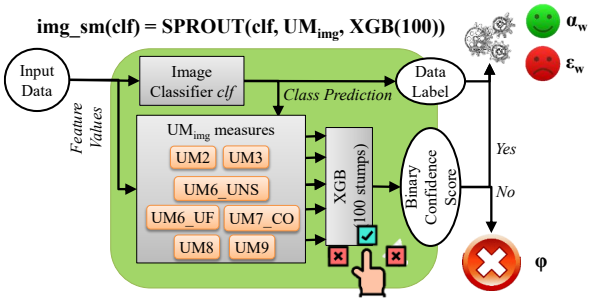


Figure 13: SPROUT for image classifiers img_sm.

7.3.3.3 *Uncertainty Measures and UM_{set} for img_sm*

According to the discussion in Section 7.1.2, the SPROUT monitor img_sm, which applies to image classifiers, will use the following uncertainty measures, which we instantiate with specific parameters when needed.

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

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

7.3.3.4 *Binary Adjudicator*

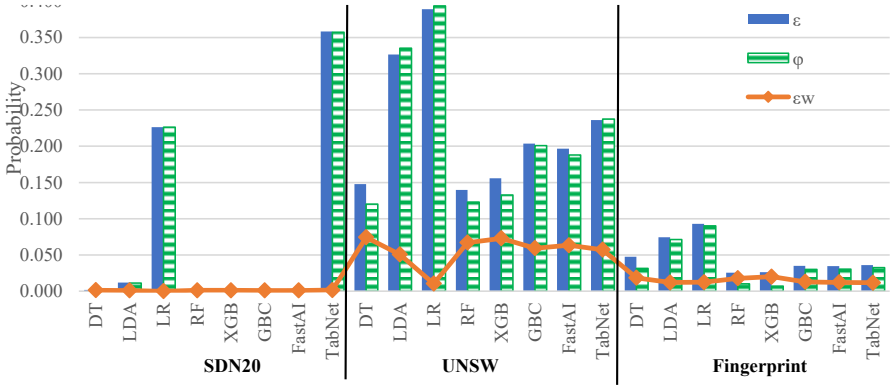
The process for training the binary adjudicator is similar to that used for *sup_tab* and *uns_tab* monitors. We exercise each of the 6 DNN image classifiers on each dataset in MTR, collecting results and merging it in a unique CSV of roughly 0.26 million data points. The CSV is used to train and validate the binary adjudicator. Supervised classifiers are tested as binary adjudicators using this dataset; results show that XGB (100 stumps) is the preferred choice as binary adjudicator for the *img_sm* monitor as they have better performance than other classifiers in suspecting misclassifications. This completes the definition of the monitor, depicted in Figure 13:

img_sm(clf) = SPROUT(clf, UM_{img}, XGB(n_stumps=100))

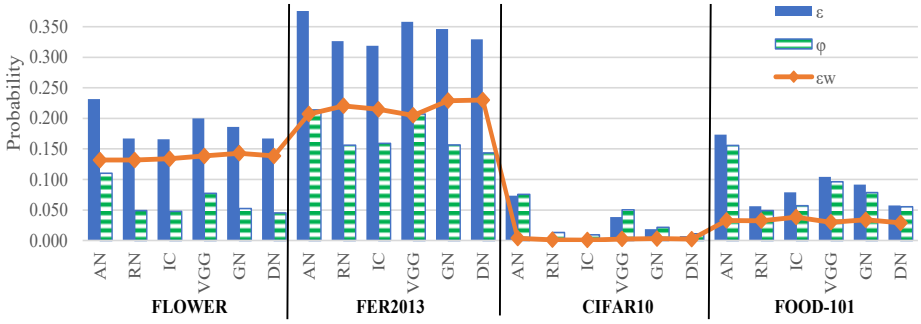
7.3.4 Results and Discussion

Once safety monitors are deployed, we can exercise them on datasets in the WTE groups (see bottom of Figure 11), collecting metric scores and comparing their behavior against traditional classifiers. Figure 14 presents a comparative analysis of a (supervised, unsupervised, image) classifier *clf* against a SPROUT

(a)



(b)



(c)

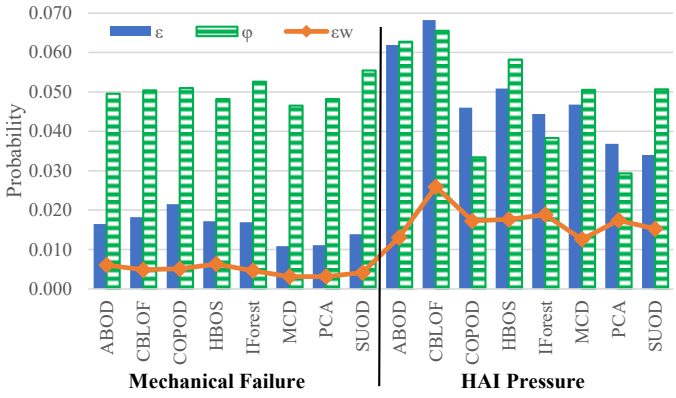


Figure 14: Misclassification probability (ε) of the classifier (solid bars), rejections (φ) (striped bars), and residual misclassification probability (εw) (line with diamonds) when applying sup_tab (up, 14a), img_sm (center, 14b), uns_tab(down, 14c) to test datasets.

monitor (respectively, *sup_tab, uns_tab, img_sm*) that uses *clf* as main classifier. The figure reports three different plots, where blue solid bars represent the misclassification probability (ε) of a *clf*, while the orange dotted line represents the residual misclassification probability (ε_w) after applying SPROUT. The green-striped bars depict the probability of rejections φ, ideally as low as possible.

Starting from the top of the figure (Figure 14a), we observe how ε_w of the *sup_tab* SPROUT monitors is always far lower than ϵ (i.e., orange bars hover on the bottom of the plot and are always lower than 0.1, whereas the blue bars may even reach 0.4) of the supervised classifier *clf*, being extremely close to the optimum $\varepsilon_w \approx 0$ on SDN20 dataset. There are cases in which applying SPROUT to a *clf* that has a high misclassification probability ε may yield to the total absence of residual misclassifications $\varepsilon_w = 0$, which is an excellent result. For instance, LR on SDN20 has $\varepsilon = 0.2263$, meaning that more than 1 out of 5 predictions of *clf* are misclassifications. Applying *sup_tab* leads to the total absence of residual misclassifications ($\varepsilon_w = 0$), which is an excellent result: the output of SPROUT is either a correct misclassification or a rejection, thus always trustworthy. As a potential drawback, *sup_tab* rejects more than 1 out of 5 predictions of the classifier ($\varphi = 0.2264$, see green-striped bar), which is not desirable; however, this high rejection probability is a direct consequence of the high ε of LR classifier on SDN20. When the *clf* to be monitored has high ε (as it happens in the UNSW dataset), an high omission probability φ is unavoidable, even when omitting all ($\varepsilon_w = 0$) and only ($\varphi_c = 0$) misclassifications. On the downside, an high φ may also point to rejections of predictions that were going to be correct classifications. In the TabNet application to the UNSW dataset, the ε and φ bars are of the same height (as in the case we discussed previously), but the ε_w quantity is far higher than zero. This means that a consistent part of rejections are unnecessary, as they represent rejections of predictions that were going to be correct.

Shifting to image classification in Figure 14b, we observe how the application of the *img_sm* monitor to the AN classifier on the Food-101 dataset allows for a substantial performance boost, with ε decreasing from 0.1669 to 0.0328 after applying SPROUT. Even better, almost all misclassifications were suspected when classifying images in the CIFAR-10 dataset, with a reasonably reduced amount of rejections (i.e., green bars are quite short). Results for the Flower and FER13 datasets show instead that misclassifications are reduced only by a small amount, with minor advantages from the SPROUT application to the classification of these images.

Table 8. Importance of uncertainty calculators for *sup_mon*, *uns_mon and img_mon* monitors to compute each measure .

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

Lastly, we comment on the application of SPROUT for unsupervised classification (*uns_tab*) in Figure 14c. The trend is similar to the previous pictures: however, ε_w is always far from 0 in the HAI Pressure dataset (right of the plot), meaning that *uns_tab* is not able to suspect and reject a relevant portion of the misclassifications of unsupervised classifiers. We explain this result as follows. First, unsupervised classifiers always have lower accuracy and higher misclassification probability than supervised classifiers, thus $\varepsilon >> 0$. Secondly, and more importantly, the uncertainty measures UM6_UNS, UM6_UF, UM7_CO that we compute for unsupervised classifiers are trained without knowledge of labels, making them less accurate in suspecting misclassifications with respect to those of *sup_tab* and *img_sm*. Even in these sub-optimal case studies, applying the *uns_mon* monitor to unsupervised classifiers at least halves their residual misclassification probability ε_w, thus $\varepsilon_w < \varepsilon / 2$.

7.3.5 Importance of Uncertainty Measures for Safety Monitors

Ultimately, we quantify the impact each uncertainty measure has on learning the model for binary adjudication of each monitor *sup_tab*, *uns_tab* and *img_sm* using Table 8. The table presents the importance of each uncertainty measure (shown in the columns) has for each safety monitor, measured via the *feature_importances_* function from *sklearn* (for Random Forest binary adjudicators, *sup_tab* and *uns_tab*) and *XGBoost* (for the adjudicator of *img_sm*) Python packages, ensuring that the contributions of all uncertainty measures sum up to 1 within each safety monitor. Missing values in the table indicate that uncertainty measures are not used by a particular SPROUT monitor i.e., they do not belong to their UM_{set}, while existing values are painted with a green gradient: the darker, the more a measure is relevant. In general, most of the variants of UM5, UM6, UM7 measures are very important for suspecting misclassifications, especially UM6_DNN for img_sm, UM6_UNS and UM6_UF for uns_tab, UM5_XGB and UM7_DT for *sup_tab*. Other uncertainty measures as UM1, UM8, UM9 have only a marginal role in suspecting misclassifications, having very low importance across the board on detecting misclassifications.

Another important metric is the time overhead that each uncertainty measure adds to the inference process. It is expected that SPROUT monitors will be slower and more resource consuming than classifiers alone, but this impact has to be estimated. The last row of Table 8 reports a qualitative estimation for the time needed to compute all uncertainty measures used by SPROUT. Some measures as UM2 and UM3 can be computed in negligible time and do not add any overhead to the classification task. Measures as UM6_DNN, UM8 and UM6_TR require heavy computations which may significantly slowdown the execution of the classification task. We are aware that the overhead generated by the application of SPROUT may constitute an obstacle in systems which are resource-limited or that have tight real-time deadlines. Should the system be tightly constrained on resource usage, the user may decide to design a SPROUT monitor that uses time inexpensive uncertainty measures: this may lower the effectiveness of safety monitors, but will also guarantee their applicability in almost every application scenario. Coming back to the SPROUT monitors presented in this thesis, with our experimental setup we found that applying *sup_tab* over a regular supervised classifier increases classification time by a factor of 4×, *uns_mon* triggers an even higher 5× overhead compared to only using an individual unsupervised classifier, while *img_sm* also results in a approx. 5× overhead, highlighting the computational trade-offs associated with each monitoring approach.

7.4 Threats to Validity and Reproducibility

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

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

Last, the usage of public data and public tools to run classifiers was a prerequisite of our analysis to allow reproducibility and to rely on proven-in-use data. All datasets are publicly available and referenced in the thesis for download at publisher's sites; code is available at [129].

7.5 Lesson Learned

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

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

Future work will focus on further reducing computational overhead, expanding support for additional uncertainty measures, and improving adaptability to dynamic, real-time environments. With its open-

source implementation [129], SPROUT is readily available for integration into autonomous decision-making pipelines, paving the way for trustworthy software-guided decisions.

8 FCCs for Trustworthy Federated Learning

In this chapter, we analyze how traditional Federated Learning (FL) limits client autonomy by requiring all participants to train the same model architecture and share model updates such as weights or gradients. This mandatory alignment of algorithms restricts flexibility and exposes sensitive model information during aggregation. Motivated by these challenges, we introduce a Trustworthy, Black-Box Federated Learning (TBB-FL) framework that allows clients to retain their preferred models as black-box components, pairs each model with a confidence calculator to reject uncertain outputs, and replaces parameter averaging with adjudication-based strategies for building a global model. This paradigm shift expands the applicability of FL, strengthens privacy, and supports more cautious and dependable collaborative learning.

8.1 Black-Box Models within Federated Learning Scenarios

Federated learning (FL) is a well-known computer system architecture in which multiple clients collaborate to solve machine learning tasks under the umbrella of a central server or aggregator [131]. This approach enables us to distribute the training process, avoiding the need to share training data and preserving data privacy via two main key principles: *local computation* and *model sharing*. The clients are treated as edge devices, collecting and locally storing data that is not requested to be transferred or shared, as illustrated in Figure 15.

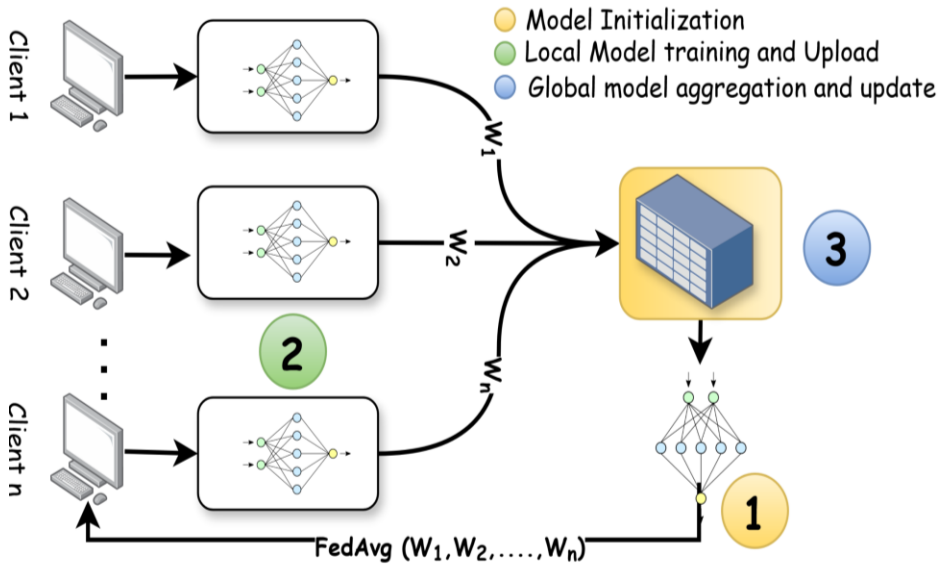


Figure 15: General Architecture of FL with the same local DNN and FedAvg.

FL has many applications across domains where data privacy, security, and regulatory compliance are critical. For example, in healthcare, multiple hospitals collaboratively train models on medical imaging data (e.g., MRI scans) without violating privacy regulations such as HIPAA and GDPR [132]. Google has deployed FL in its Gboard application to enhance next-word prediction without uploading user keystroke data to the cloud [133], and in finance, FL supports cross-institutional fraud detection without exposing confidential customer information [134]. These real-world deployments highlight FL's superiority over traditional centralized ML by preserving data confidentiality, reducing communication costs, and enabling scalable, privacy-preserving learning in diverse environments.

As a baseline behavior for FL, clients agree on a model (e.g., the same topology of Deep Neural Network, DNN [135]) that is then trained using client-specific data. Then, some details about models as weights or their gradients [136] are sent to a server for creating a global model through aggregation mechanisms, typically FedAvg, FedDyn, APFL, FedRep, FedPer [137]. Last, the server then sends the global model back to clients, who update their local model accordingly [138], possibly iterating over multiple rounds. Although FL offers exciting possibilities for building decentralized, privacy-preserving machine learning systems, it has drawbacks [139].

8.1.1 Clients need Freedom

A strong limitation that is often overlooked is that averaging techniques for deriving a global model require clients to agree on a specific topology of local model and thus share some insights. This makes clients unable to choose their preferred way to process specific information and may have a detrimental impact on desirable privacy properties since all clients know that others are using the same model, albeit trained with private data. Suppose three companies have their own strategy for detecting intrusions (i.e., a binary classifier for the sake of simplicity) by processing data gathered using a network/packet sniffing tool, and are ready to cooperate provided that data does need to be shared, which is a staple of FL. Even if they are willing to share and refine their knowledge within a collaborative learning task, within FL they are forced to start from scratch in training a brand new local model each, iterating until convergence of the training process. Under these circumstances, no FL will take place, and no enhanced global model will be produced.

Despite extensive research efforts, real-world applications of FL remain limited. **It would be more beneficial to let clients decide on the algorithm to be used to learn a local model, and redesign the**

way the global model is derived accordingly. Each client should have freedom to act on their local model, which is conceptualized as a black-box. Other clients taking part to the FL scenario should not be able to devise any insight of others' local models (i.e., no weights nor gradients), ensuring maximum privacy for both data and the model.

8.1.2 Can FL be Trusted?

Additionally, many innovative systems demand for making autonomous decisions (for which FL may be perfectly fitting) which, if incorrect, may have critical consequences [140]. Examples include, but are not limited to: fully autonomous driving, computer-guided robotic surgery, mobile robots for inspections and surveillance, optimized power management and generation, and manufacturing robots. All these cutting-edge innovations require trustworthy decision-making processes to provide their intended functionality without harming the health of people, infrastructures, environment, or causing financial losses. FL has a great potential to dealing with complex classification, regression, vision or perception tasks, but it has not been studied extensively for delivering trustworthy predictions. A typical approach [141] to design trustworthy software or components is to equip a functional component with another component that aims at triggering a fail-controlled, fail-safe, or fail-stop behaviour whenever the correct functioning is not guaranteed [142], [143]. Such self-checking component does not aim at building a perfect component; rather, it aims at suspecting and rejecting erroneous outputs, triggering adequate alerts and activating mitigation strategies instead of forwarding this erroneous output to the encompassing system. This is a proven-in-use approach that even allows for the integration of Commercial Off-The-Shelf (COTS) or non-trustable components in general into critical systems [144], [145].

8.1.3 Novelty

This chapter proposes a Trustworthy, Black-Box FL (TBB-FL) which extends traditional FL approaches by:

- allowing clients to choose the algorithm to learn their local model, which is shared as a black-box i.e., compiled executable;
- conceptualizing local models as a pair of i) (black-box) ML model and ii) confidence calculator, which quantifies to what extent the model prediction should be trusted, allowing to reject predictions at the local level.
- redesigning aggregation mechanisms [29], [31] to iteratively craft a global model to accommodate the changes above in ways other than averaging. Novel aggregation mechanisms may take advantage of confidence scores of each local model for the global prediction e.g., recovery blocks [146], delegating

[147], or decide using predictions only (e.g., voting). By design, the resulting global model will have built-in capabilities to not only output predictions, but also to decide on rejections similarly to what happens with local models.

TBB-FL allows for conducting collaborative learning scenarios ensuring the maximum privacy for both local data and models, integrating mechanisms to reject erroneous predictions to ensure improved performance with respect to local models only. Our proposal is evaluated and compared to traditional FL approaches via an experimental campaign conducting classification tasks with tabular and image datasets. Experimental results within the classification domain quantify how the integration of diverse client models and confidence-driven adjudication reduces residual misclassifications, still having comparable accuracy with traditional FL techniques. This assumes even more value as TBB-FL is evaluated under many experimental constraints (e.g., using a unique confidence calculator) and without running sensitivity analyses for hyper-parameters, which would have led to achieving even better results.

8.2 Federated Learning

Federated Learning (FL) was proposed by Google in 2016 [133] as a collaborative learning strategy where multiple decentralized clients contribute to the learning process, each retaining their private training data [138]. In other words, "*it allows for an extensive network of devices to partake in model training, each contributing with updated parameters like weights or gradients instead of exposing raw data, ensuring that personal and sensitive data remains securely within the device, aligning with rigorous data protection standards*" [148].

8.2.1 Basics

Within FL, clients first send metadata (e.g., class count, data shape) to the server, which uses it to initialize a first version of a global model. The model is sent to all clients who train it locally and return the updates, which are used to iteratively update the global model, which is then redistributed to clients. This process repeats for several rounds until convergence is achieved.

However, FL can be implemented using two main communication setups: centralized and decentralized. In the centralized approach, a single server one called the aggregator that manages the process. It sends an initial global model to all participating clients, collects the updates after they train locally on their own data, and combines these updates to refine the global model, a process often carried out using methods like Federated Averaging (FedAvg) [131]. This setup is efficient and straightforward to coordinate, but it depends heavily on the central server, making it vulnerable to a single point of failure. The decentralized

approach removes the need for a central server altogether. Instead, clients share model updates directly with one another, typically with their nearest peers, and may take turns acting as the temporary aggregator. While this peer-to-peer design improves scalability and resilience, it can also introduce additional security concerns, such as a higher risk of backdoor attacks. Common strategies for coordinating learning in decentralized systems include belief propagation [149] and consensus mechanisms [150], which help maintain consistency and trust across the network.

Federated Learning (FL) can be categorized into different types depending on how data is distributed among the participating clients. Horizontal FL applies when all clients have datasets with the same features but different users or samples. For example, two hospitals may record the same types of medical information but for different patients. Vertical FL, on the other hand, is used when clients have data about the same users but with different features. A typical case would be a collaboration between a bank and an online retailer, where both have information about the same customers but from different perspectives. Lastly, Federated Transfer Learning is useful when there is little overlap in both features and user samples across clients. In such cases, transfer learning techniques help align knowledge between domains.

8.2.2 Potential and Advantages

FL guarantees the following advantages. First, clients do not need to share their raw data to take part to the collaborative learning system; instead, they only share details of local models, keeping user data private and minimizing bandwidth usage as there is no need to share big datasets. Minimizing data exposure even helps in complying with regulations as the GDPR, HIPAA, and related laws. Second, as it happens with any collaborative learning scenario, FL allows for creating a global model that has better performance than local models individually thanks to the contribution of clients, which own and train using heterogeneous data, albeit related to the same task. Last, many clients may contribute to the learning process even when training has already been started, allowing FL to scale well even with a huge number of clients.

8.2.3 Disadvantages and Weaknesses

On the other hand, FL has some intrinsic weaknesses that are currently limiting its application in many scenarios, especially in industrial applications.

- **No Model Choice**: in centralized FL, the server shares an initial model – typically a DNN with default weights – that has to be updated locally at each iteration by clients. In decentralized FL, peers agree on a model, which will be used from now on to complete the learning process. Regardless of the specific FL setup, clients cannot choose their preferred algorithm to learn the local model: they are forced instead to train a brand new one that matches the expectation of the learning task. A client may

already have a well-performing local model at their disposal, but cannot share it within the FL scenario due to the constraint above. Moreover, all clients get to know the topology of the DNN used as global model, and may maliciously use this information to track back specific properties of local models of targeted clients.

- **Aggregation**: once a model has been agreed upon, weights or their gradients [136] are iteratively shared to update the global model through aggregation mechanisms, typically FedAvg, FedDyn, APFL, FedRep, FedPer, FedProx [137]. These strategies perform small variations of weighted averages, often trying to mitigate the problem of local optimization inherent in SGD-based approaches i.e., FedProx. However, literature [151] shows that the aggregation of different opinions or the combination of multiple models may be performed by means other than a weighted average or voting, which is the only option due to design constraints of traditional FL.
- **Handling Malicious Clients**: any distributed system or collaborative learning scenario is exposed to the problem of handling malicious of faulty clients. Those may (willingly) share data that has a detrimental impact on the learning task, deceiving the performance of the global model. Particularly, clients may show a behavior compliant with Lamport's [152] byzantine formulation, where their behavior may be inconsistently faulty, creating even more instability. Ideally, these clients should not take part to the FL task. However, once they are admitted to the client pool and start sharing data, they may lead to learning a global model that is biased, not accurate, or hiding backdoors [153].

8.2.4 Related Works

FL enables collaborative model training without centralizing data, offering strong privacy preservation and compliance with data protection regulations such as GDPR and HIPAA [131]. By keeping data localized and only sharing model updates, FL reduces the risk of direct data breaches. However, it faces significant challenges, particularly with non-independent and identically distributed (non-IID) data, where variations in client data distributions can slow convergence, reduce generalization, and in some cases cause model divergence [154]. Solutions such as FedProx [155], SCAFFOLD [156], and personalized FL approaches attempt to address these issues by correcting client drift, re-weighting updates, or tailoring parts of the model to individual clients.

Beyond heterogeneity, FL is highly vulnerable to poisoning and backdoor attacks, where adversarial clients submit malicious updates to degrade global model performance or embed hidden triggers [153]. Byzantine-robust aggregation methods (e.g., Krum, Multi-Krum, trimmed mean, FLTrust) have been proposed to mitigate such threats, yet they often trade off accuracy in benign settings. FL is also susceptible to adversarial examples at inference, which can cause incorrect predictions despite high overall accuracy. Additionally, noisy, corrupted, or mislabeled local data can propagate errors across the network, further degrading performance.

Even without raw data exchange, privacy leakage remains a risk, as model updates can be exploited through gradient inversion, membership inference, and property inference attacks. To counter this, research has explored secure aggregation protocols [131], homomorphic encryption, and differential privacy techniques [157], although these often introduce computational or accuracy overheads.

However, data shared from clients to the server may inadvertently reveal information about their private data, or malicious clients might compromise the model's integrity through attacks like poisoning or hindering its convergence [158]. Issues of FL can be partitioned into two categories: utility-centric and privacy-centric threats. Utility-centric threats aim to disrupt the integrity of data or models, ultimately diminishing accuracy. On the other hand, privacy-centric threats involve potential breaches of participant data, leading to the leakage of sensitive information. Moreover, these attacks can be further partitioned based on when they occur, either during the training phase (causative) or during the testing phase (evasion) [159].

In addition, Machine Learning (ML) faces challenges such as device and data heterogeneity, along with statistical heterogeneity. Data generated by different clients often follows distinct patterns, deviating from the independent and identically distributed (IID) assumption commonly referred to as statistical homogeneity or the closed-world assumption. Handling non-IID data is difficult, as it demands robust ML models and may lead to reduced performance [155], in addition to variations in transmission speed and computational capacity across clients, which affect FL efficiency. Nevertheless, when properly managed, data heterogeneity enables FL to derive more general and robust decision functions, making it increasingly relevant in edge computing scenarios, driven by advancements in modern devices like smartphones, wearables, and autonomous vehicles [101].

8.3 Towards a Trustworthy, Black-Box Federated Learning (TBB-FL)

The Trustworthy Black-Box Federated Learning (TBB-FL) framework is conceived to overcome key limitations of traditional Federated Learning (FL) while maintaining strict privacy guarantees and compatibility with heterogeneous models.

8.3.1 Concept

Unlike conventional FL, each model has to compute a confidence value, which is provided alongside with the prediction and quantifies to what extent the model’s output should be trusted. Additionally, TBB-FL operates exclusively with black-box local and global models, making traditional parameter-based

aggregation (e.g., FedAvg and derivatives) unfeasible. In TBB-FL, the aggregation of local models occurs not by merging weights, but by combining (black-box) model predictions and their associated confidence scores via voting, weighted confidence, recovery blocks, or other meta-learning strategies. This aggregation should leverage the strengths of diverse client models while maintaining strict isolation of their internals, reducing even more the likelihood – and the associated risk – of targeted adversarial attacks and model inversion. The confidence calculators play a central role here, as they must be designed to work alongside black-box models, thus computing confidence estimates purely from inputs and observable outputs (e.g., probability distributions, entropy measures, or auxiliary checkers) without requiring access to model-specific parameters nor additional training data.

This ensures that even in a privacy-preserving, black-box setting, the system can take advantage of improved prediction performance, reject outputs suspected to be wrong of uncertain, and ultimately achieve a behavior that could be deemed satisfactory and trustworthy for the final user.

8.3.2 Formalization

Within TBB-FL, local and global models are not simply ML models; instead, they are composed of two modules. The first is an ML model that exercises inference and acts as a black box (BBM), which has the usual train and test behavior:

(void) **bbm.train**(train_data, train_labels); (1)

pred = **bbm.predict**(test_data); (2)

Training can be supervised or unsupervised (thus, train labels are unknown). A *bbm* may perform a wide variety of tasks, ranging from classification to regression, spanning either towards natural language processing or other computer vision tasks as audio or speech processing.

The second module, which we call confidence calculator *CC*, computes confidence in a prediction via two main interfaces.

(void) **cc.train**(train_data); (3)

comp_conf = **cc.quantify**(pred, test_data, bbm) (4)

Similarly to the BBM, the CC may need to be trained as well. Once trained, the CC can be used to quantify the confidence in a prediction (*pred*) made by a specific BBM (*bbm*) processing a specific input

test_data. The bbm provided as input for quantifying confidence is intended to be an object that can be used within the *cc.quantify* function e.g., the cc may use the *bbm* to predict labels for data points other than test_data, but no internals of the *bbm* are disclosed nor required to compute the *comp_conf* value.

A pair of BBM and CC, together with a threshold value *thr*, build a Self-Checking Model SCM(bbm, cc, thr), which exposes the following functions:

<void> **SCM(bbm, cc, thr).train**(train_data, train_labels)

```
bbm.train (train_data, train_labels);
confC.train(train_data, train_labels);
```
(5)

<(x, y)> **SCM(bbm, cc, thr).predict**(test_data)

```
pred = bbm.predict(test_data);
conf = confC.quantify(test_data, pred, bbm);
return pred, conf;
```
(6)

<bool> **SCM(bbm, cc, thr).reject**(test_data)

```
pred, conf = SCM(bbm, cc, thr).predict(test_data);
return conf < thr;
```
(7)

In a nutshell, a CC acts as a safety monitor [141], [144] of a BBM: the main function is provided by the BBM, leaving the CC with the responsibility of quantifying to what extent the BBM's behavior should be trusted via a *quantify* function. Then, a threshold *thr* is applied to the confidence value devised by the *cc.quantify*, to understand whether the prediction of the BBM should be rejected. Safety monitors and, more in generally, self-checking components, are widely used in critical systems and software engineering to trigger a fail-controlled, fail-omission, or fail-stop behavior whenever the correct functioning is not guaranteed. Ideally, a self-checking component would aim to i) deliver a correct result or ii) reject those results that would have been incorrect i.e., the function should have fail-omission failures only. However,

Table 9. Notation used in the TBB-FL.

Notation	Explanation
N	Number of Clients
C	Number of Class
M	Number of Rounds
BBM	Black Box Model
CC	Confidence Calculator
SCM	Self-Checking Model (combination of BBM and ConfC)
SCM_i	A local SCM that is created and managed by the i-th client, $0 \leq i < N$
G-SCM	Global SCM
TD	Task Details
CS	Clients Set
adj	the adjudication strategy to be used for aggregation, replacing traditional averaging mechanisms
def_{CC}	a general-purpose CC that is provided to all clients

it is frequently the case in which i) incorrect predictions do not get rejected, or ii) correct predictions are erroneously rejected, making the development of this component challenging, but at the same time very rewarding.

Participants to a TBB-FL scenario will always assume to deal with SCMs, either on a local or on a global scope. This allows defining TBB-FL(TD, CS, adj, def_{CC}) as a quadruple:

- **TD** is a set of Task Details, which are general guidelines about the type/shape of data to be processed, and of the task e.g., binary/multi-class classification with c classes. The TD puts absolutely no constraint about the ML algorithm to use to build a BBM and thus, a local SCM;
- **CS** = $\{client_i, 0 \leq i < N\}$ is the set of N clients that take part to the collaborative learning scenario;
- **adj** is the adjudication strategy used for aggregation, replacing traditional averaging mechanisms;
- **def**$_{CC}$ is a general-purpose CC that is provided to all clients to build their local SCM in case they do not intend to devise their own rejection strategy.

Note that the formalization of TBB-FL is general enough to be applied to any predictive task, requiring no additional constraints nor adaptations. Clearly, the way BBMs and CCs will be implemented will depend on the data type (tabular, image, audio) and task (classification, regression, object detection, segmentation) at hand. The notation used in this section is summarized in **Table 9**.

8.3.3 Allowing Black-Box Local Models

Additionally (see Section 8.2.3), forcing clients to re-train from scratch different copies of the same local model makes it impossible to reuse existing machinery within a FL or even a collaborative learning paradigm. This is unfortunate, as companies or practitioners may be available to share their existing model for solving a specific task, but have absolutely zero willingness to retrain another version just to take part to a FL scenario. To solve that, FL should allow clients to reuse existing local models as black-boxes, and maybe refine them within the iterative learning process. This change will dramatically increase the possibility for practical FL applications and easily involve companies or practitioners that already have a working solution, increasing *diversity* [141] of local models that take part to building a global model, which is considered one of the enabling conditions for redundant, n-version, or ensemble systems to provide enhanced capabilities compared to standalone techniques.

8.3.4 Required Changes to FL

Practically speaking, there need to be two major changes:

- conceptualize local and global models as self-checking components, that pair the ML model with an additional piece of software to quantify the confidence in a prediction. The confidence value will be presented as an output alongside with the prediction, and could be used to reject uncertain outputs, or to trigger other mitigation strategies;
- allow clients to select their algorithm of choice to build the local model. This i) ensures that existing machinery could be reused without the need to train from scratch to enter a FL scenario, ii) promotes diversity, iii) designs local models as black-boxes, meaning that other clients have no clue about how other local models look like.

8.4 Methodology and Building Blocks of TBB-FL

A generic TBB-FL task develops as described in Section 8.4.1. Section 8.4.2 to 8.4.4 provide insights on key building blocks as the training rounds, confidence calculators, adjudicators and evaluation metrics, letting Section 8.4.5 to summarize the innovations of TBB-FL against traditional FL.

8.4.1 Methodology and Execution Flow

The overall execution flow of TBB-FL is illustrated and detailed in Figure 16 The methodology proceeds through the following steps, depicted from (1) to (5) in the figure.

1. Initialization. TD is delivered to all clients, which take note of the general information and prepare to using it to learn from scratch of adapt (e.g., transfer learning) an existing ML model to craft a local BBM. Also, each client stores the def_{CC} for potential future usage when building the local SCM.

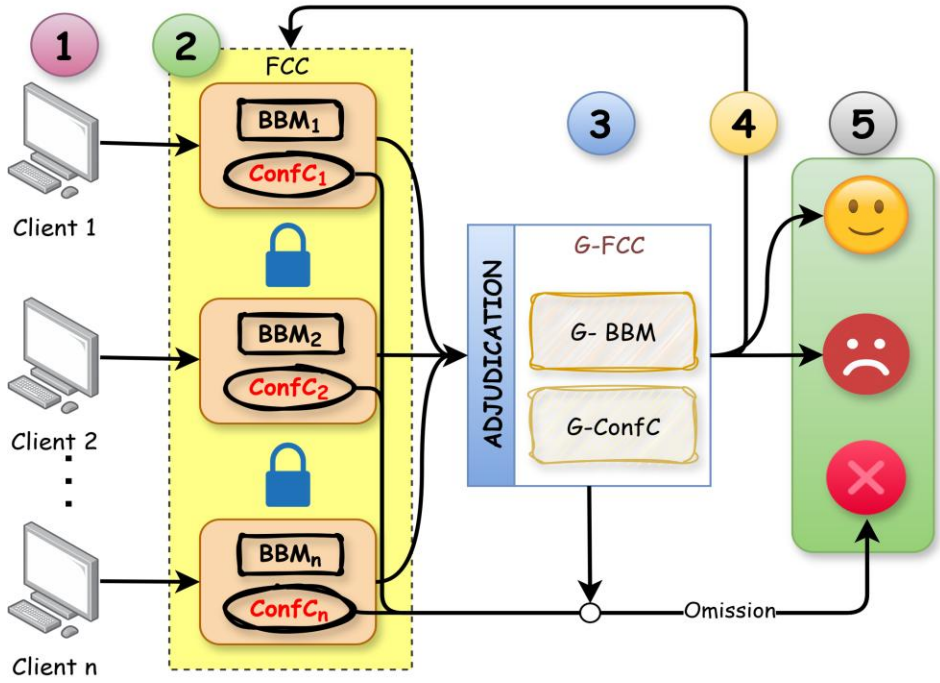


Figure 16: Architecture of TBB-FL, including FCC local models and the global model obtained through adjudication.

Table 10. Comparison between FL and TBB-FL.

Item	FL [141]	TBB-FL
I1: Client – Local Algorithm	Agreed - Same	Up to each client
I2: Global Model	FedAvg and derivatives	Adjudication via voting, recovery blocks, delegating
I3: Confidence Quantification	No Estimation	Each local model, takes part to the global model
I4: Exposure to (adversarial) attacks	Subject to targeted attacks to the specific algorithm agreed by clients	Only untargeted attacks as local algorithms are heterogeneous and private
I5: Poisoning Global Model	Need to train the global model from scratch in case a client is identified as malicious during or after training.	Can rollback to global model obtained after training rounds where the malicious client was not in the FL yet
I6: Local Model disclosure directly with clients	nothing	nothing
I7: Local Model disclosure with server	Weights (Gradients) of DNN edges	Single executable, non reverse-engineerable

2. Local SCM Construction. Each client trains their local BBM (and confidence calculator, if not using def_{CC}) using their private data and without constraints other than those in TD. The resulting BBM may even be an existing model (i.e., no training effort, no additional data required), a slight adaptation of an existing model (i.e., negligible to very low training effort), or a model trained from scratch for participating to the TBB-FL collaborative learning scenario. At the end of this step, each client will have their own SCM, which is ready to be shared as an executable.
3. Adjudication and Global Model. The server gathers executables of SCMs from clients and use them to create a global model G-SCM via a specific adjudication strategy *adj*. The adjudication function has very limited complexity as it assumes absence of data and absolutely no insights of local SCMs.
4. Global Model Deployment. As it is common in FL, the updated global model G-SCM is sent back to the clients as a singular executable. This ensures that clients cannot access the executables of other clients but just use the (black-box) global model themselves. The G-SCM should directly replace local models and allow clients to take advantage of the improved model.
5. (Self-Checking) Inference. Being SCMs, local and global models allow for rejecting predictions that are suspected to be wrong, enabling a more cautious, but also less error-prone, behavior.

8.4.2 Learning Rounds

Steps 2 to 4 of the methodology are iterated over a given number of rounds $M \geq N$. Within each round, a client may be added, removed, or share an updated local SCM. Each of these changes have an impact on the G-SCM, which is updated accordingly. As a result, the number of rounds is at minimum the amount of clients, but may be far bigger should clients update their local SCM or being connected and disconnected multiple times from the FL collaborative task.

This concept of rounds allows to keep track of the evolution of the G-SCM. A newly added client may not help in improving prediction capabilities at the global level, or may even try to damage the whole FL

collaborative task should it have malicious intent. If clients are added at the same time and the G-SCM is built only once by aggregating all local SCMs, there would be no easy way to understand what is the local SCM that deceives the G-SCM, constitutes noise, or brings no useful information. Each update of local SCMs makes for a new round, gradually evolving the G-SCM and even allow to back-track to G-SCMs obtained in previous rounds in case the predictive performance of the G-SCM at the current round is deemed inferior with respect to previous versions. When this happens, the motivation can easily attributed to the behavior of the client involved in the last round, allowing to quickly identify and potentially isolate a malicious client, which is far from trivial in traditional FL.

8.4.3 Adjudication to build a Global Model

TBB-FL allows clients for choosing their preferred way of learning a local BBM and SCM, making traditional federated averaging mechanisms unapplicable. The global model (G-SCM) has to be devised according to this paradigm shift as an adjudication of black-box local SCMs, for which only a non-reverse-engineerable, protected executable is provided. In TBB-FL, the traditional federated averaging or aggregation is replaced with a function that processes the outputs of local SCMs (prediction and confidence), and uses them to derive a global prediction via an adjudication [29] mechanism. This calls for strategies similar to those used to manage n-version, modular or redundant systems and software, where different replicas are orchestrated independently and then adjudicated via strategies as (weighted) voting [29], recovery blocks [30], n-self-checking [28], delegating [112].

8.4.4 Evaluation Metrics

Evaluating the predictive performance of a TBB-FL collaborative scenario comes down to quantify the performance of a (G-)SCM. In a nutshell, we want to maximize correct predictions, and minimize unnecessary rejections, that is, when a correct prediction gets rejected.

The concept of correct prediction, and connected evaluation metrics, depend on the task at hand. Within classification, an SCM should maximize accuracy (i.e., percentage of correct predictions [161]), trying to reject all (and only) misclassifications. In regression tasks, an SCM should aim at rejecting those predictions that are far from the true value, or that have an absolute error that is higher than acceptable. For object detection and other tasks, metrics are even more domain-specific and cannot be generalized here. Indeed, we provide a full specification of evaluation metrics for TBB-FL and SCMs intended to perform classification tasks, which are common in many critical applications, from error, intrusion, anomaly

detection to image classification and failure prediction. Let us assume that a BBM has an accuracy α_{BBM} and a misclassification probability ε_{BBM} such that $\alpha_{BBM} + \varepsilon_{BBM} = 1$. When dealing with a binary or multi-class classification task, the performance of an SCM can be evaluated relying on the quantities defined in section 5.2

An SCM will typically have lower accuracy than the BBM used within the SCM, as it does not aim at improving correct classifications, but to reduce misclassifications. The SCM provides a way to prevent a misclassified prediction from propagating through the system, potentially causing (catastrophic) failures and overall posing severe hazards to trustworthiness.

8.4.5 Comparison w.r.t. Traditional FL

The main differences of our proposal compared to traditional federated learning (FL) [157] are summarized in Table 10. The table outlines several discussion points already addressed in our work and highlights how the proposed approach enhances protection against potential attacks by design.

Local SCMs, and consequently the global SCM, are treated as black-boxes in TBB-FL, meaning that internals are not accessible. This allows for freedom to choose the algorithm to learn a local model (I1 in the table), which is shared as a single compiled executable (I6, I7), which we assume to be not reverse – engineerable (see threats to validity in Section 8.6). This means that clients have no clue about each other's models, making it even more difficult for an adversary to target a specific model as in traditional FL or single-model inference setups (I4). Common targeted attacks in conventional FL as model inversion, membership inference, gradient poisoning, and backdoor attacks are just not an issue in TBB-FL. The global model is created as an adjudication of local models (I2), meaning that the G-SCM builds its prediction as an aggregation of diverse opinions, which is a concept that is typically considered beneficial and reduces the feasibility even untargeted attacks, which are less likely to succeed [158]. Should a client suspected to be malicious during or after the training process, TBB-FL could rollback the current (or final) G-SCM to a version obtained after a previous round where such malicious client was not collaborating yet to the FL scenario (I5). This is not possible in traditional FL, where at the end of each round the global model gets updated as a combination of the updated weights or gradients of each participating client. Last, but not least, TBB-FL natively considers models to be SCMs, meaning that the prediction is paired with a confidence estimation that could be used to decide on prediction rejection, which is something that is not common in FL and will require a-posteriori adaptations. Also, having local SCMs compute confidence

independently allows to use such confidence not only to decide on rejection at a local level, but also to craft more sophisticated aggregation/adjudication strategies, enhancing the computational performance of the G-SCM.

8.5 Experimental Evaluation

Here we report the experimental setup (Section 8.5.1 to Section VI.8.5.4), discuss experimental results comparing TBB-FL against individual local models (Section 8.5.5) and compare TBB-FL against traditional FL in Section 8.5.6.

8.5.1 Experimental Methodology and Framework

Our experiments develop as follows.

- We select a set of datasets, classifiers and confidence calculators as explained in Section 8.5.2, Section 8.5.3, Section 8.5.4, respectively.
- Each dataset is partitioned with 60% used for training and 40% for testing. The training portion is further split into six disjoint subsets, with each client receiving a unique subset (10% of the initial dataset), considering the test set as shared across all clients for evaluation.
- In each experiment, training completes over six rounds, adding a new client, and local SCM, at each round. Note that we select 6 distinct classifiers for tabular data and 6 for image classification.
- At the end of each round, local SCMs are adjudicated into the G-SCM by means of a majority voting mechanisms. The final result is the one that receives the highest number of votes from the individual clients.
- All classification tasks will be primarily evaluated via (balanced) classification accuracy α, rejection rate φ and misclassification rate ε, see Section 8.4.4.

As a reference framework for FL experiments, we used the well-known Flower framework [162]. Flower provides flexibility in choosing various machine learning models and supports diverse types of datasets, making it suitable for our experimental setup. To make Flower able to run TBB-FL, we exercised minimum changes which can be found - along with supporting source code – in the anonymous GitHub [163].

All experiments were conducted on a Dell Precision 5820 Tower equipped with an Intel Xeon Gold 6250 CPU, NVIDIA Quadro RTX 6000 GPU with 24GB VRAM, 192GB RAM, running Ubuntu 18.04, with NVIDIA driver version 450.119.03 and CUDA 11.0.

8.5.2 Dataset Selection

To maximize the relevance of the evaluation, we selected datasets containing image and tabular data that are related to critical applications. Each dataset was pre-processed by partitioning it into independent and identically distributed (IID) subsets across clients, ensuring uniform data distribution for simulating a standard federated learning environment. For tabular data, preprocessing involves handling of missing values and partitioning features and labels in a client-specific manner to simulate data heterogeneity. For image data, preprocessing includes resizing, normalization, and data augmentation (such as flipping and rotation), with client-specific partitions designed to reflect IID distribution across local datasets. We selected tabular datasets related to error and intrusion detection detailed in section 3.1.1 Also, we selected 3 Image datasets such as Tencent 100K, CCTSDB2021, and NCT-CRC-HE-100K as detailed in section 3.1.2.

8.5.3 Classifier Selection

We selected 6 supervised ML algorithms from *scikit-learn* and *xgboost* to process tabular data as detailed in section 3.2.1 and for Images we selected 6 deep classifiers from *PyTorch-zoo* as described in section 3.2.2.

8.5.4 Confidence Calculators for Classifiers

Several confidence quantification strategies can be used to build the Confidence Calculator (CC) within local and global SCM. We describe four possible approaches:

- CC1 measures confidence using the inverse of the highest class probability, lower maximum values suggest higher confidence [123].
- CC2 uses the probability assigned to the predicted class (i.e., maximum probability): the closer to 1, the more confident [39].
- CC3 introduces an additional "*checker*" classifier that compares its prediction with the main classifier's output; it adjusts the confidence score depending on whether the two agree and how certain the checker is, producing values between -1 and 1.
- Finally, CC4 takes this idea further by using multiple checker classifiers and averaging their CC3 scores, resulting in a more robust, consensus-based confidence estimate. These approaches offer flexible ways to quantify confidence and can be adapted based on the needs and setup of the federated system.

For the sake of simplicity and easiness of presentation in this chapter, we set def_{CC} = CC2, and force all SCMs (both tabular and image) to use this as CC, rejecting those predictions for which the CC2 value is

lower than 0.8, or 80%. This is an arbitrary threshold that can negatively impact the behavior of TBB-FL as discussed in the threats to validity of this study

8.5.5 Results and Discussion

We perform experiments on both image and tabular dataset. Figure 17 presents a visual comparison of the performance of TBB-FL against a Best Local Model (BLM), where BLM refers to the highest-performing local BBM (thus, no confidence estimation) among all participating clients, for each dataset. Each green diagonal-striped bar corresponds to the accuracy (α), the yellow bars indicate the rejections (φ), and the red vertical-striped bars represent misclassifications (ε). It can be observed that the rejection bars vary noticeably across datasets. For instance, some datasets show relatively wider yellow bars, indicating a higher rate of rejected predictions, while others have shorter yellow bars, suggesting fewer rejections.

Across the datasets, a clear pattern emerges: a controlled use of rejections often correlates with reduced misclassification rates particularly when compared to the Basic Local Model (BLM), which lacks rejection mechanisms i.e., φ is always 0. In the CCTSDB2021 dataset, the rejection rate is notably low (φ = 0.71%), yet the misclassification rate is minimal (ε = 2.41%) compared to BLM (ε = 2.31%) with similar accuracy, indicating effective uncertainty filtering. Likewise, CICIDS18 and NCT-CRC maintain moderate rejection

Figure 17: Performance of TBB-FL and Best Local Model (BLM) across Image and Tabular Dataset depicting accuracy (α, diagonally green-striped), rejection (φ, yellow solid) and residual misclassification (ε, vertically red-striped).

rates (φ = 3.25% and 1.94%) and achieve lower misclassification (ε = 2.63% and 4.55%) compared to their BLM counterparts (ε = 2.73% and 3.37%).

On the other hand, in the ARANCINO dataset, TBB-FL yields a high misclassification rate (ε = 17.75%) despite low rejection (φ = 1.26%). Notably, it still achieves slightly higher accuracy than the BLM (ε = 19.77%), a rare occurrence that may arise due to voting-based adjudication mechanisms. In contrast, UNSW-NB15 shows a high rejection rate (φ = 14.23%), reducing misclassification (ε = 7.53%), suggesting that while many samples were rejected, not all were effectively contributing to error reduction.

These findings highlight the utility of rejection as a selective mechanism to improve decision reliability. When implemented, rejections can reduce incorrect predictions and preserve model performance particularly in heterogeneous federated environments where client models vary in capacity and confidence. Notably, the tabular datasets tend to display more prominent rejection bars compared to the image datasets, highlighting differences in rejection behavior between the two data types.

8.5.6 Benchmarking TBB-FL against Conventional FL Approaches

To enable fair comparisons, we conducted experiments using several traditional federated learning techniques on the image datasets described in Section 8.5.2. We disregarded the comparison with tabular datasets as FL is more applied for image classification than for processing tabular data. **Table 11** presents a comprehensive performance comparison, carried out using the Flower framework with its default settings for different global models as FedAvg, FedPara, FedOpt, FedAvgM. It is evident from the results that default FL methods aim at maximizing accuracy but may yield a relatively high number of misclassifications. This is effectively addressed by TBB-FL (last row of the table), which has always the lowest amount of misclassifications, with comparable – albeit not excellent due to rejections - accuracy with respect to its competitors.

Table 11. Accuracy (α) and Misclassification (ε) of Traditional Federated Learning Methods on Image Datasets

Dataset Name→	**NCT-CRC-HE-100K**			**CCTSDB2021**			**Tencent 100K**		
FL Techniques ↓	**α**	**ε**	**φ**	**α**	**ε**	**φ**	**α**	**ε**	**φ**
FedAvg	0.9544	0.0456	-	0.9758	0.0242	-	0.9322	0.0678	-
FedPara	0.9467	0.0533	-	0.9699	0.0301	-	0.9301	0.0699	-
FedOpt	0.9488	0.0512	-	0.9702	0.0298	-	0.9317	0.0683	-
FedAvgM	0.9546	0.0454	-	0.9697	0.0303	-	0.9298	0.0702	-
TBB-FL (ours)	0.9467	**0.0337**	0.0196	0.9739	**0.0188**	0.0073	0.8978	**0.0184**	**0.0838**

8.6 Threats to Validity and Limitations

Here we list possible issues or threats to validity of this study, which should not be intended as showstoppers, but may limit the generalization of our findings in specific scenarios. Some of them are already tackled as future works in the next section.

8.6.1 Confidence Calculators

Adopting SCMs and self-checking components is beneficial only when prediction errors can be suspected effectively, or when the quantification of confidence is tailored extremely well to avoid "overconfident errors", which is a key issue for any approach that revolves around uncertainty, confidence, trust. Within this chapter, we discussed Confidence Calculators CC in general, suggesting some strategies that could be applied for classification tasks i.e., CC1 to CC4. However, our experiments only discussed the behavior of TBB-FL when all SCMs use CC2, restricting the pool of possible configurations and potentially leaving behind optimal scenarios that would have made TBB-FL perform even better. However, even under this constraint, results in Section 8.5.6 show that TBB-FL outperforms local models and has far lower misclassifications than traditional FL, with comparable accuracy. The reader should see this as a limitation to this study, and a very easy way to improve already positive experimental results presented therein. Recent works showed how using ensembles of confidence measures rather than a single technique could limit this issue and result in more trustworthy confidence quantification [74], [94] and will be used as a baseline for future works.

8.6.2 Experimental Limitations

More in general, while the proposed TBB-FL framework demonstrates promising results. All experiments were conducted using a fixed set of datasets and relied on a single CC. Additionally, all clients adopted predefined classifier hyperparameters, and the evaluation was carried out over six rounds where clients are added one at a time, without simulating the removal or update of local SCMs. This design ensures consistency and simplifies the presentation in this chapter, but restricts exploration of the broader design space, such as dynamic thresholds, diverse confidence scores, and use of non-IID datasets, which is an important topic in trustworthy applications but could not fit the current page limits.

8.6.3 Performance and Bandwidth Issues

TBB-FL requires to share and exchange entire executable files with the server: in case those contain complex models, they may fill the network bandwidth and cause latency issues. The impact of this issue

largely depends on the number of clients and on how often the global model is computer and re-distributed, which is typically not frequent. Indeed, each round only updates, removes or adds a single local SCM, instead of requiring updates by all clients as in traditional FL.

8.6.4 Executables and Reverse Engineering of BBMs/SCMs

One of the assumptions of TBB-FL is that local SCMs are seen as black-boxes, since they are shared as compiled executables. It is true that – under specific circumstances – a compiled executable could be reverse engineered, potentially obtaining assembly code. However, this requires knowing the ISA of the hardware, linking options and additional parameters. Even having an assembly equivalent of the high-level code (which is very difficult in the general case and requires a very motivated and informed attacker), it will be barely possible to get back to source code that is high-level enough to understand the model and craft targeted attacks depending on this understanding. Thus, we consider this assumption to hold even if it could theoretically be broken as the likely to do so is practically negligible.

9 CONCLUSION

This thesis called for a paradigm shift in the way classifiers are conceptualized, designed, and deployed within safety-critical systems, where misclassifications may lead to catastrophic consequences. Traditional research largely aims at developing highly accurate classifiers, expecting them to operate correctly under all circumstances. However, this goal is inherently unrealistic. Classification tasks are complex, and operational environments expose systems to unknown, unexpected, or anomalous conditions for which no guarantees of correct functioning can be provided.

This thesis argues 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 desirable properties including trustworthiness.

To support this perspective, the thesis introduced Fail-Controlled Classifiers (FCCs), which exercise traditional classifiers with additional components that suspect misclassifications and reject potentially incorrect predictions. This converts unhandled content failures into omission failures that can be managed through system-level mitigation strategies, such as activating recovery routines or transitioning into a safe state. Ideally, all and only misclassifications should be rejected. Excessive rejections reduce availability, whereas insufficient rejections compromise safety. To explore this balance, nine FCC architectures i.e., SCC, WT, IP, OP, SW, RB, VT, WVT, and STK were presented, analyzed, and experimentally evaluated. Each architecture offers different trade-offs between accuracy, availability, and robustness, and must be selected based on system-level requirements, operational constraints, and tolerance for omissions.

The extensive experimental evaluation, conducted across a diverse set of tabular and image datasets, revealed further insights. The performance of classifiers especially those using Machine Learning, is sensitive to hyperparameter tuning, overfitting, feature quality, and data distribution. However, the purpose of the evaluation in this thesis was not to optimize specific classifiers but to compare them consistently against their FCC counterparts. Results demonstrated that FCCs reliably reduce misclassifications by converting them into rejections. Notably, this thesis is among the few studies evaluating trust mechanisms across dozens of datasets spanning both structured and unstructured domains, reinforcing the generality and validity of its conclusions.

Additionally, the thesis introduced SPROUT (Safety wraPper thROugh ensembles of UncertainTy measures), a black-box safety monitor designed to detect and reject predictions that cannot be trusted. Unlike existing uncertainty-based or failure-detection approaches, which often rely on white-box access, SPROUT operates without any internal knowledge of the classifier. It integrates multiple uncertainty measures into a unified confidence score and applies adjudication rules to decide when to reject a prediction. Experimental results across 35 datasets and more than 20 supervised and unsupervised classifiers showed that SPROUT can consistently identify and reject a significant proportion of misclassifications, in some cases detecting all misclassified samples of specific models. These results highlight the potential of uncertainty ensembles as a powerful foundation for black-box safety monitoring.

Lastly, this thesis further extended the concept of fail-controlled behaviour into distributed learning environments by introducing Trustworthy, Black-Box Federated Learning (TBB-FL). Traditional Federated Learning (FL) frameworks assume that clients share the same model architecture and exchange weights or gradients, which restricts flexibility, reduces privacy, and creates entry barriers for organizations with proprietary models. TBB-FL eliminates these limitations by treating each client's model as a black box and aggregating only their predictions and confidence assessments. This enables clients to participate without altering or exposing their local models. Additionally, the integration of local and global rejection mechanisms enhances resilience and reliability during aggregation, making TBB-FL particularly suitable for critical applications such as medical diagnostics, infrastructure monitoring, and cybersecurity. Experiments on both tabular and image datasets demonstrated that TBB-FL improves robustness, reduces misclassifications, and provides strong privacy-preserving guarantees compared to conventional FL approaches.

As a concluding remark "If you can't say something nice, Don't say nothing at all" (Bambi, 1942).

REFERENCES:

[1] S. M. Noble, M. Mende, D. Grewal, and A. Parasuraman, “The Fifth Industrial Revolution: How Har-monious Human–Machine Collaboration is Triggering a Retail and Service [R]evolution,” *J. Retail-Ing*, vol. 98, no. 2, pp. 199-208, Jun. 2022, doi: 10.1016/j.jretai.2022.04.003.

[2] A. S. Ami and al, “False negative-that one is going to kill you."-Understanding Industry Per-spectives of Static Analysis based Security Testing,” in *2024 IEEE Symposium on Security and Privacy (SP*, IEEE Computer Society, Oct. 2023, pp. 19–19.

[3] C. G. Keller and al, “Active pedestrian safety by automatic braking and evasive steering,” *IEEE Trans. Intell. Transp. Syst.*, vol. 12, no. 4, pp. 1292–1304, 2011.

[4] J. Guérin, R. S. Ferreira, K. Delmas, and J. Guiochet, “Unifying evaluation of machine learn-ing safety monitors,” in *2022 IEEE 33rd International Symposium on Software Reliability Engineering (ISSRE*, IEEE, Oct. 2022, pp. 414–422.

[5] A. Biondi and al, “A safe, secure, and predictable software architecture for deep learning in safety-critical systems,” *IEEE Embed. Syst. Lett.*, vol. 12, no. 3, pp. 78–82, 2019.

[6] V. Chandola, A. Banerjee, and V. Kumar, “Anomaly detection: A survey,” *ACM Comput. Surv. CSUR*, vol. 41, no. 3, pp. 1–58, 2009.

[7] A. Sarıkaya, B. G. Kılıç, and M. Demirci, “RAIDS: Robust Autoencoder-Based Intrusion Detection System Model Against Adversarial Attacks,” *Comput. Secur.*, vol. 103483, 2023.

[8] S. Ö. Arik and T. Pfister, “TabNet: Attentive Interpretable Tabular Learning,” *Proc. AAAI Conf. Artif. Intell.*, vol. 35, no. 8, Art. no. 8, May 2021, doi: 10.1609/aaai.v35i8.16826.

[9] M. J. Mirza *et al.*, “Robustness of object detectors in degrading weather conditions,” in *2021 International Intelligent Transportation Systems Conference (ITSC*, IEEE, Sep. 2021, pp. 2719–2724.

[10] M. Mathias and al, “Traffic sign recognition—How far are we from the solution?,” in *The 2013 international joint conference on Neural networks (IJCNN*, IEEE, Aug. 2013, pp. 1–8.

[11] A. Ceccarelli and F. Secci, “RGB cameras failures and their effects in autonomous driving applications,” *IEEE Trans. Dependable Secure Comput.*, 2022.

[12] R. Shwartz-Ziv and A. Armon, “Tabular Data: Deep Learning is Not All You Need,” Nov. 23, 2021, *arXiv*: arXiv:2106.03253. doi: 10.48550/arXiv.2106.03253.

[13] T. Zoppi, S. Gazzini, and A. Ceccarelli, “Anomaly-based error and intrusion detection in tabular data: No DNN outperforms tree-based classifiers,” *Future Gener. Comput. Syst.*, vol. 160, pp. 951–965, 2024.

[14] J. et A. Howard, “Fastai: a layered API for deep learning,” *Information*, vol. 11, no. 2, p. 108, 2020.

[15] “Models and pre-trained weights — Torchvision 0.20 documentation.” Accessed: Oct. 01, 2025. [Online]. Available: https://docs.pytorch.org/vision/0.20/models.html

[16] Y. Zhao, Z. Nasrullah, and Z. Li, “PyOD: A Python Toolbox for Scalable Outlier Detection,” *J. Mach. Learn. Res. JMLR*, vol. 20, no. 96, pp. 1–7, 2019.

[17] T. Zoppi, A. Ceccarelli, and A. Bondavalli, “Unsupervised algorithms to detect zero-day attacks: Strategy and application,” *Ieee Access*, vol. 9, pp. 90603–90615, 2021.

[18] T. Yin, J.-F. Ton, R. Guo, Y. Yao, M. Liu, and Y. Liu, “Fair Classifiers that Abstain without Harm,” Oct. 09, 2023, *arXiv*: arXiv:2310.06205. doi: 10.48550/arXiv.2310.06205.

[19] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr, “Basic concepts and taxonomy of dependable and secure computing,” *IEEE Trans. Dependable Secure Comput.*, vol. 1, no. 1, pp. 11–33, Jan. 2004, doi: 10.1109/TDSC.2004.2.

[20] “Interpretable and Fair Mechanisms for Abstaining Classifiers | Request PDF.” Accessed: Feb. 19, 2026. [Online]. Available: https://www.researchgate.net/publication/383568707_Interpretable_and_Fair_Mechanisms_for_Abstaining_Classifiers

[21] D. Powell, M. Chérèque, and D. Drackley, “Fault-tolerance in Delta-4,” *ACM SIGOPS Oper. Syst. Rev.*, vol. 25, no. 2, pp. 122–125, 1991.

[22] D. F. McAllister and M. A. Vouk, “Fault-tolerant software reliability engineering,” in *Handbook of Software Reliability Engineering*, 1996, pp. 567–614.

[23] “Reliable Computer Systems | Design and Evaluation, Third Edition | Dan.” Accessed: Nov. 11, 2025. [Online]. Available: https://www.taylorfrancis.com/books/mono/10.1201/9781439863961/reliable-computer-systems-daniel-siewiorek-robert-swarz

[24] V. Bandari, “Proactive Fault Tolerance Through Cloud Failure Prediction Using Machine Learning,” *Res. Rev. Sci. Technol.*, vol. 3, no. 1, pp. 51–65, 2020.

[25] V. Ocheretny, “Self-checking arithmetic logic unit with duplicated outputs,” in *2010 IEEE 16th International On-Line Testing Symposium*, IEEE, Jul. 2010, pp. 202–203.

[26] M. Psarakis, D. Gizopoulos, E. Sanchez, and M. S. Reorda, “Microprocessor software-based self-testing,” *IEEE Des. Test Comput.*, vol. 27, no. 3, pp. 4–19, 2010.

[27] I. David, R. Ginosar, and M. Yoeli, “Self-timed is self-checking,” *J. Electron. Test.*, vol. 6, pp. 219–228, 1995.

[28] “Assuring Design Diversity in N-Version Software: A Design Paradigm for N-Version Programming | SpringerLink.” Accessed: Nov. 12, 2025. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-7091-9198-9_10

[29] F. Di Giandomenico and L. Strigini, “Adjudicators for diverse-redundant components,” in *Proceedings Ninth Symposium on Reliable Distributed Systems*, IEEE, 1990, pp. 114–123.

[30] “Recovery Blocks - Randell - Major Reference Works - Wiley Online Library.” Accessed: Nov. 12, 2025. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/0471028959.sof277

[31] T. Zoppi, M. Gharib, M. Atif, and A. Bondavalli, “MetaLearning to Improve Unsupervised Intrusion Detection in CyberPhysical Systems,” *ACM Trans. Cyber-Phys. Syst. TCPS*, vol. 5, no. 4, pp. 1–27, 2021.

[32] S. Džeroski and B. Ženko, “Is combining classifiers with stacking better than selecting the best one?,” *Mach. Learn.*, vol. 54, pp. 255–273, 2004.

[33] Z. Gong, P. Zhong, and W. Hu, “Diversity in machine learning,” *Ieee Access*, vol. 7, pp. 64323–64350, 2019.

[34] “A. Avizienis and J. P. Kelly, ‘Fault Tolerance by Design Diversity Concepts and Experiments,’ IEEE Computer, Vol. 17, No. 8, August 1984, pp. 67-80. - References - Scientific Research Publishing.” Accessed: Nov. 13, 2025. [Online]. Available: https://www.scirp.org/reference/referencespapers?referenceid=33172

[35] J. C. Knight and N. G. Leveson, “An experimental evaluation of the assumption of independence in multiversion programming,” *IEEE Trans Softw Eng*, vol. 12, no. 1, pp. 96–109, Jan. 1986, doi: 10.1109/TSE.1986.6312924.

[36] B. Littlewood, P. Popov, and L. Strigini, “Modeling software design diversity: a review,” *ACM Comput Surv*, vol. 33, no. 2, pp. 177–208, Jun. 2001, doi: 10.1145/384192.384195.

[37] “Software diversity - ScienceDirect.” Accessed: Nov. 13, 2025. [Online]. Available: https://www.sciencedirect.com/science/article/abs/pii/0951832094900566

[38] I. J. Goodfellow, J. Shlens, and C. Szegedy, “Explaining and Harnessing Adversarial Examples,” Mar. 20, 2015, *arXiv*: arXiv:1412.6572. doi: 10.48550/arXiv.1412.6572.

[39] Dan Hendrycks and Kevin Gimpel. A baseline for detecting misclassified and out-of-distribution examples in neural networks. arXiv preprint arXiv:1610.02136, “Dan Hendrycks and Kevin Gimpel,” *Baseline Detect. Misclassified -- Distrib. Ex. Neural Netw.*.

[40] C. M. Bishop, *Pattern recognition and machine learning*. in Information science and statistics. New York: Springer, 2006.

[41] 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, May 2023, pp. 1–6.

[42] K. Lee, K. Lee, H. Lee, and J. Shin, “A simple unified framework for detecting out-of-distribution samples and adversarial attacks,” vol. 31. in Advances in neural information processing systems, vol. 31. 2018.

[43] Y. Zhou, “Rethinking reconstruction autoencoder-based out-of-distribution detection,” in *Proc. of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 7379–7387.

[44] B. Biggio, B. Nelson, and P. Laskov, “Poisoning attacks against support vector machines.” 2012.

[45] T. B. Brown, D. Mané, A. Roy, M. Abadi, and J. Gilmer, “Adversarial Patch,” May 17, 2018, *arXiv*: arXiv:1712.09665. doi: 10.48550/arXiv.1712.09665.

[46] N. Carlini and D. Wagner, “Towards evaluating the robustness of neural networks,” in *2017 ieee symposium on security and privacy (sp*, Ieee, May 2017, pp. 39–57.

[47] 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, Oct. 2022, pp. 390–401.

[48] 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, May 2010, pp. 305–316.

[49] 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, Oct. 2018, pp. 13–24.

[50] T. Zoppi, A. Ceccarelli, T. Puccetti, and A. Bondavalli, “Which algorithm can detect unknown attacks? Comparison of supervised, unsupervised and meta-learning algorithms for intrusion detection,” *Comput. Secur.*, vol. 127, p. 103107, 2023.

[51] 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 Netw.*, vol. 145, pp. 199–208, 2022.

[52] 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*, 2021, pp. 46–55.

[53] X.-Y. Zhang, G.-S. Xie, X. Li, T. Mei, and C.-L. Liu, “A Survey on Learning to Reject,” *Proc. IEEE*, vol. 111, no. 2, pp. 185-215, Feb. 2023, doi: 10.1109/JPROC.2023.3238024.

[54] 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 Trans. Syst. Man Cybern. Part C Appl. Rev.*, vol. 30, no. 1, pp. 84–94, Feb. 2000, doi: 10.1109/5326.827457.

[55] Y. Bahat and G. Shakhnarovich, “Confidence from Invariance to Image Transformations,” Apr. 01, 2018, *arXiv*. doi: 10.48550/arXiv.1804.00657.

[56] A. Mandelbaum and D. Weinshall, “Distance-based Confidence Score for Neural Network Classifiers,” Sep. 28, 2017, *arXiv*: arXiv:1709.09844. doi: 10.48550/arXiv.1709.09844.

[57] H. Jiang, B. Kim, M. Guan, and M. Gupta, “To trust or not to trust a classifier,” vol. 31. in Advances in neural information processing systems, vol. 31. 2018.

[58] W. Q. Meeker, G. J. Hahn, and L. A. Escobar, *Statistical intervals: a guide for practitioners and re-searchers*, vol. 541. John Wiley & Sons, 2017.

[59] W. J. Krzanowski, T. C. Bailey, D. Partridge, J. E. Fieldsend, R. M. Everson, and V. Schetinin, “Confidence in Classification: A Bayesian Approach,” *J. Classif.*, vol. 23, no. 2, pp. 199–220, 2006.

[60] B. Lakshminarayanan, A. Pritzel, and C. Blundell, “Simple and scalable predictive uncertainty esti-mation using deep ensembles,” vol. 30. in Advances in neural information processing systems, vol. 30. 2017.

[61] Z. Bilgin and M. Gunestas, “Explaining Inaccurate Predictions of Models through k-Nearest Neigh-bors,” in *ICAART (2*, 2021, pp. 228–236.

[62] “SafeML: Safety Monitoring of Machine Learning Classifiers Through Statistical Difference Measures | Model-Based Safety and Assessment.” Accessed: Mar. 27, 2025. [Online]. Available: https://dl.acm.org/doi/10.1007/978-3-030-58920-2_13

[63] G. Rossolini, A. Biondi, and G. Buttazzo, “Increasing the Confidence of Deep Neural Networks by Coverage Analysis,” Jan. 05, 2022, *arXiv*: arXiv:2101.12100. doi: 10.48550/arXiv.2101.12100.

[64] M. Hein, M. Andriushchenko, and J. Bitterwolf, “Why relu networks yield high-confidence predic-tions far away from the training data and how to mitigate the problem,” in *Proc. of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 41–50.

[65] S. Liang, Y. Li, and R. Srikant, “Enhancing the reliability of out-of-distribution image detection in neu-ral networks,” in *International Conference on Learning Representations (ICLR*, 2017. doi: https://arxiv.org/abs/1610.02136.

[66] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, “Towards deep learning models re-sistant to adversarial attacks,” in *International Conference on Learning Representations (ICLR*, 2018. doi: https://arxiv.org/abs/1706.06083.

[67] C. Guo, G. Pleiss, Y. Sun, and K. Q. Weinberger, “On calibration of modern neural networks,” in *Proceed-ings of the 34th International Conference on Machine Learning (ICML*, 2017, pp. 1321–1330. doi: https://arxiv.org/abs/1706.04599.

[68] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, “A detailed analysis of the KDD CUP 99 data set,” in *Computational Intelligence for Security and Defense Applications, 2009. CISDA 2009. IEEE Symposium on. IEEE*, 2009, pp. 1–6.

[69] A. H. Lashkari and Al, “Toward Developing a Systematic Approach to Generate Benchmark Android Malware Datasets and Classification,” in *International Carnahan Conference on Security Technolo-gy (ICCST*, IEEE, Oct. 2018, pp. 1–7.

[70] N. Moustafa and J. Slay, “UNSW-NB15: a comprehensive data set for network intrusion detection systems”,” in *Military Communications and Information Systems Conference (MilCIS), 2015. IEEE*, 2015, pp. 1–6.

[71] G. Maciá-Fernández, J. Camacho, R. Magán-Carrión, P. GarcíaTeodoro, and R. Theron, “UGR ’16: A new dataset for the evaluation of cyclostationarity-based network IDSs,” *Comput. Secur.*, vol. 73, pp. 411–424, 2018.

[72] W. Haider, J. Hu, J. Slay, B. P. Turnbull, and Y. Xie, “Generating realistic intrusion detection system dataset based on fuzzy qualitative modeling,” *J. Netw. Comput. Appl.*, vol. 87, pp. 185–192, Jun. 2017, doi: 10.1016/j.jnca.2017.03.018.

[73] M. Ring and Al, “Flow-based benchmark data sets for intrusion detection,” in *Proceedings of the 16th European Conference on Cyber Warfare and Security. ACPI*, Jun. 2017, pp. 361–369.

[74] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, “Toward Generating a New Intrusion De-tection Dataset and Intrusion Traffic Characterization,” in *ICISSP*, 2018, pp. 108–116.

[75] M. S. Elsayed, N. A. Le-Khac, and A. D. Jurcut, “InSDN: A Novel SDN Intrusion Dataset,” *IEEE Access*, vol. 8, pp. 165263–165284, 2020.

[76] B.I.T., “Biometrics Ideal Test, CASIA-FingerprintV5.” [Online]. Available: http://biometrics.idealtest.org/

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

[78] S. Koldijk, M. Sappelli, S. Verberne, M. A. Neerincx, and W. Kraaij, “The swell knowledge work dataset for stress and user modeling research,” in *Proceedings of the 16th international con-ference on multimodal interaction*, Nov. 2014, pp. 291–298.

[79] P. Schmidt, A. Reiss, R. Duerichen, C. Marberger, and K. Laerhoven, “Introducing WESAD, a multimodal dataset for Wearable Stress and Affect Detection”,” *ICMI*, 2018.

[80] A. Vajdi *et al.*, “Human Gait Database for Normal Walk Collected by Smart Phone Accelerometer.” 2019.

[81] W. R. Adams, “High-accuracy detection of early Parkinson’s Disease using multiple characteristics of finger movement while typing,” *PloS One*, vol. 12, no. 11, p. 0188226, 2017.

[82] “Voice Recognition.” Accessed: Oct. 08, 2025. [Online]. Available: https://www.kaggle.com/datasets/jeganathan/voice-recognition

[83] “Face Images with Marked Landmark Points.” Accessed: Oct. 08, 2025. [Online]. Available: https://www.kaggle.com/datasets/drgilermo/face-images-with-marked-landmark-points

[84] BackBlaze: BackBlaze Hard Drive Data (online) https:// www.backblaze.com/b2/hard-drive-test-data.html, “BackBlaze: BackBlaze Hard Drive Data (online) https:// www.backblaze.com/b2/hard-drive-test-data.html,” *Unkn. J.*.

[85] BAIDU: Baidu Smart HDD Competition (online) https:// www.kaggle.com/drtycoon/baidu-hdds-dataset-2017/version/1, “BAIDU: Baidu Smart HDD Competition (online) https:// www.kaggle.com/drtycoon/baidu-hdds-dataset-2017/version/1,” *Unkn. J.*.

[86] G. C. and al, “Prediction of Failures in the Air Pressure System of Scania Trucks Using a Random Forest and Feature Engineering,” in *Advances in Intelligent Data Analysis XV. IDA 2016. Lecture Notes in Computer Science*, vol. 9897, Cham: Springer, 2016.

[87] MechFailure: Machine Failure Prediction Competition (online), “MechFailure: Machine Failure Prediction Competition (online), https://www.kaggle.com/c/machine-failure-prediction,” *Unkn. J.*.

[88] IoT-IDS: IoT Intrusion (online) https://ieee-dataport.org/openaccess/iot-network-intrusion-dataset#files, "IoT-IDS: IoT Intrusion (online) https://ieee-dataport.org/openaccess/iot-network-intrusion-dataset#files," *Unkn. J.*.

[89] H.-K. S. HAI, W. Lee, J.-H. Yun, and H. Kim, "HAI 1.0: HIL-based Augmented ICS Security Dataset"," in *13th USENIX Workshop on Cyber Security Experimentation and Test (CSET 20*, Santa Clara, CA, 2020.

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

[91] "Challenges in Representation Learning: Facial Expression Recognition Challenge." Accessed: Oct. 07, 2025. [Online]. Available: https://kaggle.com/challenges-in-representation-learning-facial-expression-recognition-challenge

[92] L. Bossard, "Food-101 – Mining Discriminative Components with Random Forests," in *Computer Vi-sion – ECCV*, vol. 8694, D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, Eds., Cham: Springer International Publishing, 2014, pp. 446–461. doi: 10.1007/978-3-319-10599-4_29.

[93] "National Flowers." Accessed: Oct. 07, 2025. [Online]. Available: https://www.kaggle.com/datasets/shahidulugvcse/national-flowers

[94] "(PDF) Traffic-Sign Detection and Classification in the Wild." Accessed: Oct. 08, 2025. [Online]. Available: https://www.researchgate.net/publication/311610541_Traffic-Sign_Detection_and_Classification_in_the_Wild

[95] "(PDF) CCTSDB 2021: A More Comprehensive Traffic Sign Detection Benchmark." Accessed: Oct. 08, 2025. [Online]. Available: https://www.researchgate.net/publication/367334500_CCTSDB_2021_A_More_Comprehensive_Traffic_Sign_Detection_Benchmark

[96] "Predicting survival from colorectal cancer histology slides using deep learning: A retrospective multicenter study - PubMed." Accessed: Oct. 08, 2025. [Online]. Available: https://pubmed.ncbi.nlm.nih.gov/30677016/

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

[98] A. Tato and R. Nkambou, "Improving adam optimizer." 2018.

[99] B. Li *et al.*, "Trustworthy AI: From principles to practices," *ACM Comput. Surv.*, vol. 55, no. 9, pp. 1–46, 2023.

[100] "ISO/IEC TR 24028:2020 - Information technology — Artificial intelligence — Overview of trustworthiness in artificial intelligence." Accessed: Jul. 07, 2025. [Online]. Available: https://www.iso.org/standard/77608.html

[101] A. S. US, "Safety First for Automated Driving." 2019. [Online]. Available: https://group.mercedes-benz.com/documents/innovation/other/safety-first-for-automated-driving.pdf

[102] C. Cappi, S. Picard, B. Beltran, J.-C. Bianic, and M. Damour, "Identifying Challenges to the Certification of Machine Learning for Safety Critical Systems." [Online]. Available: https://www.eurocae.net/news/posts/2019/june/new-working-group-

[103] S. Mohseni, M. Pitale, V. Singh, and Z. Wang, "Practical Solutions for Machine Learn-ing Safety in Autonomous Vehicles," in *SafeAI workshop @ AAAI*, 2019.

[104] "Exploiting redundancy and path diversity for railway signalling resiliency | Request PDF," in *ResearchGate*, doi: 10.1109/ICIRT.2016.7588765.

[105] H. Alawad, S. Kaewunruen, and M. An, “Learning from accidents: Machine learning for safety at railway stations,” *IEEE Access*, vol. 8, pp. 633–648, 2019.

[106] C. Carmichael, “Triple Module Redundancy Design Techniques for Virtex FPGAs,” 2006.

[107] F. Brau, G. Rossolini, A. Biondi, and G. Buttazzo, “Robust-by-design classification via unitary-gradient neural networks,” *Proc. AAAI Conf. Artif. Intell.*, vol. 37, no. 12, pp. 14729–14737, Jun. 2023.

[108] G. Miremadi, J. Harlsson, U. Gunneflo, and J. Torin, “Two software techniques for on-line error detection,” in *[1992] Digest of Papers. FTCS-22: The Twenty-Second International Symposium on Fault-Tolerant Computing*, Jul. 1992, pp. 328–335. doi: 10.1109/FTCS.1992.243568.

[109] “Hardware-Based Built-In Security Module in System on Chip (SoC) without System Slowdowns or Loss of Productivity | Journal of Circuits, Systems and Computers.” Accessed: Feb. 19, 2026. [Online]. Available: https://www.worldscientific.com/doi/abs/10.1142/S0218126622503170

[110] M. Bramer, “Avoiding Overfitting of Decision Trees,” in *Principles of Data Mining*, M. Bramer, Ed., London: Springer, 2020, pp. 121–136. doi: 10.1007/978-1-4471-7493-6_9.

[111] T. Zoppi, A. Ceccarelli, and A. Bondavalli, “Ensembling Uncertainty Measures to Improve Safety of Black-Box Classifiers,” in *In proceedings at the 26th European Conference on Artificial Intelligence*, 2023.

[112] C. Ferri, P. Flach, and J. Hernández-Orallo, “Delegating classifiers,” in *Proceedings of the twenty-first international conference on Machine learning*, in ICML ’04. New York, NY, USA: Association for Computing Machinery, Jul. 2004, p. 37. doi: 10.1145/1015330.1015395.

[113] D. H. Wolpert, “Stacked generalization,” *Neural Netw.*, vol. 5, no. 2, pp. 241–259, Jan. 1992, doi: 10.1016/S0893-6080(05)80023-1.

[114] T. Chen and C. Guestrin, “Xgboost: A scalable tree boosting system,” in *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, Aug. 2016, pp. 785–794.

[115] L. I. Kuncheva and C. J. Whitaker, “Measures of diversity in classifier ensembles and their relationship with the ensemble accuracy,” *Mach. Learn.*, vol. 51, pp. 181–207, 2003.

[116] “1989 | City St George’s, University of London.” Accessed: Feb. 19, 2026. [Online]. Available: https://researchcentres.citystgeorges.ac.uk/software-reliability/publications/1989

[117] “Conceptual Modelling of Diverse Systems: Modelling ‘in particular.’” Accessed: Feb. 19, 2026. [Online]. Available: https://dis.ijs.si/mitjal/genre/online/data/file0529.htm

[118] S. Calzavara, L. Cazzaro, C. Lucchese, F. Marcuzzi, and S. Orlando, “Beyond robustness: Resilience verification of tree-based classifiers,” *Comput. Secur.*, vol. 121, p. 102843, 2022.

[119] F. A. Khokhar, *fahadahmedkhokhar/FCC*. (Jan. 09, 2025). Python. Accessed: Oct. 07, 2025. [Online]. Available: https://github.com/fahadahmedkhokhar/FCC

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

[121] “A survey on Image Data Augmentation for Deep Learning | Journal of Big Data | Full Text.” Accessed: Oct. 07, 2025. [Online]. Available: https://journalofbigdata.springeropen.com/articles/10.1186/s40537-019-0197-0

[122] F. A. Khokhar, T. Zoppi, A. Ceccarelli, L. Montecchi, and A. Bondavalli, “Fail-Controlled Classifiers: a Swiss-Army Knife Towards Trustworthy Systems (Appendices.” Jan. 14, 2025. doi: 10.17605/OSF.IO/YU75K.

[123] “Uncertainty identification by the maximum likelihood method - ScienceDirect.” Accessed: Oct. 16, 2025. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0022460X05004529

[124] L. Breiman, “Bagging predictors,” *Mach. Learn.*, vol. 24, no. 2, pp. 123–140, 1996.

[125] Cheung K. L. and Fu A. W. C., “Enhanced nearest neighbor search on the R-tree,” *AUM SIGMOD Rec.*, vol. 27, no. 3, pp. 16–21, 1998.

[126] Z. Xiao, Q. Yan, and Y. Amit, “Likelihood regret: An out-of-distribution detection score for variational auto-encoder,” vol. 33. in Advances in neural information processing systems, vol. 33. pp. 20685–20696, 2020.

[127] M. A. Kramer, “Nonlinear principal component analysis using autoassociative neural networks,” *AIChE J.*, vol. 37, no. 2, pp. 233–243, 1991.

[128] F. Koutenský, P. Šimánek, and M. Čepek, “Overcoming Long Inference Time of Nearest Neighbors Analy-sis in Regression and Uncertainty Prediction. SN COMPUT,” *SCI*, vol. 5, p. 486, 2024, doi: 10.1007/s42979-024-02670-2.

[129] SPROUT Repository on GitHub (online), “SPROUT Repository on GitHub (online), https://github.com/tommyippoz/SPROUT,” *Unkn. J.*.

[130] J. Bergstra, B. Komer, C. Eliasmith, D. Yamins, and D. D. Cox, “Hyperopt: a python library for mod-el selection and hyperparameter optimization,” *Comput. Sci. Discov.*, vol. 8, no. 1, p. 014008, 2015.

[131] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, “Communication-Efficient Learning of Deep Networks from Decentralized Data,” in *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, PMLR, Apr. 2017, pp. 1273–1282. Accessed: Oct. 16, 2024. [Online]. Available: https://proceedings.mlr.press/v54/mcmahan17a.html

[132] M. Sheller and others, “Federated learning in medicine,” *Sci. Rep.*, vol. 10, no. 1, p. 12598, 2020.

[133] A. Hard *et al.*, “Federated Learning for Mobile Keyboard Prediction,” *ArXiv181103604 Cs*, Feb. 2019, Accessed: Nov. 03, 2020. [Online]. Available: http://arxiv.org/abs/1811.03604

[134] Q. Yang, Y. Liu, T. Chen, and Y. Tong, “Federated Machine Learning: Concept and Applications,” *ArXiv190204885 Cs*, Feb. 2019, Accessed: Nov. 03, 2020. [Online]. Available: http://arxiv.org/abs/1902.04885

[135] S. Pal, M. Uniyal, and J. Park, “Server-Side Local Gradient Averaging and Learning Rate Acceleration for Scalable Split Learning”.

[136] Z. Li, T. Lin, X. Shang, and C. Wu, “Revisiting Weighted Aggregation in Federated Learning with Neural Networks”.

[137] P. Qi, D. Chiaro, A. Guzzo, M. Ianni, G. Fortino, and F. Piccialli, “Model aggregation techniques in federated learning: A comprehensive survey,” *Future Gener. Comput. Syst.*, vol. 150, pp. 272–293, Jan. 2024, doi: 10.1016/j.future.2023.09.008.

[138] C. Zhang, Y. Xie, H. Bai, B. Yu, W. Li, and Y. Gao, “A survey on federated learning,” *Knowl.-Based Syst.*, vol. 216, p. 106775, Mar. 2021.

[139] P. Liu, X. Xu, and W. Wang, “Threats, attacks and defenses to federated learning: issues, taxonomy and perspectives,” *Cybersecurity*, vol. 5, no. 1, p. 4, Feb. 2022.

[140] M. Quan and others, “Federated learning for cyber physical systems: a comprehensive survey,” *IEEE Commun. Surv. Tutor.*, 2025.

[141] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr, “Basic concepts and taxonomy of dependable and secure computing,” *IEEE Trans. Dependable Secure Comput.*, vol. 1, no. 1, pp. 11–33, Jan. 2004.

[142] J. Guerin, R. S. Ferreira, K. Delmas, and J. Guiochet, “Unifying Evaluation of Machine Learning Safety Monitors,” Aug. 31, 2022, *arXiv*: arXiv:2208.14660. Accessed: Oct. 18, 2024. [Online]. Available: http://arxiv.org/abs/2208.14660

[143] 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,” Jun. 18, 2024, *arXiv*: arXiv:2307.16325. Accessed: Oct. 18, 2024. [Online]. Available: http://arxiv.org/abs/2307.16325

[144] J. von Neumann, “Probabilistic Logics and the Synthesis of Reliable Organisms from Unreliable Components,” in *Automata Studies*, Princeton University Press, 1956, pp. 43–98.

[145] J. Dobson and B. Randell, “Building reliable secure computing systems out of unreliable insecure components,” in *ACSAC*, 2001, pp. 164–173. doi: 10.1109/ACSAC.2001.991534.

[146] B. Randell and J. Xu, “The evolution of the recovery block concept,” *Softw. Fault Toler.*, vol. 3, pp. 1–22, 1995.

[147] C. Ferri, P. Flach, and J. Hernández-Orallo, “Delegating classifiers,” in *Proceedings of the twenty-first international conference on Machine learning*, 2004, p. 37.

[148] J. Bhanbhro and others, “Issues in federated learning: some experiments and preliminary results,” *Sci. Rep.*, vol. 14, no. 1, pp. 1–15, 2024.

[149] A. Roy and others, “BrainTorrent: A Peer-to-Peer Environment for Decentralized Federated Learning,” *ArXiv Prepr. ArXiv190506731*, 2019.

[150] Y. Qu and others, “Decentralized Privacy Using Blockchain-Enabled Federated Learning in Fog Computing,” *IEEE Internet Things J.*, vol. 7, no. 6, pp. 5171–5183, 2020.

[151] T. Zoppi, M. Gharib, M. Atif, and A. Bondavalli, “Meta-learning to improve unsupervised intrusion detection in cyber-physical systems,” *ACM Trans. Cyber-Phys. Syst. TCPS*, vol. 5, no. 4, pp. 1–27, 2021.

[152] L. Lamport, “The weak Byzantine generals problem,” *J. ACM*, vol. 30, no. 3, pp. 668–676, 1983.

[153] E. Bagdasaryan and others, “How to backdoor federated learning,” in *AISTATS*, 2020, pp. 2938–2948.

[154] Y. Zhao, M. Li, L. Lai, N. Suda, D. Civin, and V. Chandra, “Federated Learning with Non-IID Data,” *ArXiv180600582 Cs Stat*, Jun. 2018, Accessed: Nov. 03, 2020. [Online]. Available: http://arxiv.org/abs/1806.00582

[155] T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, and V. Smith, “Federated Optimization in Heterogeneous Networks,” Apr. 21, 2020, *arXiv*: arXiv:1812.06127. Accessed: Oct. 17, 2024. [Online]. Available: http://arxiv.org/abs/1812.06127

[156] S. Karimireddy and others, “SCAFFOLD: Stochastic Controlled Averaging for Federated Learning,” in *ICML*, 2020.

[157] P. Kairouz and others, “Advances and open problems in federated learning,” *ArXiv Prepr. ArXiv191204979*, 2021.

[158] A. Blanco-Justicia, J. Domingo-Ferrer, S. Martínez, D. Sánchez, A. Flanagan, and K. E. Tan, “Achieving security and privacy in federated learning systems: Survey, research challenges and future directions,” *Eng. Appl. Artif. Intell.*, vol. 106, p. 104468, Nov. 2021.

[159] K. N. Kumar, C. K. Mohan, and L. R. Cenkeramaddi, “The Impact of Adversarial Attacks on Federated Learning: A Survey,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 46, no. 5, pp. 2672–2691, May 2024.

[160] S. G. Miremadi, J. Karlsson, U. Gunneflo, and J. Torin, “Two Software Techniques for On-line Error Detection.,” in *FTCS*, 1992, pp. 328–335.

[161] J. Lever, “Classification evaluation: It is important to understand both what a classification metric expresses and what it hides,” *Nat. Methods*, vol. 13, no. 8, pp. 603–605, 2016.

[162] D. Beutel and others, “Flower: A friendly federated learning research framework,” *ArXiv Prepr. ArXiv200714390*, 2020.

[163] GitHub, “TBB-FL: Trustworthy Black Box - Federated Learning.” 2025. [Online]. Available: https://github.com/fahadahmedkhokhar/TBB-FL.git