IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 30 December 2025, accepted 20 February 2026, date of publication 25 February 2026, date of current version 12 March 2026.

Digital Object Identifier 10.1109/ACCESS.2026.3667991

RESEARCH ARTICLE

StabilEdge, Formal Verification of Edge AI Applications

LUCA LAZZARONI[1], (Member, IEEE), RICCARDO BERTA[1], (Senior Member, IEEE), VAFALI SOLTANMURADOV[1,2], DAVID MARTÍN GÓMEZ[2], (Member, IEEE), AND FRANCESCO BELLOTTI[1], (Senior Member, IEEE)

[1]Department of Electrical, Electronic and Telecommunication Engineering (DITEN), University of Genoa, 16145 Genoa, Italy
[2]Department of Systems Engineering and Automation, LeganÃ©s, 28911 Madrid, Spain

Corresponding author: Riccardo Berta (riccardo.berta@unige.it)

ABSTRACT This paper addresses the critical need for AI trustworthiness in smart IoT systems by focusing on the stability of Deep Neural Networks (DNNs) in Edge AI. State-of-the-art model checkers guarantee feasibility of a single stability analysis. However, a comprehensive assessment requires multiple runs, considering different noise levels and output targets. This is feasible in Edge AI applications, because of the limited size of the involved DNNs. However, direct user interaction with the model checker via its command-line interface is suboptimal, with a high potential for error and inefficiency. We address this issue by proposing a higher-level tool, namely StabilEdge, that abstracts and automatically manages the technical complexities of the underlying verification engine through a graphical layer. The goal is to help designers to concentrate on application-level logic and constraints, rather than the underlying mechanics of the verification tool. StabilEdge performs parametric stability analysis by modeling realistic input perturbations, such as sensor noise/attacks, to assess if they may cause dangerous mispredictions. StabilEdge operationalizes a novel methodology unifying four complementary Edge-AI stability analyses: local robustness, focused evaluation of specific input features (sensitivity) and output target adherence (targeted robustness and sensitivity). The tool generates three key graphical outputs: stability constraint violations, class transition patterns, and input-output constraint trade-offs. It provides continuous verification feedback and specialized support for time series and vision tasks. Overall, StabilEdge automates the full verification workflow - from perturbation modeling to formal constraint generation and solver-level monitoring -, with no need for expert manual configuration. Evaluations on three edge AI tasks demonstrate that StabilEdge quantifies noise impact on DNN performance at a fine granularity, greatly enhancing the depth of NN performance assessment. This analysis yields also key insights, such as the explicit trade-off between model accuracy and its vulnerability to localized perturbations. We thus argue that the tool fulfills its goal of supporting designers in rigorously specifying DNN stability and making informed decisions on sensor selection and overall system architecture. The current implementation exploits the Marabou model checker and may integrate other engines in the future, especially to address scalability issues due to input/output and model size. To support free research in the field, StabilEdge is released open source at: https://github.com/Elios-Lab/StabilEdge

INDEX TERMS Deep learning, edge AI, formal verification, IoT, robustness, sensitivity, stability, trustworthy AI.

I. INTRODUCTION

Deep Neural Networks (DNNs) are very powerful for implementing perception and decision making functionalities, that are getting over more spread on the edge (e.g., [1], [2], [3]). Edge AI is frequently embedded in safety- or mission-critical applications (e.g., automated vehicles, wearable health monitors, etc.) [4], [5], [6], where a formal guarantee of robustness and sensitivity is not just desirable, but often necessary, also from the legal viewpoint [7]. Before deploying such systems,

The associate editor coordinating the review of this manuscript and approving it for publication was Ali Kashif Bashir.

beside the typical performance evaluation on datasets, it is also necessary to ensure trustworthiness (e.g., [8], [9]), one of which main properties is stability verification to perturbations, such as noise, attacks [10], [11]. A core challenge in design assurance, in fact, is establishing with formal rigor whether a DNN will maintain safety under all admissible perturbations within the operational domain.

Edge AI systems typically operate in proximity to sensors. This makes it important to evaluate the impact of factors such as sensor tolerances, environmental noise, and signal degradation on the DNN's behavior [12]. While the state-of-the-art offers powerful model checking tools to formally verify correctness of DNNs (e.g., Marabou [13], α-CROWN [14], β-CROWN [15]), a clear gap is given by the lack of higher-level tools and methodologies, specifically targeting edge AI, that allow designers to characterize their system's performance not only with respect to one or more datasets, but also considering possible perturbations, at various intensity levels.

Relatively small DNNs, such as those typically deployed in the field, can already be automatically verified by state-of-the-art model checkers in a relatively short time for single queries. However, a thorough stability analysis requires users to manually define low-level input and output constraints, for several queries. Common drawbacks and difficulties for edge system developers, who frequently lack in-depth expertise of formal verification, include: (i) time to configure the verifier, correctly expressing sensor noise in terms of input bounds and targets in terms of output bounds, (ii) lack of guidance on how constraints relate to application-level properties, (iii) interpretation of counterexamples in the application context, (iv) understanding the scope and limitations of reported stability properties.

We thus propose a methodology - namely, StabilEdge - to guide the formulation of verification tasks and interpretation of results, facilitating the edge AI model checking operation. We also present the homonymous tool, built atop of a state-of-the-art model checker such as Marabou, that supports an efficient deployment of the methodology by providing a Graphical User Interface (GUI) featuring intuitive abstractions, automating the verification steps, supplying the intermediate representation (i.e., reporting all the user-set constraints in the Marabou format) and visualizing results in a way that connects directly to application-level requirements.

The StabilEdge analysis involves two main aspects: robustness and sensitivity. Robustness concerns stability of a DNN outcome with respect to small perturbations in all the inputs. Sensitivity focuses the stability analysis on one or more inputs, which is useful to quantitatively assess the criticality of the considered input(s). Moreover, for both robustness and sensitivity, it is important to check (i) the noise-induced output variations in the proximity of the perturbation-less decision, and (ii) whether a user-defined (undesired) output value/range of values (i.e., targeted variations) can be provided by the system. This results in a quadrant, depicted in Fig. 1, hosting four types of analysis: Local Robustness (LR), Local Sensitivity (LS), Targeted Robustness (TR) and Targeted Sensitivity (TS). Assessing these four types of properties is particularly relevant and challenging for DNNs, that are closed boxes and show highly non-linear behavior, making it difficult to predict their reliability.

While the current release wraps Marabou as its verifier back-end, StabilEdge is a solver-agnostic methodology and tool that aims at bridging the gap between bare NN model checking (provided by Marabou and similar verifiers) and an application-level stability analysis, offering four main types of new added value: (i) a unified methodology covering local/targeted robustness/sensitivity (LR/LS/TR/TS); (ii) automated constraint generation from sensor tolerances and output requirements; (iii) a monitoring subsystem that extracts solver-level statistics unavailable in formal verifiers; and (iv) analytical outputs that support model-level insights.

Our work was mainly inspired by Grese et al. [16], that present a formally verified DNN predicting the "Takeover-time" [17] in a shared-control automated driving system. The authors use Marabou to analyze the network's sensitivity and local and contextual robustness, and to find adversarial inputs producing unsafe outputs. We abstracted their use case into a methodology including generalization (generalizing the domain-specific constraints and reframing sensor noise and environmental variability as formal verification parameters), formalization (by modeling safety requirements into targeted output conditions and verifying the DNN's expected behavior under different operating conditions), and extending it to targeted sensitivity and to regression tasks as well. We also developed a general-purpose tool that (i) automates the defined verification procedure, (ii) offers usable commands for the user to specify the formal verification parameters and (iii) provides easily interpretable graphical outputs.

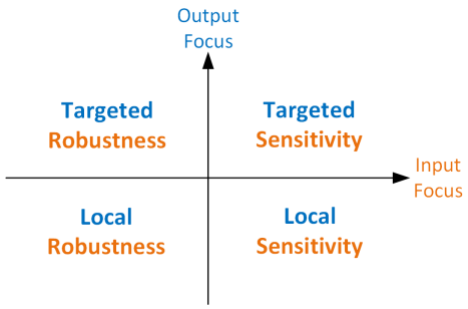


FIGURE 1. StabilEdge model checking modes. Robustness analysis considers all inputs, sensitivity analysis a subset of them. Local focus checks deviations from the perturbation-less outcome, Targeted focus checks whether the output may reach an undesired value or range of values.

Throughout this paper, we use the term "formal verification" as per the established NN verification literature, where formal guarantees are provided for individual verification queries applied to a finite set of representative inputs (e.g., tens or hundreds of samples) [18], [19]. Nevertheless, we acknowledge that evaluating a subset of inputs does not constitute a global guarantee over all the possible inputs

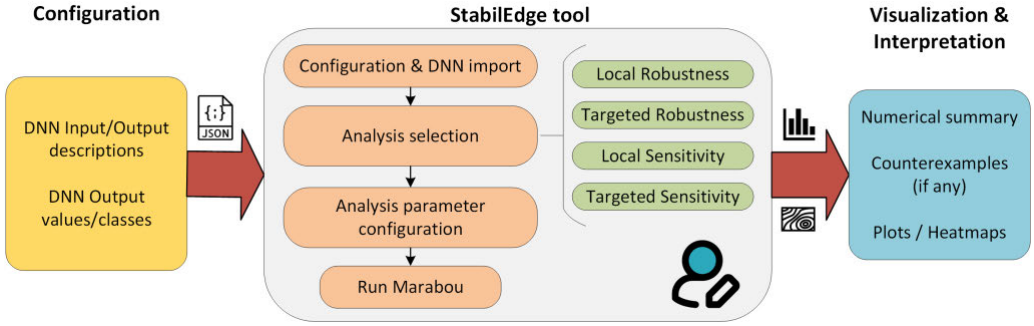


FIGURE 2. StabilEdge method workflow.

(differently from what the "formal verification" term would imply).

An overview of the workflow implementing the proposed methodology is depicted in Fig. 2. Starting from a DNN model, the user defines which features to analyze, the perturbation values (e.g., sensor noise), and the (un)desired output ranges. The StabilEdge tool then automates the creation of all the relevant verification queries and runs them in Marabou. Finally, the numeric results returned by Marabou (e.g., bounds on perturbations that break safety guarantees) are reported along with a graphic representation, allowing developers and analysts to easily verify whether the DNN is aligned with its application requirements.

While StabilEdge is solver-independent, we chose Marabou [13], [20] as the verification backend for our first release because of its expressive constraint language, complete reasoning capabilities, and ease-of-use [21]. Moreover, its Python API, documentation, and example-driven design facilitate learning and application to both classification and regression tasks.

Based on literature and working experience, the methodology targets two main use patterns. First, given real-world sensors, each one with a nominal tolerance, assess how much the DNN under examination deviates from its behavior verified in the test set. Second, given a tolerance on the DNN's outcome, understand (i) what inputs are more critical to keep the DNN's result within the outcome tolerance or (ii) what sensors from a set of candidates guarantee to meet the tolerance limits.

The remainder of the article is structured as follows. Section II gives an overview of the state of the art. Section III presents the StabilEdge methodology, while Section IV introduces the homonymous tool. The experimental set-up is presented in Section V, while results are provided and discussed in Section VI. The last section draws the conclusions on the work done and proposes possible directions for future research.

II. STATE OF THE ART

Verification of NNs is a complex task due to their non-linear and high-dimensional nature. Verification methods include Satisfiability Modulo Theories (SMT) solving [22], Mixed Integer Linear Programming (MILP) [23], and Reachability Analysis [24]. Building on such methods, some open-source tools have been recently developed to support research on automated verification, by providing metrics and interpretability for analyzing NN behaviors. VeriNet employs MILP for verification and a symbolic interval propagation-based approach for formal guarantees of safety, security, correctness, and robustness [25]. VeriNetBF is an extension to verify NN image classifiers against intensity perturbations in computer vision, also addressing scalability [26]. Marabou is another state-of-the-art tool for verifying deep NNs [13], [20]. It encodes the network and property to be verified into constraints, that are solved using a combination of linear programming (LP) and SMT. Marabou supports different NN activation functions, topologies, parallel execution, and multiple input formats. Neurify relies on symbolic interval propagation to create over-approximations, followed by a refinement strategy based on symbolic gradient information [27]. While all the previously cited algorithms are CPU-based, the latest literature has proposed very efficient GPU-based methods, such as β-CROWN [15], and multi-neuron – Branch and Bound (MN-BaB) [28]. Zhang et al. [29] deal with the Swish activation, which is promising for image classification and large language models (LLMs), and, using the linear approximation technique, turn the robustness verification problem into a constraint problem, and tackle it through the Gurobi solver [30].

A critical assessment of the state-of-the-art in NN verification has been recently published by König et al. [31]. The authors stress that no single best algorithm dominates performance across all the proposed verification problem instances and illustrate the potential of leveraging algorithm portfolios for more efficient local robustness verification. A recent benchmark for verifiers, also including hidden counterexamples, has been published by Zhou et al. [32].

Advances in addressing real-world perturbations include the DeepCert [33] framework for contextual image classifier robustness and, in [18], an adaptation of classification-oriented tools to object detection tasks. Kouvaros and Lomuscio [19] enhanced perception system robustness using MILP-based reachability analysis, while Cohen et al. [34] proposed Interval Bound Propagation Intersection over Union (IBP IoU) to effectively manage nonlinearities in object detection model verification. Liu et al.

[35] verify an aircraft collision avoidance NN trained through reinforcement learning (RL). Besides studying local robustness, the authors also analyze six schematic, critical scenarios (implemented in Marabou as sets of input constraints), verifying, with mixed results, whether the NN's single output (angular speed) could violate the expectations in those contexts. In a subsequent study, Bellotti et al. [36] transferred the analysis to the automated driving domain, verifying an agent trained through RL in a more complex simulation environment and sensor and actuator architecture, and also measuring the impact of various types and intensities of noise on the agent's behavior. In the same domain, Grese et al. [16], presented a formally verified neural network predicting the takeover-time in a shared-control automated driving system. The authors used Marabou to analyze the network's sensitivity and local and contextual robustness, and to find adversarial inputs producing unsafe outputs. Horel et al. [37] combined formal conformance testing with model checking for probabilistic collision risk estimation.

While quantization is a key feature of field-deployable DNNs, to our knowledge, none of the state-of-the-art verifiers support the processing of quantized neural networks (QNN) yet. In Marabou, verification of QNNs is possible, but only through manual encoding of the quantization semantics, exploiting the native (but undocumented) constraint-level support for quantization primitives such as *Round()* and *Clip()* [20]. Huang et al. [38] showed how quantized QNNs can be formally verified by explicitly modeling the semantics of quantization operations such as Round and Clip using integer linear programming and abstract-interpretation techniques. However, no public releases provide an operational pipeline for verifying quantized models (e.g., quantized ONNX NNs) directly into Marabou, which is a significant gap for research.

While the mentioned verification tools demonstrate significant capabilities in NN verification tasks, they present considerable barriers to adoption by practitioners. Marabou stands out as offering the most mature complete reasoning for piece-wise linear networks, with comprehensive support for various activation functions, network topologies, and constraint formulations through SMT solving. However, despite its theoretical completeness and robust verification capabilities, it requires substantial domain expertise to effectively configure verification problems. Similarly, competing tools such as VeriNet and Neurify, while offering formal guarantees through MILP and symbolic interval propagation respectively, impose equally steep learning curves and lack ready-made templates for common sensitivity and robustness queries. This complexity demands deep understanding of verification theory and manual specification of constraints, limiting accessibility for non-expert engineers and practitioners who require verification capabilities without extensive theoretical background. The gap between sophisticated verification capabilities and practical usability in engineering workflows necessitates the development of more accessible interfaces, automated constraint generation, and domain-agnostic tools that preserve formal guarantees while enabling verification to become part of everyday engineering practice [39].

TABLE 1. StabilEdge vs Solver-level verification tools comparison.

Feature	Solver-level verification tools	StabilEdge
Formal verification	✓	✓
Standalone	✓	✗
Automated constraint generation	✗	✓
Unified robustness/sensitivity workflow	✗	✓
Sensor-driven perturbation modeling	✗	✓
Real-time analysis feedback	✗	✓
Result interpretation & visualization	✗	✓

The Introduction has already mentioned the outstanding use case for local robustness verification and sensitivity analysis presented by Grese et al. [16], which was a significant source of inspiration for our work. To the best of our knowledge, StabilEdge is the first methodology and operational tool proposed for application-agnostic Edge AI stability analysis, supporting four types of investigations, allowing not only to make a thorough stability check around a datapoint, but also focusing the analysis in terms of perturbed dimensions (sensitivity) and output changes (targetness).

To better clarify the position of StabilEdge with respect to existing approaches, Table 1 provides a qualitative comparison between state-of-the-art solver-level NN verification tools (e.g., Marabou [13], α-CROWN [14], β-CROWN [15]) and StabilEdge. Solver-level tools provide provide the engine for stability query computation but require manual constraint formulation and offer limited support for application-oriented workflows. In contrast, StabilEdge builds on top of such tools to provide automated constraint generation; unified robustness and sensitivity analyses; sensor- and application-driven perturbation modeling; and structured result interpretation. This highlights that StabilEdge complements state-of-the-art verifiers by operating at a higher abstraction level, tailored to Edge-AI application design and analysis.

III. STABILEDGE METHODOLOGY

StabilEdge aims at supporting four types of formal verification analysis (namely: Local Robustness (LR), Targeted Robustness (TR), Local Sensitivity (LS), and Targeted Sensitivity (TS)), that evaluate a DNN behavior under user-defined input and output perturbation and target (respectively) bounds. Such bounds may be explored by the user for instance based on legal requirements or knowing known tolerances from real-world sensor specifications or reachable levels of adversarial attack noise. The tool leverages the Marabou SMT solver for formally verifying a set of user-defined stability properties. For each property, StabilEdge defines the corresponding set of constraints (bounds) on the input and output values. The input bounds represent a region around the datapoint under analysis, which models the maximum considered perturbation intensity. The output bounds represent either the complement of a region close to the noise-less outcome (for

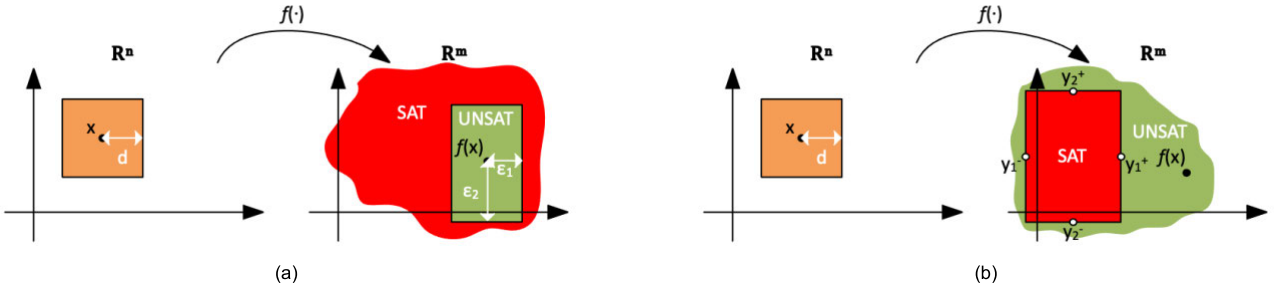


FIGURE 3. Concept of local robustness/sensitivity analysis (a) and of targeted analysis (b).

the local analysis) or an undesirable outcome region (for the targeted analysis) (Fig. 3). The desired stability property is verified if the NN cannot output any value in the region identified by the output bounds when fed with any input within the input bounds. Marabou returns SAT if the analyzed constraint is satisfiable, meaning that there exists at least one input perturbation that leads to the undesired output. In this case, the verification procedure returns also a counterexample (i.e., a datapoint in the considered input region) that violates the desired robustness or sensitivity property. Conversely, UNSAT is returned when the constraints are unsatisfiable, meaning that the NN is unable to map from the given input region to the undesired output region (thus, the investigated stability property is formally verified).

The remainder of this section details the four types of formal verification analysis supported by the proposed methodology.

A. LOCAL ROBUSTNESS

Local Robustness (LR) tests whether a network's output remains stable as all input features are perturbed within the set bounds. Given an input vector $\boldsymbol{x} \in \mathbb{R}^n$ and a network $f : \mathbb{R}^n \rightarrow R^m$, the perturbed input is defined as:

$\boldsymbol{x}' = \boldsymbol{x} + \boldsymbol{\delta}$, where $\delta_i \in [-d_i, d_i]$ for all $i \in 1, \ldots, n$, with d_i perturbation magnitude (e.g., sensor tolerance or adversarial attack bounds).

The system verifies whether $\boldsymbol{x}'$ exists such that the following holds:

- Classification: $\text{argmax}_k\ f_k\left(\boldsymbol{x}'\right) \neq \text{argmax}_k f_k(\boldsymbol{x}), k \in \{1, \ldots, m\}$, k is the index over the output classes
- Regression: $\left\|f_j\left(\boldsymbol{x}'\right) - f_j(\boldsymbol{x})\right\|_\infty > \varepsilon_j, j \in \{1, \ldots, m\}, j$ is the output index

ε_j is a user-defined output deviation threshold, for each output dimension. $\| \cdot \|_\infty$ indicates the L-infinity norm, i.e., the largest magnitude among each element of a vector.

The LR command demands the user to provide an input datapoint, the input bound d_i and the output bounds ε_j for a regression task, while no output bound is to be specified for a classification task, since we consider as "bound" a change in the output class. For multi-label problems the user may focus on a single label (i.e., output dimensions) or all of them. The outcome is SAT (the constraints can be satisfied, i.e., the constrained noise produces an undesirable output) or vice-versa, UNSAT. A typical use case is to iteratively apply the method to a set of relevant datapoints, for which to compute, given a maximum input noise threshold d (e.g., nominal sensor noise tolerance value), what the maximum deviation ε_j is in each output dimension j. In a similar stability analysis, the user may fix the desired output bounds ε_j (e.g., the maximum tolerable error) and iterate the command to find the maximum d that makes the verifier output an UNSAT value (i.e., a maximum input uniform perturbation of d does not make the output diverge more than ε_j). In a classification task the notion of overcoming the ε_j bound is replaced by simply changing the output class. Thus, the former use case becomes: given a maximum input noise threshold d_i, check in each output dimension whether a class change happened. The latter use case stays the same (i.e., find the maximum input uniform perturbation that makes the verifier output an UNSAT value), but meaning the maximum d that does not produce any class change in any output dimension.

B. TARGETED ROBUSTNESS

Targeted Robustness (TR) checks whether a specific undesired output can be produced through a perturbation of all input features. Given the same perturbed input:

$\boldsymbol{x}' = \boldsymbol{x} + \boldsymbol{\delta}$, with $\delta_i \in [-d_i, d_i]$

We evaluate:

- Classification: $\text{argmax}_k\ f_k\left(\boldsymbol{x}'\right) = t$, for a target class $+ t \neq \text{argmax}_k f_k(\boldsymbol{x})^k$
- Regression: $f_j\left(\boldsymbol{x}'\right) \notin \left[y_j^-, y_j^+\right]$, where $\left[y_j^-, y_j^+\right]$is the undesired range

The user is asked to (i) select a target output class (for classification) or value range (for regression) to verify against and (ii) specify the input perturbation bound (d), which is applied to all input features. Upon execution, the tool displays whether a counterexample exists (SAT), indicating that the network can reach the specified undesired output under the given perturbation constraints, or not (UNSAT). For a binary classification task, TR simply degenerates to LR. For multi-class classification, a fully comprehensive analysis can be performed by evaluating all possible (or only relevant) class transitions.

TR aims at assessing whether the NN can be driven toward a specific, critical failure mode or specific undesired

outcome, rather than evaluating general output stability. This is especially relevant in safety-critical systems where certain errors must be strictly avoided (e.g., false negatives in healthcare applications).

C. LOCAL SENSITIVITY

As input signals may be qualitatively different (e.g., different sensors or different ray orientations for the same Lidar sensor), sensitivity analysis perturbs only a subset of input features. Let $S \subseteq 1, \ldots, n$ be the indices of the features to perturb in the analysis:

$x_i' = x_i + \delta_i$ if $i \in S$, otherwise $x_i' = x_i$, with $\delta_i \in [-d_i, d_i]$

For Local Sensitivity (LS), it is verified:

- Classification: $\text{argmax}_k\ f_k\left(\boldsymbol{x}'\right) \neq \text{argmax}_k\ f_k(\boldsymbol{x})$
- Regression: $\left\|f_{\text{j}}\left(\boldsymbol{x}'\right) - f_{\text{j}}(\boldsymbol{x})\right\|_\infty > \varepsilon_j^k$

Similarly to LR, the user specifies the acceptable output deviation threshold ε_j for regression tasks, while for classification tasks the output deviation is simply a class change. Differently than LR, specific input features are selected by the user and perturbed within the given d_i range. The tool assesses the perturbed input subset and quantifies the corresponding change in network output. Again, a SAT outcome means that at least one input point exists such that the corresponding output violates the ε_j threshold (or induces a class change). Vice versa, an UNSAT outcome means that the output remains within the specified bounds (or no class change happens), enabling users to assess the influence and the criticality of each input (or group of them). Given the reduced number of input features considered, the expected execution time is typically lower than for LR. However, the analysis may be repeated for checking different features or groups. This analysis is typically used for assessing how much the overall prediction depends on one or more sensors.

Since sensitivity restricts its analysis to a selected subset of input features, the resulting SAT or UNSAT verdicts reflect only the robustness properties of a subset of the input space. As a consequence, its outcomes are inherently less comprehensive than those obtained via LR, which perturbs the entire input domain and thus explores a broader space of potential violations. This means that a SAT result in LS reveals the existence of a problematic perturbation within the considered subset but does not rule out additional vulnerabilities somewhere else. Conversely, an UNSAT result only guarantees robustness for the chosen features and does not imply global robustness. Therefore, the verdicts of sensitivity can be seen as a focused version of LR, providing useful (and faster) but inherently partial insights into the model's robustness. As per the name, the goal of this analysis is to quantitatively assess the criticality of specific inputs for the model under examination.

D. TARGETED SENSITIVITY

Targeted Sensitivity (TS) further restricts the analysis' focus by combining sensitivity with a targeted output. The goal is to determine whether a limited set of perturbations can cause a targeted output deviation. Let $S \subseteq 1, \ldots, n$ be the indices of the features to perturb in the analysis:

$x_i' = x_i + \delta_i$ if $i \in S$, otherwise $x_i' = x_i$, with $\delta_i \in [-d_i, d_i]$

For TS, it is verified:

- Classification: $\text{argmax}_k\ f_k\left(\boldsymbol{x}'\right) = t$, for a target class $t \neq \text{argmax}_k\ f_k(\boldsymbol{x})^k$
- Regression: $f_j\left(\boldsymbol{x}'\right) \in \left[y_j^-, y_j^+\right]$, where $\left[y_j^-, y_j^+\right]$is the undesired range

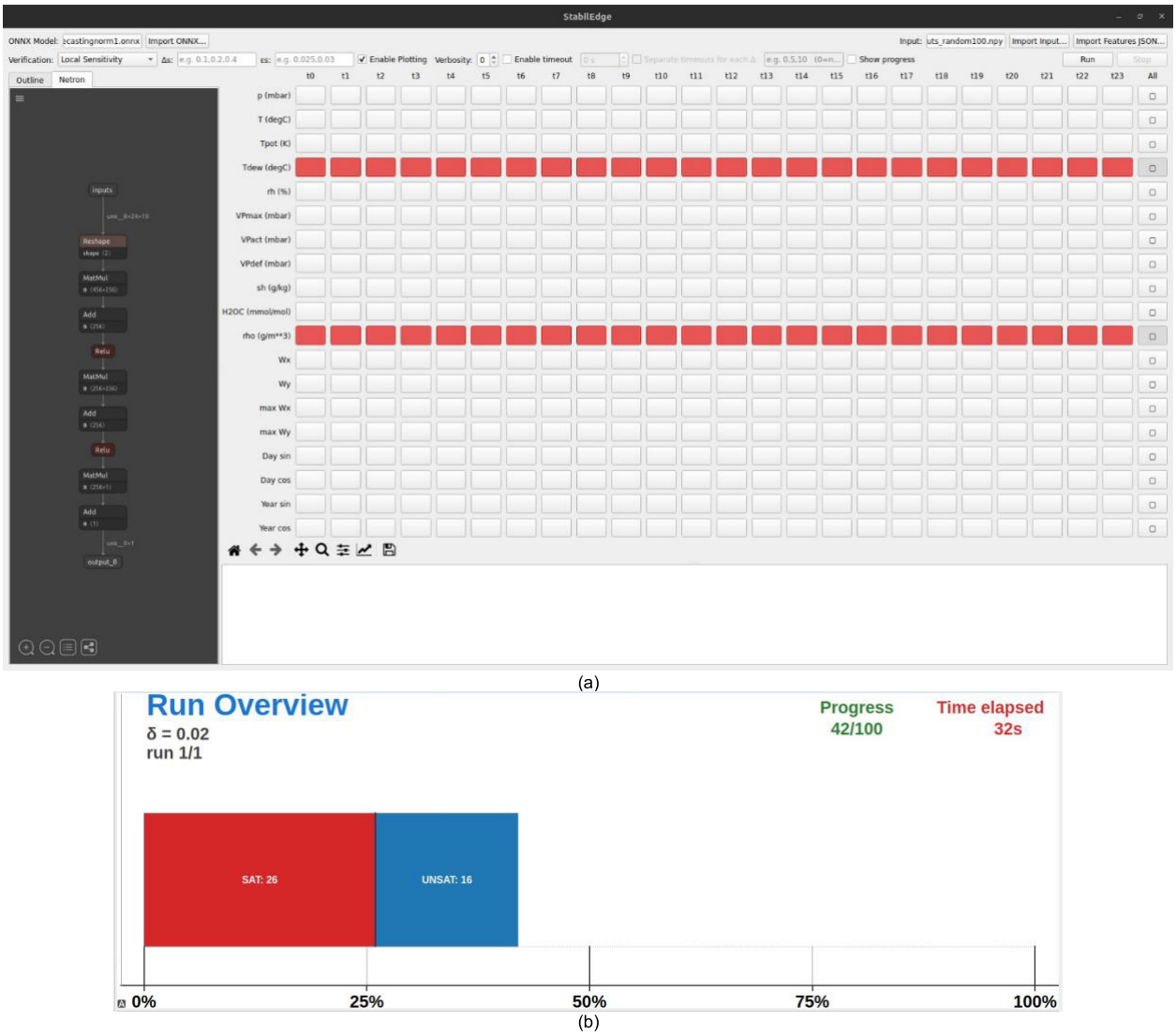
Being a targeted analysis, the tool accepts a user-defined target output (e.g., class label or regression output range). Moreover, the sensitivity analysis requires that specific input features are selected by the user and perturbed within the given d_i range. Upon completion, the tool reports whether the targeted output is reachable (SAT) or not (UNSAT) through such perturbations. Given the reduced number of considered input features and output ranges, the expected execution time is lower than for all the other modes. TS is a very specific analysis and may be repeated for different features (or group of features) and output ranges (or target class labels). The main goal is to identify the most critical features. A typical use case concerns assessing what perturbation intensity is needed in some sensor signals to trigger a dangerous behavior in the system. Like TR, in a binary classification task, TS simply degenerates to LS.

Overall, the proposed StabilEdge methodology supports both a bottom-up workflow, starting from specific, highly relevant (and relatively quick to run) TS analyses and progressing up to the most general LR analysis, or, vice versa, a top-down approach, starting from a general LR and then going into more focused analyses, if needed.

E. STABILEDGE TOOL

To support the deployment of the proposed methodology, we developed the StabilEdge graphical user interface (GUI). The interface allows users to import a NN model in the Open Neural Network Exchange (ONNX) format, and the corresponding dataset. The left part of Fig. 4 (a) shows the integrated Netron [40] visualization tool, which enables users to inspect the network architecture. StabilEdge features four operating modes, each one corresponding to one of the four types of formal verification analysis presented in the Methodology section (namely: LR, TR, LS, TS). A stability analysis is set up through a JSON configuration file, where the user can specify the essential metadata about the model, such as whether the task is classification or regression, the number of output classes, the relevant input features (e.g., whether image pixels or time series data are perturbed), the input domain constraints, and more. Parsing this document, StabilEdge automates the creation of the verification queries and manages their execution through Marabou.

For input perturbation, users can define a single d value to be applied to all the features, or multiple, distinct d_i values if preferred. Similarly, for output deviation, they can configure one or more epsilon thresholds. For convenience, the perturbation configuration can be exported as JSON and



(a)

(b)

FIGURE 4. StabilEdge GUI during: (a) visual selection of input constraints, (b) in-progress verification.

re-imported for later usage. The GUI supports both 2D (e.g., images) and 1D (e.g., time series) data. In sensitivity analyses for image models, users can select the pixels to perturb either manually, from an interactive pixel grid, or automatically, through a gradient-based method [41] and Otsu's thresholding [42], that identify meaningful regions within images (i.e., one or more areas where most of the image information is concentrated). For time series models, the interface allows selection of specific features and time steps individually or in batch through a time series grid (Fig. 4 (a)), where time steps are represented on the horizontal axis, features in vertical.

Since the verification of DNNs can be computationally and time intensive, a progress bar is available to provide real-time feedback during verification (Fig. 4 (b)). The bar displays the number of inputs already verified as SAT (in red) and UNSAT (in blue) and the number of samples still to be verified. This real-time feedback allows users to better anticipate completion times and manage computational resources more effectively.

StabilEdge incorporates the Marabou's built-in timeout mechanism, which counts wall-clock time from the start of the solving process; once the limit is reached, it aborts the search, terminates worker threads, and returns the special *TIMEOUT* result instead of SAT or UNSAT. With the default value of 0, the timeout is disabled, allowing the solver to run until it finds a conclusive answer or is stopped externally.

Since the analysis sessions may take long times, StabilEdge includes a monitoring subsystem that operates as a non-intrusive instrumentation layer, flexibly supporting the user during the solver execution. The subsystem systematically parses Marabou's unstructured textual output and translates it into structured, time-aligned statistics suitable for both adaptive timeout management and GUI visualization. This monitoring layer is necessary because Marabou's Python API does not currently expose internal solver metrics nor callback hooks, making textual-log parsing the only mechanism possible for obtaining fine-grained solver-level diagnostics.

The solver reports a set of internal metrics, including visited states, search depth, branch splits, context pops (backtracking), and stack depth. Such metrics capture the solver's traversal through the case-splits generated from the piecewise-linear constraints (e.g., ReLU activation functions) [13]. Splits quantify branching complexity, while pops reflect the extent of backtracking. To process this information, the proposed subsystem implements several data structures. A regular expression map serves as a compiled catalog mapping each metric name to its corresponding regular expression pattern, enabling efficient parsing of Marabou's outputs. Extracted metrics are accumulated in a temporary buffer during active statistical blocks, before being consolidated into a chronologically ordered history. Each snapshot in the history represents a complete view of the solver's state at that moment, with memory usage controlled by retaining only the most recent entries. At implementation level, the monitoring layer receives Marabou's verbosity-controlled log stream directly from the Python API output and processes it line by line, applying the precompiled regex patterns to extract and record the corresponding metrics.

Exploiting a set of Marabou flags (namely, *verbosity* to track the solver's state and *timeoutInSeconds* to check the timing) the subsystem continuously monitors the verification progress. Boundaries are detected by timestamps or status markers, metrics are extracted using the regex patterns, and complete solver snapshots are built upon block termination. This data is collected to support diagnostic analysis. For instance, steady increases in depth and number of visited states indicate active exploration (thus the analysis should not be terminated), whereas a long sequence of similar values is a symptom of a stagnant situation, which could be interrupted to avoid wasting time on a problematic sample, prioritizing efficient use of time. High pop to split ratios highlight frequent dead ends in the search space (typical of UNSAT cases), whereas low ratios suggest efficient solver progress, often associated with SAT cases.

At the end of a verification session, the GUI visualizes the final verification outcomes through plots, which facilitates deeper post-verification, enabling users to identify trends, compare scenarios, and derive meaningful insights from the results. Examples for the different types of AI tasks and stability analyses are provided in Section V.

IV. EXPERIMENTAL SETUP

To prove StabilEdge's ability to handle sensor-induced perturbations across various application scenarios, including perception and prediction tasks, we assessed it on three datasets, encompassing two classification (one binary and one multi-class) and one regression edge AI tasks (Table 2).

The first classification dataset concerns binary classification and consists of Time-of-Flight (TOF) [43] images acquired using the ST VL53L5CX ultralow-resolution sensor (8×8 pixels) [44] for binary classification, distinguishing between the presence or absence of a miniature mobile robot on sand and rock terrains. The corresponding CNN model includes a convolutional layer with 24 3×3 filters, max pooling, dropout regularization (0.2), a fully connected hidden layer with 32 ReLU-activated units, and a single, sigmoid-activated output neuron. The second is a multi-class dataset, namely the Infrared Thermal Dataset (ITD) [45], captured using a low-resolution MLX90640 sensor array (32×24 pixels) [46] at approximately 15°C, containing 1,770 annotated thermal images categorized into four classes: presence of (i) only one person, (ii) two people, (iii) objects, and (iv) people and objects. The classification model for ITD is a CNN consisting of three sequential convolutional blocks with an increasing number of filters (8, 16, and 32 filters respectively), each with kernel size 3×3 and ReLU activation followed by a fully connected hidden layer with 16 ReLU-activation neurons, and a 4-unit, softmax-activated output layer. Finally, for the regression test, we employed the Jena Climate Dataset [47], containing historical weather data for temperature forecasts. The regression model is a multilayer perceptron (MLP) comprising a flattening layer, two fully connected hidden layers with 256 ReLU-activated units each, and a single linear output neuron.

TABLE 2. Experiments' parameters.

Feature	TOF	ITD	Jena
Task	Binary classification	4-class classification	Regression
Input type	Image	RGB image	Time-series
Input size	(8×8)	(32×24×3)	(24×19)
NN type	CNN	CNN	MLP
Epochs	80	100	30
Batch size	16	32	32
Loss	Binary crossentropy	Sparse categorical crossentropy	Mean Squared Error
Normalization	MinMax [0, 1]	MinMax [0, 1]	MinMax [0, 1]
ONNX Opset	13	13	13

The perturbation magnitudes employed in this study were carefully set considering the actual sensor's tolerance specifications and measurement uncertainties. For the TOF dataset, perturbation values were chosen based on the ST VL53L5CX sensor's specified accuracy tolerance of ± 15 mm [44], ensuring that the experimental conditions encompass the sensor's inherent measurement noise, while also extending slightly beyond this range to assess robustness under more adverse perturbations. The ITD dataset perturbations were calibrated according to the MLX90640 thermal sensor's accuracy specification of ± 1 °C [46], with magnitudes designed to capture the sensor's thermal measurement variability across the 32×24 px resolution of the image. For the Jena dataset regression experiments, the temperature feature perturbations were established based on the PT100 temperature sensor's accuracy tolerance of ± 0.165 °C [48], while the remaining 19 meteorological features were perturbed according to their respective sensor specifications and measurement instruments, as detailed in Table 5.

The obtained models were converted to ONNX using the *tf2onnx* library [49], using operation set 13 (Table 2). All

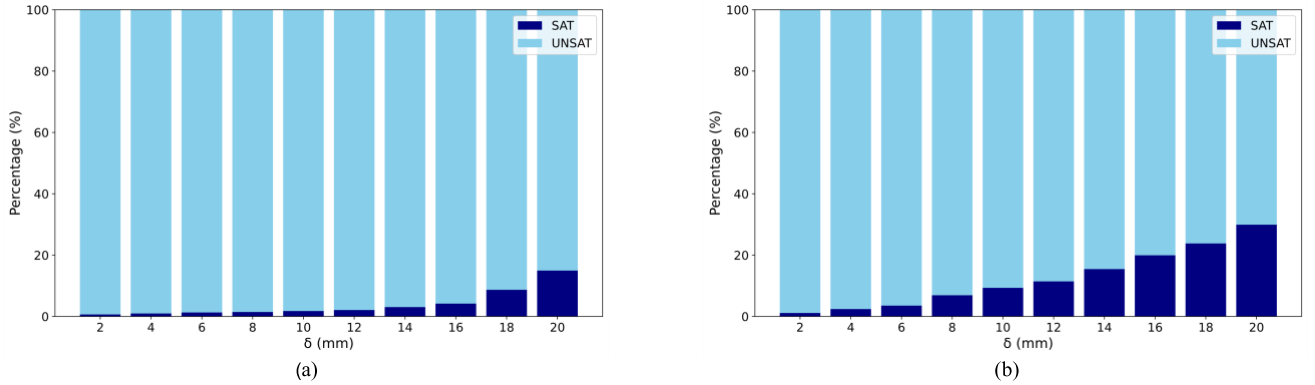


FIGURE 5. TOF – (a) Local Sensitivity (Meaningful pixels, class change) and (b) Local Robustness (all input, class change).

TABLE 3. TOF – LS vs LR comparison.

δ [mm]	LS - SAT [#]	LR - SAT [#]	LS SAT / LR SAT [%]	LS exec time [s]	LR exec time [s]
2	4	7	57.1	1,734	3,382
4	6	15	40.0	2,242	8,004
6	8	22	36.4	2,842	13,614
8	9	43	20.9	4,997	19,723
10	11	58	19.0	4,942	27,736
12	13	71	18.3	6,818	34,737
14	19	96	19.8	64,725	88,947
16	26	124	21.0	79,942	101,791
18	54	148	36.5	163,053	213,015
20	93	186	50.0	215,726	478,215

verification runs were performed with Marabou v2.0.0. For each experiment, the timeout value was set to 500,000 s; no analysis was ended prematurely.

The reported execution times have been measured on a Linux-based workstation featuring an Intel Xeon W7-2495X processor operating at 2.50 GHz with 128 GB of RAM (Marabou does not use GPUs).

V. EXPERIMENTAL RESULTS

A. TOF

Since TOF concerns a binary classification task, TR and TS degenerate to LR and LS, respectively. For the sensitivity analysis, we consider only the meaningful pixels, defined as the most informative regions of the image (computed through Otsu's thresholding [42]), that account for an average proportion of 30.68% of each image's pixels across the entire test set. The LS and LR analyses reveal distinct patterns in both verification outcomes and computational efficiency as perturbation magnitude δ increases. The employed metrics are the SAT and UNSAT percentages. The SAT percentage is the fraction of the whole dataset samples for which the perturbation leads to a violation of the desired stability property (i.e., no class change). The UNSAT percentage is simply the complement of the SAT percentage. Under the LS analysis, the SAT percentage (Fig. 5 (a)) shows, as expected, a gradual increase from 0.64% at $\delta = 2$ mm to 14.95% at $\delta = 20$ mm, representing a relatively controlled degradation in the model's robustness (evaluated on those pixels only). Correspondingly, the UNSAT percentage decreases from 99.36% to 85.05%. These results indicate that the model is robust to perturbations applied to the most semantically relevant regions of an image.

In contrast, the LR approach exhibits a significantly more pronounced vulnerability to increasing perturbation magnitudes. The SAT percentage (Fig. 5 (b)) escalates more dramatically from 1.13% at $\delta = 2$ mm to 29.90% at $\delta = 20$ mm, and the UNSAT correspondingly drop from 98.87% to 70.10%, suggesting that perturbations applied randomly across all input features compromise the model's stability more severely than those restricted to the meaningful pixels. For instance, for $\delta = 2$ mm, LS reveals 4 SAT cases, while LR 7, meaning that 3 SAT cases have been obtained by perturbing non-meaningful pixels. The worst-case scenario is with $\delta = 12$ mm, where 58 SAT cases out of 71 are from perturbations of non-meaningful pixels. Thus, non-meaningful pixels, that represent 50.24% of the inputs, account for up to 81.69% of the stability violations. This finding suggests that while a focus on meaningful areas during training enhances a DNN's accuracy (it is the principle of the attention mechanism [50]), it appears to significantly increase the network's susceptibility to noise in other regions of the image.

By definition, LS produces less SAT outcomes than LR. However, Table 3 shows that the ratio between the two quantities has a non-linear trend. SAT instances initially increase more quickly for LR than for LS (i.e., among meaningful pixels only), until a δ of about 12 mm. Then, the ratio grows, reaching similar values as at low noise levels. This means that the maximum gap in perturbation robustness between meaningful and non-meaningful pixels happens for medium-intensity perturbations.

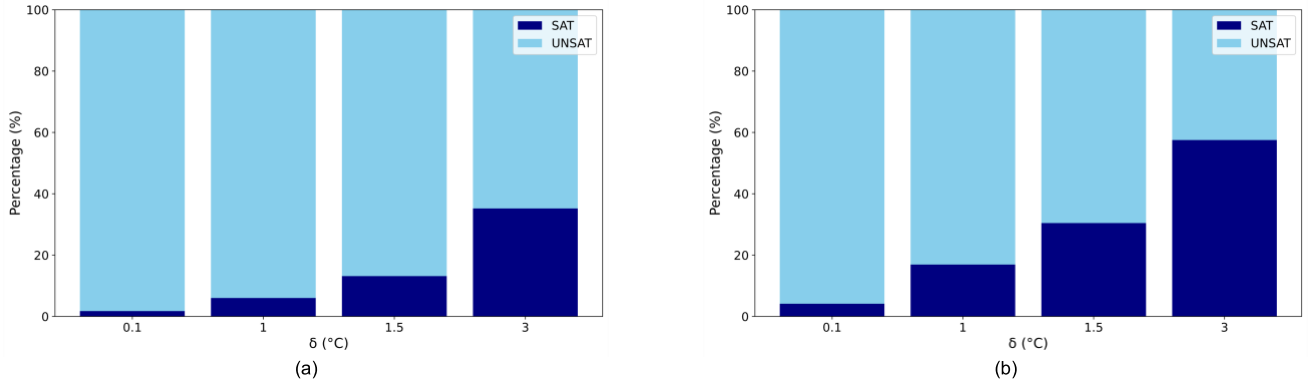


FIGURE 6. ITD – (a) Local Sensitivity (meaningful px, class change) and (b) Local Robustness (all input, class change).

TABLE 4. ITD – LS vs LR comparison.

δ [mm]	LS - SAT [#]	LR - SAT [#]	LS SAT / LR SAT [%]	LS exec time [s]	LR exec time [s]
0.1	5	11	45.5	885	2,664
1.0	16	45	35.6	2,265	6,365
1.5	35	81	43.2	5,178	12,006
3	75	153	49.0	11,185	20,825

Meaningful pixels are identified through a clustering method that separates an image into foreground and background by maximizing the between-class variance. The method searches for the threshold that best separates the two classes in terms of intensity histogram. However, low-intensity pixels may be relatively more heavily affected by physical sensor noise. To experimentally assess this issue, we repeated the LS analysis by perturbing only the low-intensity pixels (i.e., those below the Otsu threshold). For small perturbation magnitudes, the resulting SAT counts were comparable to those obtained when perturbing meaningful pixels (e.g., 1 vs. 4 SAT cases at δ =2 mm, 5 vs. 8 at δ =6 mm, and 7 vs. 9 at δ =8 mm), indicating that background regions can influence the model when noise is mild. However, as δ increases, the meaningful pixels become noticeably more influential (e.g., 15 vs. 19 SAT cases at δ =14 mm, 36 vs. 54 at δ =18 mm, and 64 vs. 93 at δ =20 mm). This suggests that while low-intensity regions contribute to instability at low noise levels, the dominant vulnerabilities of the model emerge in the more informative regions as perturbations grow. Therefore, our experiment confirms that focusing the LS analysis on meaningful pixels remains appropriate for characterizing the most impactful instability sources.

Columns 5 and 6 of Table 3 report the times needed to complete the LR and LS analyses on the whole dataset. It appears that the response time systematically increases with δ. This is not straightforward, since increasing δ increases the number of SATs, and finding a SAT stops the analysis for that sample, while an UNSAT outcome requires a complete space exploration. Since the percentage of SAT is always limited (Fig. 5), response times are dominated by the UNSAT instances, thus by the space exploration, which is larger for larger δ. For the same reason, also LR takes always longer than LS, as it requires exploring more input dimensions.

As a general observation, LS may be considered also as a partial alternative to LR, when LR takes too much to complete (particularly for large values of δ) or the LR's data structure (including the whole input size) exceeds the memory capacity of the host computer. On the other hand, while presented results are obtained through analysis processing the whole dataset, data could also be sampled, to reduce the computational burden.

From a physical point of view, the proposed method shows that the analyzed NN is quite robust to perturbations, especially if they concern the most significant pixels. It must be highlighted that the proposed analysis is pessimistic. In fact, it is very likely that the dataset already includes noisy measurements, and our analysis adds further noise to them. Knowing the distribution of the noise would add more information to improve the analysis and better interpret the results. The same applies also to the other real-world datasets discussed in this paper.

B. ITD

The ITD dataset problem, which addresses a multi-class classification task, allows the evaluation of all four verification methods (LS, LR, TS, and TR) across varying perturbation magnitudes (δ). Like for TOF, the sensitivity analysis is performed on the meaningful pixels of each image, which constitute an average proportion of 4.89% of each image's pixels across the entire test set. At $\delta = 0.1$, LR yields 11 SAT cases (i.e., number of samples for which the applied perturbation leads to a violation of the desired stability property, which is no class change). This represents 4.1% of the dataset population (Fig. 6 (b)). On the other hand, LS has only 5 SAT cases (1.7% in Fig. 6 (a)). At $\delta = 3.0$, LR reaches 153 SAT cases (57.5%), LS 75 (35.2%).

The targeted analysis reveals more nuanced patterns in class-specific misclassification behavior. For the sake of

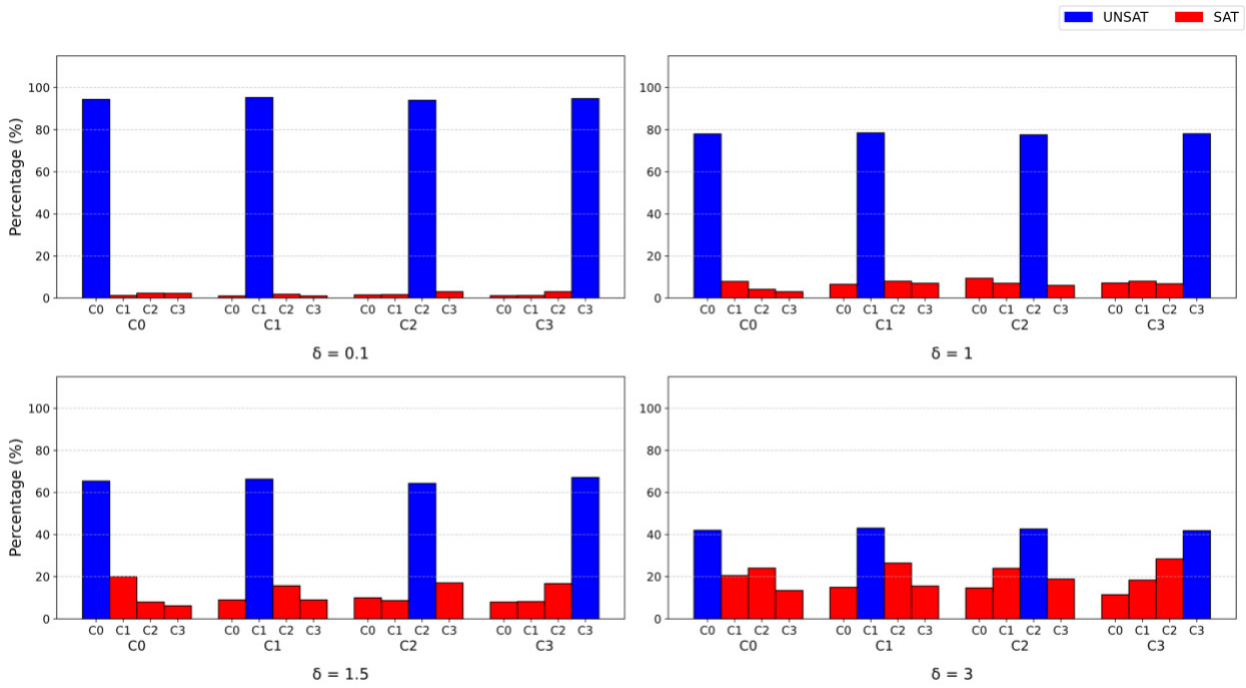


FIGURE 7. ITD – Targeted Robustness (all input, one-vs-all analysis).

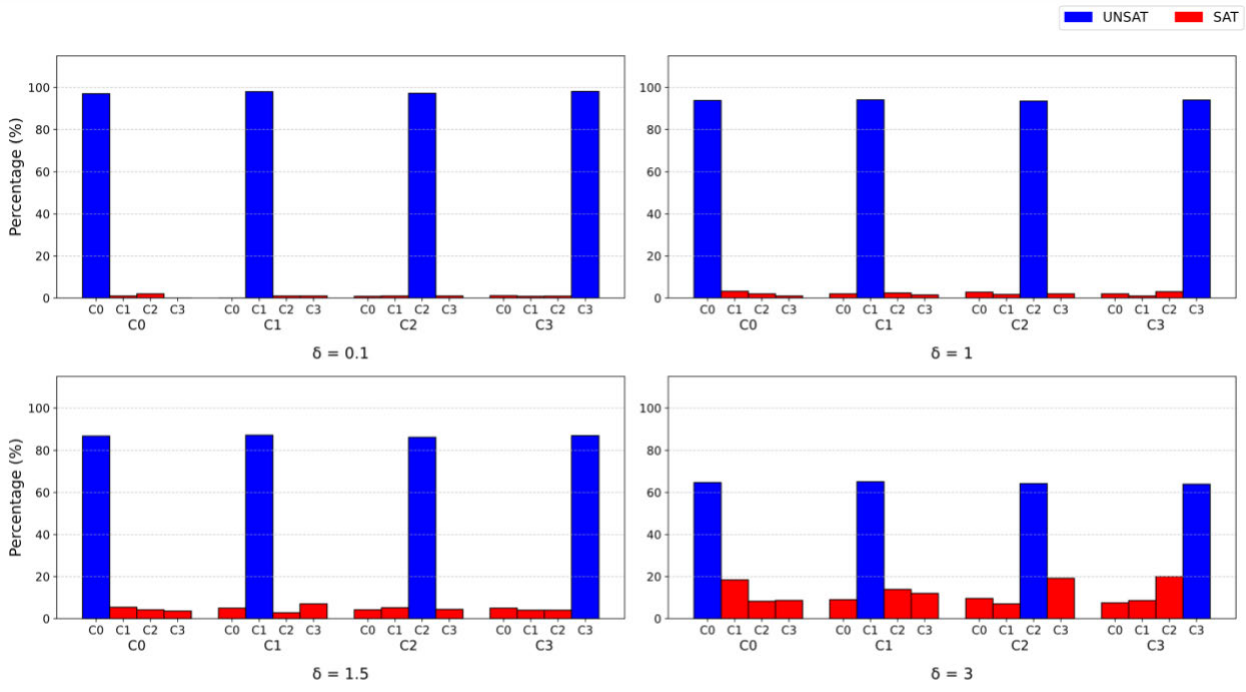


FIGURE 8. ITD – Targeted Sensitivity (meaningful pixels, one-vs-all analysis).

completeness, we performed the targeted analysis for each class, in a one-vs-all manner. The class code is as follows: C1 one person; C2 two people; C3 objects; C4 people and objects. Under TR conditions, classification accuracy degrades significantly as δ increases (Fig. 7). At $\delta = 0.1$, all classes maintain accuracy levels above 93%, with class C1 achieving the highest accuracy at 95.2%. However, at $\delta = 3.0$, accuracy drops dramatically across all classes, with the best-performing class (C0) predicting correctly only in 42% of the cases. The confusion patterns indicate that misclassifications become increasingly distributed among alternative classes, and Fig. 7 gives a quantitative measurement of the

TABLE 5. Specifications and accuracy of the sensors used in the Jena dataset collection.

Feature	Symbol	Instrument	Accuracy	Reference
Air temperature	T (°C)	AKM.00.F148.520.7S8	±0.165 °C	[46]
Relative air humidity	rh (%)	AKM.00.F148.520.7S8	±2%	[46]
Air pressure	P (mbar)	PTB101B	±0.5 mbar	[51]
Wind velocity	wv_x, wv_y (ms^{-1})	Ultrasonic Anemometer 2D	±0.1 m/s	[52]
Wind direction	wd_x, wd_y (ms^{-1})	Ultrasonic Anemometer 2D	±1 °C	[52]
Shortwave incoming radiation	Tpot (K)	CM11	±3 Wm^{-2}	[53]
Dew Point Temperature	Tdew (°C)	DMP8	±0.1 °C	[54]
Mixing Ratio	sh (g/kg)	EE210	±2.5%	[55]
Water Vapor Density	ρ (g/m**3)	EE210	±2.5%	[55]
Molar Mixing Ratio	H2OC (mmol/mol)	5100P	±2%	[56]
Saturation Water Vapor Pressure	VPmax (mbar)	AKM.00.F148.520.7S8	±2%	[46]
Actual Water Vapor Pressure	VPact (mbar)	AKM.00.F148.520.7S8	±2%	[46]
Water Vapor Pressure Deficit	VPdef (mbar)	AKM.00.F148.520.7S8	±2%	[46]
Photosynthetically Active Radiation	PAR ($mmolm^{-2}s^{-1}$)	PAR Lite	±10 $mmolm^{-2}s^{-1}$	[57]
Precipitation yes/no	–	Precipitation Transmitter	±3%	[58]
Precipitation spectra	rain (mm)	OTT Parsivel	±1mm	[59]
CO2-concentration	CO2 (Ppm)	LI6262	±1 ppm	[60]
Day sin	Day sin	CR1000X	±3 min/year	[61]
Day cos	Day cos			
Year sin	Year sin			
Year cos	Year cos			

intuition that large perturbations effectively randomize the decision boundaries. Notably, at all perturbation levels the different classes have similar robustness performance (i.e., percentage of UNSAT), while the transitions among classes tend to have different patterns.

For TS, thus limiting the perturbation to the meaningful pixels, Fig. 8 shows similar trends but also a much stronger robustness than when all pixels are perturbed, especially at moderate noise intensities. This performance differential confirms the critical importance of perturbation/attack localization in adversarial robustness evaluation.

The patterns for LR and LS on the ITD dataset, shown in Table 4, are consistent with those described for TOF in Table 3. Although the fraction of meaningful pixels considered in LS differs between the two datasets (∼30% for TOF vs. ∼5% for ITD), this difference does not translate into noticeable changes in the SAT ratios or execution time ratios.

Also in this case, we repeated the LS analysis by perturbing only the low-intensity background pixels. SAT counts were consistently lower than those obtained when perturbing the meaningful thermal regions but followed the same increasing trend as δ grew (e.g., 2 vs. 5 SAT cases at $\delta = 0.1$, 9 vs. 16 at $\delta = 1$, 21 vs. 35 at $\delta = 1.5$, and 57 vs. 75 at $\delta = 3$). This indicates that background regions do contribute to instability, although to a lesser extent. Interestingly, this behavior differs between the TOF and ITD datasets. For TOF, perturbing low-intensity pixels produces SAT counts that are very similar to those obtained on meaningful pixels at small and medium perturbation levels, and only becomes noticeably lower at higher δ values. In contrast, for the thermal dataset the low-intensity perturbations yield consistently fewer SAT cases than the meaningful regions across all δ values. This difference is plausibly explained by the very different distributions of low-intensity pixels that we measured in the two cases: in the TOF images, low-intensity pixels occupy on average 70.32% of the frame, whereas in ITD they cover ∼95.11%. However, while in ITD the perturbations applied to low-intensity pixels affect almost the entire input, this region carries little discriminative structure because it is nearly uniform and exhibits extremely low variance across the dataset images. Perturbing such a large but homogeneous background produces only limited changes, which explains why the number of SAT cases remains consistently below that obtained when perturbing the high-intensity regions. In TOF, instead, low-intensity pixels cover a much smaller and highly variable portion of the frame, with a standard deviation of 18.23% across the dataset images, compared to 2.47% in ITD. This indicates that background pixels in TOF often include edges or depth transitions that interact more strongly with the model's feature extraction. As a result, perturbations to TOF background pixels can influence decision-relevant features and hence yield SAT counts that are closer to those observed when perturbing meaningful regions, at least for small and medium δ. Overall, meaningful regions remain the primary drivers of instability, but the background still exhibits a non-negligible effect.

From a physical point of view, considering the MLX90640 thermal sensor's accuracy specification of ±1°C [46], the proposed method shows that the analyzed NN is quite robust to perturbations, especially if they are limited to the most significant pixels. However, Fig. 6 (b) (LR) shows that actual sensor failures (i.e., errors much greater than the measurement tolerance) are disruptive. The targeted analysis further specifies the performance check. Particularly, the TR (Fig. 7) suggests a kind of transition pattern between C3 (people and objects) and C2 (objects), while confusion between C0 (one person) and C1 (two people) is weaker and depends on the perturbation intensity. As anticipated, high noise values are disruptive, and, for instance, a stronger confusion between C1 (two people) and C2 (objects) emerges. Somehow similar

TABLE 6. δ vectors employed for the LR analysis on the Jena dataset.

Feature symbol	δ_0	δ_1	δ_2	δ_3	δ_4	δ_5	δ_6	δ_7	δ_8
T	0.025	0.050	0.100	0.200	0.400	0.600	0.800	1.000	2.000
rh	0.010	0.050	0.100	0.200	0.400	0.600	0.800	1.000	2.000
p	0.050	0.100	0.200	0.400	0.400	0.600	1.000	2.000	4.000
wv_x, wv_y	0.025	0.050	0.100	0.200	0.400	0.600	0.800	1.000	2.000
wd_x, wd_y	0.050	0.100	0.200	0.400	0.600	0.800	1.000	2.000	4.000
Tpot	0.025	0.050	0.100	0.200	0.400	0.600	0.800	1.000	2.000
Tdew	0.025	0.050	0.100	0.200	0.400	0.600	0.800	1.000	2.000
sh	0.010	0.050	0.100	0.200	0.400	0.600	0.800	1.000	2.000
ρ	0.010	0.050	0.100	0.200	0.400	0.600	0.800	1.000	2.000
H2OC	0.010	0.050	0.100	0.200	0.400	0.600	0.800	1.000	2.000
VPmax	0.050	0.100	0.200	0.400	0.600	0.800	1.000	2.000	4.000
VPact	0.050	0.100	0.200	0.400	0.600	0.800	1.000	2.000	4.000
VPdef	0.050	0.100	0.200	0.400	0.600	0.800	1.000	2.000	4.000
Day sin Day cos Year sin Year cos	0.001	0.005	0.010	0.050	0.100	0.200	0.400	0.800	1.000

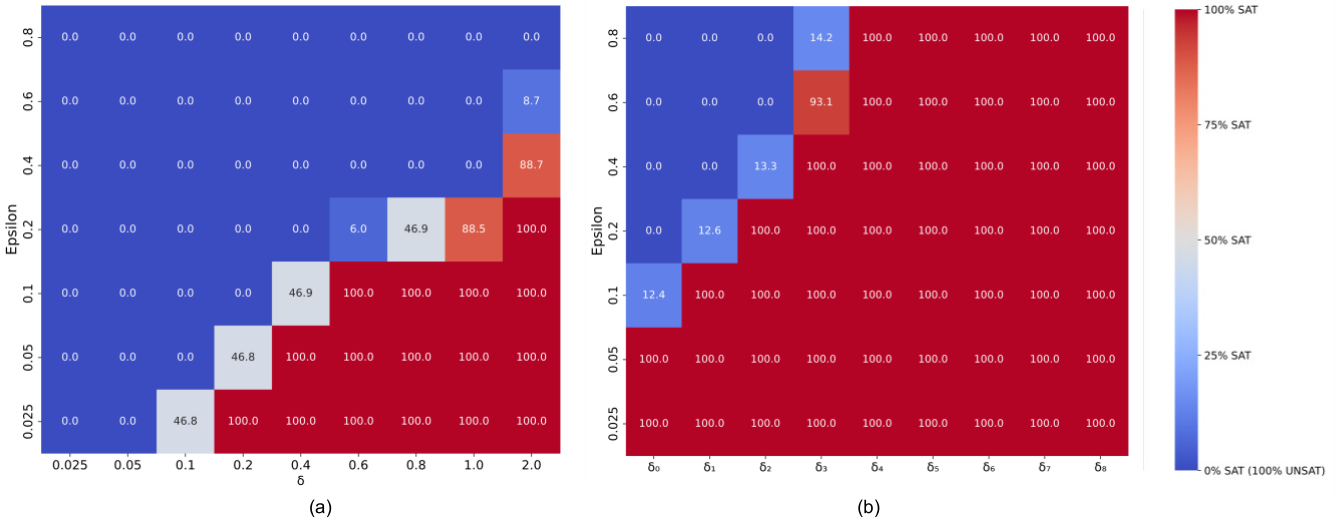


FIGURE 9. Jena – (a) Local Sensitivity (noise on temperature only) and (b) Local robustness.

information could be got by analyzing the confusion matrix, but the proposed method makes the noise dimension explicit.

C. JENA

For the Jena dataset, our evaluation focuses on LS and LR analyses. Targeted analyses were omitted, as in a temperature forecasting task there is no reason for constraining to or excluding specific output ranges, given the continuous nature of the predicted values.

Although the Jena inputs are provided to the MLP as flattened vectors, each vector corresponds to an entire multivariate time-series window, preserving all the temporal dependencies present in the dataset. The perturbation vectors (δ) defined in Table 6 (according to each sensor specification (Table 5)) are applied consistently across all time steps of each feature, not independently per scalar element. As a result, the verification process does not disrupt the intrinsic temporal structure of the signal; rather, it evaluates the effect of sensor-level perturbations that are propagated uniformly through the whole time window. This distinction ensures that the verification respects the real-world temporal behavior of the sensing process.

In the LS experiments, we applied perturbations to the air temperature feature only (Tables 5 and 6). The verification outcomes exhibit a progressive degradation pattern as the noise intensity increases. Fig. 9 (a) presents a comprehensive heatmap visualization of the verification results, displaying the distribution of SAT instances across varying input perturbation magnitudes (δ) and output deviation tolerances (ε), where the color coding indicates the percentage of SAT outcomes out of 1,000 test cases. Differently from TOF and ITD, we did not test the whole Jena test set (7010 samples), to limit the execution time.

For the most restrictive output deviation of $\varepsilon = 0.025$, the system maintains complete robustness (0% SAT, depicted in dark blue regions) for input perturbations up to $\delta = 0.05$. However, a critical transition occurs at $\delta = 0.1$, where 46.8% of instances become satisfiable, indicating that the model's output begins to exceed the specified tolerance threshold. This transition zone represents

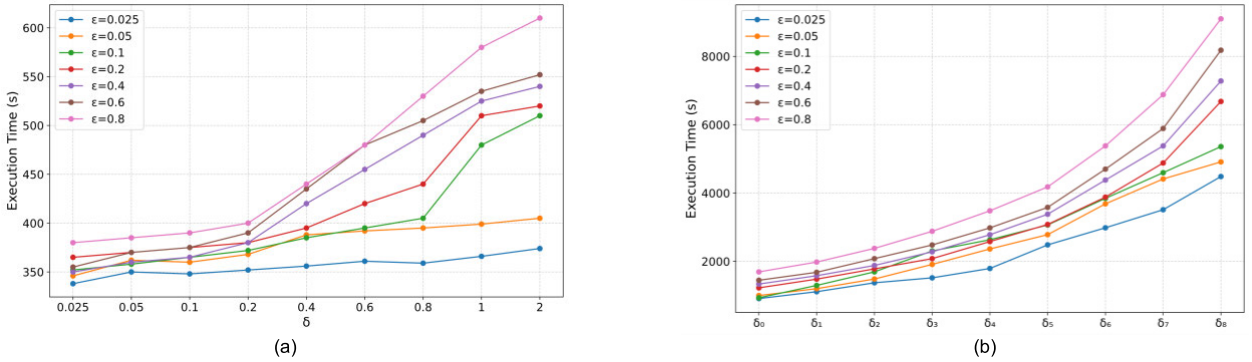


FIGURE 10. Jena – Execution times for (a) Local Sensitivity and (b) Local Robustness.

the robustness boundary where input perturbations begin to induce output deviations beyond acceptable limits. Beyond $\delta = 0.2$, all the 1,000 test instances result in SAT outcomes (100% SAT, shown in dark red), demonstrating complete loss of local robustness under these conditions. The heatmap effectively captures the trade-off relationship between input perturbation bounds and output stability requirements, quantifying how much increasing the output deviation tolerance extends the robustness region toward larger perturbation magnitudes.

As the output deviation tolerance increases, the system exhibits greater resilience to input perturbations. For $\varepsilon = 0.05$, the threshold shifts to $\delta = 0.2$, where mixed SAT/UNSAT results first appear (468 SAT, 532 UNSAT), while complete robustness loss occurs at $\delta = 0.4$. This pattern continues progressively, with $\varepsilon = 0.1$ maintaining robustness until $\delta = 0.4$, $\varepsilon = 0.2$ showing initial vulnerability at $\delta = 0.6$, and higher tolerance values ($\varepsilon = 0.4, 0.6, 0.8$) demonstrating remarkable stability, with robustness breakdown occurring only at the largest perturbation magnitude of $\delta = 2$.

The computational cost in the LS configuration (Fig. 10 (a)) remains relatively modest, with execution times ranging from approximately 300 to 600 s across all experimental conditions. Notably, the execution time exhibits a general upward trend with increasing perturbation/target magnitudes (both δ and ε). This is coherent with the intuition (already confirmed in TOF and ITD) that larger deviations expand the search space and thus slow down the analysis, but the growth is limited, because only one feature is varied.

The LR experiments, involving perturbations across all 19 input features simultaneously, present a quantitatively different landscape. The multi-dimensional perturbation space, characterized by δ vectors with feature-specific values (δ_0 to δ_8 vectors in Table 6, referring to the features introduced in Table 5), introduces substantially greater computational complexity, while revealing different robustness characteristics compared to the single-feature LS approach.

For the smallest output deviation tolerances ($\varepsilon = [0.025, 0.05]$), Fig. 9 (b) shows that all perturbation levels result in complete robustness failure (1,000 SAT outcomes), indicating that even minimal perturbations across the entire input space can cause output deviations exceeding the specified threshold. This behavior contrasts with the LS results, where significant robustness was maintained under comparable conditions. The same results are obtained for $\varepsilon = 0.05$, suggesting that low output deviation tolerances are fundamentally incompatible with full input perturbation in this system.

The robustness characteristics begin to emerge at $\varepsilon = 0.1$, where the smallest perturbation level (δ_0) produces mixed results (124 SAT, 876 UNSAT), while larger perturbations continue to result in complete robustness failure. As the output tolerance increases to $\varepsilon = 0.2$, a clear progression becomes evident: δ_0 achieves complete robustness (1,000 UNSAT), δ_1 produces mixed results (126 SAT, 874 UNSAT), and higher perturbation levels result in total robustness loss. This pattern continues systematically through $\varepsilon = 0.4$, 0.6, and 0.8, with the robustness boundary progressively shifting to accommodate larger perturbation magnitudes.

The computational demands of the LR configuration (Fig. 10 (b)) are substantially higher than those observed in LS (Fig. 10 (a)), with execution times ranging from approximately 1,000 to over 9,000 s. Computational cost increases systematically with both perturbation magnitude (δ) and output deviation tolerance (ε). Similarly to what was observed for the classification datasets, the execution time progression from δ_0 to δ_8 demonstrates a clear scaling relationship. The LR's execution time exceeds that of LS by approximately one order of magnitude, underscoring the considerable computational burden introduced by full-dimensional input perturbation. The dimensionality effect is also qualitatively reflected in the curve shapes: LS grows nearly linearly until $\delta = 1$, where the slope starts to decline, while LR exhibits steady exponential growth. In LR, UNSAT samples - although fewer in number - more heavily dominate the execution time than in LS, because of the much larger space to explore in 19 dimensions instead of 1. The memory footprint of the LR analysis is significantly higher than for LS (in our experiments, on average 5 GB for LS and 12 GB for LR), dramatically slowing down the analysis. This highlights how the dimensionality of the

perturbation space governs the overall scalability of the verification process.

Comparing the LR and LS results, on the other hand, allows observing and quantitatively assessing how much is the NN more fragile when perturbations are applied across multiple input dimensions simultaneously. The consistently higher SAT rates in LR experiments suggest that multi-dimensional perturbations exploit cumulative vulnerabilities across the feature space, leading to more frequent violations of the stability constraints (i.e., evading the ε area) even under smaller individual feature perturbations. Comparing LR and LS highlights the stabilizing role of the unperturbed features: if the model's prediction remains correct despite perturbing a single feature, this indicates that the remaining features collectively exert a strong influence on the network's decision. Thus, these results suggest that the temperature feature itself, while semantically significant (it is the same quantity as the target), does not look to constitute a major vulnerability source.

The physical effect analysis is more complex for the Jena dataset, since it involves several sensors. The sensitivity analysis concerns the PT100 temperature sensor, which has an accuracy tolerance of ± 0.165 °C [48], while the remaining 19 meteorological features were perturbed according to their respective sensor specifications and measurement instruments, as detailed in Table 5 ([48], [51], [52], [53], [54], [55], [56], [57], [58], [59], [60], [61]).

Fig. 9 (a) (LS) highlights the robustness of the system prediction, since a 0.1° perturbation is needed to cause errors greater than 0.025°, while a 0.2° perturbation never creates an error greater than 0.1°.

On the other hand, when all the features are perturbed (LR in Fig. 9 (b)), even the least perturbation of 0.025 ° (in the δ_0 disturbance vector in Table 6) creates issues up to a prediction tolerance of 0.1°, while δ_4 (0.4° for the temperature feature) is sufficient to always exceed a 0.8° output tolerance. On the other hand, an error within 0.8° is guaranteed with the δ_2 perturbations (0.1° for the temperature feature).

D. SENSOR SELECTION

The proposed methodology provides actionable insights for sensor selection in edge AI systems by linking formal robustness results to real sensor accuracy specifications. This sub-section assesses this feature by comparing two representative sensors for each task under consideration. Sensor specifications are provided in Table 7 for TOF and ITD and in Table 8 for Jena.

For TOF considering the ST VL53L5CX sensor's specified accuracy tolerance of ± 15 mm [44], StabilEdge analysis indicates that the evaluated NN is robust to realistic perturbations, particularly when noise is confined to the most significant pixels. In this configuration, Local Sensitivity results show only about 4% SAT cases at $\delta = 16$ mm. In contrast, 8×8 TOF sensors such as the VL53L7CX, which exhibit ranging errors up to 20 mm [62], correspond to larger perturbation magnitudes. Under these conditions, the LS analysis reveals a substantially higher fraction of instability (approximately 15% SAT cases), suggesting that such sensors may be less suitable for this task.

Similarly, for ITD, considering the MLX90640 thermal sensor's accuracy specification of ± 1 °C [46], the proposed analysis shows that the network remains robust to perturbations within the sensor's nominal error, especially when perturbations are limited to the most significant pixels (approximately 6% SAT cases at $\delta = 1$ °C). However, Fig. 6(b) (LR) highlights that perturbations well beyond the nominal accuracy (which are representative of sensor faults or severe environmental effects) are highly disruptive. On the other hand, the operational range of lower-accuracy thermal imaging modules, such as the Omron D6T-32L-01A (± 3 °C to ± 5 °C) [63], corresponds to much larger effective perturbations, yielding to markedly higher instability (about 28% SAT cases at $\delta = 3$ °C). This makes such sensors strongly discouraged for edge AI applications in this range.

A similar conclusion emerges from the Jena regression task (Table 8). Fig. 9(a) (LS) shows that, when employing the PT100 temperature sensor (tolerance of ± 0.165 °C [48]), a perturbation of 0.2 °C never produces errors exceeding 0.1 °C. In contrast, lower-accuracy temperature sensors such as the DHT22 (± 0.5 °C) and DHT11 (± 2 °C) [64] correspond to perturbation levels at which the analysis predicts a rapid degradation of robustness, even for moderate output tolerances (<0.4 °C and <0.8 °C, respectively).

TABLE 7. Sensor selection – classification.

Dataset	Sensor	Tolerance	LS SAT
TOF	VL53L5CX	±15 mm	4%
	VL53L7CX	±20 mm	15%
ITD	MLX90640	±1°C	6%
	D6T-32L-01A	[±3, ±5]°C	28%

TABLE 8. Sensor selection – regression.

Dataset	Sensor	Tolerance	LS – min ε for 0% SAT
Jena	PT100	±0.165°C	0.025°C
	DHT22	±0.5°C	0.4°C
	DHT11	±2°C	0.8°C

E. BACKEND DISCUSSION

For the verification back-end, the current implementation of StabilEdge relies on Marabou [20], [13], which provides sound and complete reasoning for a wide range of activation functions, network topologies, and constraint types, enabling the uniform instantiation of all four analysis modes (LR, LS, TR, and TS) within a single verification engine. Marabou's Python API and easy-to-use tooling [21] support the encoding of expressive input and output constraints.

At the same time, the choice of Marabou entails scalability limitations. SMT-based reasoning with case splitting and backtracking can become computationally expensive as network size and input dimensionality grow (see Section V-F). This trade-off is inherent to complete verification approaches and motivates the consideration of alternative back-end solvers.

Other state-of-the-art verifiers, such as α-CROWN [14] and β-CROWN [15], represent promising complementary back-end candidates. These solvers can leverage GPU acceleration to scale to larger and deeper NNs and thus often provide significantly faster analysis, but at the cost of either incompleteness (α-CROWN) or increased implementation complexity and learning curve (β-CROWN), as shown in an earlier work [21].

The StabilEdge methodology and tool is independent of the wrapped back-end. However, different solvers have different performance characteristics, with an impact on usability. For instance, incomplete back-ends could be used to quickly identify likely instability regions or counterexamples, while complete solvers could be reserved for final certification of critical cases. However, adopting multiple back-ends also raises nontrivial engineering challenges, including the need for mapping StabilEdge's heterogeneous stability queries onto solver-specific specification languages and reconcile potential differences in supported output constraints and verification semantics. Exploring hybrid configurations and alternative verification engines as interchangeable back-ends represents a natural and promising direction for future extensions of the framework.

F. SCALABILITY

To analyze how verification time scales with input resolution, we considered a single CNN architecture operating on grayscale images from the Melanoma Skin Cancer dataset [65] and evaluated three input resolutions: 64×64, 128×128, and 224×224. The architecture consists of two 3×3 convolutional layers with 32 and 64 filters, respectively, each followed by max pooling. A 128-units dense layer follows, then a sigmoid-activated single unit dense layer performs the binary classification. The model was analyzed using LS considering low-intensity pixels (i.e., darker regions of skin more likely affected by melanoma). The tested perturbation magnitudes δ (expressed in pixel intensity units), were 1, 5, and 10. To bound the computational cost, the analysis was limited to 100 test samples, and a timeout of 2 days was enforced.

For an input size of 64×64, the model has 406,337 parameters. Loading the ONNX model into Marabou required approximately 7 s and 8 GB of RAM. After loading, the model could be kept resident in memory, and runtime memory usage during verification stabilized at ~20 GB.

With perturbation level $\delta = 1$, verifying the 100 images required approximately 4 hours, and all cases resulted in UNSAT. At $\delta = 5$ the LS analysis again resulted in all UNSAT cases, requiring ~8 hours. At $\delta = 10$, 3 SAT cases were found, and the total execution time increased to ~11 hours.

For an input size of 128×128, the same architecture expands to 1,848,129 parameters. Loading the ONNX model into Marabou required ~36 s, with a peak memory usage of ~18 GB during parsing. After loading, the model could be kept resident in memory, and runtime memory usage during verification stabilized at ~40 GB.

Even for the smallest perturbation magnitude ($\delta = 1$), the LS analysis reached the timeout threshold. Given the more than $4\times$ increase in parameter count and the significantly larger input domain compared to the 64×64 case, LS verification at 128×128 would require multiple days even at the smallest perturbation level, while also pushing runtime memory consumption close to system limits (i.e., 120 GB of RAM). Executing such analysis is therefore computationally intensive and potentially prohibitive, as it requires very high-end hardware.

For an input size of 224×224, the architecture grows to 5,976,897 parameters. Model loading alone required ~120 s and more than 60 GB of RAM. During verification, memory consumption exceeded the available 120 GB limit, leading to segmentation faults before any LS query could be executed.

Overall, these results highlight a fundamental scalability limitation of complete SMT-based verification when applied to high-resolution vision models. Even under LS, which is the least computationally demanding analysis mode considered in this work, verification time grows rapidly as perturbation magnitudes increase, while memory consumption becomes the dominant bottleneck as input resolution grows. The infeasibility observed at 128×128 and 224×224 resolutions is therefore due to the combined effect of increased input dimensionality and the exhaustive nature of complete verification. While this limits the applicability of Marabou-based verification to low- and medium-resolution vision models, using GPU-based or incomplete verifiers, such as β-CROWN [15] or α-CROWN [14], could significantly extend the StabilEdge's scalability by trading completeness and hardware availability for performance.

VI. CONCLUSION AND FUTURE WORK

AI trustworthiness is a key requirement for smart IoT systems and applications. This paper focuses on stability, which is a trustworthiness' aspect particularly critical in Edge AI, concerning the closed box nature of DNNs. The main original contribution advancing the state-of-the-art is the proposal of a novel methodology and a graphical tool that efficiently exploits a leading-edge DNN model checker (Marabou) to support the edge application designer in two main tasks: a rigorous specification of a DNN's stability, and informed decisions on sensor selection and system architecture design, aligned with the application's requirements. To the best of our knowledge, this is the first release of an application-oriented, domain-independent Edge AI stability verification tool, able to support wide-spectrum, fine-grain analyses.

Carefully scaffolding user interaction, particularly but not exclusively facilitating vision and time series tasks, StabilEdge supports the assessment of whether realistic perturbations, arising from sensor imperfections or operational disturbances, could lead to mispredictions or unsafe decisions. The tool performs parametric stability analysis, enabling focused evaluation of specific input features and

output target adherence. The tool produces three types of graphical charts, that show respectively: (i) the percentage of stability constraint violations (e.g., Fig. 5), (ii) the class transition patterns for classification tasks (e.g., Fig. 7), and (iii) the trade-off between input and output constraints (e.g., Fig. 9). Tested on three real-world classification and regression tasks, the tool allowed to quantitatively measure the effects of noise on a DNN's performance, also providing higher-level insights. For instance, the findings suggested that while a focus on meaningful areas enhances DNN accuracy, it seems to particularly expose the network to vulnerability to noise in other parts of the image, highlighting the critical importance of perturbation/attack localization in vision tasks. A timing analysis on a mid-range workstation highlights a scalability issue due to input/output dimensionality and model size, which can become dramatic also given the associated memory footprint, and may be addressed through focused explorations.

Overall, the StabilEdge tool focuses on increasing the designer work's efficiency. In Edge AI, given the relatively compact size of the DNNs, a single stability analysis is relatively short (5-15 minutes), which makes the problem tractable. However, it is important to combine several analyses (multiple input and output constraints) to understand the impact of different noise intensities and the criticality of the features, especially with respect to dangerous output variations. StabilEdge allows the engineer to efficiently manage the process at application level, abstracting the technical details of the underlying model checker and AI task (e.g., image classification or time-series prediction) and providing graphical interaction.

The current StabilEdge implementation relies on Marabou, as it is a well-established and documented NN formal verifier. However, the proposed tool is general and could interface other engines in future. Criteria for selection could involve speed, completeness of analysis, type of solver, coverage of NN layer types and activation functions. Another important direction for future development is the support for end-to-end verification of quantized NNs, that are very spread in the embedded domain as they enable significant reductions in inference times and power consumption. Integrating quantization semantics directly into the model-loading pipeline would allow StabilEdge to produce safety guarantees also for INT8 edge deployments. Other follow-ups could explore more complex models and tasks (e.g., object detection).

In the proposed tests, either the whole dataset or 1,000 random points were employed. Future research may explore how to better select from a dataset a set of adequately representative datapoints, which would be beneficial to address development time constraints. To improve efficiency, an adaptive algorithm could be implemented to abort the evaluation of samples with dynamically forecasted excessive computational cost, thereby optimizing the trade-off between exploration coverage and runtime.

To support free research in the field, StabilEdge is released open source at: https://github.com/Elios-Lab/StabilEdge.

ACKNOWLEDGMENT

The authors are grateful to the editor and anonymous reviewers for their insightful feedback, which has greatly enhanced the quality of the manuscript.

REFERENCES

[1] B. Akdemir, H. Faheem Shahid, M. A. K. Brix, J. Lääkkölä, J. Islam, T. Kumar, J. Reponen, M. T. Nieminen, and E. Harjula, "From technical prerequisites to improved care: Distributed edge AI for tomographic imaging," *IEEE Access*, vol. 13, pp. 14317–14343, 2025, doi: 10.1109/ACCESS.2025.3530297.

[2] M. Kang and D. Park, "Flexible edge-AI software execution architecture based on cloud-connected incremental learning," *IEEE Access*, vol. 13, pp. 120772–120784, 2025, doi: 10.1109/ACCESS.2025.3586940.

[3] F. Xu, Z. Wu, S. You, Y. Sun, W. Tang, and J. Qi, "Self-supervised denoising with edge perception in OCT images," *Comput. Electr. Eng.*, vol. 124, May 2025, Art. no. 110360, doi: 10.1016/j.compeleceng.2025.110360.

[4] M. Fresta, F. Bellotti, I. Bochenko, L. Lazzaroni, G. Merlhiot, F. Tango, and R. Berta, "Deep learning-based real-time driver cognitive distraction detection," *IEEE Access*, vol. 13, pp. 26589–26607, 2025, doi: 10.1109/ACCESS.2025.3539392.

[5] P. H. Regalado and T. Han, "MediVerse: A secure and scalable IoT-MR framework for real-time health and performance monitoring," *IEEE Access*, vol. 13, pp. 97511–97528, 2025, doi: 10.1109/ACCESS.2025.3570731.

[6] R. Berta, L. Lazzaroni, A. Capello, M. Cossu, L. Forneris, A. Pighetti, and F. Bellotti, "Development of deep-learning-based autonomous agents for low-speed maneuvering in unity," *J. Intell. Connected Vehicles*, vol. 7, no. 3, pp. 229–244, Sep. 2024, doi: 10.26599/jicv.2023.9210039.

[7] Y. Liu, X. Zhou, H. Kou, Y. Zhao, X. Xu, X. Zhang, and L. Qi, "Privacy-preserving point-of-interest recommendation based on simplified graph convolutional network for geological traveling," *ACM Trans. Intell. Syst. Technol.*, vol. 15, no. 4, pp. 1–17, Jul. 2024, doi: 10.1145/3620677.

[8] C. Chakraborty, S. M. Nagarajan, R. Rajesh, U. Kumaran, and G. G. Devarajan, "Trustworthy-based user behavior model: Integrity-based resilient deep learning approach in distributed networks for next-generation applications," *IEEE Trans. Computat. Social Syst.*, vol. 12, no. 6, pp. 5245–5254, Dec. 2025, doi: 10.1109/TCSS.2025.3530772.

[9] B. Chander, C. John, L. Warrier, and K. Gopalakrishnan, "Toward trustworthy artificial intelligence (TAI) in the context of explainability and robustness," *ACM Comput. Surveys*, vol. 57, no. 6, pp. 1–49, Feb. 2025, doi: 10.1145/3675392.

[10] J. Li, Y. Tan, J. Yang, Z. Li, H. Ye, C. Xia, and Y. Li, "Unsupervised adversarial example detection of vision transformers for trustworthy edge computing," *ACM Trans. Multimedia Comput., Commun., Appl.*, vol. 21, no. 8, pp. 1–19, Aug. 2025, doi: 10.1145/3674981.

[11] A. Pighetti, F. Bellotti, R. Berta, A. Cavallaro, L. Lazzaroni, and C. Oh, "RAFT: Regularized adversarial fine-tuning to enhance deep reinforcement learning for self-parking," *IEEE Sensors Lett.*, vol. 9, no. 9, pp. 1–4, Sep. 2025, doi: 10.1109/LSENS.2025.3600982.

[12] Z. Huang, G. Ye, P. Yang, and W. Yu, "Application of multi-sensor fusion localization algorithm based on recurrent neural networks," *Sci. Rep.*, vol. 15, no. 1, p. 8195, Mar. 2025, doi: 10.1038/s41598-025-90492-4.

[13] G. Katz, D. A. Huang, D. Ibeling, K. Julian, C. Lazarus, R. Lim, P. Shah, S. Thakoor, H. Wu, A. Zeljić, D. L. Dill, M. J. Kochenderfer, and C. Barrett, "The marabou framework for verification and analysis of deep neural networks," in *Computer Aided Verification*. Cham, Switzerland: Springer, 2019, pp. 443–452.

[14] K. Xu, H. Zhang, S. Wang, Y. Wang, S. Jana, X. Lin, and C.-J. Hsieh, "Fast and complete: Enabling complete neural network verification with rapid and massively parallel incomplete verifiers," presented at the Int. Conf. Learn. Represent., Oct. 2020. [Online]. Available: https://openreview.net/forum?id=nVZtXBI6LNn

[15] S. Wang, H. Zhang, K. Xu, X. Lin, S. Jana, C.-J. Hsieh, and J. Zico Kolter, "Beta-CROWN: Efficient bound propagation with per-neuron split constraints for complete and incomplete neural network robustness verification," 2021, *arXiv:2103.06624*.

[16] J. M. Grese, C. Pasareanu, and E. Pakdamanian, "Formal analysis of a neural network predictor inShared-control autonomous driving," in *Proc. AIAA Scitech Forum*, Jan. 2021, doi: 10.2514/6.2021-1580.

[17] H. Deng, G. Xiang, J. Pan, X. Wu, C. Fan, K. Wang, and Y. Peng, "How to design driver takeover request in real-world scenarios: A systematic review," *Transp. Res. Part F: Traffic Psychol. Behaviour*, vol. 104, pp. 411–432, Jul. 2024, doi: 10.1016/j.trf.2024.06.012.

[18] Y. Y. Elboher, A. Raviv, Y. L. Weiss, O. Cohen, R. Assa, G. Katz, and H. Kugler, "Formal verification of deep neural networks for object detection," 2024, *arXiv:2407.01295*.

[19] P. Kouvaros and A. Lomuscio, "Formal verification of CNN-based perception systems," 2018, *arXiv:1811.11373*.

[20] H. Wu, O. Isac, A. Zeljić, T. Tagomori, M. Daggitt, W. Kokke, I. Refaeli, G. Amir, K. Julian, S. Bassan, P. Huang, O. Lahav, M. Wu, M. Zhang, E. Komendantskaya, G. Katz, and C. Barrett, "Marabou 2.0: A versatile formal analyzer of neural networks," 2024, *arXiv:2401.14461*.

[21] C. Brix, S. Bak, C. Liu, and T. T. Johnson, "The fourth international verification of neural networks competition (VNN-COMP 2023): Summary and results," 2023, *arXiv:2312.16760*.

[22] H. Song, Y. Lin, Y. Huang, H. Du, Q. Xiang, Y. Chen, L. Kong, Q. Li, F. Le, and J. Shu, "Toward verifying and interpreting learning-based networking systems with SMT," *IEEE Trans. Netw.*, vol. 33, no. 4, pp. 1469–1483, Aug. 2025, doi: 10.1109/TON.2025.3540640.

[23] M. Ataei, E. Hasaj, J. Gipp, and S. Forouzi, "Mathematical programming models for exact and interpretable formulation of neural networks," 2025, *arXiv:2504.14356*.

[24] S. W. Choi, Y. Li, X. Yang, T. Yamaguchi, B. Hoxha, G. Fainekos, D. Prokhorov, and H.-D. Tran, "Reachability analysis of recurrent neural networks," *Nonlinear Analysis: Hybrid Syst.*, vol. 56, May 2025, Art. no. 101581, doi: 10.1016/j.nahs.2025.101581.

[25] P. Henriksen and A. Lomuscio, "Efficient neural network verification viaa adaptive refinement and adversarial search," in *Frontiers in Artificial Intelligence and Applications*. Amsterdam, The Netherlands: IOS Press, 2020, doi: 10.3233/FAIA200385.

[26] P. Henriksen, K. Hammernik, D. Rueckert, and A. Lomuscio, "Bias field robustness verification of large neural image classifiers," presented at the BMVC, Apr. 2021. [Online]. Available: https://openreview.net/forum?id=Hk6KlJsGQC

[27] S. Wang, K. Pei, J. Whitehouse, J. Yang, and S. Jana, "Efficient formal safety analysis of neural networks," 2018, *arXiv:1809.08098*.

[28] C. Ferrari, M. N. Müller, N. Jovanovic, and M. Vechev, "Complete verification via multi-neuron relaxation guided branch-and-bound," 2022, *arXiv:2205.00263*.

[29] Z. Zhang, J. Liu, G. Liu, J. Wang, and J. Zhang, "Robustness verification of swish neural networks embedded in autonomous driving systems," *IEEE Trans. Computat. Social Syst.*, vol. 10, no. 4, pp. 2041–2050, Aug. 2023, doi: 10.1109/TCSS.2022.3179659.

[30] *Gurobi Optimizer*. Accessed: Aug. 1, 2025. [Online]. Available: https://www.gurobi.com/solutions/gurobi-optimizer/

[31] M. König, A. W. Bosman, H. H. Hoos, and J. N. van Rijn, "Critically assessing the state of the art in neural network verification," *J. Mach. Learn. Res.*, vol. 25, no. 12, pp. 1–53, 2024.

[32] X. Zhou, K. Shen, A. Xu, H. Xu, C.-J. Hsieh, H. Zhang, and Z. Shi, "SoundnessBench: A soundness benchmark for neural network verifiers," 2024, *arXiv:2412.03154*.

[33] C. Paterson, H. Wu, J. Grese, R. Calinescu, C. S. Păsăreanu, and C. Barrett, "DeepCert: Verification of contextually relevant robustness for neural network image classifiers," in *Computer Safety, Reliability, and Security*. Cham, Switzerland: Springer, 2021, pp. 3–17.

[34] N. Cohen, M. Ducoffe, R. Boumazouza, C. Gabrea, C. Pagetti, X. Pucel, and A. Galametz, "VerifIoU—Robustness of object detection to perturbations," in *Proc. AIAA DATC/IEEE 44th Digit. Avionics Syst. Conf. (DASC)*, Sep. 2025, pp. 1–10, doi: 10.1109/DASC66011.2025.11257223.

[35] C. Liu, D. Cofer, and D. Osipychev, "Verifying an aircraft collision avoidance neural network with marabou," in *NASA Formal Methods*, vol. 13903. Cham, Switzerland: Springer, 2023, pp. 79–85, doi: 10.1007/978-3-031-33170-1_5.

[36] F. Bellotti, R. Berta, V. Soltanmuradov, D. M. Gómez, A. Dhonthi, V. Hashemi, and L. Lazzaroni, "Robustness verification of a reinforcement learning-based agent for automated car parking," in *Applications in Electronics Pervading Industry, Environment and Society*. Cham, Switzerland: Springer, 2025, pp. 139–147.

[37] J.-B. Horel, P. Ledent, L. Marsso, L. Müller, C. Laugier, R. Mateescu, A. Paigwar, A. Renzaglia, and W. Serwe, "Verifying collision risk estimation using autonomous driving scenarios derived from a formal model," *J. Intell. Robotic Syst.*, vol. 107, no. 4, p. 59, Apr. 2023, doi: 10.1007/s10846-023-01808-3.

[38] H. Pei, H. Wu, Y. Yang, I. Daukantas, M. Wu, Y. Zhang, and C. Barrett, "Towards efficient verification of quantized neural networks," in *Proc. AAAI Conf. Artif. Intell.*, Mar. 2024, vol. 38, no. 19, pp. 21152–21160, doi: 10.1609/aaai.v38i19.30108.

[39] L. C. Cordeiro, M. L. Daggitt, J. Girard-Satabin, O. Isac, T. T. Johnson, G. Katz, E. Komendantskaya, A. Lemesle, E. Manino, A. Šinkarovs, and H. Wu, "Neural network verification is a programming language challenge," in *Programming Languages and Systems*. Cham, Switzerland: Springer, 2025, pp. 206–235.

[40] L. Roeder. (2025). *Lutzroeder/Netron*. Accessed: Jul. 29, 2025. [Online]. Available: https://github.com/lutzroeder/netron

[41] J. Canny, "A computational approach to edge detection," *IEEE Trans. Pattern Anal. Mach. Intell.*, vols. PAMI-8, no. 6, pp. 679–698, Nov. 1986, doi: 10.1109/TPAMI.1986.4767851.

[42] N. Otsu, "A threshold selection method from gray-level histograms," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-9, no. 1, pp. 62–66, Jan. 1979, doi: 10.1109/TSMC.1979.4310076.

[43] J. Pleterski, G. Škulj, C. Esnault, J. Puc, R. Vrabič, and P. Podržaj, "Miniature mobile robot detection using an ultralow-resolution time-of-flight sensor," *IEEE Trans. Instrum. Meas.*, vol. 72, pp. 1–9, 2023, doi: 10.1109/TIM.2023.3318710.

[44] *VL53L5CX | Product—STMicroelectronics*. Accessed: Dec. 12, 2025. [Online]. Available: https://www.st.com/en/imaging-and-photonics-solutions/vl53l5cx.html

[45] S. Zhu, T. Voigt, D. F. Perez-Ramirez, and J. Eriksson, "A low-resolution infrared thermal dataset and potential privacy-preserving applications," in *Proc. 19th ACM Conf. Embedded Networked Sensor Syst.* New York, NY, USA: Association for Computing Machinery, Nov. 2021, pp. 552–555, doi: 10.1145/3485730.3493692.

[46] *Far Infrared Thermal Sensor Array (32×24 RES)*. Accessed: Mar. 31, 2025. [Online]. Available: https://www.melexis.com/en/product/MLX90640/Far-Infrared-Thermal-Sensor-Array

[47] *Max-Planck-Institut Fuer Biogeochemie—Wetterdaten*. Accessed: Mar. 27, 2025. [Online]. Available: https://www.bgc-jena.mpg.de/wetter/

[48] *C48_EN*. Accessed: Jun. 20, 2025. [Online]. Available: https://www.deltastrumenti.it/images/pdf/galltec_mela/c48_en.pdf

[49] (2025). *Onnx/Tensorflow-Onnx*. Accessed: Dec. 17, 2025. [Online]. Available: https://github.com/onnx/tensorflow-onnx

[50] Z. Niu, G. Zhong, and H. Yu, "A review on the attention mechanism of deep learning," *Neurocomputing*, vol. 452, pp. 48–62, Sep. 2021, doi: 10.1016/j.neucom.2021.03.091.

[51] *PTB100*. Accessed: Jun. 20, 2025. [Online]. Available: https://archive.eol.ucar.edu/docs/isf/facilities/isff/sensors/vaisala/ptb100/PTB100.pdf

[52] *One Stop Wind Shop: Accredited Wind Measurement Equipment and Services*. Accessed: Dec. 16, 2025. [Online]. Available: http://shop.profec-ventus.com/index.php

[53] *KIPP & Zonen CM 11 Instruction Manual PDF Download*. Accessed: Jun. 20, 2025. [Online]. Available: https://www.manualslib.com/manual/2425442/Kipp-And-Zonen-Cm-11.html

[54] *Dew Point and Temperature Probe DMP8 | Vaisala*. Accessed: Aug. 29, 2025. [Online]. Available: https://www.vaisala.com/en/products/instruments-sensors-and-other-measurement-devices/instruments-industrial-measurements/dmp8

[55] *Datasheet_EE210*. Accessed: Aug. 29, 2025. [Online]. Available: https://www.epluse.com/fileadmin/data/product/ee210/datasheet_EE210.pdf

[56] K. Bates. *TLDAS Trace Moisture Analyzer, The Barracuda MODEL 4010LX*. Accessed: Aug. 29, 2025. [Online]. Available: https://www.amio2.com/moisture-analyzer/model-4010lx/

[57] *KIPP & Zonen Par Lite Instruction Manual PDF Download*. Accessed: Jun. 20, 2025. [Online]. Available: https://www.manualslib.com/manual/1837981/Kipp-And-Zonen-Par-Lite.html

[58] *5.4032.35.007-011_5.4032.45.008-009_Precipitation_Transmitter_En*. Accessed: Jun. 20, 2025. [Online]. Available: https://www.thiesclima.com/db/dnl/5.4032.35.007-011_5.4032.45.008-009_Precipitation_Transmitter_en.pdf

[59] *OTT Parsivel^2—Laser Weather Sensor*. Accessed: Jun. 20, 2025. [Online]. Available: https://www.ott.com/products/meteorological-sensors-26/ott-parsivel2-laser-weather-sensor-2392/

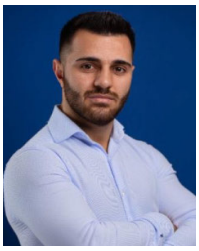
[60] *Licor_6262_Manual*. Accessed: Jun. 20, 2025. [Online]. Available: https://www.aoml.noaa.gov/ocd/ocdweb/brown/Licor_6262_Manual.pdf

[61] *CR1000X—Measurement and Control Datalogger*. Accessed: Jul. 11, 2025. [Online]. Available: https://www.campbellsci.com/cr1000x

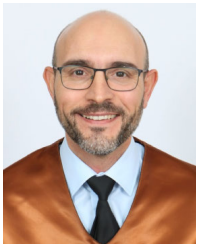
[62] *VL53L7CX | Product—STMicroelectronics*. Accessed: Dec. 12, 2025. [Online]. Available: https://www.st.com/en/imaging-and-photonics-solutions/vl53l7cx.html
[63] *D6T | MEMS Thermal Sensors|OMRON Device & Module Solutions—Europe*. Accessed: Dec. 12, 2025. [Online]. Available: https://components.omron.com/eu-en/products/sensors/D6T
[64] *DHT11, DHT22 and AM2302 Sensors | Adafruit Learning System*. Accessed: Dec. 12, 2025. [Online]. Available: https://learn.adafruit.com/dht/downloads
[65] *Melanoma Skin Cancer Dataset of 10000 Images*. Accessed: Dec. 22, 2025. [Online]. Available: https://www.kaggle.com/datasets/hasnainjaved/melanoma-skin-cancer-dataset-of-10000-images

LUCA LAZZARONI (Member, IEEE) received the B.Sc. degree in electronic engineering and information technologies, the M.Sc. degree in electronic engineering, and the Ph.D. degree in science and technology for electronic and telecommunication engineering (STIET) from the University of Genoa, Genoa, Italy, in 2017, 2020, and 2024, respectively. He is currently an Assistant Professor with the ELIOS Laboratory, Department of Electrical, Electronic, Telecommunication Engineering and Naval Architecture, University of Genoa. He has co-authored about 50 papers in international journals and conferences. His research interests include deep learning, tiny machine learning, and automated driving.

RICCARDO BERTA (Senior Member, IEEE) is currently an Associate Professor with the ELIOS Laboratory, DITEN, University of Genoa. He is also the DITEN Department Delegate for Research. He has been responsible for industrial research projects, particularly in the field of big data and driving automation. His main research interests include the applications of electronic systems, particularly in the fields of tiny machine learning, big data management, automated driving, and the Internet of Things (IoT). He is a member of the Board of Directors of Italian Electronics Society (SIE). He is a Founding Member and the Publications Chair of the Applications in Electronics Pervading Industry, Environment and Society (ApplePies) International Conference.



VAFALI SOLTANMURADOV received the B.Sc. degree in instrumentation engineering and information technologies and the M.Sc. degree in engineering technologies for strategy and security. He is currently pursuing the joint Ph.D. degree with the University of Genoa and Universidad Carlos III de Madrid. His research interests include deep learning, deep reinforcement learning, automated driving, formal verification, and explainable artificial intelligence.



DAVID MARTÍN GÓMEZ (Member, IEEE) received the degree in industrial physics (automation) from the National University of Distance Education (UNED), Spain, in 2002, and the joint Ph.D. degree in computer science from Spanish Council for Scientific Research (CSIC) and UNED, in 2008. He has been a member of the Intelligent Systems Laboratory, since 2011. He is currently a Professor and a Postdoctoral Researcher with the Carlos III University of Madrid. In 2014, he received the VII Barreiros Foundation Award for the Best Research in the automotive field. In 2015, the IEEE Society awarded him as the Best Reviewer of the 18th IEEE International Conference on Intelligent Transportation Systems.

FRANCESCO BELLOTTI (Senior Member, IEEE) received the M.Sc. degree (cum laude) in electronic engineering and the Ph.D. degree in electrical engineering from the University of Genoa, in 1997 and 2001, respectively. He is currently an Associate Professor with DITEN, University of Genoa, and the Coordinator of the Electronic Engineering M.Sc. Program. He also teaches cyber-physical systems, computational intelligence, and machine learning for automated driving. He has co-authored more than 280 scientific publications in journals, international books, and conference proceedings. He has been responsible for the design and implementation WPs of several European and Italian industrial research projects, particularly in the field of big data and driving automation. His main research interests include automated driving, machine learning, cyber-physical systems, and embedded systems. He is a Founding Member of the Applications in Electronics Pervading Industry, Environment and Society (ApplePies) International Conference.