Monitoring Robustness and Individual Fairness

Ashutosh Gupta
IIT Bombay
Mumbai, India
akg@iitb.ac.in

Thomas A. Henzinger
Institute of Science and Technology Austria
Klosterneuburg, Austria
tah@ist.ac.at

Konstantin Kueffner
Institute of Science and Technology Austria
Klosterneuburg, Austria
konstantin.kueffner@ist.ac.at

Kaushik Mallik
IMDEA Software Institute
Madrid, Spain
kaushik.mallik@imdea.org

David Pape
Department of Computer Science,
Paris Lodron University of Salzburg
Salzburg, Austria
david.pape@stud.plus.ac.at

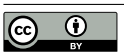
Abstract

In automated decision-making, it is desirable that outputs of decision-makers be robust to slight perturbations in their inputs, a property that may be called *input-output robustness*. Input-output robustness appears in various different forms in the literature, such as robustness of AI models to adversarial or semantic perturbations and individual fairness of AI models that make decisions about humans. We propose runtime monitoring of input-output robustness of deployed, black-box AI models, where the goal is to design monitors that would observe one long execution sequence of the model, and would raise an alarm whenever it is detected that two similar inputs from the past led to dissimilar outputs. This way, monitoring will complement existing offline "robustification" approaches to increase the trustworthiness of AI decision-makers. We show that the monitoring problem can be cast as the fixed-radius nearest neighbor (FRNN) search problem, which, despite being well-studied, lacks suitable online solutions. We present our tool Clemont[1], which offers a number of lightweight monitors, some of which use upgraded online variants of existing FRNN algorithms, and one uses a novel algorithm based on binary decision diagrams—a data-structure commonly used in software and hardware verification. We have also developed an efficient parallelization technique that can substantially cut down the computation time of monitors for which the distance between input-output pairs is measured using the L_∞ norm. Using standard benchmarks from the literature of adversarial and semantic robustness and individual fairness, we perform a comparative study of different monitors in Clemont, and demonstrate their effectiveness in correctly detecting robustness violations at runtime.

CCS Concepts

• **Computing methodologies** → **Artificial intelligence**; • **Applied computing** → Law, social and behavioral sciences; • **Software and its engineering** → **Formal software verification**.

[1]https://github.com/ariez-xyz/clemont (DOI: https://doi.org/10.5281/zenodo.15552183)



KDD '25, Toronto, ON, Canada

ACM ISBN 979-8-4007-1454-2/2025/08
https://doi.org/10.1145/3711896.3737054

Keywords

Monitoring, individual fairness, adversarial robustness, semantic robustness, fixed-radius nearest neighbor search, trustworthy AI

ACM Reference Format:
Ashutosh Gupta, Thomas A. Henzinger, Konstantin Kueffner, Kaushik Mallik, and David Pape. 2025. Monitoring Robustness and Individual Fairness. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.2 (KDD '25), August 3–7, 2025, Toronto, ON, Canada.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3711896.3737054

1 Introduction

AI decision-makers are being increasingly used for making critical decisions in a wide range of areas, including banking [48, 64], hiring [53], object recognition [68], and autonomous driving [19, 80]. It is therefore crucial that they are reliable and trustworthy. One of the general yardsticks of reliability is (global) *input-output robustness*, which stipulates that similar inputs to the given AI model must lead to similar outputs. This subsumes a number of widely used metrics, namely *adversarial robustness* of image classifiers [57], requiring images that are pixel-wise similar be assigned similar labels, *semantic robustness* of image classifiers [26], requiring images that capture similar semantic objects are assigned similar labels, and *individual fairness* of human-centric decision-makers [32, 67], requiring individuals with similar features receive similar treatments.

Currently, input-output robustness of AI models is evaluated *offline*, i.e., before seeing the actual inputs to be encountered during the deployment [32, 52], and it is required that the model be robust either with high probability with respect to a given input data distribution—the *probabilistic* setting, or against all possible inputs—the *worst-case* setting. In practice, these offline robustness requirements of AI models are impossible to achieve due to various reasons. For example, probabilistic robustness is problematic under data distribution shifts [69], and worst-case robustness is tricky in classification tasks due to output transitions near class boundaries [44].

We propose a practical, *runtime* variant of input-output robustness where robustness needs to be achieved on specific (finite) runs of deployed models, and a given *run* violates robustness if two similar inputs from the past produced dissimilar outputs. It is easy to see that (worst-case) offline input-output robustness implies runtime input-output robustness, but not the other way round: an

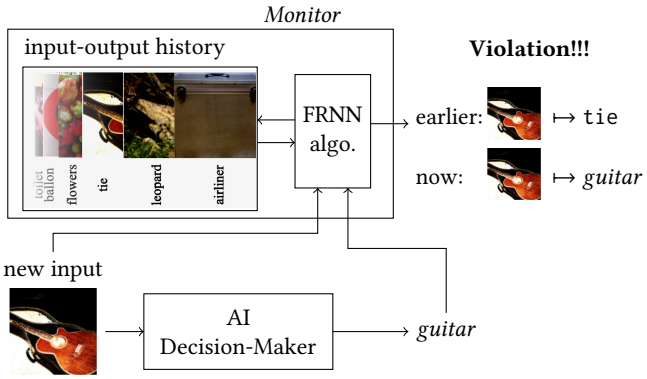


Figure 1: Schematic diagram of input-output robustness monitors. The monitor stores the history of seen input-output pairs, and after arrival of each new input and the respective output of the AI decision-maker, uses a fixed-radius nearest neighbor (FRNN) search algorithm to check if any "close" input from the past gave rise to a "distant" output. The predictions in the figure are from the AlexNet [50] model.

AI model that is unrobust in the offline setting can still produce robust runs if the unrobust input pairs do not appear in practice. Naturally, runtime input-output robustness is immune to data distribution shifts, and remains unaffected by class boundaries if inputs near the boundaries do not appear at runtime. This way, runtime input-output robustness accounts for only those inputs that matter.

We propose *monitoring* of runtime input-output robustness. The objective is to design algorithms—or monitors—which would observe one long input-output sequence of a given black-box decision-maker, and, after each new observation, would raise an alarm if runtime robustness has been violated by the current run. In addition, after detecting a violation, the monitor would present a *witness set*, which is the set of every past input that is similar to the current input but produced a dissimilar output. The witness set can then be scrutinized by human experts, and measures can be taken if needed.

Monitoring has been extensively used to improve trustworthiness of software systems in other areas of computer science, including safety assurance in embedded systems [8] and bias mitigation in human-centric AI [1, 39, 40]. Like in these applications, monitoring is meant to *complement*—and not replace—the existing offline measures of input-output robustness. In fact, our experiments empirically show that offline robustness improves runtime robustness by a significant margin. Yet, AI models that were designed using state-of-the-art offline robust algorithms still showed considerable runtime robustness violations. Without monitoring, these violations would go undetected. Some offline algorithms *verify* the absence of robustness violations of trained models using formal methods inspired approaches [49, 59], which usually do not scale for large and complex models. As monitors treat the monitored systems as black-boxes, their performances remain unaffected by model complexities, making them essential tools when no verification approach would scale. In fact, our monitors are shown to scale for examples up to more than 100,000 feature dimensions and for neural networks with more than 350 million parameter. Such systems are beyond the reach of static verification approaches.

We show that the algorithmic problem of monitoring input-output robustness boils down to solving the well-known fixed-radius nearest neighbor (FRNN) search problem at each step of seeing a new input-output pair, as illustrated in Figure 1. In FRNN, we are given a point p, a set of points S, and a constant $\epsilon > 0$, where p and S belong to the same metric space, and the objective is to compute the set $S' \subseteq S$ which is the set of all points that are at most ϵ-far from p. For our monitoring problem, S and p are respectively the past and current input-output pairs at any given point, and the underlying metric space is designed in a way that two points are close to each other if they correspond to the violation of robustness.

Even though FRNN has been studied extensively, most existing algorithms consider the *static* setting where S remains fixed. Usually, the static FRNN algorithms from the literature are concerned with building fast *indexing schemes* for S, such that the process of computing S' is efficient. For monitoring, we need the *dynamic* variant, where S is growing with incoming input-output pairs, and p is the current input-output pair. The naïve approach to go from the static to the dynamic setting would be to recompute the index at each step after the latest point is added to S, but this will cause a substantial computational overhead in practice.

Our contributions are as follows.

- We present a practical solution for upgrading existing static FRNN algorithms to the dynamic setting through periodic recomputation of indices.
- We present a new dynamic FRNN algorithm based on the symbolic data structure called binary decision diagram (BDD) used in hardware and software verification.
- We present a parallelized FRNN algorithm that substantially boosts the computational performances of our monitors.
- We implemented our monitors in the tool CLEMONT, and show that it can detect violations within fraction of a second to a few seconds (per decision) for real-world models with more than 350M parameter and 150.5k input dimensions.

2 Related Work

The property of robustness has been studied across various domains in computer science, most prominently, automata theory and AI. Robustness in automata theory appears in the study of transducers [42], I/O-systems [42], and sequential circuits [31]. The notion of robustness used are structurally similar to the ones used in AI and include ϵ-robustness, (ϵ, δ)-robustness, or Lipschitz robustness [17]. In AI we are interested in the robustness of a single model. Here we differentiate between local or global robustness [57], which differ by the domain where the robustness requirements must hold. Depending on applications, the existing robustness definitions go by names like semantic robustness [26], adversarial robustness [57], or individual fairness [32, 67].

The two general approaches ensuring the robustness of machine learning models are training [7] and verification [59]. Training robust models is done using techniques such as regularization, curriculum learning, or ensemble learning [7]. Those techniques often fail to provide strong robustness guarantees and those that

do, mostly do so in expectation [11, 51, 67]. Verifying models for robustness is done using tool such as SMT solvers [46, 47], abstract interpretation [36], mixed-integer programming [71], or branch-and-bound [73]. A verified model is guaranteed to satisfy either local [11] and global robustness [13, 43, 72, 76]. Those strong guarantees come at the cost of high computation time. In particular, they do not scale as the complexity of the classifier increases, and usually fail for networks with more than 1000 neurons [11, 13, 34, 59].

Monitoring is a well-established topic in runtime verification of hardware, software, and cyber-physical systems [8]. Recently, monitoring has been extended to verify *group* fairness properties of deployed AI decision-makers [1, 39–41], though monitoring individual fairness (an instance of i.o.r.) properties has appeared rarely [1]. The difference between monitoring group fairness and individual fairness is in the past information that needs to be kept stored: while individual fairness (and i.o.r. by extension) needs all decisions from the past, for group fairness, the explicit past decisions can be discarded and only some small statistics about them needs to be kept [1, 39–41]. The only known work on monitoring individual fairness proposes a simple solution similar to our brute-force monitor [1], which we show to not suffice in many benchmarks.

We will see that some of our monitors build upon existing FRNN search algorithms, a survey of which is deferred to Section 4.

3 The Monitoring Problem

3.1 Input-Output Robustness (I.O.R.)

We consider input-output robustness of AI classifiers, though our formulation can be easily extended for regression models. AI *classifiers* are modeled as functions of the form $D\colon X \to Z$, where X is the *input* space and Z is the *output* space, with the respective distance metrics d_X and d_Z. Each $(x, D(x))$ pair will be referred to as a *decision* of D. *Input-output robustness*, or i.o.r. in short, of D requires that a small difference in inputs must not result in a large difference in outputs.

Definition 3.1. Let D be a classifier. For given constants $\epsilon_X, \delta_Z > 0$ and two inputs $x, x' \in X$, the classifier D is (ϵ_X, δ_Z)-*input-output robust*, or (ϵ_X, δ_Z)-i.o.r.[2] in short, for x and x' if:

$$d_X(x, x') \leq \epsilon_X \implies d_Z(D(x), D(x')) \leq \delta_Z. \tag{1}$$

We drop the constants "ϵ_X" and "δ_Z" if irrelevant or unambiguous. Usually, i.o.r. is not defined with a pair of fixed inputs like in Definition 3.1, but rather as local or global requirements on the system, and these local and global variants can be retrieved from our definition of i.o.r. as follows. The classifier D satisfies *local* i.o.r. with respect to a given input x, if for every $x' \in X$, D is i.o.r. for x and x', and D satisfies *global* i.o.r., if for every pair of inputs $x, x' \in X$, D is i.o.r. for x, x' [17, 43, 52]. I.o.r. appears in different forms in the literature, which are reviewed below.

Adversarial robustness. The definition of adversarial robustness [26] exactly mirrors i.o.r. in (1) with X usually being real-coordinate spaces with either L_2 or L_∞ norm.

Semantic robustness. A decision-maker is *semantically robust* if a small semantic change in its input does not significantly change the output [26], where two input images or input texts are semantically close if their semantic meanings are similar, although the distance between their feature values can be large. For measuring semantic robustness of the classifier $D\colon X \to Z$, we use a separate AI model S that maps every input $x \in X$ to a point y in an intermediate lower-dimensional semantic embedding space Y. Two inputs $x, x' \in X$ are then semantically close if $S(x)$ and $S(x')$ are close to each other according to a given distance metric. Usually, X and Y are real-coordinate spaces with either L_2 or L_∞ norms, and therefore semantic robustness reduces to i.o.r. by defining $d_X(x, x') = \|S(x) - S(x')\|$ with the respective norm $\|\cdot\|$, and ϵ_X is assumed to be specified.

Individual fairness. Individual fairness is a global robustness property defined to assess the fairness of classifiers making decisions about humans. Among many alternate definitions [32, 43, 49, 67], we use the one of Biswas et.al. [13].

3.2 The New *Runtime* Variant

The existing local and global variants of i.o.r. are *offline* properties of classifiers, meaning they are evaluated before observing the actual inputs seen at runtime. In practice, a classifier that is *not* locally or globally i.o.r. may still be acceptable, as long as the pairs of inputs that witness the unrobust behaviors do not appear at runtime. This motivates us to introduce the third, *runtime* variant of i.o.r., which is a property of a given decision sequence, and *not* a property of the underlying classifier. Here, a decision sequence of the classifier $D\colon X \to Z$ is any finite input-output sequence $(x_1, z_1), \ldots, (x_n, z_n) \in (X \times Z)^n$, for any $n > 0$, such that for every $i \in [1; n]$, $D(x_i) = z_i$.

Definition 3.2. Let D be a classifier and let $\epsilon_X, \delta_Z > 0$. A decision sequence $(x_1, z_1) \ldots (x_n, z_n)$ of D is *runtime* (ϵ_X, δ_Z)-*i.o.r.* if

$$\forall i, j \in [1; n] \,.\, d_X(x_i, x_j) \leq \epsilon_X \implies d_Z(D(x_i), D(x_j)) \leq \delta_Z. \tag{2}$$

It is straightforward to show that runtime i.o.r. is *weaker* than global i.o.r.:

Theorem 3.3. *Suppose $\epsilon_X, \delta_Z > 0$ are constants and X is infinite.*

(1) *Every decision-sequence of every globally (ϵ_X, δ_Z)-i.o.r. classifier is runtime (ϵ_X, δ_Z)-i.o.r..*
(2) *If the decision-sequence of a classifier is runtime (ϵ_X, δ_Z)-i.o.r., the classifier is not necessarily globally (ϵ_X, δ_Z)-i.o.r..*

The proof is in Appendix B. Claim (1) of Theorem 3.3 implies that if a given decision sequence of a classifier is *not* runtime i.o.r., then the classifier is surely *not* globally i.o.r. On the other hand, Claim (2) suggests that if the decision sequence *is* runtime i.o.r., we will not be able to conclude whether the classifier is globally i.o.r. or not. Furthermore, from Definition 3.2, as soon as a decision sequence violates runtime i.o.r., so will every future extension of the sequence, regardless of the decisions that will be made in future.

3.3 Monitoring Runtime I.O.R.

We consider the problem of *online* monitoring of runtime i.o.r. of classifiers. The goal is to design a function—the *monitor*—that observes one long decision sequence of a black-box classifier, and after observing each new decision (x, z), outputs the set of every past decision (x', z') such that x and x' are close but z and z' are

[2] The acronym "i.o.r." will represent both the noun "input-output robustness" and the adjective "input-output robust."

not. If the monitor always outputs the empty set while observing a given long decision sequence, then the sequence is runtime i.o.r.

Problem 1 (Monitoring runtime i.o.r.). *Let $D\colon X \to Z$ be an arbitrary (black-box) classifier and let $\epsilon_X, \delta_Z > 0$ be constants. Compute the function $M\colon (X \times Z)^+ \times (X \times Z) \to 2^{(X\times Z)}$ such that for every finite sequence of past decisions $\rho = (x_1, z_1)\dots(x_n, z_n)$ of D, and for every new decision (x_{n+1}, z_{n+1}),*

$$M(\rho, (x_{n+1}, z_{n+1})) = \big\{(x_i, z_i), i \in [1; n] \mid d_X(x_{n+1}, x_i) \le \epsilon_X \wedge d_Z(z_{n+1}, z_i) > \delta_Z\big\}.$$

The function M will be called the i.o.r. monitor.

Monitoring runtime i.o.r. offers an added level of trustworthiness in AI decision making, especially when the underlying decision maker is not known to be globally i.o.r. One possibility is that the outputs of the monitor can be sent for scrutiny by human experts, so that necessary steps can be taken. Without monitoring, robustness violations would go undetected, and could manifest in greater risks and loss of trustworthiness.

3.4 Reduction to Fixed-Radius Nearest Neighbor

Problem 1 reduces to the online *fixed-radius nearest neighbor* (FRNN) problem stated below:

Problem 2 (FRNN monitoring). *Let Q be a set equipped with the distance metric d_Q and $\epsilon_Q > 0$ be a given constant. Compute the function $M\colon Q^+ \times Q \to 2^Q$ such that for every sequence of past points $\rho = q_1 \dots q_n \in Q^+$, and for every new point $q_{n+1} \in Q$,*

$$M(\rho, q) = \{q_i, i \in [1; n] \mid d_Q(q_{n+1}, q_i) \le \epsilon_Q\}.$$

The function M will be called the FRNN monitor.

Problem 1 reduces to Problem 2 by using $Q = X \times Z$, $\epsilon_Q = \epsilon_X$, and by defining the metric d_Q as follows: For every $(x, z), (x', z') \in Q$,

$$d_Q((x, z), (x', z')) := \begin{cases} d_X(x, x') & \text{if } d_Z(z_{n+1}, z_i) \ge \delta_Z \\ \infty & \text{otherwise.} \end{cases}$$

The advantage of stating the monitoring problem using Problem 2 instead of using Problem 1 is simplicity, and from now on, the "monitoring problem" will refer to Problem 2 unless stated otherwise.

4 Preliminaries of FRNN Algorithms

We review the existing FRNN algorithms with a focus on the ones used by our monitors. We use the notation from Problem 2, where Q is a set with the distance metric d_Q, and $\epsilon_Q > 0$ is given.

4.1 Brute-Force (BF) FRNN

The most straightforward solution of Problem 2 is the brute-force algorithm, which simply stores the set of seen points in memory, and after seeing a new point from Q, performs a brute-force search to collect all ϵ_Q-close points. If Q is a d-dimensional real-coordinate space, then clearly the time complexity at the n-th step is $O(d \cdot n)$, since the new point must be compared with n other d-dimensional points. In Section 4.2, we will review some FRNN algorithms with asymptotically better complexities but significantly higher overhead costs. Because of this, for small n, an efficient implementation of the brute-force approach is capable of outperforming other alternatives. This will be visible in our experiments as well where we use the highly optimized brute-force similarity search algorithm implemented in Meta's Faiss library [30].

4.2 *Static* FRNN with Indexing

Although FRNN is a well-studied problem, most existing *non*-brute-force approaches consider the static, one-shot version of Problem 2, namely the setting where the nearest neighbors will be searched once, and the set of seen points is not accumulating. In Section 5, we will present FRNN monitors that use static FRNN algorithm as their back-ends, and, in principle, *any* off-the-shelf static FRNN algorithm can be used. For concrete experimentation and as a proof-of-concept, we chose two representative static algorithms, namely the classic k-dimensional tree or *k-d tree algorithm* and the *sorting-based nearest neighbor* algorithm. These algorithms improve over the brute-force alternative by storing the given points in efficient data structures, aka *indexes*, such that searching for nearest neighbors becomes efficient. We will assume that in Problem 2, $Q = \mathbb{R}^d$ for some dimension $d \in \mathbb{N}$, and d_Q is either the L_2-norm or the L_∞-norm. Although there are FRNN algorithms for general metric spaces [24, 78], they will be redundant for most use cases of i.o.r.. Let $\{q_1, \dots, q_n\} =: \mathcal{D} \subseteq \mathbb{R}^d$ be the given set of past points, and q_{n+1} be the new point as described in Problem 2.

k-d trees. k-d trees are binary trees for storing the given set of points $\mathcal{D}$ in a k-dimensional space (for us $k = d$ and the space is Q). Each leaf node of a k-d tree contains a set of points in $\mathcal{D}$. Each internal node divides the space Q into two halves using a hyperplane, and points in $\mathcal{D}$ that are on the left side of the hyperplane are stored in the left sub-tree, whereas the points that are on the right side are stored in the right sub-tree. The construction of a k-d tree from $\mathcal{D}$ takes $O(dn \log n)$ time. k-d trees are not optimized for finding all neighbors within a given radius, but rather for identifying a fixed number nearest neighbors, which takes $O(\log n)$ time on an average and still $O(n)$ time in the worst case. If we modify k-d trees for the purpose of FRNN queries, then each query would take $O(d \cdot n^{1-1/d} + d \cdot m)$ time, where m is the number of neighbors which are ϵ_Q-close to the given input point. If m is large and approaches n, then the time complexity approaches $O(n)$—the same as the brute-force approach. In practice, most data sets are sparse and m is usually small. Furthermore, k-d tree-based FRNN is superior to brute-force when d is small but n is large, whereas both become equivalent (modulo the additional indexing overhead of k-d trees) when d is large and n is small.

Sorting-based nearest neighbor (SNN). The recently developed SNN algorithm uses a sorting-based indexing scheme that is faster to build than k-d trees. In particular, the indexing step requires $O(n \log n + nd^2)$-time. The algorithm computes an ascending sequence of key values, each value corresponding to a point in the dataset $\mathcal{D}$. The property satisfied by the key values is, that if two points have key values ϵ-far apart, then their L_2 distance must also be greater than ϵ. The FRNN queries exploit this relationship. First, the key value of the input point is computed, requiring $O(d^2)$-time. Second the algorithm can safely discards all points with key values ϵ-far from the key value of the input point. This can be done efficiently using binary search in $O(\log n)$-time. Then a brute force

search is performed on the remaining points. The soundness of SNN relies on the chosen norm, although they are not restricted to the L_2 norm, the L_∞ norm is not supported [20].

4.3 Survey of Other FRNN Algorithms

We chose k-d tree [10] and SNN [20] as representative indexing algorithms, where the indexing of k-d trees uses an implicit partitioning over the input space, while the indexing of SNN uses a partition-free approach. Other algorithms using partitioning-based indexing include R trees [27, 38], ball trees [61], cover trees [12], general metric trees [22], and GriSpy [18], and other algorithms using partition-free indexing include the work by Connor et.al. [25]. Any of these algorithms could be used in our monitor, and as a general rule of thumb, the partition-based approaches will face higher computational blow-up than the partition-free methods for monitoring systems with high-dimensional input spaces [20].

The key aspect in monitoring runtime i.o.r. is scalability, which will require us to use optimized FRNN algorithms that are fast even for high-dimensional data, such that the nearest neighbor search consistently ends *before* the arrival of the next input. Multiple strategies could be used for improving scalability.

A first alternative would be to use off-the-shelf *parallelized FRNN algorithms*, which include works on both CPU-based [14, 16, 21, 45, 58, 77] and GPU-based [56, 60, 63, 79] parallelization. These algorithms have their own strengths and weaknesses, and the choice of the appropriate algorithm will ultimately be driven by the application's requirements. For instance, algorithms based on recent developments in GPU hardware tend to be limited to 3 dimensions [56], while CPU-based algorithm are more flexible [14].

In Section 5, we will build FRNN monitors by periodically recomputing indexes of static FRNN algorithms. A faster alternative would be to use *dynamic FRNN algorithms* permitting incremental updates to the indexing structure [14, 23, 33, 35, 77]. Recent works present incremental indexing of Hamming weight trees for FRNN in Hamming spaces [33] and of k-d trees for FRNN in Euclidian metric spaces [14, 77]. Most of these works provide approximate solutions (explained below), whereas we intend to find the exact set of nearest neighbors; the few existing algorithms [14, 77] for the exact setting will be incorporated in future editions of our monitors.

A third alternative would be to use *approximate FRNN algorithms*, which would trade off monitoring accuracy with performance and may occasionally output false positives (reporting robustness violations even if there is none) or false negatives (not reporting robustness violations even if there are some). The literature on approximate FRNN is vast, and includes approximation schemes that are either data-dependent [5] or data-independent [3], and use various techniques ranging from input space dimensionality reduction [6] to approximate tree-based space partitioning [12, 54, 65] to the hierarchical navigable small world search algorithm [55]. All these approaches can be integrated within our monitor, and may be useful if occasional false outputs are acceptable.

5 FRNN Monitoring via Periodic Indexing

Most existing, non-brute-force FRNN algorithms build index structures for storing the set of input points, and these indexes usually do not support incremental updates that would be suitable for monitoring input *sequences*. Recomputing the entire index at each step would incur a substantial computational cost and is infeasible. We resolve this in Algorithm 1 by using a simple practical approach, namely re-indexing the FRNN data structure only periodically, after the interval of a given fixed number of inputs $\tau > 0$. The algorithm stores the past inputs in two separate memories, namely a long-term memory L and a short-term memory S. The long-term memory L is updated periodically and stores past inputs that appeared *before* the last update of L, while the short-term memory S is the "buffer" that stores inputs that appeared *after* the last update of L. Every time a new input q appears, we need to search for its neighbors in the set $L \cup S$. We delegate the search over S to the brute-force approach and the search over L separately to a static FRNN approach. Every time the size of S reaches τ, we transfer the points in S to L, reset S to the empty set, rebuild the index of the static FRNN algorithm using the updated L, and continue with the next input.

In our experiments, for the static part, we compared k-d tree-based FRNN and SNN as proof-of-concept, although *any* FRNN algorithm could be used. For a fixed dimension of the input space and for a large set of past input points, the performances of k-d trees and SNN are significantly better than the brute-force approach, making them suitable for searching over L. The hyper-parameter τ needs to be selected in a way that it creates a balance between the increasing query time of the brute-force algorithm and the cost of re-indexing the static FRNN algorithm. The following method can be applied. Let n be a given number of input-output pairs, and suppose $f(n)$, $g(n)$, and $h(n)$ represent, respectively, the worst-case time complexities of computing the FRNN index, the search over the long-term memory, and the search over the short-term memory. Typically, $f(n) > h(n) > g(n)$, and there is the tradeoff between indexing too frequently (low τ), paying the high price of $f(n)$ more often, and indexing too rarely (high τ), paying the price of $h(n)$ more often while the lower amount $g(n)$ could be used. We can express the overall amortized complexity of the monitoring algorithm (per execution step) as $T(\tau) = (f(n+\tau) + \tau g(n) + \sum_{i=1}^{\tau} h(i))/\tau$. The functions f, g, and h will depend on the particular algorithms being employed, but usually they are simple enough that we can find the optimal τ such that $T(\tau)$ is minimized.

6 FRNN Monitoring using Binary Decision Diagrams (BDD)

We now present a new FRNN monitoring algorithm that is suitable in the dynamic setting of monitoring, where the seen inputs are incrementally updated over time. Our new algorithm gives rise to monitors that in many cases outperform the monitors using off-the-shelf FRNN algorithms with periodic indexing. The ides is to perform a bi-level search, where the top-level search is *fast but approximate*, and uses an indexing scheme with *binary decision diagrams* (BDD), and the bottom-level search is *slow but exact*, and uses the brute-force algorithm. While the top-level search would quickly narrow down the search space, it can generate false positives, which would then be eliminated by the bottom-level brute-force search. This algorithm works only for L_∞ distance; extensions to other metrics is left for future work.

Algorithm 1 FRNN Monitor: Static FRNN with Periodic Indexing

Input: Space Q, distance metric d_Q, constant $\epsilon_Q > 0$

$L \leftarrow \emptyset$ ▷ initialize long-term memory
$S \leftarrow \emptyset$ ▷ initialize short-term memory
$F \leftarrow StaticFRNN(Q, d_Q, \epsilon_Q)$ ▷ initialize static FRNN object
$F.ComputeIndex(L)$ ▷ *initial indexing*
while *true* **do** ▷ monitoring begins
  $q \leftarrow GetNewInput()$ ▷ q is the new input
  $U \leftarrow F.FRNN(q)$ ▷ static FRNN
  $V \leftarrow BruteForceFRNN(S, q; d_Q, \epsilon_Q)$ ▷ brute-force FRNN
  output $U \cup V$ ▷ monitor's output
  $S \leftarrow S \cup \{q\}$ ▷ update short-term memory
  if $|S| = \tau$ **then** ▷ *recompute index* for static FRNN?
    $L \leftarrow L \cup S$ ▷ update long-term memory
    $S \leftarrow \emptyset$ ▷ reset short-term memory
    $F.ComputeIndex(L)$ ▷ recompute index
  end if
end while

BDDs for indexing. BDDs have been extensively used in hardware and software verification in computer science. The syntactic description of BDDs is irrelevant and out of scope; see the work of Bryant [15] for reference. We will use BDDs essentially as black-boxes to build an efficient data structure for the FRNN search. To this end, a BDD is essentially a function of the form $f\colon \{0,1\}^n \to \{0,1\}$, where $n > 0$ is the number of boolean input variables. Let Q, d_Q, and ϵ_Q be as defined in Problem 2.

Suppose Q has some real and some categorical dimensions. Assuming that the real dimensions have a bounded range, we will discretize the real dimensions into *finitely* many (non-overlapping) intervals of width ϵ_Q. Each interval has a representative point as its label, and every other points inside this interval is "approximated" by the same label. For example, in a data set on humans, "age" can be a real-valued feature, whose realistic range could be 0–120. If $\epsilon_Q = 1$, the discrete intervals become $[0,1), [1,2), \ldots, [119,120)$. For each interval, we can choose the lower bound as its label, i.e., the label of $[34,35)$ is 34. Given a real feature value x, we will use $\overline{x}$ to represent its respective label. The categorical dimensions (such as "gender") are assumed to have only finitely many values, and the "label" of each possible value is the value itself. This way, we discretized Q using a set of finitely many vectors of labels, call it $\overline{Q}$, where the i-th element of each vector is the respective label in the i-th feature dimension. For a given input $q \in Q$, let $\overline{q}$ represent its corresponding label vector.

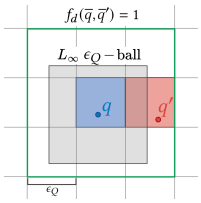


Figure 2: Discretization of $Q = \mathbb{R}^2$. The points have neighboring label, but are ϵ_Q apart.

We now introduce the the BDD $f\colon \{0,1\}^n \to \{0,1\}$ which will store the set W of past inputs using their discrete label vectors. For this, we will require $n = \log(|\overline{Q}|)$ bits to encode the label vectors. At each point, for a given $q \in Q$, and assuming b_q is the boolean encoding of $\overline{q}$, $f(b) = 1$ iff q has been seen in the past. After seeing every new input, f can be updated using standard BDD operations [15]. We also maintain a dictionary Δ, that maps each label vector $v \in \overline{Q}$ to the set of seen points $q \in Q$ with label v. We need another BDD for encoding the distance function d_Q, denoted as $f_d\colon \{0,1\}^{2n} \to \{0,1\}$, which takes as inputs two binary encoded label vectors b, b' of two inputs q, q', and outputs $f_d(b,b') = 1$ iff each vector entry of $\overline{q}$ and $\overline{q'}$ are either all the same or are adjacent labels to each other; in our example of "age," 34 and 35 are adjacent labels. The BDD f_d is constructed statically in the beginning using standard BDD operations [15].

The sketch of the algorithm. The FRNN monitor uses a *hierarchical FRNN search*: The BDDs sit at the top-level, and after each new input q is observed, quickly checks if any past point had the same or adjacent labels. There are three possible outcomes: (a) Neither $\overline{q}$ nor its neighbors appeared before, in which case the monitor outputs $M(\cdot, q) = \emptyset$, a case that we expect to experience most of the time. (b) The vector $\overline{q}$ appeared before but none of its neighbors did, in which case the monitor outputs $M(\cdot, q) = \Delta(\overline{q})$, since every two points with the same label vector are at most ϵ_Q apart. (c) Both (a) and (b) are false, i.e., some neighbor $\overline{q'}$ of $\overline{q}$ appeared before, which could mean either a true positive or false positive, since the distance between two points with neighboring labels may or may not be smaller than ϵ_Q; see Figure 2 for an illustration.

When Option (c) is true and the result of the top-level search is inconclusive, the bottom-level brute-force search comes to rescue. But now, the brute-force algorithm only needs to search within the set of seen inputs that have the neighboring labels of $\overline{q}$, which in most case will be significantly faster than the regular brute-force search over the entire set of seen inputs. The pseudocode of the full algorithm is included in Appendix A, which includes elementary BDD operations like disjunction and membership query. Even though these operations have exponential complexity in the number of BDD variables n [15], i.e., linear complexity with respect to $|\overline{Q}|$, still in practice, modern BDD libraries use a number of smart heuristics and have superior scalability.

7 Performance Optimization: Parallelized FRNN

Many existing FRNN algorithms, including k-d trees and our BDD-based algorithm, use partitioning of the input space, which causes a blow-up in the complexity with growing search space dimension [4]. We present a parallelization scheme for performing nearest neighbor search over real-coordinate spaces equipped with the L_∞ distance metric. The idea is that for the L_∞ metric, two points are ϵ-close iff they are ϵ-close in *each* dimension. This inspired us to decompose the given FRNN problem instance into multiple sub-instances of FRNN with fewer dimensions than the original problem. Each sub-problem can be solved independently, and therefore in parallel, and it is made sure that when the solutions of the sub-problems are composed in a certain way, we obtain a solution for the original FRNN problem. Our parallelization scheme works as a wrapper on any FRNN monitoring algorithm. In our experiments, we demonstrate the efficacy of the parallelized version of both k-d tree-based FRNN monitors and BDD-based FRNN monitors.

For simplicity, we explain the parallelized algorithm using two parallel decompositions of the given problem; the extension to arbitrarily many decompositions is straightforward. Before describing

Algorithm 2 Parallelized FRNN Monitoring

Input: Space Q, distance metric d_Q, constant $\epsilon_Q > 0$

1: $S \leftarrow \emptyset$ ▷ initialize memory
2: **while** *true* **do** ▷ monitoring begins
3: $q \leftarrow GetNewInput()$ ▷ q is the new input
4: $\overline{S} \leftarrow AssignUniqueLabels(S)$ ▷ $\overline{S} \subset \mathbb{R}^{2n} \times \mathbb{N}$
5: $S_A \leftarrow \{s \in \mathbb{R}^n \times \mathbb{N} \mid \exists q \in \overline{S} \,.\, (q.\text{A}, q.\text{id}) = s\}$ ▷ projection on A
6: $S_B \leftarrow \{s \in \mathbb{R}^n \times \mathbb{N} \mid \exists q \in \overline{S} \,.\, (q.\text{B}, q.\text{id}) = s\}$ ▷ projection on B
7: **Do in parallel**
8: $T_A \leftarrow \text{FRNN}(S_A, (p.\text{A}, p.\text{id}); d_Q, \epsilon_Q)$ ▷ local FRNN in A
9: $T_B \leftarrow \text{FRNN}(S_B, (p.\text{B}, p.\text{id}); d_Q, \epsilon_Q)$ ▷ local FRNN in B
10: **End parallel**
11: $T \leftarrow \{t \in \mathbb{R}^{2n} \mid \exists i \in \mathbb{N} \,.\, \exists a \in T_A \,.\, \exists b \in T_B \,.\, (t.\text{A}, i) = a, (t.\text{B}, i) = b\}$ ▷ composition of outputs of local FRNN monitors
12: **output** T
13: $S \leftarrow S \cup \{q\}$ ▷ update memory
14: **end while**

the algorithm, we introduce some notation. We consider FRNN problems on the metric space $(\mathbb{R}^{2n}, d_\infty)$ where d_∞ is the L_∞ norm. For a given point $q = (r_1, \ldots, r_n, r_{n+1}, \ldots, r_{2n}) \in \mathbb{R}^{2n}$, we will write $q.\text{A}$ and $q.\text{B}$ to respectively denote the projections $(r_1, \ldots, r_n)$ and $(r_{n+1}, \ldots, r_{2n})$. We will use the augmented space $(\mathbb{R}^{2n} \times \mathbb{N}, d'_\infty)$, where the extra dimension $\mathbb{N}$ will be used to add unique labels to the points in $\mathbb{R}^{2n}$, and d'_∞ equals to d_∞ with the labels of the points ignored. For a given point $q = (r_1, \ldots, r_n, r_{n+1}, \ldots, r_{2n}, m)$ in the augmented set, we will write $q.\text{id}$ to denote the label m of q, while $q.A$ and $q.B$ will as usual represent $(r_1, \ldots, r_n)$ and $(r_{n+1}, \ldots, r_{2n})$, respectively. We define the function *AssignUniqueLabels* which takes as input a given *finite* set $S \subset \mathbb{R}^{2n}$, and outputs a set $\overline{S}$ in the augmented space $\mathbb{R}^{2n} \times \mathbb{N}$ such that $\overline{S}$ exactly contains the elements of S which are now assigned unique labels.

The parallelized algorithm is presented in Algorithm 2, and we explain it using a simple example in Figure 3. Suppose $n = 1$, i.e., the original FRNN problem is in the metric space $(\mathbb{R}^2, d_\infty)$. Let the inputs be the set $S = \{q_1, q_2, q_3\}$ and the new point p, as shown in Figure 3, where the shaded region around p represents its ϵ_Q-neighborhood for a given $\epsilon_Q > 0$ (as in Problem 2). Clearly, the FRNN algorithm should output $T = \{q_3\}$ as the set of ϵ-neighbors of p.

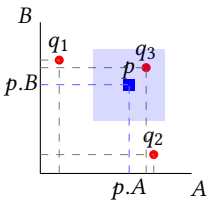


Figure 3: Parallelized FRNN monitoring.

To solve this problem in parallel, we first assign unique labels (Line 4 in Algorithm 2) to the points q_1, q_2, q_3; let $q_i.\text{id} = i$ for every $i \in \{1, 2, 3\}$. Then we project S to the two individual dimensions A and B, giving us the lower dimensional sets S_A and S_B (Lines 5 and 6), where we ensure that the labels of the points are preserved in the projections. Now we solve—in parallel—the two single-dimensional FRNN instances $(S_A, (p.\text{A}, p.\text{id}), \epsilon_Q)$ and $(S_B, (p.\text{B}, p.\text{id}), \epsilon_Q)$, giving us the lower-dimensional sets of ϵ_Q-close points T_A and T_B, respectively (Lines 8 and 9). Finally we compose T_A and T_B to obtain the final answer T (Line 11). The key insight is that the d_∞-norm suggests that two points p and q are ϵ_Q-close iff they are ϵ_Q-close in *all* dimensions. Therefore, if there is a point q that is ϵ_Q-close to p in A but not in B or vice versa, then q is not ϵ-close to p and hence is not included in T; this case applies to both points q_1 and q_2 in Figure 3. As the point q_3 is ϵ-close to p in both dimensions, it is added to T. Note that the additional identification labels of the points help us to perform this synchronized check across both A and B dimensions.

8 Experimental Evaluation

We implemented our algorithms in the tool Clemont, and use it to monitor well-known benchmark models from the literature. On one hand, we demonstrate the monitors' effectiveness on real-world benchmarks (Section 8.1), and on the other hand we demonstrate their feasibility in terms of computational resources (Section 8.2).

Different parts of the experiments were run on different machines. Our monitors for all our examples run on CPUs, and GPU-based implementations (especially the parallellized FRNN algorithm) are left for future works. The only place GPUs were used are for training the models used in our experiments. Our monitors were evaluated on personal laptops with 8GB memory for all examples other than the adversarial robustness example with ImageNet model, whose feature space is too large (150,000 features) for personal laptops, and we used machines with 256GB memory for this one experiment.

8.1 Practical Applications of I.O.R. Monitoring

The experimental setup and the corresponding results are provided in Table 1. For adversarial and semantic robustness, we picked a number of image data sets provided by RobustBench [26], which is a standardized benchmark suite for comparing robustness of AI models. For each of the data sets, we picked the best and the worst performing models, and provided them with a sequence of input images, some of which were deliberately modified with adversarial or semantic corruptions. For individual fairness, we picked the standard fairness data sets, and used baseline and fair models from various existing works from the literature. These models were then given a sequence of input features of individuals. In all these experiments, we deployed our monitors to track the violation of the respective robustness or fairness conditions by the AI models.

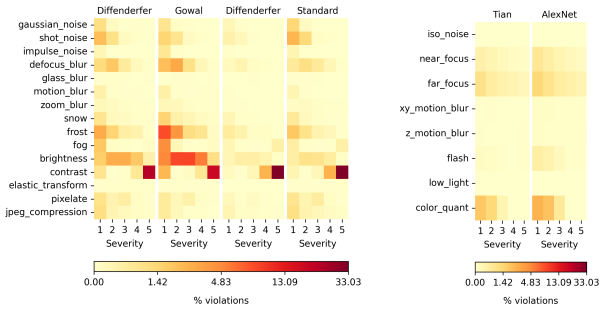


Figure 4: Semantic robustness violations identified during monitoring. Left to right: Robust [29] and base [37] CIFAR100C model, robust [29] and base [26] CIFAR10C model, robust [70] and base [26] ImageNet model.

Key Takeaways. The results are summarized in Table 1, and some concrete instances of detected semantic robustness violations are displayed in Appendix C. Our monitors always output correct answers by design, and therefore the only interesting quality metric is the violations rate, representing the average number of inputs for which the monitor detected i.o.r. violations. Except for a few cases of individual fairness, the violation rate is always positive, highlighting the need for monitoring as an additional safeguard.

The violation rate also confirms that robust or fair training algorithms do improve the runtime i.o.r. in practice, which is expected.

For semantic robustness, we observe some interesting trends. As the severity of the corruption increases, in most cases the violation rate goes down. We suspect that this is because higher corruption increases the distance between the corrupted and the original image in the embedding space, and differences in output labels are not considered as robustness violation anymore. However, for corruption of contrast, this trend does not hold.

We report the time and the memory requirements for monitoring the sequence of all the inputs from the entire benchmark data sets. We observe that the brute-force algorithm outperforms all other approaches in every category, which can be explained by the small number of data points for which brute-force excels (see Section 8.2).

8.2 Computational Performances of Monitors

It is expected that the average computation time of monitors will grow with respect to the length of decision sequences and the number of dimensions in the data, due to the increase in FRNN search complexities. We demonstrate these trends empirically.

Length of decision sequences. We used the HIGGS data set [75] because of its large volume of 10.5 million entries. For each entry we generated a synthetic output, and then used our different monitors to sequentially run over the 10.5 million decisions. We repeated this experiment with 24 and 12 dimensions, and for different values of $\epsilon_Q \in \{0.01, 0.025, 0.05\}$. In Figure 5, we report the rolling average processing time per input (with 100k window size) with respect to increasing length of the decision sequence. We observe that BDD-based monitors are fastest for low dimensions and large values of ϵ_Q, outperforming k-d trees with L_∞-norm. This is surprising given the stellar performance of k-d trees for the L_2 norm. Both SNN and brute force perform reasonably well across all our experiments.

Furthermore, the BDD-based algorithm shows a non-monotonic trend with respect to ϵ_Q: As ϵ_Q increases, the number of partitions decreases, so the BDDs get smaller and more efficient. However, this introduces more false positives, requiring the lower-level brute-force routine to engage more frequently, causing a decrease in performance. Intuitively, the BDD-based monitor performs well when the input data is sparse, so that the false positives are less frequent. This is expected for high-dimensional data, although higher dimension would increase the computational cost. This can be balanced by parallelizing with a just enough number of parallel workers, s.t. the data in each parallel FRNN remains sparse.

Number of dimensions and parallel processing units. We augmented ImageNet data [28] with Gaussian noise and synthetic decisions, and ran our monitor on sequences of $10,000$ labeled samples with varying number of dimensions, obtained by sampling random pixels from the image. We compare various FRNN monitoring algorithms, both without parallelization and measuring computational time starting from an initial history of length 100k, and with parallelization and measuring computational time from the beginning (see Algorithm 2). In Figure 5, we can observe that BDD-based monitors are the slowest as the dimensions increase, and k-d trees and SNN are somewhat comparable. Moreover, the plots show that parallelization drastically increases the viability of monitoring in high dimensions. The number of threads should be chosen as a function of the dimension, as we can observe that there exists a sweet spot in the trade-off between the dimensionality reduction and parallelization overhead. This behavior is especially pronounced for BDD-based monitoring, as was explained earlier.

9 Discussions

We propose *runtime* i.o.r. as a new variant of i.o.r. properties that include adversarial robustness, semantic robustness, and individual fairness in one umbrella. Runtime i.o.r. requires the current run of a given AI decision maker be robust, and therefore is weaker than the traditional local or global i.o.r. properties that require robustness to be satisfied even for inputs that may never appear in practice. We propose *monitors* for the detection of runtime i.o.r. violations by deployed black-box AI models. Our monitors build upon FRNN algorithms and use various optimizations, and their effectiveness and feasibility are demonstrated on real-world benchmarks.

Several future directions exist. Firstly, we will incorporate more advanced FRNN algorithms in our monitors, like the ones with dynamic indexing and approximate solutions. Secondly, our robustness (semantic and adversarial) case studies are only on image data sets, but there are other possibilities. We plan to build monitors for spam filters that would warn the user if different verdicts were made for semantically similar texts from the past. Finally, we will address various engineering questions about the monitoring aspect, like buffering new inputs while computation of previous inputs is still running, and distributed monitoring for networks of AI models.

Acknowledgments

This work was supported in part by the ERC project ERC-2020-AdG 101020093 and the SBI Foundation Hub for Data Science & Analytics, IIT Bombay.

References

[1] Aws Albarghouthi and Samuel Vinitsky. 2019. Fairness-aware programming. In *Proceedings of the Conference on Fairness, Accountability, and Transparency*. 211–219.
[2] Sajjad Amini, Mohammadreza Teymoorianfard, Shiqing Ma, and Amir Houmansadr. 2024. MeanSparse: Post-Training Robustness Enhancement Through Mean-Centered Feature Sparsification. *arXiv preprint arXiv:2406.05927* (2024).
[3] Alexandr Andoni and Piotr Indyk. 2008. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. *Commun. ACM* 51, 1 (2008), 117–122.
[4] Alexandr Andoni and Piotr Indyk. 2017. Nearest neighbors in high-dimensional spaces. In *Handbook of Discrete and Computational Geometry*. Chapman and Hall/CRC, 1135–1155.
[5] Alexandr Andoni, Piotr Indyk, Huy L Nguyen, and Ilya Razenshteyn. 2014. Beyond locality-sensitive hashing. In *Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms*. SIAM, 1018–1028.
[6] Alexandr Andoni, Piotr Indyk, and Ilya Razenshteyn. 2018. Approximate nearest neighbor search in high dimensions. In *Proceedings of the International Congress of Mathematicians: Rio de Janeiro 2018*. World Scientific, 3287–3318.
[7] Tao Bai, Jinqi Luo, Jun Zhao, Bihan Wen, and Qian Wang. 2021. Recent Advances in Adversarial Training for Adversarial Robustness. (2021).

	Data Set	n	d	Norm	ϵ	Base	# Param.	Violations (%)	Robust	#Param.	Violations (%)	Time per sample (ms)			Memory (MB)		
												BF	k-d	k-d (16t)	BF	k-d	k-d (16t)
Adv. Robust.	CIFAR-10	$20k$	$3.1k$	L_∞	$\frac{8}{255}$	[26]	36M	0.948	[9]	366M	0.196	3.71	6.09	1.87	2191	2852	1946
	CIFAR-100					[66]	11M	0.344	[74]	267M	0.316	5.47	32.04	9.73	2057	2878	1945
	ImageNet	$10k$	$150.5k$		$\frac{4}{255}$	[26]	26M	0.767	[2]	198M	0.186	0.26s	7.73s	0.62s	64GB	75GB	65GB
												BF	k-d	SNN	BF	k-d	SNN
Sem. Robust.	CIFAR-10-C	$20k$	384^*	L_2	7.5	[26]	36M	Fig 4	[29]	268M	Fig 4	2.36	83.75	12.75	388	519	573
	CIFAR-100-C					[37]	267M		[29]	269M		4.53	59.26	37.80	396	519	573
	ImageNet	$10k$			12.5	[50]	61M		[70]	86M		32.78	0.2s	60.25	376	373	401
	($*$) from DINOv2[62] embedding						[67] [49]	[67] [49]		[67] [49]	[67] [49]	BF	k-d	BDD	BF	k-d	BDD
Ind. Fair.	German	$1k$	31	L_∞	0.16	[67] [49]	3.3k 1.5k	0.0 2.9	[67] [49]	3.3k 1.5k	0.0 0.0	0.31	0.30	0.74	220	223	235
	Adult	$48.8k$	15				5.1k 1.1k	0.1 23.2		5.1k 1.1k	0.0 2.5	0.69	1.21	5.38	250	360	275
	COMPAS	$6.2k$	18				1.5k 0.7k	1.8 55.4		1.5k 0.7k	0.2 40.6	0.31	0.49	4.50	223	235	236

Table 1: Experimental setup and performance summary for robustness monitoring applications. For each experimental setting we compare: the detected i.o.r. violations for the *base model* and the *robust model*; the processing time per input and the total memory required by our monitor implemented with various FRNN algorithms.

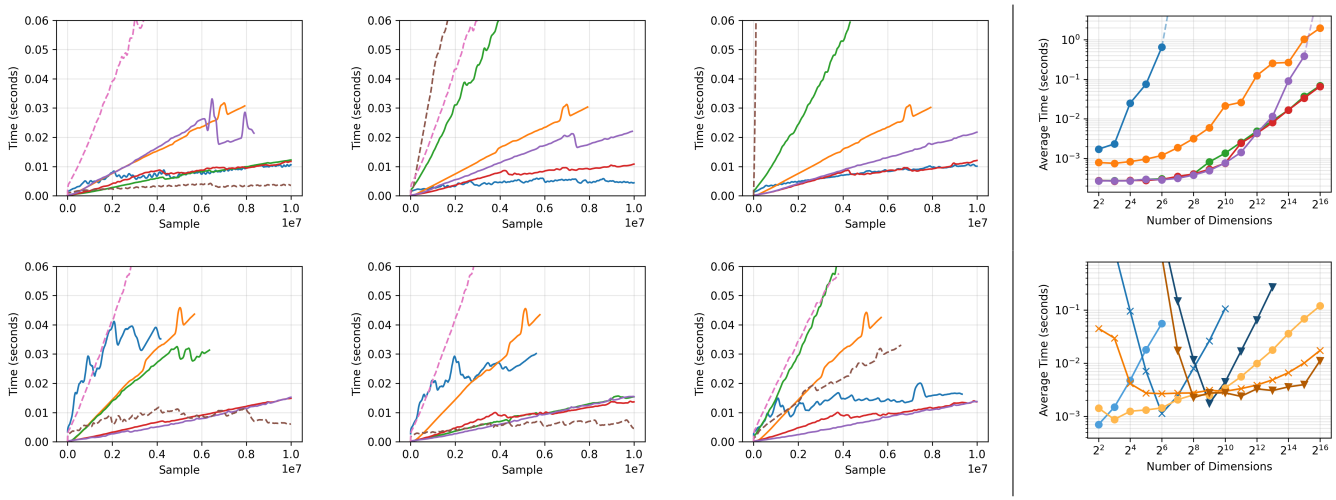


Figure 5: LEFT: Performance comparison on the HIGGS dataset with 10 million entries. The rows correspond to 12 and 24 dimensional inputs respectively. The columns correspond to a ϵ of 0.01, 0.025, and 0.05 respectively. Legend: BDD (——), Brute Force (——), Kd-tree L_∞ (——), Kd-tree L_2 (——), SNN (——), 2-threaded BDD (– – –), 2-threaded Kd-tree (– – –). RIGHT: The plot shows average processing time for 10k images from ImageNet: without parallelization after pre-loading 100k images (top); with parallelization after pre-loading 0 images (bottom). Parallelization plot only: BDD at 1 thread (——), BDD at 16 threads (——), BDD at 96 threads (——), Kd-tree at 1 thread (——), Kd-tree at 16 threads (——), Kd-tree at 96 threads (——)

[8] Ezio Bartocci and Yliès Falcone. 2018. *Lectures on runtime verification.* Springer.

[9] Brian R Bartoldson, James Diffenderfer, Konstantinos Parasyris, and Bhavya Kailkhura. 2024. Adversarial Robustness Limits via Scaling-Law and Human-Alignment Studies. *arXiv preprint arXiv:2404.09349* (2024).

[10] Jon Louis Bentley. 1975. Multidimensional binary search trees used for associative searching. *Commun. ACM* 18, 9 (1975), 509–517.

[11] Elias Benussi, Andrea Patane, Matthew Wicker, Luca Laurenti, and Marta Kwiatkowska. 2022. Individual Fairness Guarantees for Neural Networks. In *31st International Joint Conference on Artificial Intelligence, IJCAI 2022.* International Joint Conferences on Artificial Intelligence (IJCAI), 651–658.

[12] Alina Beygelzimer, Sham Kakade, and John Langford. 2006. Cover trees for nearest neighbor. In *Proceedings of the 23rd international conference on Machine learning.* 97–104.

[13] Sumon Biswas and Hridesh Rajan. 2023. Fairify: Fairness verification of neural networks. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE).* IEEE, 1546–1558.

[14] Guy E Blelloch and Magdalen Dobson. 2022. Parallel Nearest Neighbors in Low Dimensions with Batch Updates. In *2022 Proceedings of the Symposium on Algorithm Engineering and Experiments (ALENEX).* SIAM, 195–208.

[15] Randal E Bryant. 2018. Binary decision diagrams. *Handbook of model checking* (2018), 191–217.

[16] Yu Cao, Xiaojiang Zhang, Boheng Duan, Wenjing Zhao, and Huizan Wang. 2020. An improved method to build the KD tree based on presorted results. In *2020 IEEE 11th International Conference on Software Engineering and Service Science (ICSESS).* IEEE, 71–75.

[17] Marco Casadio, Ekaterina Komendantskaya, Matthew L Daggitt, Wen Kokke, Guy Katz, Guy Amir, and Idan Refaeli. 2022. Neural network robustness as a verification property: a principled case study. In *International conference on computer aided verification.* Springer, 219–231.

[18] Martín Chalela, Emanuel Sillero, Luis Pereyra, Mario Alejandro García, Juan B Cabral, Marcelo Lares, and Manuel Merchán. 2021. Grispy: A python package for fixed-radius nearest neighbors search. *Astronomy and Computing* 34 (2021), 100443.

[19] Li Chen, Penghao Wu, Kashyap Chitta, Bernhard Jaeger, Andreas Geiger, and Hongyang Li. 2024. End-to-end autonomous driving: Challenges and frontiers. *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2024).

[20] Xinye Chen and Stefan Güttel. 2024. Fast and exact fixed-radius neighbor search based on sorting. *PeerJ Computer Science* 10 (2024), e1929.

[21] Byn Choi, Rakesh Komuravelli, Victor Lu, Hyojin Sung, Robert L Bocchino Jr, Sarita V Adve, and John C Hart. 2010. Parallel SAH kD tree construction.. In *High performance graphics*. Citeseer, 77–86.

[22] Paolo Ciaccia, Marco Patella, Pavel Zezula, et al. 1997. M-tree: An efficient access method for similarity search in metric spaces. In *Vldb*, Vol. 97. Citeseer, 426–435.

[23] Paolo Ciaccia, Marco Patella, Pavel Zezula, et al. 1997. M-tree: An efficient access method for similarity search in metric spaces. In *Vldb*, Vol. 97. Citeseer, 426–435.

[24] Kenneth L Clarkson. 1997. Nearest neighbor queries in metric spaces. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*. 609–617.

[25] Michael Connor and Piyush Kumar. 2010. Fast construction of k-nearest neighbor graphs for point clouds. *IEEE transactions on visualization and computer graphics* 16, 4 (2010), 599–608.

[26] Francesco Croce, Maksym Andriushchenko, Vikash Sehwag, Edoardo Debenedetti, Nicolas Flammarion, Mung Chiang, Prateek Mittal, and Matthias Hein. 2020. RobustBench: a standardized adversarial robustness benchmark. *arXiv preprint arXiv:2010.09670* (2020). Accessed: 2024-12-01.

[27] Sanjoy Dasgupta and Kaushik Sinha. 2013. Randomized partition trees for exact nearest neighbor search. In *Conference on learning theory*. PMLR, 317–337.

[28] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 248–255.

[29] James Diffenderfer, Brian Bartoldson, Shreya Chaganti, Jize Zhang, and Bhavya Kailkhura. 2021. A winning hand: Compressing deep networks can improve out-of-distribution robustness. *Advances in neural information processing systems* 34 (2021), 664–676.

[30] Matthijs Douze, Alexandr Guzhva, Chengqi Deng, Jeff Johnson, Gergely Szilvasy, Pierre-Emmanuel Mazaré, Maria Lomeli, Lucas Hosseini, and Hervé Jégou. 2024. The faiss library. *arXiv preprint arXiv:2401.08281* (2024).

[31] Laurent Doyen, Thomas A Henzinger, Axel Legay, and Dejan Nickovic. 2010. Robustness of sequential circuits. In *2010 10th International Conference on Application of Concurrency to System Design*. IEEE, 77–84.

[32] Cynthia Dwork, Moritz Hardt, Toniann Pitassi, Omer Reingold, and Richard Zemel. 2012. Fairness through awareness. In *Proceedings of the 3rd innovations in theoretical computer science conference*. 214–226.

[33] Sepehr Eghbali, Hassan Ashtiani, and Ladan Tahvildari. 2019. Online nearest neighbor search using hamming weight trees. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 42, 7 (2019), 1729–1740.

[34] Yizhak Yisrael Elboher, Justin Gottschlich, and Guy Katz. 2020. An abstraction-based framework for neural network verification. In *Computer Aided Verification: 32nd International Conference, CAV 2020, Los Angeles, CA, USA, July 21–24, 2020, Proceedings, Part I 32*. Springer, 43–65.

[35] Ada Wai-chee Fu, Polly Mei-shuen Chan, Yin-Ling Cheung, and Yiu Sang Moon. 2000. Dynamic vp-tree indexing for n-nearest neighbor search given pair-wise distances. *The VLDB Journal* 9 (2000), 154–173.

[36] Timon Gehr, Matthew Mirman, Dana Drachsler-Cohen, Petar Tsankov, Swarat Chaudhuri, and Martin Vechev. 2018. Ai2: Safety and robustness certification of neural networks with abstract interpretation. In *2018 IEEE symposium on security and privacy (SP)*. IEEE, 3–18.

[37] Sven Gowal, Chongli Qin, Jonathan Uesato, Timothy Mann, and Pushmeet Kohli. 2020. Uncovering the limits of adversarial training against norm-bounded adversarial examples. *arXiv preprint arXiv:2010.03593* (2020).

[38] Antonin Guttman. 1984. R-trees: A dynamic index structure for spatial searching. In *Proceedings of the 1984 ACM SIGMOD international conference on Management of data*. 47–57.

[39] Thomas Henzinger, Mahyar Karimi, Konstantin Kueffner, and Kaushik Mallik. 2023. Runtime monitoring of dynamic fairness properties. In *Proceedings of the 2023 ACM Conference on Fairness, Accountability, and Transparency*. 604–614.

[40] Thomas A Henzinger, Mahyar Karimi, Konstantin Kueffner, and Kaushik Mallik. 2023. Monitoring algorithmic fairness. In *International Conference on Computer Aided Verification*. Springer, 358–382.

[41] Thomas A Henzinger, Konstantin Kueffner, and Kaushik Mallik. 2023. Monitoring algorithmic fairness under partial observations. In *International Conference on Runtime Verification*. Springer, 291–311.

[42] Thomas A Henzinger, Jan Otop, and Roopsha Samanta. 2014. Lipschitz Robustness of Finite-state Transducers. In *34th International Conference on Foundation of Software Technology and Theoretical Computer Science (FSTTCS 2014)*. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 431–443.

[43] Philips George John, Deepak Vijaykeerthy, and Diptikalyan Saha. 2020. Verifying individual fairness in machine learning models. In *Conference on Uncertainty in Artificial Intelligence*. PMLR, 749–758.

[44] Anan Kabaha and Dana Drachsler Cohen. 2024. Verification of Neural Networks' Global Robustness. *Proceedings of the ACM on Programming Languages* 8, OOPSLA1 (2024), 1010–1039.

[45] Ibrahim Kamel and Christos Faloutsos. 1992. Parallel R-trees. *ACM SIGMOD Record* 21, 2 (1992), 195–204.

[46] Guy Katz, Clark Barrett, David L Dill, Kyle Julian, and Mykel J Kochenderfer. 2017. Reluplex: An efficient SMT solver for verifying deep neural networks. In *Computer Aided Verification: 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part I 30*. Springer, 97–117.

[47] Guy Katz, Derek A Huang, Duligur Ibeling, Kyle Julian, Christopher Lazarus, Rachel Lim, Parth Shah, Shantanu Thakoor, Haoze Wu, Aleksandar Zeljić, et al. 2019. The marabou framework for verification and analysis of deep neural networks. In *Computer Aided Verification: 31st International Conference, CAV 2019, New York City, NY, USA, July 15-18, 2019, Proceedings, Part I 31*. Springer, 443–452.

[48] Orçun Kaya, Jan Schildbach, Deutsche Bank AG, and Stefan Schneider. 2019. Artificial intelligence in banking. *Artificial intelligence* (2019).

[49] Haitham Khedr and Yasser Shoukry. 2023. Certifair: A framework for certified global fairness of neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 37. 8237–8245.

[50] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems* 25 (2012).

[51] Preethi Lahoti, Krishna P Gummadi, and Gerhard Weikum. 2019. ifair: Learning individually fair data representations for algorithmic decision making. In *2019 ieee 35th international conference on data engineering (icde)*. IEEE, 1334–1345.

[52] Klas Leino, Zifan Wang, and Matt Fredrikson. 2021. Globally-robust neural networks. In *International Conference on Machine Learning*. PMLR, 6212–6222.

[53] Lan Li, Tina Lassiter, Joohee Oh, and Min Kyung Lee. 2021. Algorithmic hiring in practice: Recruiter and HR Professional's perspectives on AI use in hiring. In *Proceedings of the 2021 AAAI/ACM Conference on AI, Ethics, and Society*. 166–176.

[54] Yi Lin and Yongho Jeon. 2006. Random forests and adaptive nearest neighbors. *J. Amer. Statist. Assoc.* 101, 474 (2006), 578–590.

[55] Yu A Malkov and Dmitry A Yashunin. 2018. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE transactions on pattern analysis and machine intelligence* 42, 4 (2018), 824–836.

[56] Durga Keerthi Mandarapu, Vani Nagarajan, Artem Pelenitsyn, and Milind Kulkarni. 2024. Arkade: k-Nearest Neighbor Search With Non-Euclidean Distances using GPU Ray Tracing. In *Proceedings of the 38th ACM International Conference on Supercomputing*. 14–25.

[57] Ravi Mangal, Aditya V Nori, and Alessandro Orso. 2019. Robustness of neural networks: A probabilistic and practical approach. In *2019 IEEE/ACM 41st International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER)*. IEEE, 93–96.

[58] Ziyang Men, Zheqi Shen, Yan Gu, and Yihan Sun. 2024. Pkd-tree: Parallel k d-tree with Batch Updates. *arXiv preprint arXiv:2411.09275* (2024).

[59] Mark Huasong Meng, Guangdong Bai, Sin Gee Teo, Zhe Hou, Yan Xiao, Yun Lin, and Jin Song Dong. 2022. Adversarial robustness of deep neural networks: A survey from a formal verification perspective. *IEEE Transactions on Dependable and Secure Computing* (2022).

[60] Vani Nagarajan, Durga Mandarapu, and Milind Kulkarni. 2023. Rt-knns unbound: Using rt cores to accelerate unrestricted neighbor search. In *Proceedings of the 37th International Conference on Supercomputing*. 289–300.

[61] Stephen M Omohundro. 1989. Five balltree construction algorithms. (1989).

[62] Maxime Oquab, Timothée Darcet, Théo Moutakanni, Huy Vo, Marc Szafraniec, Vasil Khalidov, Pierre Fernandez, Daniel Haziza, Francisco Massa, Alaaeldin El-Nouby, et al. 2023. Dinov2: Learning robust visual features without supervision. *arXiv preprint arXiv:2304.07193* (2023).

[63] Sushil K Prasad, Michael McDermott, Xi He, and Satish Puri. 2015. GPU-based Parallel R-tree Construction and Querying. In *2015 IEEE International Parallel and Distributed Processing Symposium Workshop*. IEEE, 618–627.

[64] Mahfuzur Rahman, Teoh Hui Ming, Tarannum Azim Baigh, and Moniruzzaman Sarker. 2023. Adoption of artificial intelligence in banking services: an empirical analysis. *International Journal of Emerging Markets* 18, 10 (2023), 4270–4300.

[65] Parikshit Ram and Kaushik Sinha. 2019. Revisiting kd-tree for nearest neighbor search. In *Proceedings of the 25th acm sigkdd international conference on knowledge discovery & data mining*. 1378–1388.

[66] Leslie Rice, Eric Wong, and Zico Kolter. 2020. Overfitting in adversarially robust deep learning. In *International conference on machine learning*. PMLR, 8093–8104.

[67] Anian Ruoss, Mislav Balunovic, Marc Fischer, and Martin Vechev. 2020. Learning certified individually fair representations. *Advances in neural information processing systems* 33 (2020), 7584–7596.

[68] Alex Serban, Erik Poll, and Joost Visser. 2020. Adversarial examples on object recognition: A comprehensive survey. *ACM Computing Surveys (CSUR)* 53, 3 (2020), 1–38.

[69] Rohan Taori, Achal Dave, Vaishaal Shankar, Nicholas Carlini, Benjamin Recht, and Ludwig Schmidt. 2020. Measuring robustness to natural distribution shifts in image classification. *Advances in Neural Information Processing Systems* 33 (2020), 18583–18599.
[70] Rui Tian, Zuxuan Wu, Qi Dai, Han Hu, and Yu-Gang Jiang. 2022. Deeper Insights into the Robustness of ViTs towards Common Corruptions. *arXiv preprint arXiv:2204.12143* (2022).
[71] Vincent Tjeng, Kai Y Xiao, and Russ Tedrake. 2017. Evaluating Robustness of Neural Networks with Mixed Integer Programming. In *International Conference on Learning Representations*.
[72] Caterina Urban, Maria Christakis, Valentin Wüstholz, and Fuyuan Zhang. 2020. Perfectly parallel fairness certification of neural networks. *Proceedings of the ACM on Programming Languages* 4, OOPSLA (2020), 1–30.
[73] Shiqi Wang, Huan Zhang, Kaidi Xu, Xue Lin, Suman Jana, Cho-Jui Hsieh, and J Zico Kolter. 2021. Beta-crown: Efficient bound propagation with per-neuron split constraints for neural network robustness verification. *Advances in Neural Information Processing Systems* 34 (2021), 29909–29921.
[74] Zekai Wang, Tianyu Pang, Chao Du, Min Lin, Weiwei Liu, and Shuicheng Yan. 2023. Better diffusion models further improve adversarial training. In *International Conference on Machine Learning*. PMLR, 36246–36263.
[75] Daniel Whiteson. 2014. HIGGS. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C5V312.
[76] Samuel Yeom and Matt Fredrikson. 2020. Individual Fairness Revisited: Transferring Techniques from Adversarial Robustness. In *Twenty-Ninth International Joint Conference on Artificial Intelligence*.
[77] Rahul Yesantharao, Yiqiu Wang, Laxman Dhulipala, and Julian Shun. 2021. Parallel Batch-Dynamic k d-Trees. *arXiv preprint arXiv:2112.06188* (2021).
[78] Peter N Yianilos. 1993. Data structures and algorithms for nearest neighbor search in general metric spaces. In *Soda*, Vol. 93. 311–21.
[79] Simin You, Jianting Zhang, and Le Gruenwald. 2013. Parallel spatial query processing on gpus using r-trees. In *Proceedings of the 2Nd ACM SIGSPATIAL international workshop on analytics for big geospatial data*. 23–31.
[80] Ekim Yurtsever, Jacob Lambert, Alexander Carballo, and Kazuya Takeda. 2020. A survey of autonomous driving: Common practices and emerging technologies. *IEEE access* 8 (2020), 58443–58469.

Appendices

A Pseudocode of FRNN Monitoring Algorithm using BDDs

Algorithm 3 FRNN Monitoring Algorithm using BDDs

Input: Space Q, distance metric d_Q, constant $\epsilon_Q > 0$

BDD $f \leftarrow$ bddZero ▷ initialize BDD for storing seen label vectors
BDD $f_d \leftarrow$ bddZero ▷ initialize BDD for encoding the adjacency relation two label vectors
$\overline{Q} \leftarrow Discretize(Q, d_Q, \epsilon_Q)$ ▷ discretize the space Q into boxes
for $\overline{q}, \overline{q'} \in \overline{Q}$ **do** ▷ compute f_d (one-time process)
  if $\overline{q}$ and $\overline{q'}$ are adjacent **then**
    $f_d \leftarrow f_d \vee f_{(\overline{q},\overline{q'})}$
  end if
end for
while true **do** ▷ monitoring begins
  $q \leftarrow GetNewInput()$ ▷ q is the next input point
  BDD $g \leftarrow f_d(\cdot, \overline{q})$ ▷ g represents the labels that are adjacent to $\overline{q}$ (the label of q)
  BDD $h \leftarrow f \wedge g$ ▷ h represents the labels that are in g and also have appeared before
  if $h =$ bddZero **then**
    $R = \emptyset$ ▷ no neighbor close to q appeared in the past
  else
    if $(h(b) = 1 \Leftrightarrow b = \overline{q})$ **then**
      $R \leftarrow \Delta(\overline{q})$ ▷ the past neighbors of $\overline{q}$ all had the label $\overline{q}$
    else
      $S \leftarrow getPoints(h)$ ▷ collect past neighboring labels (some can be false positives and are probably not coming from true neighbors of q)
      $R \leftarrow BruteForceFRNN(\Delta(S), q; d_Q, \epsilon_Q)$ ▷ low-level brute-force search to eliminate false positives from S
    end if
  end if
  $f \leftarrow f \vee f_{\overline{q}}$ ▷ update the list of seen labels
  $\Delta(\overline{q}) \leftarrow \Delta(\overline{q}) \cup \{q\}$ ▷ update the dictionary that maps seen labels to seen points
  output $M(\rho, q) = R$ ▷ the output of the monitor
end while

B Proof of Theorem 3.3

Theorem B.1. *Suppose $\epsilon_X, \delta_Z > 0$ are constants and X is infinite.*

(1) *Every decision-sequence of every globally (ϵ_X, δ_Z)-i.o.r. classifier is runtime (ϵ_X, δ_Z)-i.o.r..*
(2) *If the decision-sequence of a classifier is runtime (ϵ_X, δ_Z)-i.o.r., the classifier is not necessarily globally (ϵ_X, δ_Z)-i.o.r..*

Proof. Claim (1): Let D be an (ϵ_X, δ_Z)-i.o.r. classifier and let $x_1, \ldots, x_n \in X^n$ be a sequence of inputs. We know that i.o.r. is satisfied for every two states in X. Hence, it must also be satisfied for $\{x_1, \ldots, x_n\} \subseteq X$. Claim (2): Let $\rho = (x_1, z_1), \ldots, (x_n, z_n) \in X^n$ be a runtime (ϵ_X, δ_Z)-i.o.r. decision sequence. Let D be a classifier generating ρ. Let $x, x' \in X$ such that $d_x(x, x') \leq \epsilon_X \setminus \{x_1, \ldots, x_n\}$. We define D' identical to D, but for x and x' where we set $d_Z(D(x), D(x')) \geq \delta_Z$, i.e., D' is not globally (ϵ_X, δ_Z)-i.o.r.. □

C Counterexample pairs in semantic robustness monitoring

Figure 6: Sample of flagged inputs for semantic robustness on CIFAR-10C, Frost corruption, severity 2, baseline model [26].

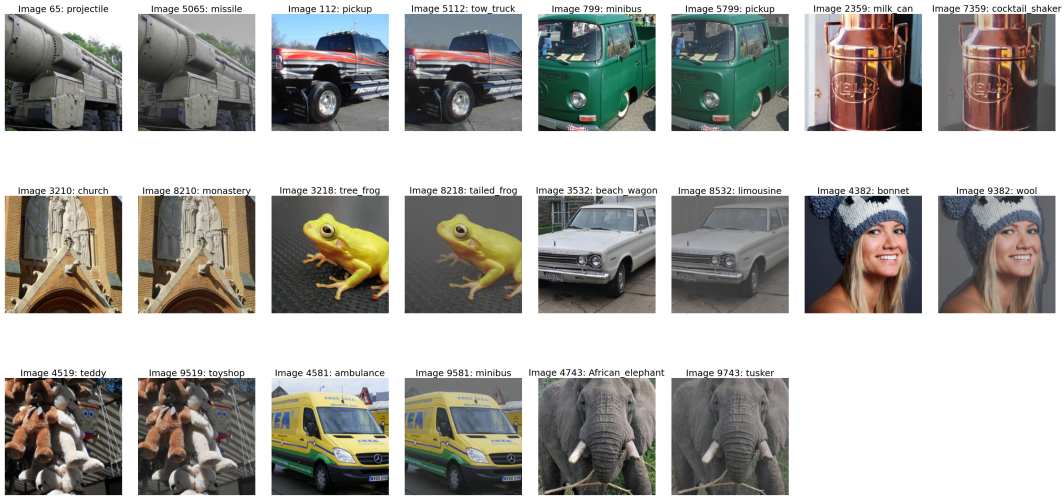


Figure 7: Flagged inputs for semantic robustness on Imagenet-3DCC, Fog corruption, severity 2, top model [70].