SMLE: Safe Machine Learning via Embedded Overapproximation

Matteo Francobaldi, Michele Lombardi

DISI, University of Bologna
{matteo.francobaldi2, michele.lombardi2}@unibo.it

Abstract

Despite the extent of recent advances in Machine Learning (ML) and Neural Networks, providing formal guarantees on the behavior of these systems is still an open problem, and a crucial requirement for their adoption in regulated or safety-critical scenarios. We consider the task of training differentiable ML models guaranteed to satisfy designer-chosen properties, stated as input-output implications. This is very challenging, due to the computational complexity of rigorously verifying and enforcing compliance in deep neural models. We provide an innovative approach based on: 1) a general, simple architecture enabling efficient verification with a conservative semantic; 2) a rigorous training algorithm based on the Projected Gradient Method; 3) a formulation of the problem of searching for strong counterexamples. The proposed framework, being only marginally affected by model complexity, scales well to practical applications, and produces models that provide full property satisfaction guarantees. We evaluate our approach on properties defined by linear inequalities in regression, and on mutually exclusive classes in multi-label classification. Our approach is competitive with a baseline that includes property enforcement in preprocessing (on training data) and postprocessing (on model predictions). Finally, our contributions establish a framework that opens up multiple research directions and potential improvements.

Code — https://github.com/Francobaldi/SMLE-AAAI2025
Extended Version — https://arxiv.org/abs/2409.20517

Introduction

Recent years have seen a rapid expansion in the deployment of AI and Machine Learning (ML) systems, so that their robustness and safety have become a matter of public concern. In safety-critical or regulated contexts such as automation, healthcare, and risk assessment, AI solutions must comply with specific properties set by designers. In non-critical settings, the ability of AI systems to meet user expectations is still an important factor for their acceptance. The AI act recently passed by the European Union is considered by many as the first of many legal frameworks that will stress the importance of compliance for AI systems in high-risk sectors.

However, training robust models is challenging, for multiple reasons. The theoretical basis for most training formulations allows for some level of error and uncertainty. It's almost impossible to ensure that the training samples are fully representative of real-world usage, leading to unpredictable behavior when out-of-distribution inputs are encountered. Finally, a body of research (Chakraborty et al. 2021; Tseng et al. 2024) suggests that complex AI models can be fragile and susceptible to adversarial attacks.

Several research directions addressing these issues have emerged, but the problem remains largely open. Verification approaches focus on checking compliance for a given property (Liu et al. 2021), but for most architectures this problem is NP-hard and very difficult. Other techniques introduce loss terms linked to undesirable properties (e.g. discrimination), or by adversarial training, which generates and accounts for counterexamples at training time (Muhammad and Bae 2022); both methods struggle to provide out-of-distribution guarantees, either for structural reasons or due to the computational complexity of generating and resolving counterexamples. Some methods enforce property compliance on the training data; others work at inference time, e.g. via input randomization (Cohen, Rosenfeld, and Kolter 2019) or by adjusting the model output to ensure property satisfaction (Yu, Xu, and Zhang 2022).

We tackle the problem of training ML models that are robust w.r.t. formal properties specified as implications. To this end, we introduce *a general neural architecture* that is simple to implement and enables efficient verification with a conservative semantic. The architecture is referred to as Safe ML via Embedded overapproximation (SMLE), since it works by augmenting a backbone network with a low-complexity, trainable overapproximator. For certain classes of properties and design choices, conservative verification of SMLE networks has polynomial time complexity.

We use SMLE as a building block for *a training framework based on the Projected Gradient Method*. Our method *guarantees satisfaction of the desired properties upon convergence*. While the computational cost is noticeable, it is *only marginally affected by the backbone network* and remains affordable for practical-scale problems. We ground our framework for two classes of properties. First, we consider properties defined via *linear inequalities*, a large class with applications such as risk assessment, stable time-

series forecasting, collision avoidance. Second, we consider a much more complex (but more specific) combinatorial property, namely *mutual exclusions in multi-label classification*. We evaluate our approach on synthetic and real-world datasets, considering both random and realistic properties. As a strong baseline for the comparison, we use pre-processing and post-processing solutions relying on exact maximum-a-posteriori computation. While our method shows slightly decreased accuracy compared to the baselines, it consistently achieves safety guarantees without increasing the complexity of the inference process. Finally, our contributions open up multiple research directions.

Related Work

Two trends mainly arise from the literature on AI safety and robustness: verification and robust model training.

Verification methods attempt to formally certify the validity of properties in already-trained models. They rely either on Optimization and Searching or on Reachability Analysis. The former work in a declarative fashion: they encode the ML system into a chosen modeling language, such as Mixed-Integer Linear Programming (Tjeng, Xiao, and Tedrake 2017; Bunel et al. 2018; Fischetti and Jo 2018; Anderson et al. 2020; Tsay et al. 2021) or Satisfability Modulo Theory (Ehlers 2017; Huang et al. 2017; Katz et al. 2017, 2019), hence they search for a counterexample that falsifies the assertion. These methods can perform exact verification, but usually fail to scale to real-world use cases. The latter adopt instead algorithmic approaches: they analyze the layer-by-layer propagation of a set of inputs through a neural network, in order to reconstruct the set of reachable outputs, hence to check whether any of them falls into an unsafe region. By maintaining an outer-approximation of the input set during the propagation (Singh et al. 2018; Xiang, Tran, and Johnson 2018; Gehr et al. 2018; Li et al. 2019; Singh et al. 2019), these methods offer increased tractability, but at the price of a loss of completeness. A comprehensive survey on AI Verification is provided in (Liu et al. 2021).

Robust training methods, on the other hand, address safety and robustness in ML systems directly in the design of the model. Some of these methods work in a post-processing fashion, by equipping the main model with auxiliary mechanisms that operate right before or immediately after the actual inference. The former detect and purify, or reject, malicious inputs (Dhillon et al. 2018; Samangouei, Kabkab, and Chellappa 2018; Yang et al. 2019; Pang et al. 2018; Metzen et al. 2017; Xu, Evans, and Qi 2017); the latter correct the output to enforce constraint satisfaction (Wabersich and Zeilinger 2021; Yu, Xu, and Zhang 2022). Another class of methodologies, widely recognized as one of the most effective to improve (local) robustness, is the so-called Adversarial Training (Madry et al. 2018; Zhang et al. 2019; Shafahi et al. 2019; Wang et al. 2020; Zhang et al. 2020; Wong, Rice, and Kolter 2020; Kim, Lee, and Lee 2021). Here the model is trained over a combination of the original datapoints and their worst adversarial examples. The main challenge of adversarial training arises from the generation of adversarial examples, an NP-Hard problem that should be solved iteratively during the training loop. Thus, the focus of this research branch is on designing clever ways to approximate this problem to speed up the computations. A clear overview on Adversarial Training is proposed in (Bai et al. 2021).

The key difference between verification and robust model training is that verification offers formal guarantees when a property is satisfied but provides no guidance for correcting the model when it is violated, while robust training actively promotes property satisfaction but lacks formal certification. This work aims to bridge the gap between these two lines, by introducing a methodology to enforce and formally guarantee the satisfaction of properties at training time.

Robust Training Framework

Formal Properties and Robust Training Let X and Y be random variables respectively representing the model input and the quantity to be predicted; let $\mathcal{X}, \mathcal{Y}$ be their supports – assumed to be bounded – and $P(X, Y)$ their joint distribution. Finally, let $f(x; \theta)$ be a deterministic and differentiable ML model, such as a neural network, with parameter vector θ. We consider properties stated as implications in the form:

$$\forall x \in \mathcal{X}, Q(x) \Rightarrow R(f(x; \theta)) \tag{1}$$

where Q and R are logical predicates defined respectively over $\mathcal{X}$ and $\mathcal{Y}$. This class of properties includes consistency in classification (e.g. "a dog is also an animal", mutual exclusive or forbidden labels), bounding the variability of predictions in multi-step time series forecasting, and safety properties for collision avoidance systems. More examples can be found in the VNN competition (Brix et al. 2023).

We will not consider properties defined via local perturbations, such as classical adversarial examples or local monotonicity. While predicates modeling these properties for a *fixed* set of examples can be constructed, true robustness in these settings should account for the actual input distribution $P(X)$, which is typically inaccessible. While this limitation is common to all robustness methods, it is especially at odds with our approach, which emphasizes full guarantees.

Training a robust model, then, amounts to solving:

$$\arg\min_\theta \left\{ \mathrm{E}_{x,y \sim P(X,Y)} \left[L(y, f(x; \theta)) \right] \text{ s.t. eq. (1)} \right\} \tag{2}$$

where L is the loss function for an individual example and the expectation is usually approximated via the sample average over the training set. Equation (2) is a constrained optimization problem with a differentiable loss; it can be solved to local optimality, for example, via the Projected Gradient Method (Parikh, Boyd et al. 2014). This approach pairs every gradient update with a geometrical projection in feasible space. Formally, the model parameters are updated via:

$$\theta^{(k+1)} = \mathrm{proj}_f(\theta^{(k)} - \eta^{(k)} \cdot \nabla \tilde{L}(\theta^{(k)})) \tag{3}$$

where the superscripts refer to k-th iteration, $\tilde{L}$ is the expectation from eq. (2), and $\eta^{(k)}$ is the learning rate vector. The projection operator is defined as:

$$\mathrm{proj}_f(\theta) = \arg\min_{\theta'} \{ \|\theta' - \theta\|_2^2 \text{ s.t. eq. (1) for } \theta' \} \tag{4}$$

Intuitively, we seek the smallest parameter adjustment that guarantees the satisfaction of the desired property. The main

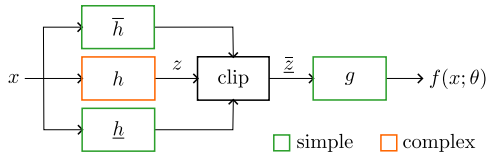


Figure 1: A depiction of a SMLE architecture.

challenge with this approach is that solving eq. (4) can be very expensive from a computational perspective, due to: 1) the universal quantification in eq. (1); 2) the high non-linearity and large size of modern neural models. In fact, even just checking the validity of eq. (1) is a very difficult NP-hard problem (Katz et al. 2017).

SMLE Architecture We address the mentioned difficulties by introducing a general neural architecture that is simple to implement and enables efficient *conservative* verification for eq. (1). We start by viewing the ML model f as a composition of an arbitrary embedding function h and a linear (more precisely, affine) output function g:

$$f(x;\theta) \equiv g(z;\theta_g) \circ h(x;\theta_h) \tag{5}$$

where z is the output of the embedding function and $\theta \equiv (\theta_g, \theta_h)$. This decomposition is both natural and common for many neural networks – including classifiers, by focusing on their logit output. Then, we augment the model by applying clipping to the output of h. In detail, we introduce:

$$\text{clip}(z; l; u) = \max(l, \min(u, z)) \tag{6}$$

which is employed to obtain:

$$g(\bar{z};\theta_g) \circ \text{clip}(z; \underline{h}(x;\theta_{\underline{h}}); \overline{h}(x;\theta_{\overline{h}})) \circ h(x;\theta_h) \tag{7}$$

The embedding z is processed to obtain a clipped embedding $\bar{z}$, with lower and upper bounds computed by two *auxiliary models* $\underline{h}$ and $\overline{h}$. The clipped embedding is then transformed by g to provide the model output. The auxiliary models can be freely chosen, as long as they are differentiable and significantly simpler than the embedding function h. In the simplest case, $\underline{h}$ and $\overline{h}$ can be constant vectors, but they could be implemented as fully-fledged neural networks. The structure of our architecture is depicted in fig. 1.

The structure of eq. (7) ensures that the input to the g function is contained in the box:

$$\overline{\underline{H}}(x;\theta_{\underline{h}},\theta_{\overline{h}}) = [\underline{h}_1(x), \overline{h}_1(x)] \times \ldots \times [\underline{h}_n(x), \overline{h}_n(x)] \tag{8}$$

where n is the size of the embedding vector. In other words, our models *include a trainable overapproximation*. For this reason, we refer to the architecture from eq. (7) as Safe Machine Learning via Embedded overapproximation (SMLE). Many neural architectures employed in AI research and real-world applications can be easily adapted to this structure.

The SMLE architecture allows for efficient, albeit conservative, property verification. This can be framed as searching for a counterexample, i.e. an input value that violates eq. (1):

$$\exists x \in \mathcal{X} : Q(x) \wedge \neg R(f(x;\theta)) \tag{9}$$

With the SMLE architecture, the test can be replaced by:

$$\exists x \in \mathcal{X}, \bar{z} \in \overline{\underline{H}}(x;\theta_{\underline{h}},\theta_{\overline{h}}) : Q(x) \wedge \neg R(g(\bar{z};\theta_g)) \tag{10}$$

Algorithm 1: PDCG$(\theta_g, n_{it}, n_{xs})$

$C =$ empty queue
for $k = 1..n_{it}$ **do**
  search for a counterexample $(x^*, \bar{z}^*)$
  if no counterexample is found **then**
    return success
  if $|C| = n_{xs}$ **then**
    remove the first example in C
  append $\bar{z}^*$ to C
  obtain $\theta_g^{(k)}$ by solving eq. (12)
return failure

Since $\overline{\underline{H}}$ is by construction an overapproximation, if no counterexample for eq. (10) can be found, then the SMLE model is guaranteed to satisfy the desired property. However, the procedure is incomplete and has a *conservative* semantic: if a counterexample is found, the test is inconclusive.

The main appeal of eq. (10) is that is involves only components whose complexity can be *freely controlled*. Regardless of the complexity of the embedding function h, by choosing sufficiently simple $\underline{h}$, $\overline{h}$, and g, conservative verification can be made efficient *by construction*. For example, if $\underline{h}, \overline{h}$ are linear, and Q and R represent polytope membership tests, then eq. (10) is a linear system of inequalities that can be solved in polynomial time. The main drawback is the loss of completeness, which is however less problematic when the focus is on enforcing property satisfaction at training time, as it does not jeopardize guarantees.

Conservative Projection The SMLE architecture provides the basis for building an efficient, conservative, version of the projection operator from eq. (4). As a first step in this direction, we restrict projection to the output parameters θ_g and replace eq. (1) with its conservative SMLE version:

$$\arg\min_{\theta'_g} \|\theta'_g - \theta_g\|_2^2 \tag{11a}$$

$$\text{s.t. } R(g(\bar{z};\theta'_g)) \qquad \forall x \in \mathcal{X} : Q(x), \forall \bar{z} \in \overline{\underline{H}}(x) \tag{11b}$$

In the formulation, the parameters $\theta_{\underline{h}}, \theta_{\overline{h}}$ are omitted from $\overline{\underline{H}}$ since they are not affected by projection. Equation (11) is considerably simpler to solve than the original projection operator, since it involves only simple functions and much fewer parameters. The problem remains challenging, however, since it contains an infinite number of constraints due to the use of universal quantification.

We address this issue via a delayed constraint generation approach. We iteratively introduce constraints associated to counterexamples, which can be efficiently found thanks to the SMLE architecture. Given a finite collection of counterexamples C, eq. (11) is then replaced by:

$$\arg\min_{\theta'_g} \{\|\theta'_g - \theta_g\|_2^2 \text{ s.t. } R(g(\bar{z};\theta'_g)), \forall \bar{z} \in C\} \tag{12}$$

For improved efficiency, and without loss of generality, we restrict our attention to the value of the clipped embedding $\bar{z}$ of each counterexample $(x, \bar{z})$, since its input x does not appear in the body of eq. (11b).

This Projection with Delayed Constraint Generation process is outlined in algorithm 1. As the convergence properties of the process have not yet been analyzed in detail, we introduce two mitigation strategies to ensure that the algorithm cost is manageable. First, to reduce the memory requirements, we store a finite number of counterexamples, discarding older ones according to a FIFO policy. Second, we enforce an iteration limit: if this is reached, guarantees do not hold and the projection is deemed unsuccessful, though the model still becomes comparatively mode robust.

Counterexample Generation In principle, counterexample generation in algorithm 1 could be efficiently handled by solving eq. (10). In practice, however, that leaves no control on which kind of counterexample is obtained: if these are too weak, the projection process could become unreasonably slow. Unfortunately, defining "strong" counterexamples is non-trivial, especially without making assumptions on the nature of the property to be satisfied. In particular, since we treat Q and R as logical predicates, they have no associated continuous measure of constraint violation.

We argue that stronger counterexamples are those requiring large parameter adjustments for their resolution. Hence, given a pair $(x, \underline{\bar{z}})$ satisfying $Q(x)$, we propose to define its strength as a counterexample by means of the L1 norm:

$$\|\theta^*_g - \theta_g\|_1 \tag{13}$$

where θ_g is the current weight vector for the g function and θ^*_g is defined as:

$$\theta^*_g = \arg\min_{\theta'_g} \left\{ \|\theta'_g - \theta_g\|_2^2 \text{ s.t. } R(g(\underline{\bar{z}}; \theta'_g)) \right\} \tag{14}$$

For a pair $x, \underline{\bar{z}}$ that already satisfies the property, we have $\theta^*_g - \theta_g = 0$, while we have $\|\theta^*_g - \theta_g\|_1 > 0$ for a true counterexample. In eq. (13) we use the L1, rather than L2, norm for its lower computational complexity. To the same end, we also propose to restrict the type of adjustment used to evaluate the counterexample strength. In particular, since g is an affine function, by restricting the allowed changes to those affecting the offset (i.e. translation), we have that:

$$g(\underline{\bar{z}}; \theta'_g) = g(\underline{\bar{z}}; \theta_g + \delta) = g(\underline{\bar{z}}; \theta_g) + \delta \tag{15}$$

where δ is the difference $\theta'_g - \theta_g$. We now frame the problem of generating a strong counterexample as that of finding *a pair $x, \underline{\bar{z}}$ whose resolution requires the largest translation*:

$$x^*, \underline{\bar{z}}^* = \arg\max_{x, \underline{\bar{z}}} \|\delta^*\|_1 \tag{16a}$$

$$\text{s.t.: } \delta^* = \arg\min_{\delta} \{\|\delta\|_2^2 \text{ s.t. } R(g(\underline{\bar{z}}; \theta_g) + \delta)\} \tag{16b}$$

$$Q(x) \wedge \underline{\bar{z}} \in \overline{\underline{H}}(x) \tag{16c}$$

where $\|\delta^*\|_1$ and $\|\delta'\|_2^2$ are equivalent to $\|\theta^*_g - \theta_g\|_1$ and $\|\theta'_g - \theta_g\|_2^2$, since we are restricted to translation. While translation alone might be insufficient to solve the projection from eq. (4), here we are interested in resolving *a single* pair $x, \underline{\bar{z}}$. This can always be done by translation if the R predicate has at least one positive assignment. Moreover, we have that $\|\delta\|_1 = 0$ iff no counterexample exists.

The formulation from eq. (16) is partially heuristic in nature, therefore further simplifications can be done for specific properties if they bring computational or quality advantages. In any case, the approach is fairly general, built on a solid rationale, and should result in strong counterexamples. As a challenge, the equation describes a bilevel optimization problem for which defining a solution approach is non-trivial. We will discuss specific groundings of this formulation for two practically relevant settings in the next section.

Algorithm 2: RT(*training-data*, *SGD-params*, θ, n_{it}, n_{xs})

for every usual SGD iteration **do**
  perform a gradient descent update
  run PDCG($\theta^{(k)}, 1, n_{xs}$)
return PDCG($\theta^{(k)}, n_{it}, n_{xs}$)

Robust Training Algorithm We can now introduce our robust training procedure, described in algorithm 2. We start by training a SMLE architecture via Stochastic Gradient Descent (SGD) as usual in deep learning, but after every gradient update we perform a single iteration of our projection algorithm. This phase is meant to improve the model accuracy, while accounting for the need to satisfy the desired property without an excessive computational cost. Once convergence is reached according to usual SGD criteria, we attempt a full projection. If the process succeeds, the resulting SMLE model is guaranteed to satisfy the property, without any further intervention. One goal of our empirical evaluation is investigating the success rate or algorithm 2.

Framework Groundings

Grounding our framework for a given setting requires: 1) choosing a SMLE architecture; 2) defining an implementable formulation for the projection problem from eq. (12); and 3) doing the same for the counterexample generation problem from eq. (16). While the first step is generally easy, the latter two are non-trivial and depend on the considered properties. Here, we discuss viable choices for two settings: 1) the case where Q and R are defined via linear inequalities; and 2) mutual exclusions in multi-label classification. We will focus on mathematical formulations, but obtaining a formally correct grounding requires also to account for solver tolerances and floating-point error propagation. Tolerance-aware formulations can be found in the supplemental material, but we leave robust handling of floating point errors for future work.

Linear Inequalities This setup was chosen since it captures a wide range of practically relevant properties (Brix et al. 2023). In this case, Q and R represent polytope membership tests and are defined via the inequalities:

$$Q(x) \equiv Qx \leq q, \quad R(\hat{y}) \equiv R\hat{y} \leq r \tag{17}$$

where $\hat{y}$ is the model output, Q, R are coefficient matrices, and q, r are the corresponding right-hand side vectors. The projection problem from eq. (12) corresponds to:

$$\arg\min_{\theta'_g} \|\theta'_g - \theta_g\|_2^2 \tag{18a}$$

$$\text{s.t. } R\hat{y}_i \leq r \qquad \forall \underline{\bar{z}}_i \in C \tag{18b}$$

$$\hat{y}_i = \theta_{g,0} + \theta_{g,1:n}\, \underline{\bar{z}}_i \qquad \forall \underline{\bar{z}}_i \in C \tag{18c}$$

where $\theta_{g,0}$ is the offset vector and $\theta_{g,1:n}$ the coefficient matrix for the affine transformation g. Equation (18) is a polytime solvable Quadratic Program.

The process for obtaining a formulation for the counterexample generation problem is more involved and can be found in the supplemental material. The key observation is that resolving a counter example $(x, \underline{\bar{z}})$ violating a given linear inequality requires a translation that is proportional to the classical Linear Programming notion of constraint violation. Accordingly, we generate counterexamples by solving:

$$\arg\max_{x,\underline{\bar{z}}} \sum_{k=1}^{K} \max(0, R_k\hat{y} - r_k) \tag{19a}$$

$$\text{s.t. } Q(x) \tag{19b}$$

$$\underline{h}(x;\theta_{\underline{h}}) \leq \underline{\bar{z}} \leq \max(\underline{h}(x;\theta_{\underline{h}}), \bar{h}(x;\theta_{\bar{h}})) \tag{19c}$$

$$\hat{y} = \theta_{g,0} + \theta_{g,1:n}\,\underline{\bar{z}} \tag{19d}$$

where K is the number of rows in R. The problem can be stated as Mixed Integer Linear Program by linearizing the $\max$ operator in Equation (19a), ensuring that non-violated constraints are correctly treated as not needing resolution, while Equation (19c) accounts for the possible degeneracy of the clipping operation, occurring when $\underline{h}(x;\theta_{\underline{h}}) \geq \bar{h}(x;\theta_{\bar{h}})$. Due to simplifications made in our grounding process, eq. (19) is not equivalent to eq. (16). However, the problem is still capable of generating strong counterexamples, and the objective is still 0 if no counterexample exists.

Mutual Exclusive Classes The second setup find applications when tagging content, or when determining the traits of biological samples, if certain tag or trait combinations should never be predicted together. It was chosen as an example of a *combinatorial* property in a classification setting, which comes with unique challenges. Since SMLE assumes the last layer of the architecture is linear, the Q and R predicates are defined in this case as:

$$Q(x) \equiv \text{TRUE} \tag{20a}$$

$$R(\hat{y}) \equiv I^+(\hat{y}_h) + I^+(\hat{y}_k) \leq 1 \qquad \forall h,k \in F \tag{20b}$$

where $\hat{y}$ represents the logit output of the multilabel classifier, F is the set of mutually exclusive class pairs, and the indicator funtion $I^+(\hat{y}_h) = \text{TRUE}$ iff $\hat{y}_h \geq 0$ and FALSE otherwise. In fact, if a sigmoid layer is used to obtain the actual classifier output, class h will predicted as true iff $\hat{y}_h \geq 0$. The projection problem from eq. (12) corresponds to:

$$\arg\min_{\theta'_g} \|\theta'_g - \theta_g\|_2^2 \tag{21a}$$

$$\text{s.t. } I^+_{i,h} + I^+_{i,k} \leq 1 \qquad \forall h,k \in F, \forall \underline{\bar{z}}_i \in C \tag{21b}$$

$$MI^+_{i,k} \geq \hat{y}_{i,k} \qquad \forall k \in O, \forall \underline{\bar{z}}_i \in C \tag{21c}$$

$$\hat{y}_i = \theta_{g,0} + \theta_{g,1:n}\,\underline{\bar{z}}_i \qquad \forall \underline{\bar{z}}_i \in C \tag{21d}$$

$$I^+_{i,k} \in \{0,1\} \qquad \forall k \in O, \forall \underline{\bar{z}}_i \in C \tag{21e}$$

where O is the set of output classes and auxiliary variables are used to model the $I^+(\hat{y}_{i,k})$ predicates; M is a sufficiently large constant that is used to ensure that $\hat{y}_{i,k} < 0$ if $I^+_k = 0$. Ways to choose the value of M and to handle strict inequalities are discussed in the supplemental material. Equation (21) is a Mixed Integer Quadratic Programming problem, which modern mathematical programming solvers are capable of addressing.

We propose counterexample generation problem based on the observation that, if mutually exclusive classes h and k are predicted at the same time, then adjusting any of $\hat{y}_h$ or $\hat{y}_k$ so that it is less than 0 resolves the violation.

$$\arg\max_{x,\underline{\bar{z}}} I^m_{h,k} \min(\hat{y}_h, \hat{y}_k) \tag{22a}$$

$$\text{s.t. } Q(x) \tag{22b}$$

$$\underline{h}(x;\theta_{\underline{h}}) \leq \underline{\bar{z}} \leq \max(\underline{h}(x;\theta_{\underline{h}}), \bar{h}(x;\theta_{\bar{h}})) \tag{22c}$$

$$\hat{y} = \theta_{g,0} + \theta_{g,1:n}\,\underline{\bar{z}} \tag{22d}$$

$$MI^+_k - M \leq \hat{y}_k \qquad \forall k \in O \tag{22e}$$

$$I^m_{h,k} \leq \frac{1}{2}(I^+_h + I^+_k) \qquad \forall h,k \in F \tag{22f}$$

$$I^+_k \in \{0,1\}\,\forall k \in O, \;\; I^m_{h,k} \in \{0,1\}\,\forall h,k \in F \tag{22g}$$

Equation (22f) ensures that the additional binary variables $I^m_{h,k}$ can be set to 1 only if mutually exclusive classes h and k are predicted. Equation (22e) ensures that the indicator variables I^+_k can be set to 1 only if the corresponding logit otuput is at least 0. The $\min$ operator in the objective can be linearized by introducing a fresh continuous variable and two inequalities, as shown in the supplemental material. Equation (22) is Mixed Integer Linear Program and matches the semantic from eq. (16), since adjusting $\hat{y}_k$ so that it equals 0 corresponds to choosing $\delta_k = -\hat{y}_k$.

Experimentation

We conduct an empirical study designed around the following research questions: *(Q1, Accuracy)* Can our approach achieve an acceptable prediction accuracy, while providing full guarantees? *(Q2, Guarantees)* How effective is our method at providing guarantees compared to existing approaches? *(Q3, Ablation Study)* How is our framework accuracy impacted by its key hyperparameters?

Compared Approaches The baseline for our comparison relies on a *maximum-a-posteriori* operator, defined as:

$$\text{map}(y) = \arg\max_{y'}\{\mathcal{L}(y' \mid y) \text{ s.t. } R(y')\} \tag{23}$$

where $\mathcal{L}$ denotes the likelihood of an adjusted prediction y' w.r.t. a reference prediction y. Intuitively, we correct an infeasible output y, by projecting it to its closest feasible point.

Our baseline consists of two competitors, `preprocess` and `postprocess`. In `preprocess` we apply map to enforce property satisfaction in the training labels, then we train the model on this modified dataset. In `postprocess` we apply map to enforce the property in the model predictions at inference time. In `preprocess`, the highest computational cost, i.e. property enforcement, is paid offline, then any learning algorithm can be used to obtain a model with fast inference. As a disadvantage, this approach does not guarantee the validity of the property. Notably, the resulting model is vulnerable to adversarial attacks. On the other hand, `postprocess` provides full satisfaction guarantees, but might be very inefficient at inference time; in the

case of mutually exclusive classes, for example, eq. (23) is an NP-Hard problem that should be solved for each input. As an optimistic reference, we consider a theoretical approach, `oracle`, which works by applying map to the ground truth labels, and which we use to: 1) obtain an upper bound on the achievable accuracy; and 2) quantify the difficulty of satisfying a given property on a dataset, by measuring the accuracy drop caused by the application of map.

Our approach `smle` is trained through algorithm 2, after having enforced the property on the training data, as for `preprocess`. This pre-training step was not adopted for `postprocess`, since it did not provide any significant advantage in that case, as revealed by preliminary experiments.

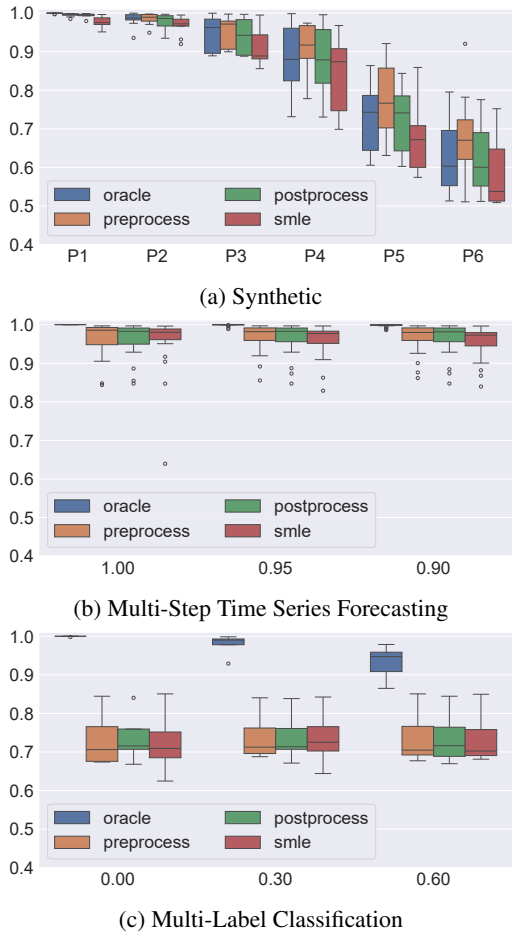
All our experiments, methods and benchmarks are implemented in Python, by relying on the libraries TensorFlow (Abadi et al. 2015), Keras (Chollet et al. 2015) and Scikit-Learn (Pedregosa et al. 2011) for the ML components, and on Pyomo (Hart, Watson, and Woodruff 2011; Bynum et al. 2021), OMLT (Ceccon et al. 2022; Zhang et al. 2024) and Gurobi (Gurobi Optimization, LLC 2024), for the optimization ones. The implementation details are provided in our code, which is publicly available together with our data.

Benchmarks We cover the two groundings described above, regression and multi-label classification, by adopting two benchmarks for the former, called *synthetic regression* and *multi-step time series forecasting*, and one for the latter, simply called *multi-label classification*.

Synthetic Regression. We design 9 learning tasks, consisting in estimating a vector of the form $F_K(x) = ((\sum_{j=1}^{n} x_j)^k)_{k \in K}$. For each task, we train and test on randomly generated data, and we enforce 6 randomly generated linear properties, defined as in eq. (17).

Multi-Step Time Series Forecasting. We consider the problem of estimating the m consecutive values $y = (s_{t+1}, \ldots, s_{t+m})$ of a time series, by observing the n previous values $x = (s_{t-n+1}, \ldots, s_t)$ of the same series $S = (s_t)_{t=0}^{t=N}$. We train and test on 25 time series selected from a public repository (Makridakis, Spiliotis, and Assimakopoulos 2020), each representing a single learning task and split according to a chronological 80%-20% criterion. For each series S, we consider 3 properties defined as: $\forall x, |y_i - y_{i+1}| \leq \Delta_q, \forall i = 1, \ldots, m-1$, for $q = 0.90, 0.95, 1.00$, where Δ_q denotes the q-quantile of the set of deviations $\{|s_t - s_{t+1}|\}_{t=0}^{t=N}$. These properties, which can be encoded as in eq. (17), prevent unreasonably high deviations between two consecutive predictions.

Multi-Label Classification. We train and test on 5 datasets selected from a public repository (Moyano 2024), each representing a single learning task and split according to a random 80%-20% criterion. For each dataset D, we consider 3 properties defined as in eq. (20) with $F = \{(c_0, c_1) \in O_D \mid \text{freq}_D(c_0, c_1) \leq \Delta_q\}$, where O_D represents the set of possible classes, $\text{freq}_D(c_0, c_1)$ the frequency at which the pair (c_0, c_1) occurs among the true labels, while Δ_q denotes the q-quantile of the set of pair frequencies $\{\text{freq}_D(c_0, c_1)\}_{c_0, c_1 \in O_D}$, for $q = 0.0, 0.3, 0.6$. These properties prevent the prediction of unusual combinations.



(a) Synthetic

(b) Multi-Step Time Series Forecasting

(c) Multi-Label Classification

Figure 2: Predictive performance of the considered competitors aggregated across properties of increasing difficulty.

Q1 Results In Figure 2, we compare the predictive performance of our framework against its competitors over the three benchmarks. We evaluate the models with a value in the $[0, 1]$ interval, corresponding to the R^2 in regression, and to the *average class accuracy* in classification. The reported results are obtained on the test sets, and aggregated across the considered properties, sorted by difficulty.

Figure 2a shows that, as expected, the performance of the considered models decreases with the difficulty of the enforced property. Perhaps counter-intuitively, `preprocess` appears to outperform `oracle`, especially as the difficulty of the properties increases. In fact, this is just a side effect of the inability of `preprocess` to provide full guarantees: since `oracle` can be outperformed only by violating the property, this performance gap simply indicates that `preprocess` is leading to significant violations. These two trends do not evidently arise in Figures 2b and 2c. The reason is that, while properties in the Synthetic benchmark are randomly generated, in the other two they are chosen realistically with respect to the data; as a result, they tend to be more consistent with the natural data distribution.

The `postprocess` approach can achieve a high predic-

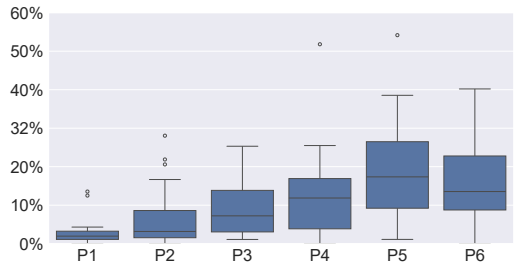


Figure 3: Property violation of `preprocess` on Synthetic.

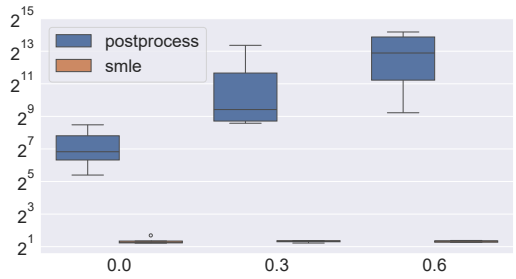


Figure 4: Inference slowdown of `smle` and `postprocess` relative to `preprocess` on Multi-Label Classification.

tive performance, and provides satisfaction guarantees. As already discussed, however, the downside is a more complex and computationally expensive inference process.

Finally, our framework achieves very promising results: in most cases, it performs very closely to its competitors, and in some cases even better (fig. 2c). Remarkably, although Algorithm 2 may fail to reach convergence, all `smle` models presented in this computational study were able to achieve it, and hence to provide full satisfaction guarantees. This required, in algorithm 1, a memory size of a single counterexample for linear properties, and only 10 counterexamples for mutually exclusive classes.

Q2 Results For a trained SMLE model, reduced accuracy is the only price paid to improve robustness. This is not the case for our competitors, with `preprocess` being unreliable in terms of property satisfaction and `postprocess` having a higher computional cost of inference. On the Synthetic benchmark, the most critical for `preprocess`, we count the percentage of test examples where the desired property is violated by this model. On the Classification benchmark, the most critical for `postprocess`, we compute the slowdown, relative to `preprocess` (the fastest model), of the inference time per test example, for both `postprocess` and `smle`. The results of these experiments are reported in figs. 3 and 4, respectively.

As shown in Figure 3, `preprocess` exhibits a very high violation rate, exceeding 50% in the worst cases. In contrast, `smle` guarantees 0% violation rate on the same properties and tasks. While `postprocess` can achieve the same level of guarantees, Figure 4 shows that it requires a significant computational effort at inference time, with a slowdown rising up to 2^{13} on the most demanding properties. In contrast,

Synthetic

Aux. Complexity		Emb. Size	
Constant	Linear	Small	Large
0.814	0.819	0.819	0.823

Multi-Step Time Serie Forecasting

Aux. Complexity		Emb. Size		Emb. Type	
Constant	Linear	Small	Large	ReLU	LSTM
0.958	0.953	0.962	0.958	0.958	0.950

Table 1: Performance of `smle` with different configurations.

`smle` is, regardless of property difficulty, only around twice as slow as `preprocess` at inference time, evidently due to the effect of the introduced overapproximator architecture.

Q3 Results Finally, we investigate the impact in terms of predictive capabilities of two key design choices in our framework, namely, the auxiliary models $\underline{h}$, $\bar{h}$ and the backbone model h. In particular, on the regression benchmarks, we compare two overapproximators with different complexities (constant versus linear), as well as two embedding models with different depth and width (small versus large). On the Forecasting benchmark, we also compare to two different types of backbone (ReLU versus LSTM).

Table 1, where we display the results over the test sets in terms of R^2, aggregated across tasks and properties, shows that our framework is quite robust to different design choices, with only small differences reported between each considered setup. This suggests that defining the hyperparameters for our framework should not be significantly more difficult than for regular scenarios.

Conclusion

We introduced the SMLE architecture, which augments a backbone network with an embedded, trainable overapproximator, and enables conservative property verification with *controllable complexity*. We use our architecture in a framework designed for training models with property satisfacion guarantees, consisting of: 1) a projection algorithm with delayed constraint generation; and 2) a method to generate strong counterexamples. We showed how to ground our method on two classes of properties, and demonstrated its effectiveness against both preprocessing and postprocessing methods, while offering stronger guarantees compared to the former and simpler inference compared to latter.

Our contributions open up several directions for future research. First, there are opportunities to reduce the computational cost of our method, e.g. by replacing the first phase of algorithm 2 with traditional adversarial training, by simplifying the counterexample generation problem, or by using a linear cost at projection time. Second, we believe that our framework might be adapted to address properties involving more than one example, such as fairness constraints, or global monotonicity. Third, the SMLE architecture itself might be extended to deal with variable input sizes via Transformers or Graph Neural Networks.

Acknowledgements

The project leading to this application has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No. 101070149.

References

Abadi, M.; Agarwal, A.; Barham, P.; Brevdo, E.; Chen, Z.; Citro, C.; Corrado, G. S.; Davis, A.; Dean, J.; Devin, M.; Ghemawat, S.; Goodfellow, I.; Harp, A.; Irving, G.; Isard, M.; Jia, Y.; Jozefowicz, R.; Kaiser, L.; Kudlur, M.; Levenberg, J.; Mané, D.; Monga, R.; Moore, S.; Murray, D.; Olah, C.; Schuster, M.; Shlens, J.; Steiner, B.; Sutskever, I.; Talwar, K.; Tucker, P.; Vanhoucke, V.; Vasudevan, V.; Viégas, F.; Vinyals, O.; Warden, P.; Wattenberg, M.; Wicke, M.; Yu, Y.; and Zheng, X. 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. Software available from tensorflow.org.

Anderson, R.; Huchette, J.; Ma, W.; Tjandraatmadja, C.; and Vielma, J. P. 2020. Strong mixed-integer programming formulations for trained neural networks. *Mathematical Programming*, 183: 3–39.

Bai, T.; Luo, J.; Zhao, J.; Wen, B.; and Wang, Q. 2021. Recent Advances in Adversarial Training for Adversarial Robustness. In Zhou, Z.-H., ed., *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, 4312–4321. International Joint Conferences on Artificial Intelligence Organization.

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

Bunel, R.; Turkaslan, I.; Torr, P. H.; Kohli, P.; and Kumar, M. P. 2018. A unified view of piecewise linear neural network verification. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, 4795–4804. Curran Associates Inc.

Bynum, M. L.; Hackebeil, G. A.; Hart, W. E.; Laird, C. D.; Nicholson, B. L.; Siirola, J. D.; Watson, J.-P.; and Woodruff, D. L. 2021. *Pyomo–optimization modeling in python*, volume 67. Springer Science & Business Media, third edition.

Ceccon, F.; Jalving, J.; Haddad, J.; Thebelt, A.; Tsay, C.; Laird, C. D.; and Misener, R. 2022. OMLT: Optimization & Machine Learning Toolkit. *Journal of Machine Learning Research*, 23(349): 1–8.

Chakraborty, A.; Alam, M.; Dey, V.; Chattopadhyay, A.; and Mukhopadhyay, D. 2021. A survey on adversarial attacks and defences. *CAAI Transactions on Intelligence Technology*, 6(1): 25–45.

Chollet, F.; et al. 2015. Keras. https://keras.io.

Cohen, J.; Rosenfeld, E.; and Kolter, Z. 2019. Certified adversarial robustness via randomized smoothing. In *international conference on machine learning*, 1310–1320. PMLR.

Dhillon, G. S.; Azizzadenesheli, K.; Lipton, Z. C.; Bernstein, J.; Kossaifi, J.; Khanna, A.; and Anandkumar, A. 2018. Stochastic activation pruning for robust adversarial defense. *arXiv preprint arXiv:1803.01442*.

Ehlers, R. 2017. Formal verification of piece-wise linear feed-forward neural networks. In *Automated Technology for Verification and Analysis: 15th International Symposium, ATVA 2017, Pune, India, October 3–6, 2017, Proceedings 15*, 269–286. Springer.

Fischetti, M.; and Jo, J. 2018. Deep neural networks and mixed integer linear optimization. *Constraints*, 23(3): 296–309.

Gehr, T.; Mirman, M.; Drachsler-Cohen, D.; Tsankov, P.; Chaudhuri, S.; and Vechev, M. 2018. AI2: Safety and Robustness Certification of Neural Networks with Abstract Interpretation. In *2018 IEEE Symposium on Security and Privacy (SP)*, 3–18.

Gurobi Optimization, LLC. 2024. Gurobi Optimizer Reference Manual.

Hart, W. E.; Watson, J.-P.; and Woodruff, D. L. 2011. Pyomo: modeling and solving mathematical programs in Python. *Mathematical Programming Computation*, 3(3): 219–260.

Huang, X.; Kwiatkowska, M.; Wang, S.; and Wu, M. 2017. Safety Verification of Deep Neural Networks. In Majumdar, R.; and Kunčak, V., eds., *Computer Aided Verification*, 3–29. Springer International Publishing.

Katz, G.; Barrett, C.; Dill, D. L.; Julian, K.; and Kochenderfer, M. J. 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*, 97–117. Springer.

Katz, G.; Huang, D. A.; Ibeling, D.; Julian, K.; Lazarus, C.; Lim, R.; Shah, P.; Thakoor, S.; Wu, H.; Zeljić, A.; Dill, D. L.; Kochenderfer, M. J.; and Barrett, C. 2019. The Marabou Framework for Verification and Analysis of Deep Neural Networks. In Dillig, I.; and Tasiran, S., eds., *Computer Aided Verification*, 443–452. Springer International Publishing.

Kim, H.; Lee, W.; and Lee, J. 2021. Understanding catastrophic overfitting in single-step adversarial training. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, 8119–8127.

Li, J.; Liu, J.; Yang, P.; Chen, L.; Huang, X.; and Zhang, L. 2019. Analyzing Deep Neural Networks with Symbolic Propagation: Towards Higher Precision and Faster Verification. In Chang, B.-Y. E., ed., *Static Analysis*, 296–319. Springer International Publishing.

Liu, C.; Arnon, T.; Lazarus, C.; Strong, C.; Barrett, C.; and Kochenderfer, M. J. 2021. *Algorithms for Verifying Deep Neural Networks*.

Madry, A.; Makelov, A.; Schmidt, L.; Tsipras, D.; and Vladu, A. 2018. Towards Deep Learning Models Resistant to Adversarial Attacks. In *International Conference on Learning Representations*.

Makridakis, S.; Spiliotis, E.; and Assimakopoulos, V. 2020. The M4 Competition: 100,000 time series and 61 forecasting methods. *International Journal of Forecasting*, 36(1): 54–74.

Metzen, J. H.; Genewein, T.; Fischer, V.; and Bischoff, B. 2017. On Detecting Adversarial Perturbations. In *International Conference on Learning Representations*.

Moyano, J. M. 2024. Multi-Label Classification Dataset Repository.

Muhammad, A.; and Bae, S.-H. 2022. A survey on efficient methods for adversarial robustness. *IEEE Access*, 10: 118815–118830.

Pang, T.; Du, C.; Dong, Y.; and Zhu, J. 2018. Towards Robust Detection of Adversarial Examples. In Bengio, S.; Wallach, H.; Larochelle, H.; Grauman, K.; Cesa-Bianchi, N.; and Garnett, R., eds., *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc.

Parikh, N.; Boyd, S.; et al. 2014. Proximal algorithms. *Foundations and trends® in Optimization*, 1(3): 127–239.

Pedregosa, F.; Varoquaux, G.; Gramfort, A.; Michel, V.; Thirion, B.; Grisel, O.; Blondel, M.; Prettenhofer, P.; Weiss, R.; Dubourg, V.; Vanderplas, J.; Passos, A.; Cournapeau, D.; Brucher, M.; Perrot, M.; and Duchesnay, E. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12: 2825–2830.

Samangouei, P.; Kabkab, M.; and Chellappa, R. 2018. Defense-GAN: Protecting Classifiers Against Adversarial Attacks Using Generative Models. In *International Conference on Learning Representations*.

Shafahi, A.; Najibi, M.; Ghiasi, M. A.; Xu, Z.; Dickerson, J.; Studer, C.; Davis, L. S.; Taylor, G.; and Goldstein, T. 2019. Adversarial training for free! *Advances in neural information processing systems*, 32.

Singh, G.; Gehr, T.; Mirman, M.; Püschel, M.; and Vechev, M. 2018. Fast and Effective Robustness Certification. In Bengio, S.; Wallach, H.; Larochelle, H.; Grauman, K.; Cesa-Bianchi, N.; and Garnett, R., eds., *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc.

Singh, G.; Gehr, T.; Püschel, M.; and Vechev, M. 2019. An abstract domain for certifying neural networks. *Proceedings of the ACM on Programming Languages*, 3(POPL): 1–30.

Tjeng, V.; Xiao, K.; and Tedrake, R. 2017. Evaluating Robustness of Neural Networks with Mixed Integer Programming. *arXiv preprint arXiv:1711.07356*.

Tsay, C.; Kronqvist, J.; Thebelt, A.; and Misener, R. 2021. Partition-Based Formulations for Mixed-Integer Optimization of Trained ReLU Neural Networks. In Beygelzimer, A.; Dauphin, Y.; Liang, P.; and Vaughan, J. W., eds., *Advances in Neural Information Processing Systems*.

Tseng, T.; McLean, E.; Pelrine, K.; Wang, T. T.; and Gleave, A. 2024. Can Go AIs be adversarially robust? *arXiv preprint arXiv:2406.12843*.

Wabersich, K. P.; and Zeilinger, M. N. 2021. A predictive safety filter for learning-based control of constrained nonlinear dynamical systems. *Automatica*, 129: 109597.

Wang, Y.; Zou, D.; Yi, J.; Bailey, J.; Ma, X.; and Gu, Q. 2020. Improving Adversarial Robustness Requires Revisiting Misclassified Examples. In *International Conference on Learning Representations*.

Wong, E.; Rice, L.; and Kolter, J. Z. 2020. Fast is better than free: Revisiting adversarial training. In *International Conference on Learning Representations*.

Xiang, W.; Tran, H.-D.; and Johnson, T. T. 2018. Output reachable set estimation and verification for multilayer neural networks. *IEEE transactions on neural networks and learning systems*, 29(11): 5777–5783.

Xu, W.; Evans, D.; and Qi, Y. 2017. Feature squeezing: Detecting adversarial examples in deep neural networks. *arXiv preprint arXiv:1704.01155*.

Yang, Y.; Zhang, G.; Katabi, D.; and Xu, Z. 2019. ME-Net: Towards Effective Adversarial Robustness with Matrix Estimation. In *Proceedings of the 36th International Conference on Machine Learning (ICML)*.

Yu, H.; Xu, W.; and Zhang, H. 2022. Towards safe reinforcement learning with a safety editor policy. *Advances in Neural Information Processing Systems*, 35: 2608–2621.

Zhang, D.; Zhang, T.; Lu, Y.; Zhu, Z.; and Dong, B. 2019. You only propagate once: Accelerating adversarial training via maximal principle. *Advances in neural information processing systems*, 32.

Zhang, J.; Xu, X.; Han, B.; Niu, G.; Cui, L.; Sugiyama, M.; and Kankanhalli, M. 2020. Attacks which do not kill training make adversarial learning stronger. In *International conference on machine learning*, 11278–11287.

Zhang, S.; Campos, J. S.; Feldmann, C.; Sandfort, F.; Mathea, M.; and Misener, R. 2024. Augmenting optimization-based molecular design with graph neural networks. *Computers & Chemical Engineering*, 186: 108684.