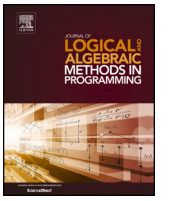
J. Log. Algebraic Methods Program. 137 (2024) 100941

Contents lists available at ScienceDirect

Journal of Logical and Algebraic Methods in Programming

journal homepage: www.elsevier.com/locate/jlamp



Bridging formal methods and machine learning with model checking and global optimisation

Saddek Bensalem [a], Xiaowei Huang [b,*], Wenjie Ruan [b], Qiyi Tang [b], Changshun Wu [a], Xingyu Zhao [b]

[a] University Grenoble Alpes, VERIMAG, Grenoble, France
[b] Department of Computer Science, University of Liverpool, Liverpool, L69 3BX, UK

ARTICLE INFO

Keywords:
Formal methods
Machine learning
Model checking
Global optimisation

ABSTRACT

Formal methods and machine learning are two research fields with drastically different foundations and philosophies. Formal methods utilise mathematically rigorous techniques for software and hardware systems' specification, development and verification. Machine learning focuses on pragmatic approaches to gradually improve a parameterised model by observing a training data set. While historically, the two fields lack communication, this trend has changed in the past few years with an outburst of research interest in the robustness verification of neural networks. This paper will briefly review these works, and focus on the urgent need for broader and more in-depth communication between the two fields, with the ultimate goal of developing learning-enabled systems with excellent performance and acceptable safety and security. We present a specification language, MLS^2, and show that it can express a set of known safety and security properties, including generalisation, uncertainty, robustness, data poisoning, backdoor, model stealing, membership inference, model inversion, interpretability, and fairness. To verify MLS^2 properties, we promote the global optimisation-based methods, which have provable guarantees on the convergence to the optimal solution. Many of them have theoretical bounds on the gap between current solutions and the optimal solution.

1. Introduction

Recent advances in machine learning have enabled the development of complex, intelligent software systems with human-level performance. Notable examples include image classification and natural language processing, among many others. However, even if many machine learning systems (or models) have been successfully applied to industrial applications, they are not rigorously engineered. Instead, their design and implementation are based on developers' experience and have been frequently referred to as "dark art". Compounded with the intensively discussed safety and security issues discovered through various adversarial attacks, such as [1–7] and the growing expectation that machine learning models will be applied to safety-critical applications, it is clear that rigorous engineering methods are urgently needed [8].

Successful experience from industrial software engineering, which produced software currently applied in, e.g., automotive and avionic applications, suggests that, to develop high-quality and low-cost software in a limited production time, a software devel-

* Corresponding author.
E-mail address: xiaowei.huang@liverpool.ac.uk (X. Huang).

https://doi.org/10.1016/j.jlamp.2023.100941
Received 29 January 2023; Received in revised form 12 December 2023; Accepted 20 December 2023

Available online 28 December 2023

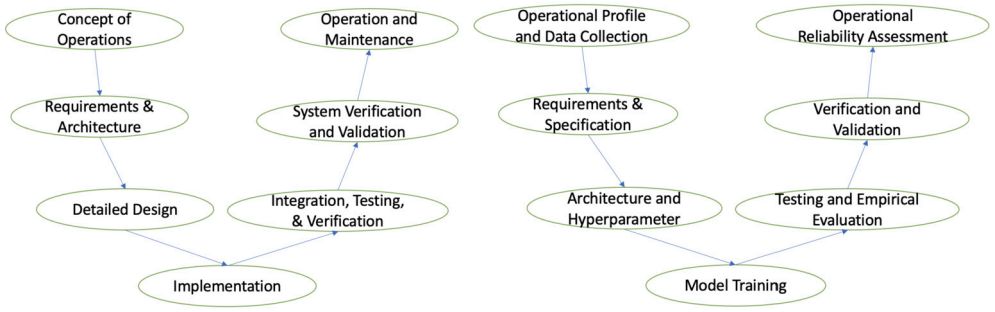


Fig. 1. V-Models for Software [9] (Left) and Machine Learning (Right) Development.

opment life cycle (SDLC) process is required. Considering the V-model of an SDLC, as shown in the left diagram [9] of Fig. 1, the development of a software system starts from the understanding and documentation of the operational concepts, the development of the requirement and architecture, and the conduction of detailed design before its actual implementation (i.e., coding). Once coded, the software must pass unit-level testing and system-level, verification, and validation before deployment. After the deployment, maintenance is still required throughout the life of the software. It should be noted that, during the execution of the V-model, feedback from a later stage may often trigger the modification to an earlier stage. Therefore, the development is an iterative process rather than a sequential process. For machine learning models, the current V-model is no longer suitable, because the machine learning models are over-parameterised with complex, fixed structures. Due to the lack of operational semantics, there is no explainable way of decomposing the architecture and parameters. A new V-model, as shown in the right diagram of Fig. 1, can still be considered, and we will discuss its details in Section 2.

Formal methods are essential tools throughout software development (e.g., by following the V-model) as they provide mathematical rigour (and therefore provable guarantees) to the performance, safety, and security of the final software products. However, unlike software systems where formal methods have been developed on the specification, development, and verification of a system, formal methods are only developed on the robustness verification of neural networks [10,11,8]. Only local robustness defects of a neural network can be dealt with. This paper argues that, to support the new V-model, more formal method-based techniques are needed to deal with various safety and security defects of neural networks.

While compelling, the development of formal methods for neural networks is non-trivial because the machine learning community currently deals with safety and security defects in an ad-hoc way by, e.g., discovering the weakness with various attacks or improving the models with enhanced training methods. That is, dedicated algorithms are developed for specific defects. In this paper, we first attempt to describe a dynamic model for learning-enabled systems and then use a common specification language, named MLS^2, to specify a set of safety and security properties on the dynamic model. Based on the specification language and the dynamic model, the verification of MLS^2 properties can be reduced to the computation of a few constructs, including the data distribution, the posterior distribution, the distance between distributions, and the quantification over a set of inputs.

We consider global optimisation (GO) methods for the verification algorithms to compute the language constructs. GO methods can make efficient computations and converge to the optimal results. Many GO methods have a provable bound on the error between the obtained optimal solutions. For the computation of the posterior distribution, we may consider Markov Chain Monte Carlo (MCMC), which approximates the distribution with a Markov chain, or Variational Inference, which finds the best-known distribution that approximates the unknown distribution. For the universal quantification over a set of inputs, we may consider Lipschitzian optimisation [12,13] as we did for the robustness verification [14–16].

The organisation of the paper is as follows. In the next section, we will provide notations for the following sections. Then, we introduce the new V-model in Section 2, and a new dynamic model for learning-enabled systems in Section 3. These are followed by introducing new specification language in Section 4 and how it can be used to express known properties in Section 5. We will explain how the language can be verified in Section 6. Finally, we introduce related work in Section 7 and conclude in Section 8.

2. Preliminaries and V-model for machine learning

We use $f_{\mathbf{w}}$ to denote a neural network with weight matrix $\mathbf{w}$. We use $\mathbf{x}$ to range over the input domain $\mathbb{R}^n$, and y to range over the labels C. Given $f_{\mathbf{w}}$ and $\mathbf{x}$, we write $f_{\mathbf{w}}(\mathbf{x})$ for the probability distribution of classifying $\mathbf{x}$, such that $f_{\mathbf{w}}(\mathbf{x})(c)$ for $c \in C$ is the probability of classifying $\mathbf{x}$ with label c. Moreover, we write $\hat{y} = \arg\max f_{\mathbf{w}}(\mathbf{x})$ for the predictive label. Given a label y, we write $\mathbf{y}$ for its one-hot encoding. A one-hot encoding can be seen as a probability distribution.

In this section, we explain the phases in the new V-model as shown in the right diagram of Fig. 1.

Operational profile and data collection Different from the "Concept of Operations" phase of the traditional V-model where, the project manager communicates with the customers to understand their needs and expectations, the development of an ML model also requires the interaction with the customers to retrieve the operational profile and the training data.

A neural network $f_{\mathbf{w}} : \mathbb{R}^n \to [0,1]^{|C|}$ is to simulate a target function $h : \mathbb{R}^n \to \{1,...,|C|\}$. For a real-world function h, the appearance of an input $\mathbf{x}$ in the space $\mathbb{R}^n$ is not uniform. We assume that there is a probability density function $\mathcal{D} : \mathbb{R}^n \to [0,1]$,

which returns the probability $D(\mathbf{x})$ for each input $\mathbf{x}$. We may also abuse the notation and write $D((\mathbf{x}, y))$, with the assumption that $D((\mathbf{x}, y)) = D(\mathbf{x})$ because h is a deterministic function. The distribution D is called *data distribution*. It is usually required that the training data includes a set of inputs sampled, i.i.d. from the data distribution.

When designing a neural network for some critical applications, we may need to consider the environment in which it will be deployed. In the operational environment, we also have a distribution $\mathcal{O} : \mathbb{R}^m \to [0, 1]$, which we call *operational profile* by inheriting the terminology from the reliability engineering area [17]. In [18,19], methods have been proposed to learn an approximate operational profile $\hat{\mathcal{O}}$ through a set of data collected from the operational environment. The main challenges are to accurately estimate the input space of high dimensions and to provide provable guarantees on the gap between $\hat{\mathcal{O}}$ and $\mathcal{O}$.

Requirement and specification Unlike the "Requirements and Architecture" phase of the traditional V-model where the project team determines the exact requirements and has a high-level design, the project team will not have the exact or complete requirements and specifications. Instead, the team must figure out a set of properties the final ML system should satisfy.

Usually, the target function h is unlikely to correctly and precisely specify. The training data can be seen as partial specification, with positive and negative examples. Beyond training data, we may need other requirements that cannot be explicitly expressed with the training data, such as robustness, privacy, and security properties that we will discuss in Section 5. A common practice in formal methods is to design a specification language (as we will do in Section 4), show that it can express desirable properties (Section 5), and design verification algorithms that can work with any specification expressible with the language (Section 6).

Architecture and hyperparameter Similar to the "Detailed Design" phase of the traditional V-model where the team will design concrete modules and unit tests, machine learning systems require network architecture and hyper-parameters as the detailed design. However, such design does not entirely rely on the information from previous stages. Instead, it requires additional domain expertise and experience from the designer, frequently refer as the "dark art".

The detailed design of a neural network is based on its architecture and hyper-parameters. Various criteria have been utilised to determine the quality of a trained model, and generalisation is the most popular one. We write G_f^{0-1} for the 0-1 generalisation error of neural network f, and $\mathcal{F}$ for the set of neural networks. Then, G_f^{0-1} can be decomposed as follows:

$$G_f^{0-1} = \underbrace{G_f^{0-1} - \inf_{f \in \mathcal{F}} G_f^{0-1}}_{\text{Estimation error of } f} + \underbrace{\inf_{f \in \mathcal{F}} G_f^{0-1} - G_{\mathbf{d}_{train}}^{0-1,*}}_{\text{Approximation error of } \mathcal{F}} + \underbrace{G_{\mathbf{d}_{train}}^{0-1,*}}_{\text{Bayes error}} \tag{1}$$

where $G_{\mathbf{d}_{train}}^{0-1,*}$ is the 0-1 generalisation error of the Optimal Bayes classifier over the training dataset $\mathbf{d}_{train}$. The *Bayes error* is the lowest and irreducible error over all possible classifiers for the given classification problem [20]. It is non-zero if the true labels are not deterministic (e.g., an image is labelled as y_1 by one person but as y_2 by others), thus intuitively, it captures the uncertainties in the dataset $\mathbf{d}_{train}$ and the true distribution D when aiming to solve a real-world problem with machine learning. The *approximation error of* $\mathcal{F}$ measures how far the best classifier, in $\mathcal{F}$ is from the overall optimal classifier, after isolating the Bayes error. The set $\mathcal{F}$ is determined by the architecture of the machine learning model. Thus the activities at this stage are to minimise this error with optimised architecture and hyper-parameters.

Model training Unlike the "implementation" where the programmers write the software code, machine learning systems rely on the optimisation algorithm to find the solution automatically. Once the architecture and the hyper-parameters are selected, the training is an optimisation process to reduce the estimation error. The *estimation error of* f measures how far the learned classifier f is from the best classifier in $\mathcal{F}$. Lifecycle activities at the **model training** stage essentially aim to reduce this error.

Both the approximation and estimation errors are reducible. The *ultimate goal* of all lifecycle activities is to reduce the two errors to 0. This is analogous to the "possible perfection" notion of traditional software as pointed to by Rushby and Littlewood [21,22]. That is assurance activities, e.g., performed in support of DO-178C, can be best understood as developing evidence of possible perfection. Similarly, for a safety-critical machine learning model, we believe its lifecycle activities should be considered as aiming to train a "possibly perfect" model regarding the two *reducible* errors. Thus, we may have some confidence that the two errors are both 0 (equivalently, prior confidence in the *irreducible* Bayes error since the other two are 0), which indeed is supported by ongoing research into finding globally optimised DNNs [23]. When GO is not achievable, heuristic methods to achieve estimation errors as minor as possible should also be considered, such as the adversarial training methods such as [24–26].

Testing and empirical evaluation Unlike integration testing in software, which tests the interfacing between modules, machine learning has less precise semantics and much more complex relations between modules (such as layers, filters, and neurons). A testing engineer of machine learning is expected to conduct the following two groups of testing activities.

First, it has been a common practice in machine learning that some empirical evaluation methods, such as test accuracy and ROC curve, are taken to understand the performance of a trained model roughly. This step is still required to efficiently rule out some poorly performed architecture and hyperparameters. Second, the testing methods, such as [27–30], are also developed to utilise automatically generated test cases to estimate the performance of the trained model. When using a testing method, we must evaluate test adequacy, i.e., when to terminate the test case generation process. Two main approaches can be utilised, including the behaviours of a machine learning model in the inference stage and the data instances that might appear in the operational phase.

For these approaches, objective metrics are needed to determine the extent to which an analysis technique has been conducted, as discussed in the later ALARP principle.

Verification and validation This step is similar to traditional software and machine learning, where the team tests whether the obtained system complies to the functional and non-functional requirements. This process can provide a rigorous analysis, and can conclude the compliance with probable guarantees.

Neither empirical evaluation nor testing methods can have a provable guarantee of the result. For safety-critical applications, formal verification may be needed. Formal verification requires mathematically rigorous proof to argue for or against the satisfiability of a property on a given neural network. Existing verification algorithms are mainly focused on point-wise robustness, i.e., the robustness of the model over a given input. The algorithms can be roughly categorised into constraint-solving based methods [11], abstract interpretation based methods [31–33], GO based methods [10,14,34], and game-based methods [35,36]. The first two categories treat deep learning as a white box, with the computation needed on all neurons. This results in the scalability issue due to the complexity of the problem and the size of the deep learning. The latter two categories can work with real-world deep learning models but are still subject to the curse of dimensionality. Currently, the verification is focused on the robustness problem, and we will discuss how to work with other properties in Section 6.

Operational reliability assessment Due to the incomplete requirements, this stage can be more involved than the "Operation and Maintenance" phase in the V-model for software. A machine learning model is likely adaptive, and can run in an environment that is deviated from its training environment. In these cases, maintenance engineers must closely monitor the system execution, detect deviations, and take action when risks occur.

In [18,19], we know that it is possible to compute the reliability in a given operational profile $\mathcal{O}$. However, the deployment environment may change, rendering the reliability assessment inappropriate. Nevertheless, experience may be learned from reliability engineering for conventional software, where techniques have been developed to monitor the changes between different software versions to ensure the maintenance and improvement of reliability [37–39]. For the machine learning models, it is suggested that we monitor two data distributions (or operational profiles), one for the original environment and the other for the new environment. Since the reliability of the result is on the original environment, and we can measure the distance between two data distributions, techniques can be developed to predict the reliability in the new environment conservatively.

3. A dynamic model for learning-enabled systems

We assume a learning-enabled autonomous system with a perception component and an attacker. Let C be the set of class labels for the perception component and $Prop$ a set of atomic propositions. The system is modeled as a labeled probabilistic transition system $M = (S, A, P, \pi)$, where

- S is a set of states,
- $A = \{(y_1, y_2) \mid y_1, y_2 \in C\} \cup \{\chi, \tau\}$ is a set of actions,
- $P : S \times A \times S \to [0, 1]$ is a probabilistic transition relation such that $\sum_{a \in A} \sum_{s' \in S} P(s, a, s') = 1$ for any s, and
- $\pi : S \to 2^{Prop}$ is a labeling function.

Depending on who leads the behaviour, we categorise system transitions into three classes as follows:

- *internal transition*, denoted by $P(s_1, \tau, s_2)$, which describes the internal transition behaviour of the system excluding the perception component, e.g., it can express the update of system objective, the high-level planning, the low-level mechanical movement, etc.;
- *perception-led transition*, denoted by $P(s_1, (y_1, y_2), s_2)$, which is a system transition based on a prediction – with the perception component – on state s_1 (or the input to the perception component that is determined by the state s_1), such that the ground truth is class y_1 and the predictive label is y_2;
- *attacker-led transition*, denoted by $P(s_1, \chi, s_2)$, which means a system transition led by an attacker to update a private dataset $\mathbf{d}_{adv}$.

For the security and privacy properties, we assume every state includes several components, i.e., $\mathbf{w}$, $\mathbf{d}_{train}$, $\mathbf{d}_{test}$, $\mathbf{d}_{adv}$, where $\mathbf{w}$ is the current weight of the perception component, $\mathbf{d}_{train}$ is the training dataset, $\mathbf{d}_{test}$ is the test dataset, and $\mathbf{d}_{adv}$ is a dataset maintained by the attacker. Usually $\mathbf{d}_{train}$ and $\mathbf{d}_{test}$ do not change. Moreover, the state may include some Boolean variables:

- *inference* turns *True* if and only if the perception component is applied the next time. Once *inference* = *True*, the state will include an instance $(\mathbf{x}, y)$.
- *training*, which turns *True* if and only if the last internal transition is for the training of the perception component.

Based on M, we can construct a path $s_0 a_1 s_1 a_2 s_2 \ldots$ such that $P(s_i, a_{i+1}, s_{i+1}) > 0$ for all $i \geq 0$. Let $Path(M)$ be the set of paths. We need some notations. For every input instance $\mathbf{x}$, we use

- $y(\mathbf{x})$ to denote the ground truth label;
- $\hat{y}(\mathbf{x})$ to denote the predictive label based on the current perception model;
- $y^*(\mathbf{x})$ to denote a robust predictive label such that it is the predictive label with the greatest number of corresponding instances within a norm ball;
- and $y'(\mathbf{x})$ to denote a monitor-enforced label.

We may omit the postfix $(\mathbf{x})$ and only write $y, \hat{y}, y^*, y'$ if there is no confusion.

3.1. Transition relation

In the following, we provide more details on the transition relation $P(s_1, a, s_2)$, according to the actions $a \in A$.

3.1.1. Transitions for attacker to collect data

The transition $P(s_1, \chi, s_2)$ is a probabilistic transition to update the variable $\mathbf{d}_{adv}$. The update depends on the need of the attacker and can be, e.g., including the past inference result by letting $\mathbf{d}_{adv} = \mathbf{d}_{adv} \cup \{(\mathbf{x}, \hat{y})\}$, where $\hat{y}$ is the predictive label obtained by the attack when inputting instance $\mathbf{x}$ on state s_1. By collecting operational data in this way, attackers can reconstruct the distribution of the training dataset used by the model to achieve their attack goals, such as performing membership inference.

3.1.2. Transitions for perception component

First, we consider the case where the perception component always makes correct predictions. This assumption was made by much existing literature on the specification and evaluation of autonomous systems, such as [40,41]. Technically, let $(\mathbf{x}, y)$ be the instance on a state s_1, we let

$$P(s_1, (y, y), s_2) \triangleq 1 \tag{2}$$

which suggests that $P(s_1, (y', y'), s_2) = 0$ for any $y' \neq y$.

Now we consider the case where the perception component can be imperfect. Certainly, we may directly use the predictive label, and therefore, the transition $P(s_1, (y, \hat{y}), s_2)$ is defined as follows:

$$P(s_1, (y, \hat{y}), s_2) \triangleq P(s_1, (\hat{y}, \hat{y}), s_2) \times P(\hat{y}|y, s_1) \tag{3}$$

where $P(\hat{y}|y, s_1)$ is the predictive probability of the perception component on the instance $(\mathbf{x}, y)$ contained in the state s_1, and $P(s_1, (\hat{y}, \hat{y}), s_2)$ expresses the probability of transitioning from s_1 to s_2 with a decision $\hat{y}$ made by the perception component.

3.2. When perception component protected by statistical verification method

For an imperfect perception, the autonomous system may use a robust label, and in this case, the transition is $P(s_1, (y, y^*), s_2)$ such that

$$P(s_1, (y, y^*), s_2) \triangleq P(s_1, (y^*, y^*), s_2) \times \sum_{\hat{y} \in C} P(\hat{y}|y, s_1) \times P(y^*|\hat{y}, s_1) \tag{4}$$

where $P(y^*|\hat{y}, s_1)$ can be obtained through statistical verification of neural networks such as [42,43], $P(\hat{y}|y, s_1)$ is the prediction probability, and $P(s_1, (y^*, y^*), s_2)$ expresses the probability of transitioning from s_1 to s_2 with a decision y^* made by the perception component.

3.3. When perception component protected by runtime monitor

We may replace the neural network verification with the runtime monitor by using $P(y'|\hat{y}, s_1)$ to approximate the robust labeling $P(y^*|\hat{y}, s_1)$. Therefore, we assume the autonomous system uses the monitored labeling, i.e.,

$$P(s_1, (y, y'), s_2) \triangleq P(s_1, (y', y'), s_2) \times \sum_{\hat{y} \in C} P(\hat{y}|y, s_1) \times P(y'|\hat{y}, s_1) \tag{5}$$

Remark 1. To avoid the dependencies on the states s_1 that require the computation of probabilistic transitions over all individual states, we may use $P(\hat{y}|y)$, $P(y^*|\hat{y})$, and $P(y'|\hat{y})$, instead of $P(\hat{y}|y, s_1)$, $P(y^*|\hat{y}, s_1)$, and $P(y'|\hat{y}, s_1)$, in the above expressions. This way, we approximate the local conditional probabilities over states with the global conditional probabilities.

3.4. Illustration examples

This section presents several small examples to illustrate the transition systems defined in the previous sections. We consider an autonomous vehicle and use $s_1 = (\text{running}, \mathbf{x})$ and $s_2 = (\text{stopped}, \mathbf{x})$ to denote the vehicle's driving state and the scene captured by

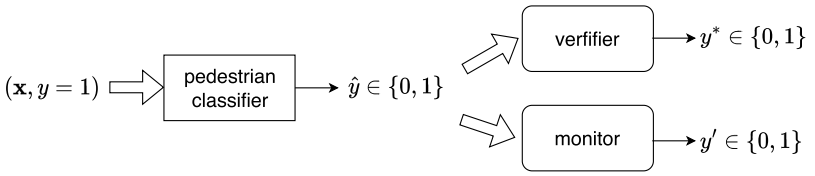


Fig. 2. The scheme of pedestrian classifier protected by a statistical verifier and a runtime monitor.

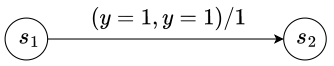


Fig. 3. The probabilistic transitions between s_1 and s_2 are defined in the case of a perfect pedestrian classifier.

Table 1
Confusion matrix for the pedestrian classifier.

	$\hat{y}=0$	$\hat{y}=1$
y=0	0.97	0.03
y=1	0.02	0.98

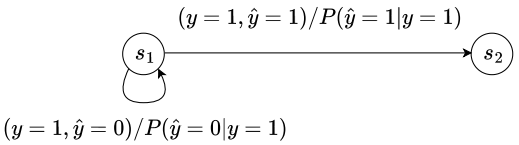


Fig. 4. The probabilistic transitions between s_1 and s_2 are defined in the case of an imperfect pedestrian classifier.

its camera (we assume the camera is perfect here, i.e., without hardware error). The captured scene $\mathbf{x}$ is the vehicle's perception components input. Furthermore, we use ground truth $y = 1$ ($y = 0$) to indicate a pedestrian is captured (not captured).

Specifically, we consider a pedestrian classifier, one of the perception components, which can produce a predictive label $\hat{y} \in \{0, 1\}$ for the input $\mathbf{x}$, where 1 and 0 indicate whether or not there exists a pedestrian. In some cases, the prediction may be protected by a statistical verifier and a runtime monitor, which can enforce the prediction into robust labels y^* and y'. The diagram of this protection scheme is shown in Fig. 2.

For simplicity and without loss of generality, we make the following assumptions: i) the vehicle is running, and a pedestrian is captured, i.e., the vehicle is in state s_1 with the ground truth $y = 1$ for the classifier's input $\mathbf{x}$; ii) the output from the perception module (classifier, verifier, or monitor) will be rightly executed by the controller as follows: if a pedestrian is detected, i.e., the output is 1; the controller will stop the running vehicle; iii) $P(s_i, (\hat{y}, \hat{y}), s_j) = \hat{c}$, $P(s_i, (y^*, y^*), s_j) = c^*$, and $P(s_i, (y', y'), s_j) = c'$, where $\hat{c} = c^* = c' = 1$;

Example 1. Consider the pedestrian classifier as perfect. The classifier's output $\hat{y}$ is always consistent with the ground truth. Thus, in state s_1, there exists only one possibility of state transition, given the existence of pedestrians, that is, a transition from s_1 to s_2 led by an action $(y = 1, \hat{y} = 1)$ with probability 1. This transition is graphically represented in Fig. 3. In Fig. 3 and the following examples, we use the notation a/p on the transition between two states s_1 and s_2 to denote the execution of action a with probability p to transit from s_1 to s_2.

However, in practice, perception components could be imperfect.

Example 2. Let us consider the case where the pedestrian classifier is imperfect and not protected by either a verifier or a monitor. The output $\hat{y}$ can take any value in $\{0, 1\}$ with a probability defined in the classifier's predictive confusion matrix shown in Table 1.

Thus, in state s_1, there exist two possibilities of state transition, given the existence of pedestrians. First, if the controller gets the output $\hat{y} = 1$, the action $(y = 1, \hat{y} = 1)$ is taken and leads a transition from s_1 to s_2 with probability $P(\hat{y} = 1|y = 1) = 0.98$. We note that, the transition probability $P(s_1, (y = 1, \hat{y} = 1), s_2) = P(\hat{y} = 1|y = 1)$, according to Equation (3) and the assumption that $P(s_i, (\hat{y}, \hat{y}), s_j) = 1$. Second, if the output is $\hat{y} = 0$, the action $(y = 1, y \hat{=} 0)$ makes a self-loop in s_1 with probability $P(\hat{y} = 0|y = 1) = 0.02$. These transitions are graphically shown in Fig. 4.

Now we consider the cases where a statistical verifier or a runtime monitor protects the imperfect classifier.

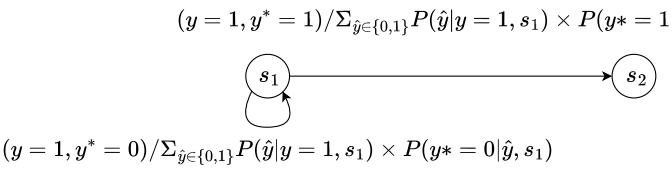


Fig. 5. The probabilistic transitions between s_1 and s_2 are defined in the case of the pedestrian classifier protected by a statistical verifier.

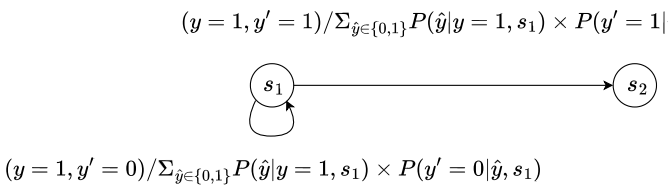


Fig. 6. The probabilistic transitions between s_1 and s_2 are defined in the case of the pedestrian classifier protected by a runtime monitor.

Example 3. Using a protected statistical verifier defines two state transitions for state s_1, similar to the direct use of an imperfect classifier, except the calculation of transition probabilities defined by e.g., (4). The graphical transitions defined in this can be found in Fig. 5. Similarly, we note that, the transition probability $P(s_1, (y = 1, y^* = 1), s_2) = \sum_{\hat{y}\in\{0,1\}} P(\hat{y}|y = 1, s_1) \times P(y^* = 1|\hat{y}, s_1)$, according to Equation (4) and the assumption that $P(s_i, (y^*, y^*), s_j) = 1$.

As for the monitoring case, for simplicity, we present the graphical illustration by Fig. 6 without a detailed explanation since it can be easily understood by replacing the part of the verifier with the monitor.

4. Specification language

We introduce a specification language MLS^2, abbreviated for Machine Learning Safety and Security, which will be used in the next section to specify a number of known properties.

Definition 1. The syntax of the language MLS^2 is

$$
\begin{array}{l}
\mu := P(W|\mathbf{d}) \mid P(Y|\mathbf{x}, \mathbf{w}) \mid P(\hat{Y}|\mathbf{d}, \mathbf{w}) \mid \mathbf{y} \\
u := (\mathbf{x}, y) \sim D \mid \mathbf{w} \sim P(W|\mathbf{d}) \\
v := y \in C \mid (\mathbf{x}, y) \in \mathbf{d} \mid \mathbf{x} \in Mut(\mathbf{x}) \\
\gamma := c \mid \mu(c) \mid D_{KL}(\mu, \mu) \mid ||\mu - \mu||_p \mid \mathbb{E}_u(\gamma) \mid \mathbb{V}_u(\gamma) \mid \mathbb{E}_v(\gamma) \mid \mathbb{V}_v(\gamma) \mid \gamma + \gamma \mid \gamma - \gamma \mid |\gamma| \\
\phi := \gamma \leq c \mid \mathbf{d} \subseteq \mathbf{d} \mid \neg\phi \mid \phi \vee \phi \mid \exists \mathbf{t} : \phi \\
\varphi := \phi \mid \neg\varphi \mid \varphi \vee \varphi \mid \bigcirc \varphi \mid \varphi \mathbf{U} \varphi
\end{array}
\tag{6}
$$

where the **bold** lower capital letters $\mathbf{x}$, $\mathbf{w}$, $\mathbf{d}$, and $\mathbf{y}$ denote an input, a weight matrix, a set of data instances, and a probability distribution, respectively. We use $\mathbf{t}$ to range over $\{\mathbf{x}, \mathbf{d}, \mathbf{y}\}$. Capital letters W and Y denote random variables for the weight matrix and the label, respectively. Moreover, we use c to express either a constant value or some concrete value of a random variable. We use variable u to range over the support of distribution (i.e., $(\mathbf{x}, y) \sim D$ or $\mathbf{w} \sim P(W|\mathbf{d})$), and use variable v to range over the instances in a known set (i.e., $(\mathbf{x}, y) \in \mathbf{d}$ or $y \in C$ or $\mathbf{x} \in Mut(\mathbf{x})$), where Mut is a set of mutations that maps an input instance $\mathbf{x}$ into a set of new input instances.

Intuitively, $P(W|\mathbf{d})$ expresses the posterior distribution of the models parameterised with W, when they are trained on a dataset $\mathbf{d}$. The expression $P(Y|\mathbf{x}, \mathbf{w})$ is the predictive distribution when the model is parameterised with known $\mathbf{w}$. The input instance is $\mathbf{x}$, and $P(\hat{Y}|\mathbf{d}, \mathbf{w})$ denotes the probability distribution of predictive labels $\hat{Y}$ over a set of data $\mathbf{d}$. These three distributions serve as the most fundamental elements of the properties. We can also have the distributions from the labels' one-hot encoding $\mathbf{y}$.

The formulas γ return real values. Specifically, $\mu(c)$ is the probability density of the distribution μ on c. For example, we can write $P(Y|\mathbf{x}, \mathbf{w})(y)$ for some $y \in C$ to denote the probability of predicting $\mathbf{x}$ as label y when the model is parameterised with $\mathbf{w}$. The formulas $D_{KL}(\mu, \mu)$ and $||\mu - \mu||_p$, for $p \geq 0$, express the distance between two distributions with the KL divergence and the norm distance, respectively. There are different measurements to measure the distance between two distributions. In this paper, we take KL divergence as an example and believe the formalism is generic and can be extended to other measurements. The formulas $\mathbb{E}_u(\gamma)$ and $\mathbb{V}_u(\gamma)$ return the mean and the variance, respectively, of γ. We will use the same notations, $\mathbb{E}_{(\mathbf{x},y)\in\mathbf{d}}(\gamma)$ and $\mathbb{V}_{(\mathbf{x},y)\in\mathbf{d}}(\gamma)$, when considering a dataset $\mathbf{d}$. Moreover, we may use the linear combination of γ, i.e., $\gamma + \gamma$ and $\gamma - \gamma$, and the absolute value $|\gamma|$.

Once having γ, the formula ϕ can be formed by asserting relational relations between γ and a constant c, then composing a Boolean formula with Boolean operators. In the definition, we only write the relational operation $\leq$ and a minimum set of Boolean operators, i.e., $\{\neg, \vee\}$ and it is straightforward to extend them to other relational operators $\{<, \geq, >\}$ and other Boolean operators $\{\wedge, \Rightarrow, \Longleftrightarrow\}$. Similarly, we only write the existence quantifier $\exists$ in the definition but will use $\forall$ in the following.

Moreover, we use the usual linear temporal logic (LTL) operators $\mathbf{U}$ and $\bigcirc$ to consider the dynamic evolution of the system. We will also freely use other LTL operators, such as $\Box$ and $\Diamond$, which can be expressed with $\mathbf{U}$ and Boolean operators.

The semantics of the language can be obtained by first having the standard semantics for the language constructs and then following the syntax.

- $s \vDash P(W|\mathbf{d})(\mathbf{w}) \leq c$ iff the probability of a given weight $\mathbf{w}$ on the posterior distribution $P(W|\mathbf{d})$ is no greater than a constant c.
- $s \vDash P(Y|\mathbf{x},\mathbf{w})(y) \leq c$ iff, given a neural network with weight $\mathbf{w}$, the probability of predicting the input $\mathbf{x}$ as y is no greater than a constant c.
- $s \vDash P(\hat{Y}|\mathbf{d},\mathbf{w})(c)$ iff, given a neural network with weight $\mathbf{w}$, the probability of a label y over a set of data $\mathbf{d}$ is no greater than a constant c.
- $s \vDash \mathbf{y}(y) \leq c$ iff the probability of a label y in a distribution (i.e., one-hot encoding) $\mathbf{y}$ is no greater than a constant c.
- $s \vDash D_{KL}(\mu_1,\mu_2) \leq c$ iff the KL divergence between two distributions μ_1 and μ_2 is no greater than a constant c.
- $s \vDash ||\mu_1 - \mu_2||_p \leq c$ iff the p-norm distance between two distributions μ_1 and μ_2 is no greater than a constant c.
- $s \vDash \mathbb{E}_{(\mathbf{x},y)\sim D} P(Y|\mathbf{x},\mathbf{w})(y) \leq c$ iff, when given a neural network with weight $\mathbf{w}$, the mean of predictive probability on a given label y, over a data distribution D, is no greater than a constant c.
- $s \vDash \mathbb{E}_{\mathbf{w}\sim P(W|\mathbf{d})} P(Y|\mathbf{x},\mathbf{w})(y) \leq c$ iff, on a given input $\mathbf{x}$, the mean of predictive probability on a given label y, over a posterior distribution $P(W|\mathbf{d})$ of weight $\mathbf{w}$ on a given dataset $\mathbf{d}$, is no greater than a constant c.
- $s \vDash \mathbb{E}_{(\mathbf{x},y)\in\mathbf{d}} P(Y|\mathbf{x},\mathbf{w})(y) \leq c$ iff, when given a neural network with weight $\mathbf{w}$, the mean of predictive probability on a given label y, over a known dataset $\mathbf{d}$, is no greater than a constant c.
- $s \vDash \mathbb{E}_{\mathbf{x}\in Mut(\mathbf{x}))} P(Y|\mathbf{x},\mathbf{w})(y)) \leq c$ iff, when given a neural network with weight $\mathbf{w}$, the mean of predictive probability on a given label y, over a known set of mutations Mut on an input $\mathbf{x}$, is no greater than a constant c.
- We omit other operators for their standard semantics, including the mathematical operator $+$ and $-$, the absolute value operator $||$, the relational operators $\subseteq$, and logic operators such as $\neg, \vee, \exists, \neg, \bigcirc, \mathbf{U}$.

Note that, the computation of $P(W|\mathbf{d})$ will be explained in Section 6.1 and 6.2, and the computation of $P(\hat{Y}|\mathbf{d},\mathbf{w})$ will be explained in Section 6.3. $P(Y|\mathbf{x},\mathbf{w})(y)$ is the prediction probability that can be obtained directly by reading the output of the classifier, and $\mathbf{y}$ is a known distribution. The KL divergence and norm distance computation are discussed in Section 6.5. The computation of mean and variance is discussed in Section 6.4. The semantics above can be easily extended to deal with formulas such as $\mathbb{E}_{(\mathbf{x},y)\sim D}(||P(Y|\mathbf{x},\mathbf{w}) - \mathbf{y}||_2)$ and $\mathbb{E}_{(\mathbf{x},y)\in\mathbf{d}}(1 - P(Y|\mathbf{x},\mathbf{w})(y))$. The computation of $\exists$ and $\forall$ is explained in Section 6.6. Finally, the diffusion model in Section 6.7 can be utilised to resolve the quantifiers $\exists$ and $\forall$ and the posterior distribution $P(W|\mathbf{d})$.

5. Safety, security, and privacy properties

This section formalises several safety, security, and privacy properties with the specification language described in the previous section.

5.1. Generalisation

Generalisation concerns the performance of the machine learning model on unseen data (or on the underlying data distribution). The following formula ϕ^1_{gen} expresses that the expected loss, measured over the difference between the prediction $P(Y|\mathbf{x},\mathbf{w})$ and the ground truth y on the data distribution D, is lower than a pre-specified threshold. Formally,

$$\phi^1_{gen}(\mathbf{w}) \triangleq \mathbb{E}_{(\mathbf{x},y)\sim D}(||P(Y|\mathbf{x},\mathbf{w}) - \mathbf{y}||_2) < \epsilon^1_{gen} \tag{7}$$

where $\mathbf{y}$ is the one-hot encoding of the label y and $\epsilon^1_{gen} > 0$ is a small constant. Note that $\mathbf{w}$ is the only free variable of the above expression, i.e., the evaluation of $||P(Y|\mathbf{x},\mathbf{w}) - \mathbf{y}||_2$ can be done over a state where there is an instance $(\mathbf{x}, y)$. Beyond that, for the entire formula ϕ^1_{gen}, Section 6.4 explains how it can be evaluated after estimating the data distribution D (as in Section 6.3).

Moreover, generalisation is a concept close to overfitting, which suggests that a model may perform (much) better on the training data than on the test data (i.e., the data distribution). We can specify this view – generalisation gap – with the following formula

$$\phi^2_{gen}(\mathbf{w}) \triangleq |\mathbb{E}_{(\mathbf{x},y)\sim D}(||P(Y|\mathbf{x},\mathbf{w}) - \mathbf{y}||_2) - \mathbb{E}_{(\mathbf{x},y)\in\mathbf{d}_{train}}(||P(Y|\mathbf{x},\mathbf{w}) - \mathbf{y}||_2)| < \epsilon^2_{gen} \tag{8}$$

where $\mathbf{d}_{train}$ is the set of training data and $\epsilon^2_{gen} > 0$ is a pre-specified small constant. Intuitively, ϕ^2_{gen} requires that the gap between the performance on the data distribution, i.e., $\mathbb{E}_{(\mathbf{x},y)\sim D}(||P(Y|\mathbf{x},\mathbf{w}) - \mathbf{y}||_2)$, and the performance on the training dataset, i.e., $\mathbb{E}_{(\mathbf{x},y)\in\mathbf{d}_{train}}(||P(Y|\mathbf{x},\mathbf{w}) - \mathbf{y}||_2)$, is bounded and insignificant. Similarly, $\mathbf{w}$ is the only free variable of the above expression, and the formula ϕ^2_{gen} can be evaluated over a state.

Finally, since a model with good generalisation ability may refer to either of the above cases, we use the following formula

$$\phi_{gen}(\mathbf{w}) \triangleq \Box(training \Rightarrow \phi^1_{gen}(\mathbf{w}) \vee \phi^2_{gen}(\mathbf{w})) \tag{9}$$

to ensure that the perception component has a good generalisation ability whenever it is trained.

5.2. Uncertainty

The ability to output a prediction and confidence is critical to downstream tasks. While a neural network may output a prediction, it is desirable to understand its confidence in making such a prediction. For example, when the neural network is for a perception task, a planning module may consider confidence (to understand if the result from the perception module is trustable) when determining the actions to be taken in the following steps.

Assume we have a model with weight matrix $\mathbf{w}$ and an input $\mathbf{x}$. First of all, we may ascertain the data uncertainty through the predictive probability, i.e.,

$$\phi^1_{unc}(\mathbf{w}, \mathbf{x}) \triangleq \mathbb{V}_{y \in C}(P(Y|\mathbf{x}, \mathbf{w})(y)) > \epsilon^1_{unc} \tag{10}$$

which requires the variance of the output probability to be greater than a positive constant ϵ^1_{unc}. We note that, a small variance may suggest that the predictive probability values of different classes are close to each other [44,45]. Therefore, a greater variance can represent a lower uncertainty in this case. Note that the formula ϕ^1_{unc} includes two free variables $\mathbf{w}$ and $\mathbf{x}$, which suggests that it can be evaluated on a state by providing an input $\mathbf{x}$.

Moreover, we may work with the total uncertainty, i.e., the data and the model uncertainties. In this case, the following formula may be considered:

$$\phi^2_{unc}(\mathbf{x}) \triangleq \mathbb{V}_{\mathbf{w} \sim P(W|\mathbf{d})}(P(Y|\mathbf{x}, \mathbf{w})(\hat{y})) < \epsilon^2_{unc} \tag{11}$$

where $\hat{y}$ is the predicted label of $\mathbf{x}$ with the model parameterised with $\mathbf{w}$. Intuitively, it is to determine if the variance of the prediction on $\hat{y}$ over the posterior distribution is smaller than a positive constant ϵ^2_{unc}. Note that, after the estimation of posterior distribution $P(W|\mathbf{d})$ (details in Section 6.1 and Section 6.2), the formula can be evaluated on a state.

Finally, we may use the following formula:

$$\phi_{unc}(\mathbf{w}, \mathbf{x}) \triangleq \square(\mathit{inference} \Rightarrow \phi^1_{unc}(\mathbf{w}, \mathbf{x}) \wedge \phi^2_{unc}(\mathbf{x})) \tag{12}$$

to ensure that, whenever an inference occurs on instance $\mathbf{x}$, the uncertainty over the prediction of $\mathbf{x}$ on the model $\mathbf{w}$ is low.

5.3. Robustness

Robustness concerns whether or not a perturbation may lead to a drastic change in the output. Simply speaking, a model with weight matrix $\mathbf{w}$ is robust over an input $\mathbf{x}$ can be expressed as

$$\phi^1_{rob}(\mathbf{w}, \mathbf{x}) \triangleq \forall \mathbf{r} : ||\mathbf{r}||_2 \leq c \Rightarrow |P(Y|\mathbf{x} + \mathbf{r}, \mathbf{w})(\hat{y}) - P(Y|\mathbf{x}, \mathbf{w})(\hat{y})| \leq \epsilon_{rob} \tag{13}$$

where $\mathbf{r} \in \mathbb{R}^n$ denotes the perturbation. Intuitively, it says that as long as the perturbation is within a range, the predictive probability on $\hat{y}$ is bounded by ϵ_{rob}. Note that $||\mathbf{r}||_2$ is not in the syntax but is a syntax sugar for $||\mathbf{r} - \mathbf{0}||_2$, where $\mathbf{0}$ is an all-zero vector of the same shape with $\mathbf{r}$. Similar to the formulas for uncertainty, robustness concerns a certain prediction, and therefore there are two free variables, $\mathbf{w}$ and $\mathbf{x}$, in the formula ϕ^1_{rob}.

Finally, we may use the following formula:

$$\phi_{rob}(\mathbf{w}, \mathbf{x}) \triangleq \square(\mathit{inference} \Rightarrow \phi^1_{rob}(\mathbf{w}, \mathbf{x})) \tag{14}$$

to ensure that, whenever an inference occurs, the robustness over the prediction of $\mathbf{x}$ on the model $\mathbf{w}$ is satisfied.

5.4. Data poisoning

Data poisoning suggests that by adding a set $\mathbf{d}'$ of poisoning data to the training dataset $\mathbf{d}$, it can make the model predict a specific input $\mathbf{x}_{adv}$ as a target label y_{adv}. Formally, we can write the following formula

$$\phi^1_{poi}(\mathbf{d}, \mathbf{d}') \triangleq \forall y : \mathbb{E}_{\mathbf{w} \sim P(W|\mathbf{d} \cup \mathbf{d}')}(P(Y|\mathbf{x}_{adv}, \mathbf{w})(y_{adv})) \geq \mathbb{E}_{\mathbf{w} \sim P(W|\mathbf{d} \cup \mathbf{d}')}(P(Y|\mathbf{x}_{adv}, \mathbf{w})(y)) \tag{15}$$

where $\mathbb{E}_{\mathbf{w} \sim P(W|\mathbf{d} \cup \mathbf{d}')} P(Y|\mathbf{x}_{adv}, \mathbf{w})(y_{adv})$ returns the probability of labelling $\mathbf{x}_{adv}$ with y_{adv} after poisoning. Intuitively, $\phi^1_{poi}(\mathbf{d}, \mathbf{d}')$ suggests that y_{adv} is the predictive label. This expression is stronger than the usual definition of data poisoning as we utilise the Bayesian view and obtain the prediction for $\mathbf{x}_{adv}$ after considering the posterior distribution.

While the targeted classification is required, we may also need to ensure that the poisoning does not affect the performance of the training data, i.e.,

$$\phi^2_{poi}(\mathbf{d}, \mathbf{d}') \triangleq (\mathbb{E}_{\mathbf{w} \sim P(W|\mathbf{d} \cup \mathbf{d}')}(\mathbb{E}_{(\mathbf{x}, y) \in \mathbf{d}}(||P(Y|\mathbf{x}, \mathbf{w}) - \mathbf{y}||_2))) \leq \epsilon^2_{poi} \tag{16}$$

Intuitively, when weighted over the posterior distribution, the average loss on the training data is no greater than a positive constant ϵ^2_{poi}.

Now, to ensure resistance to the data poisoning attacks, we require the dataset maintained by the attacker, i.e., $\mathbf{d}_{adv}$, cannot successfully poison the usual training on the training dataset $\mathbf{d}_{train}$. Then, by defining the formula

$$\phi_{poi}(\mathbf{d}, \mathbf{d}') \triangleq \Box(training \Rightarrow \neg\phi^1_{poi}(\mathbf{d}, \mathbf{d}') \vee \neg\phi^2_{poi}(\mathbf{d}, \mathbf{d}')) \tag{17}$$

we concretise $\mathbf{d}$ and $\mathbf{d}'$ into $\mathbf{d}_{train}$ and $\mathbf{d}_{adv}$, respectively, and expect $\phi_{poi}(\mathbf{d}_{train}, \mathbf{d}_{adv})$ to be satisfiable on the system M, which expresses that, whenever training is executed, either of the above formulas fails.

5.5. Backdoor

A Backdoor attack is to determine the existence of a trigger $\mathbf{r}$, by which all inputs $\mathbf{x}$ will be classified as a specific label y_{adv}. Formally, we have the following formula

$$\phi^1_{bac}(\mathbf{w}) \triangleq \neg\exists\mathbf{r}\forall\mathbf{x}\forall y : P(Y|\mathbf{x}+\mathbf{r}, \mathbf{w})(y_{adv}) \geq P(Y|\mathbf{x}+\mathbf{r}, \mathbf{w})(y) \tag{18}$$

to express the resistance to the backdoor attack. We can also take the Bayesian view, and write

$$\phi^2_{bac}(\mathbf{d}) \triangleq \neg\exists\mathbf{r}\forall\mathbf{x}\forall y : \mathbb{E}_{\mathbf{w}\sim P(W|\mathbf{d})}(P(Y|\mathbf{x}+\mathbf{r}, \mathbf{w})(y_{adv})) \geq \mathbb{E}_{\mathbf{w}\sim P(W|\mathbf{d})}(P(Y|\mathbf{x}+\mathbf{r}, \mathbf{w})(y)) \tag{19}$$

Then, we may have the formula

$$\phi_{bac}(\mathbf{w}, \mathbf{d}_{train}) \triangleq \Box(training \Rightarrow \phi^1_{bac}(\mathbf{w}) \wedge \phi^2_{bac}(\mathbf{d}_{train})) \tag{20}$$

which expresses that both of the above backdoor definitions are satisfied when training a new learning component. It is also possible that the backdoor might be achieved through data poisoning, and we may write

$$\phi_{bac}(\mathbf{w}, \mathbf{d}_{train}, \mathbf{d}_{adv}) \triangleq \neg\Diamond(training \wedge \phi^2_{bac}(\mathbf{d}_{train}) \wedge \neg\phi^2_{bac}(\mathbf{d}_{train} \cup \mathbf{d}_{adv})) \tag{21}$$

which expresses that, there does not exist any time in the future that the model is resistant to the backdoor trigger if trained on the usual training dataset but is not resistant if trained on the poisoned dataset.

5.6. Model stealing

Model stealing is to reconstruct a model that is functionally equivalent to the original model. The reconstruction can be conducted by first querying a set of data instances and then training a surrogate model with the data instances. First of all, we define the following formula

$$\phi^1_{ste}(\mathbf{d}_1, \mathbf{d}_2) \triangleq D_{KL}(P(W|\mathbf{d}_1), P(W|\mathbf{d}_2)) < \epsilon^1_{ste} \tag{22}$$

to express that the two posterior distributions trained on two different datasets $\mathbf{d}_1$ and $\mathbf{d}_2$, i.e., $P(W|\mathbf{d}_1)$ and $P(W|\mathbf{d}_2)$, are similar. Second, the following formula expresses that the dataset $\mathbf{d}$ is classified well by the original model whose weight matrix is $\mathbf{w}$, i.e.,

$$\phi^2_{ste}(\mathbf{d}, \mathbf{w}) \triangleq \mathbb{E}_{(\mathbf{x},y)\in\mathbf{d}}(1 - P(Y|\mathbf{x}, \mathbf{w})(y)) < \epsilon^2_{ste} \tag{23}$$

where ϵ^2_{ste} is a small positive constant. We can use $\phi^2_{ste}(\mathbf{d}, \mathbf{w})$ to express that $\mathbf{d}$ is from the same distribution as the underlying data distribution.

Finally, the resistance to the model stealing is to make the following formula hold:

$$\phi_{ste}(\mathbf{d}_{train}, \mathbf{d}_{adv}, \mathbf{w}) = \Box(\neg\phi^2_{ste}(\mathbf{d}_{adv}, \mathbf{w}) \vee \neg\phi^1_{ste}(\mathbf{d}_{train}, \mathbf{d}_{adv})) \tag{24}$$

which expresses that either the adversarial dataset is not on the distribution or the reconstruction fails with high probability. That is, at any time when the attacker collects a set of adversarial data $\mathbf{d}_{adv}$ that are on the data distribution, training on $\mathbf{d}_{adv}$ has a significant difference from training on $\mathbf{d}_{train}$.

5.7. Membership inference

Membership inference is to determine if an instance $(\mathbf{x}_{adv}, y_{adv})$ is in the training dataset or not. Formally, we use

$$\phi^1_{mem}(\mathbf{x}, y, \mathbf{d}) \triangleq \mathbb{E}_{\mathbf{w}\sim P(W|\mathbf{d})}(P(Y|\mathbf{x}, \mathbf{w})(y)) > \epsilon^{1,e}_{mem} \wedge \mathbb{V}_{\mathbf{w}\sim P(W|\mathbf{d})}(P(Y|\mathbf{x}, \mathbf{w})(y)) < \epsilon^{1,v}_{mem} \tag{25}$$

to express that the data instance $(\mathbf{x}, y)$ is on the same distribution as $\mathbf{d}$. Intuitively, the formula $\mathbb{E}_{\mathbf{w}\sim P(W|\mathbf{d})}(P(Y|\mathbf{x}, \mathbf{w})(y)) > \epsilon^{1,e}_{mem}$ says that, once a model is trained with dataset $\mathbf{d}$, the expected predictive probability of labelling $\mathbf{x}$ with y is higher than $\epsilon^{1,e}_{mem}$. If the constant $\epsilon^{1,e}_{mem}$ is close to 1, the satisfiability of formula $\phi^1_{mem}(\mathbf{x}, y, \mathbf{d})$ suggests that the instance $(\mathbf{x}, y)$ is on the same distribution with $\mathbf{d}$. The other expression $\mathbb{V}_{\mathbf{w}\sim P(W|\mathbf{d})}(P(Y|\mathbf{x}, \mathbf{w})(y)) < \epsilon^{1,v}_{mem}$ imposes a stronger condition that the variance needs to be small when the positive constant $\epsilon^{1,v}_{mem}$ is close to 0.

Based on the above, we have the following formula

$$\begin{aligned}\phi^2_{mem}(\mathbf{d}_{adv}) \triangleq \forall\mathbf{d}_1\forall\mathbf{d}_2 : \quad & (\mathbf{d}_1 \subseteq \mathbf{d}_{adv} \wedge \mathbf{d}_2 \subseteq \mathbf{d}_{adv} \wedge \phi^2_{ste}(\mathbf{d}_1) \wedge \phi^2_{ste}(\mathbf{d}_2)) \Rightarrow \\ & (\phi^1_{mem}(\mathbf{x}_{adv}, y_{adv}, \mathbf{d}_1) \iff \phi^1_{mem}(\mathbf{x}_{adv}, y_{adv}, \mathbf{d}_2))\end{aligned} \tag{26}$$

which expresses that for any two datasets $\mathbf{d}_1$ and $\mathbf{d}_2$ that are both subsets of $\mathbf{d}_{adv}$, if they are both on the underlying data distribution, the decision on whether the sample $(\mathbf{x}_{adv}, y_{adv})$ is in the training data is unambiguous.

Finally, the resistance to the membership inference is expressed with the following formula:

$$\phi_{mem}(\mathbf{d}_{adv}) \triangleq \Box \phi^2_{mem}(\mathbf{d}_{adv}) \tag{27}$$

which suggests that the attacker can never utilise the collected data $\mathbf{d}_{adv}$ to achieve the membership inference successfully.

5.8. Model inversion

During inference, a model inversion attack infers sensitive information about an instance. Without loss of generality, we assume that each data instance includes m features $X_1, ..., X_m$, and X_1 is the sensitive feature to be inferred. Then, given partial information about a data instance $\mathbf{x}$ (e.g., values of features $X_2, ..., X_n$), and its predictive label $\hat{y}$ by a machine learning model f, it is to infer the value for sensitive feature X_1.

We write x_i as the i-th element of $\mathbf{x}$. The following formula expresses that the input $\mathbf{x}$, with the feature X_1 being x_1, is on the data distribution, i.e.,

$$\phi^1_{inv}(\mathbf{x}[X_1 \leftarrow x_1], \mathbf{d}) \triangleq \exists y : \phi^1_{mem}(\mathbf{x}[X_1 \leftarrow x_1], y, \mathbf{d}) \tag{28}$$

Note that, we quantify away the label y because any label y satisfying $\phi^1_{mem}(\mathbf{x}[X_1 \leftarrow x_1], y, \mathbf{d})$ will be sufficient.

Now, assume that we have two datasets $\mathbf{d}_1$ and $\mathbf{d}_2$, and two input instances $\mathbf{x}$ and $\mathbf{x}'$ whose only difference is on the feature X_1, we define

$$\phi^2_{inv} \triangleq D_{KL}(P(W|\mathbf{d}_1), P(W|\mathbf{d}_2)) < \epsilon^{2,l}_{inv} \Rightarrow |x_1 - x'_1| < \epsilon^{2,r}_{inv} \tag{29}$$

which expresses that, as long as the posterior distributions are close to each other, the values of the sensitive feature X_1 are also close to each other.

Finally, the resistance to the model inversion is to find the smallest number k such that the following formula $\phi_{inv}(k)$ holds:

$$\begin{aligned}\phi^3_{inv}(\mathbf{d}_{adv}) \triangleq \forall \mathbf{d}_1 \forall \mathbf{d}_2 : ((\mathbf{d}_1 \subseteq \mathbf{d}_{adv} \wedge \mathbf{d}_2 \subseteq \mathbf{d}_{adv} \wedge \phi^2_{ste}(\mathbf{d}_1) \wedge \phi^2_{ste}(\mathbf{d}_2)) \Rightarrow \\ \forall x_1 \forall x_2 : ((\phi^1_{inv}(\mathbf{x}[X_1 \leftarrow x_1], \mathbf{d}_1) \wedge \phi^1_{inv}(\mathbf{x}[X_1 \leftarrow x'_1], \mathbf{d}_2)) \Rightarrow \phi^2_{inv}))\end{aligned} \tag{30}$$

Intuitively, the second line of Equation (30) says that, as long as $\mathbf{x}[X_1 \leftarrow x_1]$ is on the same distribution with $\mathbf{d}_1$, and $\mathbf{x}[X_1 \leftarrow x'_1]$ is on the same distribution with $\mathbf{d}_2$, the similarity of two posterior distributions (i.e., $D_{KL}(P(W|\mathbf{d}_1), P(W|\mathbf{d}_2)) < \epsilon^{2,l}_{inv}$) will lead to the requirement that the two values of feature X_1, i.e., x_1 and x'_1, are very close. In other words, obtaining any dataset of size k will lead to an unambiguous inference of the feature X_1.

Finally, the resistance to the model inversion attack is to determine the largest k such that the following formula holds:

$$\phi_{inv}(\mathbf{d}_{adv}) \triangleq \Box \phi^3_{inv}(\mathbf{d}_{adv}) \tag{31}$$

to make sure that the model inversion attack can never succeed.

5.9. Interpretability

There are many different definitions of interpretability. Here, we follow a popular definition that maps an n-dimensional instance $\mathbf{x}$ into a weighted map $exp(\mathbf{x}) : \{1, ..., n\} \rightarrow [0, 1]$ on the input features. The weighted map $exp(\mathbf{x})$ can then be displayed as a saliency map, as the explanation of the decision made by the machine learning model on $\mathbf{x}$. The weighted map can be normalised into a probability distribution, and we assume so in the following discussion.

First, we may require as a criterion of a good explanation that the output of the neural network does not have a major change (expressed as the L_∞ norm distance less than a constant ϵ^1_{int}) when masking less important input features according to the weighted map $\mathbf{x}$. Formally, we let

$$\phi^1_{int}(\mathbf{x}) \triangleq \mathbb{V}_{\mathbf{x}' \in Mut_{exp(\mathbf{x})}(\mathbf{x})}(||P(Y|\mathbf{x}', \mathbf{w}) - \mathbf{y}||_\infty) < \epsilon^1_{int} \tag{32}$$

where $Mut_{exp(\mathbf{x})}(\mathbf{x})$ is a set of mutations that mask less important input features from $\mathbf{x}$ according to the weighted map $exp(\mathbf{x})$. We may require the masking of important features to lead to a significant change in the predictive output, and we omit the details for the space limitation.

Beyond the correctness criterion as set up in Equation (32), there is also research on requiring the robustness of explanations, i.e.,

$$\phi^2_{int}(\mathbf{x}) \triangleq \forall \mathbf{r} : ||\mathbf{r}||_2 \leq c \Rightarrow D_{KL}(exp(\mathbf{x} + \mathbf{r}), exp(\mathbf{x})) < \epsilon^2_{int} \tag{33}$$

which states that the explanation, expressed as a probability distribution $exp(\mathbf{x})$, does not change significantly when subject to perturbations.

Finally, a specification for the interpretability of an instance $\mathbf{x}$ by a weighted map $exp(\mathbf{x})$ can be expressed as follows:

$$\phi_{int}(\mathbf{x}) \triangleq \Box(inference \Rightarrow \phi^1_{int}(\mathbf{x}) \wedge \phi^2_{int}(\mathbf{x})) \tag{34}$$

5.10. Fairness

Fairness concerns whether certain sensitive features may have a causality relation with the decision-making. Without loss of generality, we assume X_1 is the sensitive feature. Then, the fairness can somewhat be re-stated as that the predictive distribution of letting the sensitive feature X_1 have the value x_1 is the same as the predictive distribution of letting the sensitive feature X_1 have the value x_1', i.e.,

$$\phi^1_{fai}(\mathbf{w}, \mathbf{x}) \triangleq \forall x_1 \forall x_1' : D_{KL}(P(\hat{Y}|\mathbf{x}[X_1 \leftarrow x_1], \mathbf{w}), P(\hat{Y}|\mathbf{x}[X_1 \leftarrow x_1'], \mathbf{w})) < \epsilon_{fai} \tag{35}$$

for ϵ_{fai} a small constant. Intuitively, this means that any value of X_1 does not make a noticeable difference. Similar as the previous properties, we may write

$$\phi_{fai}(\mathbf{w}, \mathbf{x}) \triangleq \square(training \Rightarrow \phi^1_{fai}(\mathbf{w}, \mathbf{x})) \tag{36}$$

to ensure that the fairness can be preserved whenever a new perception component is trained.

6. Verification of properties

From the language MLS^2, we can see that most of the constructs can be easily evaluated. For example, $P(Y|\mathbf{x}, \mathbf{w})$ can be obtained by simply querying the model of a wight matrix $\mathbf{w}$ with the input $\mathbf{x}$, and $\mathbb{E}_v(\gamma)$ and $\mathbb{V}_v(\gamma)$ can be obtained by enumerating over all elements in the finite set if we know how to evaluate γ. Nevertheless, a few other constructs might require significant computational effort to evaluate, which we will discuss below.

6.1. Estimation of posterior distribution $P(W|\mathbf{d})$ through MCMC

MCMC is a family of algorithms to sample a probability distribution usually defined in a high-dimensional space. These algorithms perform Monte Carlo estimates by constructing a Markov chain with the desired distribution as its stationary distribution. The more samples we draw, the more closely the distribution of the samples matches the desired distribution.

Given an unknown distribution $P(W|\mathbf{d})$, different MCMC algorithms will construct Markov chains with different probability transition matrices. In the following, we briefly describe the Metropolis-Hastings algorithm, the most common MCMC algorithm, and the Markov chain constructed by this algorithm.

A Markov chain is a tuple $M(W|\mathbf{d}) = (S, \mathbf{w}_0, T, L)$, where S is a set of states, $\mathbf{w}_0 \in S$ is an initial state, $T : S \times S \to [0, 1]$ is a probability transition matrix, and L is a labelling function. The construction of $M(W|\mathbf{d})$ proceeds by first sampling $\mathbf{w}_0$ from $P(W|\mathbf{d})$ as the initial state, and then gradually adding states $\mathbf{w}_{n+1}$ to S and updating the transition matrix T until it converges. Let H be a transition matrix for any irreducible Markov chain whose state space supports of $P(W|\mathbf{d})$. Suppose the last sample we draw is $\mathbf{w}_n$. We generate a new sample $\mathbf{w}_{n+1}$ as follows:

1. Choose a proposal state $\mathbf{w}'$ according to the probability distribution given by $H(\mathbf{w}'|\mathbf{w}_n)$.
2. Calculate the acceptance probability of the proposal $\mathbf{w}'$ as

$$A(\mathbf{w}'|\mathbf{w}_n) = \min\left(1, \frac{P(\mathbf{w}'|\mathbf{d})H(\mathbf{w}_n|\mathbf{w}')}{P(\mathbf{w}_n|\mathbf{d})H(\mathbf{w}'|\mathbf{w}_n)}\right)$$

3. Let $u \sim \text{Uniform}([0, 1])$. Accept the proposed value as the new sample if $u \leq A(\mathbf{w}'|\mathbf{w}_n)$, that is, let $\mathbf{w}_{n+1} = \mathbf{w}'$. Reject the proposed value otherwise, that is, let $\mathbf{w}_{n+1} = \mathbf{w}_n$.

The probability transition matrix T of the Markov chain constructed by the Metropolis-Hastings algorithm has the following probability transition matrix:

$$T(\mathbf{w}'|\mathbf{w}) = \begin{cases} H(\mathbf{w}'|\mathbf{w})A(\mathbf{w}'|\mathbf{w}) & \text{if } \mathbf{w} \neq \mathbf{w}' \\ 1 - \sum_{\mathbf{w} \neq \mathbf{w}'} H(\mathbf{w}'|\mathbf{w})A(\mathbf{w}'|\mathbf{w}) & \text{otherwise} \end{cases}$$

This Markov chain satisfies (1) the uniqueness of the stationary distribution and (2) the detailed balance convening the desired distribution $P(W|\mathbf{d})$, that is, $T(\mathbf{w}'|\mathbf{w})P(\mathbf{w}|\mathbf{d}) = T(\mathbf{w}|\mathbf{w}')P(\mathbf{w}'|\mathbf{d})$. These conditions guarantee that the constructed Markov chain has the desired distribution $P(W|\mathbf{d})$ as its stationary distribution. MCMC algorithms, such as simulated annealing, can converge to a global optimum.

6.2. Estimation of posterior distribution $P(W|\mathbf{d})$ through variational inference

MCMC can be computationally expensive when dealing with large dimensional problems. In this case, we may consider an alternative approach, i.e., variational inference (VI), which casts the computation of the distribution $P(W|\mathbf{d})$ as an optimisation problem. VI assumes a class of tractable distributions $\mathcal{Q}$ and intends to find a $q(W) \in \mathcal{Q}$ that is closest to $P(W|\mathbf{d})$. Once we have the distribution $q(W)$, we can use it for any computation that involves $P(W|\mathbf{d})$.

We use the KL divergence to measure the distance between $q(W)$ and $P(W|\mathbf{d})$. Formally, we have the following:

$$\begin{aligned} & D_{KL}(q(W)||P(W|\mathbf{d})) \\ &= \int q(W)\log\frac{q(W)}{P(W|\mathbf{d})}dW \\ &= \mathbb{E}_{q(W)}(\log\frac{q(W)}{P(W|\mathbf{d})}) \\ &= \mathbb{E}_{q(W)}(\log\frac{q(W)}{P(\mathbf{d}|W)P(W)}P(\mathbf{d})) \\ &= \mathbb{E}_{q(W)}(\log\frac{q(W)}{P(\mathbf{d}|W)P(W)}) + \log P(\mathbf{d}) \\ &= D_{KL}(q(W)||P(W)) - \mathbb{E}_{q(W)}(\log P(\mathbf{d}|W)) + \log P(\mathbf{d}) \end{aligned} \tag{37}$$

To minimise this, we can minimise the negative log evidence lower bound

$$\mathcal{L}_{VI} = D_{KL}(q(W)||P(W)) - \mathbb{E}_{q(W)}(\log P(\mathbf{d}|W)) \tag{38}$$

The expectation value $\mathbb{E}_{q(W)}(\log P(\mathbf{d}|W))$ can be approximated with Monte Carlo integration. Therefore, the optimisation

$$\hat{q}(\mathbf{W}) \triangleq \arg\min_{q(W)\in\mathcal{Q}} \mathcal{L}_{VI} \tag{39}$$

can be conducted by iteratively improving a candidate $q(W)$ until convergence. There are GO algorithms for VI, such as [46], which guarantees to converge to the ϵ-global variational lower bound on the log-likelihood.

The computational complexity of VI depends on the distribution class $\mathcal{Q}$. For example, suppose it is mean field approximation. In this case, the complexity is in polynomial time concerning the number of input features. If it is Gaussian process, the complexity is exponential for the number of input features.

6.3. Estimation of data distribution $\mathcal{D}$ and distribution of predictive labels $P(\hat{Y}|\boldsymbol{d},\boldsymbol{w})$

The estimation of data distribution $\mathcal{D}$ usually is based on a set of known data points $\{(\mathbf{x}_1, y_1), ..., (\mathbf{x}_n, y_n)\}$. This can be done through, e.g., Kernel density estimation. We have

$$\hat{D}(\mathbf{x}) = \frac{1}{nh}\sum_{i=1}^{n} K(\frac{\mathbf{x}-\mathbf{x}_i}{h}) \tag{40}$$

K is a non-negative function called the kernel, and $h > 0$ is a smoothing parameter called the bandwidth. The normal kernel is often used, where $K(x)$ is the standard normal density function. In this case, the obtained estimation $\hat{D}(\mathbf{x})$ is a multivariate Gaussian mixture model. GO methods for KDE, such as [47], can converge to a global optimum.

The distribution $P(\hat{Y}|\mathbf{d},\mathbf{w})$ of predictive labels can also be estimated in this way as having the known data points $\{\hat{y}|(\mathbf{x},y)\in\mathbf{d}\}$, where $\hat{y}$ is the predictive label of $\mathbf{x}$ over network parameterised with $\mathbf{w}$.

6.4. $\mathbb{E}_u(\gamma)$ or $\mathbb{V}_u(\gamma)$

Given u can be either $\mathbf{w}\sim P(W|\mathbf{d})$ or $(\mathbf{x},y)\sim\mathcal{D}$, and we have suggested in Sections 6.1, 6.2, and 6.3 a few methods to estimate distributions $P(W|\mathbf{d})$ and $\mathcal{D}$ with error bounds, the expressions $\mathbb{E}_u(\gamma)$ and $\mathbb{V}_u(\gamma)$ can be evaluated.

6.5. $D_{KL}(\mu,\mu)$ or $||\mu-\mu||_p$

A direct computation of the KL divergence or the norm distance of two unknown high-dimensional distributions, such as the posterior distributions, will be hard. However, we can apply VI to estimate two distributions $q_1(\mathbf{W})$ and $q_2(\mathbf{W})$, one for each of the distributions. Then, because $q_1(\mathbf{W})$ and $q_2(\mathbf{W})$ are known, we can compute KL divergence analytically. Alternatively, we can compute two Markov chains M_1 and M_2 with MCMC and then their distance.

6.6. $\exists\boldsymbol{t}$ and $\forall\boldsymbol{t}$

The quantifiers $\exists\mathbf{t}$ and $\forall\mathbf{t}$ will cause a significant increase in computational complexity when they are alternating and when $\mathbf{t}$ represents a high-dimensional variable for either an input $\mathbf{x}$ or a set of inputs $\mathbf{d}$. We note that the properties in the previous section require at most one alternating between $\exists$ and $\forall$.

In robustness verification, GO has been applied in, e.g., [35,14–16,36], but it is mainly for $\mathbf{t}$ to represent an input $\mathbf{x}$. For the cases where $\mathbf{t}$ represents the output y, for example, Equation (15), we can enumerate all possible values of y, as the class C is usually fixed and finite. For the cases where $\mathbf{t}$ represents a set $\mathbf{d}$ of inputs, we can apply similar techniques as in [35,14,36].

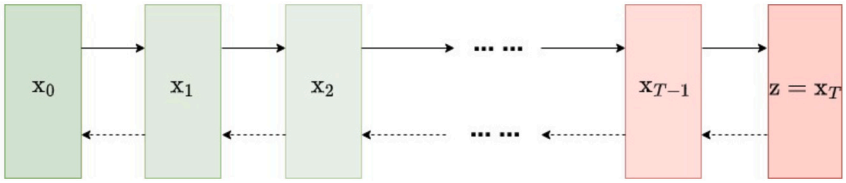


Fig. 7. Forward and reverse diffusion processes.

6.7. Diffusion model

The generative model in deep learning has shown great potential in many fields, especially the latest results from diffusion models, such as DALLE2 [48], GLIDE [49], IMAGEN [50]. Diffusion models are two-process models as shown in Fig. 7: a forward process (with noise added) and a reverse process (with noise removed). A certain amount of noise is added to the input in T steps through the forward process. The input data will become a standard normal distribution when T is large enough. After this, new inputs can be produced by sampling from the obtained normal distribution and denoising procedures. Typical diffusion models include *denoising diffusion probability models* (DDPM) [51] and *denoising score matching* [52,53]. Here, we introduce the former as an example.

Forward diffusion process Specifically, DDPM can be regarded as a T-step Markov chain. Let $\mathbf{x}_t$ denote the sample at time t. $\mathbf{x}_t$ is obtained from $\mathbf{x}_{t-1}$ by adding the following Gaussian noise:

$$q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t\mathbf{I})$$

where $\beta_t \in [0,1]$ is the variance and controls the step size (in practice, β_t is monotonic over t, i.e., $\beta_1 < \beta_2 < \cdots < \beta_T$). Thanks to some nice properties [51], the samples at any time can be expressed explicitly by the input as follows:

$$\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\epsilon$$

where $\bar{\alpha}_t$ is a constant equal to $\Pi_{i=1}^{i=t}(1-\beta_i)$ and $\epsilon \sim \mathcal{N}(0,\mathbf{I})$.

Reverse diffusion process Contrary to the forward process, the reverse process is a denoising process. It is proved [54] that as long as β_t is small enough, the reverse conditional probability $q(\mathbf{x}_{t-1}|\mathbf{x}_t)$ is also a Gaussian distribution. However, it is difficult to calculate and usually is transformed and simplified into a noise prediction problem, that is, the noise added at time t steps. This is achieved by training a neural network (parameterised by θ) by repeatedly taking inputs from the dataset and doing the following steps until it converges: i) first uniformly sample t from $[1,T]$; ii) second sample a value ϵ from $\mathcal{N}(0,\mathbf{I})$; then take gradient descent step on $\nabla_\theta||\epsilon - \epsilon_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\epsilon, t)||^2$.

After training, new samples can be generated as follows: i) first sample a noise $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0},\mathbf{I})$; ii) second, for $t \in [T,1]$ do $\epsilon \sim \mathcal{N}(\mathbf{0},\mathbf{I})$ if $t > 1$ else $\epsilon = 0$ and calculate $\mathbf{x}_{t-1} = \frac{1}{\sqrt{\bar{\alpha}_t}}(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon_\theta(\mathbf{x}_t,t)) + \sigma_t\epsilon$; iii) finally return $\mathbf{x}_0$.

In short, the core of this algorithm is to design the noise added at each step in the forward diffusion process and then train a network to predict this noise so that one can sample from the standard normal distribution and generate new samples through a series of noise-removing calculations.

Application of diffusion models in verification We discuss how to use the diffusion models to support the verification. The diffusion model is a generative model for x, where we can instantiate x by either a model, an input, or a set of inputs. The diffusion model will allow us to resolve the quantifiers ∃ and ∀. Suppose such a generative model can be obtained for an object of interest. In that case, some system properties can be statistically verified by proposing some hypothesis on the considered objects and then using the generative model to generate data for the statistical test. Consider a concrete example that verifies whether the generalisation of an object detection module in an autonomous driving system is sufficiently good for corner cases. To this end, a corner case dataset needs to be designed according to some requirements, such as being fair enough regarding the number and diversity of scenes. A possible solution is as follows: i) first, design such a generative model based on the limited existing data; 2) second, use this generative model as a simulator (sampling from the **z** latent space in Fig. 7) to perform hypothesis testing on the generalisation ability of the considered object detector model to corner cases; 3) finally, one can analyse the hypothesis testing results and conclude if the generalisation is good enough or not. In addition, it can also be an alternative technique for the posterior distribution $P(W|\mathbf{d})$.

6.8. Error bounds

MCMC and VI, as statistical methods, are subject to errors, unlike formal proofs that yield binary answers. Nevertheless, errors can be theoretically bounded. The accuracy measure and error bounds of MCMC have been well studied. See, for example, Chapter 12 of [55]. While the classic variational inference (VI) is usually criticised for its lack of theoretically justified post-hoc accuracy measures, some recent research shows that it is possible to have validated variational inference with mild assumptions and improved workflow. For example, [56] shows that rigorous bounds on the error can be obtained using their improved workflow.

For KDE, the discussion on its error bound can be found in textbooks such as [57].

7. Related works

We review some work that formalises the specification for learning-enabled autonomous systems. [58] formalises requirements for the runtime verification of an autonomous unmanned aircraft system based on an extension of propositional LTL, where temporal operators are augmented with timing constraints. It uses atomic propositions such as "$horizontalIntruderDistance > 250$" to express the result of the perception module without considering the sensory input and the possible failure of getting the exact value for the variable $horizontalIntruderDistance$. [41] introduces a specification language based on LTL, which utilises event-based abstraction to hide the details of the neural network structure and parameters. It starts considering the potential failure of the perception component and uses a predicate $pedestrian(\mathbf{x})$ to express if $\mathbf{x}$ is a pedestrian (referring to the ground truth). However, it does not consider the predicate's potential vulnerabilities, such as robustness and uncertainty. [59] proposes Timed Quality Temporal Logic (TQTL) to express monitorable [60] spatio-temporal quality properties of perception systems based on neural networks. It considers object detectors such as YOLO and uses for example $D_0 : d_1 : (ID, 1), (class, car), (pr, 0.9), (bb, B1)$ to denote an object d_1 in a frame D_0 such that it has an index 1, a predictive label car, the prediction probability 0.9, and is in a bounding box B_1. Therefore, every state may include multiple such expressions and then a TQTL formula can be written by referring to the components of the expressions in a state. Unlike previous attempts, our specification language for machine learning considers not only the functionality (i.e., the relation between input and output) of a trained model but also the training process (where objects such as training datasets, model parameters, and distance between posterior distributions are considered). With this, the language can express the safety and security properties that describe the attacks during the lifecycle stages.

We also mention the current efforts on the verification of deep learning. The algorithms can be roughly categorised into constraint-solving based methods [11], abstract interpretation based methods [31,32], global optimisation based methods [10,14,34], and game-based methods [35,36]. However, these works are focused on robustness, and it is unclear whether and how the techniques can be extended to consider other properties described with MLS^2.

8. Conclusions

This paper makes an attempt to use a formal specification language to describe ten different safety and security properties of machine learning models. The language can describe input-output relations and the relations between training data and trained models. To verify properties expressed with the language, we suggest global optimisation methods to deal with basic constructs such as posterior distributions. We hope this forms a new step toward the communication between formal methods and machine learning.

This new step opens many opportunities for the future exploration of this topic. The first is to figure out algorithms with lower computational complexity in theory. Two aspects that require attention, one on the estimation of the posterior distribution $P(W|\mathbf{d})$ and the other on the quantifier $\forall\mathbf{t}$. For $P(W|\mathbf{d})$, it is highly dimensional due to the sizes of $\mathbf{d}$ and W, which will render the estimation algorithms such as MCMC and VI become less accurate. For $\forall\mathbf{t}$, we need to consider all possible instantiations of $\mathbf{t}$, which can have a very significant number considering the high dimensionality of $\mathbf{t}$. Even if a theoretical improvement cannot be achieved, any practical method to reduce computational intensiveness would also help. The second is to figure out the error bounds for any specification formula of MLS^2. The discussion on the error bounds in Section 6.8 is only for the atomic propositions, and we still need to know how the error bounds propagate through the operators in MLS^2.

CRediT authorship contribution statement

Saddek Bensalem: Conceptualization, Formal analysis, Writing – review & editing. **Xiaowei Huang:** Conceptualization, Methodology, Writing – original draft, Writing – review & editing. **Wenjie Ruan:** Methodology, Writing – review & editing. **Qiyi Tang:** Formal analysis, Methodology, Writing – original draft, Writing – review & editing. **Changshun Wu:** Formal analysis, Methodology, Writing – original draft, Writing – review & editing. **Xingyu Zhao:** Formal analysis, Methodology, Writing – original draft, Writing – review & editing.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Xiaowei Huang reports financial support was provided by EU Framework Programme for Research and Innovation Euratom. Xiaowei Huang reports financial support was provided by Engineering and Physical Sciences Research Council. Saddek Bensalem reports financial support was provided by EU Framework Programme for Research and Innovation Euratom.

Acknowledgement

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 956123. Moreover, XH is also supported by the UK EPSRC under projects [EP/R026173/1, EP/T026995/1].

References

[1] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, R. Fergus, Intriguing properties of neural networks, in: ICLR, Citeseer, 2014.

[2] A. Demontis, M. Melis, M. Pintor, M. Jagielski, B. Biggio, A. Oprea, C. Nita-Rotaru, F. Roli, Why do adversarial attacks transfer? Explaining transferability of evasion and poisoning attacks, in: 28th USENIX Security Symposium (USENIX Security 19), USENIX Association, Santa Clara, CA, 2019, pp. 321–338, https://www.usenix.org/conference/usenixsecurity19/presentation/demontis.

[3] T. Orekondy, B. Schiele, M. Fritz, Knockoff nets: stealing functionality of black-box models, in: IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019, Computer Vision Foundation / IEEE, 2019, pp. 4954–4963.

[4] Z. Yang, J. Zhang, E.-C. Chang, Z. Liang, Neural network inversion in adversarial setting via background knowledge alignment, in: Proc. of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS '19, ACM, New York, NY, USA, 2019, pp. 225–240.

[5] X. Yin, W. Ruan, J. Fieldsend, Dimba: discretely masked black-box attack in single object tracking, Mach. Learn. (2022) 1–19.

[6] R. Mu, W. Ruan, L. Soriano Marcolino, Q. Ni, Sparse adversarial video attacks with spatial transformations, in: The 32nd British Machine Vision Conference (BMVC'21), 2021.

[7] Y. Zhang, W. Ruan, F. Wang, X. Huang, Generalizing universal adversarial attacks beyond additive perturbations, in: 2020 IEEE International Conference on Data Mining (ICDM'20), IEEE, 2020, pp. 1412–1417.

[8] X. Huang, D. Kroening, W. Ruan, J. Sharp, Y. Sun, E. Thamo, M. Wu, X. Yi, A survey of safety and trustworthiness of deep neural networks: verification, testing, adversarial attack and defence, and interpretability, Comput. Sci. Rev. 37 (2020) 100270, https://doi.org/10.1016/j.cosrev.2020.100270, http://www.sciencedirect.com/science/article/pii/S1574013719302527.

[9] Wikipedia: V-model, https://en.wikipedia.org/wiki/V-model#cite_note-FHWA_05-1.

[10] X. Huang, M. Kwiatkowska, S. Wang, M. Wu, Safety verification of deep neural networks, in: International Conference on Computer Aided Verification, Springer, 2017, pp. 3–29.

[11] G. Katz, C. Barrett, D.L. Dill, K. Julian, M.J. Kochenderfer, Reluplex: an efficient SMT solver for verifying deep neural networks, in: International Conference on Computer Aided Verification, Springer, 2017, pp. 97–117.

[12] D.R. Jones, C.D. Perttunen, B.E. Stuckman, Lipschitzian optimization without the Lipschitz constant, J. Optim. Theory Appl. 79 (1993) 157–181.

[13] D.R. Jones, J.R.R.A. Martins, The DIRECT algorithm: 25 years later, J. Glob. Optim. 79 (3) (2021) 521–566.

[14] W. Ruan, X. Huang, M. Kwiatkowska, Reachability analysis of deep neural networks with provable guarantees, in: IJCAI, 2018, pp. 2651–2659.

[15] F. Wang, P. Xu, W. Ruan, X. Huang, Towards verifying the geometric robustness of large-scale neural networks, in: Proceedings of the AAAI Conference on Artificial Intelligence (AAAI'23), 2023.

[16] C. Zhang, W. Ruan, P. Xu, Reachability analysis of neural network control systems, in: Proceedings of the AAAI Conference on Artificial Intelligence (AAAI'23), 2023.

[17] J. Musa, Operational profiles in software-reliability engineering, IEEE Softw. 10 (2) (1993) 14–32.

[18] X. Zhao, W. Huang, A. Banks, V. Cox, D. Flynn, S. Schewe, X. Huang, Assessing reliability of deep learning through robustness evaluation and operational testing, in: AISafety2021, 2021.

[19] Y. Dong, W. Huang, V. Bharti, V. Cox, A. Banks, S. Wang, X. Zhao, S. Schewe, X. Huang, Reliability assessment and safety arguments for machine learning components in system assurance, ACM Trans. Embed. Comput. Syst. (2022), https://doi.org/10.1145/3570918.

[20] K. Fukunaga, Introduction to Statistical Pattern Recognition, Elsevier, 2013.

[21] B. Littlewood, J. Rushby, Reasoning about the reliability of diverse two-channel systems in which one channel is "possibly perfect", IEEE Trans. Softw. Eng. 38 (5) (2012) 1178–1194.

[22] J. Rushby, Software verification and system assurance, in: 7th Int. Conf. on Software Engineering and Formal Methods, IEEE, Hanoi, Vietnam, 2009, pp. 3–10.

[23] S.S. Du, J.D. Lee, H. Li, L. Wang, X. Zhai, Gradient descent finds global minima of deep neural networks, arXiv e-prints, arXiv:1811.03804, 2018.

[24] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, A. Vladu, Towards deep learning models resistant to adversarial attacks, in: ICLR'18, 2018.

[25] G. Jin, X. Yi, L. Zhang, L. Zhang, S. Schewe, X. Huang, How does weight correlation affect the generalisation ability of deep neural networks, in: NeurIPS'20, 2020.

[26] G. Jin, X. Yi, W. Huang, S. Schewe, X. Huang, Enhancing adversarial training with second-order statistics of weights, in: CVPR2022, 2022.

[27] Y. Sun, M. Wu, W. Ruan, X. Huang, M. Kwiatkowska, D. Kroening, Concolic testing for deep neural networks, in: Automated Software Engineering (ASE), 33rd IEEE/ACM International Conference on, 2018.

[28] Y. Sun, X. Huang, D. Kroening, Testing deep neural networks, CoRR, arXiv:1803.04792 [abs], 2018.

[29] Y. Sun, M. Wu, W. Ruan, X. Huang, M. Kwiatkowska, D. Kroening, Deepconcolic: testing and debugging deep neural networks, in: 41st ACM/IEEE Int. Conf. on Software Engineering (ICSE'19), 2019.

[30] W. Huang, Y. Sun, X. Zhao, J. Sharp, W. Ruan, J. Meng, X. Huang, Coverage-guided testing for recurrent neural networks, IEEE Trans. Reliab. 71 (3) (2022) 1191–1206, https://doi.org/10.1109/TR.2021.3080664.

[31] T. Gehr, M. Mirman, D. Drachsler-Cohen, P. Tsankov, S. Chaudhuri, M. Vechev, AI2: safety and robustness certification of neural networks with abstract interpretation, in: Security and Privacy (SP), 2018 IEEE Symposium on, 2018.

[32] J. Li, J. Liu, P. Yang, L. Chen, X. Huang, L. Zhang, Analyzing deep neural networks with symbolic propagation: towards higher precision and faster verification, in: SAS2019, 2019, pp. 296–319.

[33] R. Mu, W. Ruan, L.S. Marcolino, Q. Ni, 3dverifier: efficient robustness verification for 3d point cloud models, Mach. Learn. (2022) 1–28.

[34] W. Ruan, M. Wu, Y. Sun, X. Huang, D. Kroening, M. Kwiatkowska, Global robustness evaluation of deep neural networks with provable guarantees for the Hamming distance, in: IJCAI2019, 2019, pp. 5944–5952.

[35] M. Wicker, X. Huang, M. Kwiatkowska, Feature-guided black-box safety testing of deep neural networks, in: International Conference on Tools and Algorithms for the Construction and Analysis of Systems, Springer, 2018, pp. 408–426.

[36] M. Wu, M. Wicker, W. Ruan, X. Huang, M. Kwiatkowska, A game-based approximate verification of deep neural networks with provable guarantees, Theor. Comput. Sci. (2020).

[37] P. Bishop, A. Povyakalo, Deriving a frequentist conservative confidence bound for probability of failure per demand for systems with different operational and test profiles, Reliab. Eng. Syst. Saf. 158 (2017) 246–253.

[38] R. Pietrantuono, P. Popov, S. Russo, Reliability assessment of service-based software under operational profile uncertainty, Reliab. Eng. Syst. Saf. 204 (2020) 107193.

[39] K. Salako, L. Strigini, X. Zhao, Conservative confidence bounds in safety, from generalised claims of improvement & statistical evidence, in: 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN'21, IEEE/IFIP, Taipei Taiwan, 2021, pp. 451–462.

[40] J. Anderson, M. Hekmatnejad, G. Fainekos, Pyforel: a domain-specific language for formal requirements in temporal logic, in: 2022 IEEE 30th International Requirements Engineering Conference (RE), 2022, pp. 266–267.

[41] S. Bensalem, C.-H. Cheng, X. Huang, P. Katsaros, A. Molin, D. Nickovic, D. Peled, Formal specification for learning-enabled autonomous systems, in: FoMLAS2022, 2022.

[42] C. Huang, Z. Hu, X. Huang, K. Pei, Statistical certification of acceptable robustness for neural networks, in: I. Farkaš, P. Masulli, S. Otte, S. Wermter (Eds.), Artificial Neural Networks and Machine Learning – ICANN 2021, Springer International Publishing, Cham, 2021, pp. 79–90.

[43] T. Zhang, W. Ruan, J.E. Fieldsend, Proa: a probabilistic robustness assessment against functional perturbations, in: Joint European Conference on Machine Learning and Knowledge Discovery in Databases (ECML/PKDD'22), 2022.

[44] P. Xu, W. Ruan, X. Huang, Towards the quantification of safety risks in deep neural networks, CoRR, arXiv:2009.06114 [abs], 2020, arXiv:2009.06114, https://arxiv.org/abs/2009.06114.

[45] P. Xu, W. Ruan, X. Huang, Quantifying safety risks of deep neural networks, Complex Intell. Syst. (2022).
[46] H. Saddiki, A.C. Trapp, P. Flaherty, A deterministic global optimization method for variational inference, https://doi.org/10.48550/ARXIV.1703.07169, 2017, https://arxiv.org/abs/1703.07169.
[47] O. Wirjadi, T. Breuel, A branch and bound algorithm for finding the modes in kernel density estimates, Int. J. Comput. Intell. Appl. 08 (01) (2009) 17–35, https://doi.org/10.1142/S1469026809002461, arXiv:https://doi.org/10.1142/S1469026809002461.
[48] A. Ramesh, P. Dhariwal, A. Nichol, C. Chu, M. Chen, Hierarchical text-conditional image generation with clip latents, arXiv preprint, arXiv:2204.06125, 2022.
[49] A. Nichol, P. Dhariwal, A. Ramesh, P. Shyam, P. Mishkin, B. McGrew, I. Sutskever, M. Chen, Glide: towards photorealistic image generation and editing with text-guided diffusion models, arXiv preprint, arXiv:2112.10741, 2021.
[50] C. Saharia, W. Chan, S. Saxena, L. Li, J. Whang, E. Denton, S.K.S. Ghasemipour, B.K. Ayan, S.S. Mahdavi, R.G. Lopes, et al., Photorealistic text-to-image diffusion models with deep language understanding, arXiv preprint, arXiv:2205.11487, 2022.
[51] J. Ho, A. Jain, P. Abbeel, Denoising diffusion probabilistic models, Adv. Neural Inf. Process. Syst. 33 (2020) 6840–6851.
[52] Y. Song, S. Ermon, Generative modeling by estimating gradients of the data distribution, Adv. Neural Inf. Process. Syst. 32 (2019).
[53] S. Lyu, Interpretation and generalization of score matching, arXiv preprint, arXiv:1205.2629, 2012.
[54] W. Feller, On the theory of stochastic processes, with particular reference to applications, in: Selected Papers I, Springer, 2015, pp. 769–798.
[55] C.P. Robert, G. Casella, Monte Carlo Statistical Methods, Springer, 2004.
[56] J. Huggins, M. Kasprzak, T. Campbell, T. Broderick, Validated variational inference via practical posterior error bounds, in: International Conference on Artificial Intelligence and Statistics, 2019.
[57] A.B. Tsybakov, Introduction to Nonparametric Estimation, Springer, 2008.
[58] A. Dutle, C.A. Muñoz, E. Conrad, A. Goodloe, L. Titolo, I. Perez, S. Balachandran, D. Giannakopoulou, A. Mavridou, T. Pressburger, From requirements to autonomous flight: an overview of the monitoring ICAROUS project, in: Proc. 2nd Workshop on Formal Methods for Autonomous Systems, in: EPTCS, vol. 329, 2020, pp. 23–30.
[59] A. Balakrishnan, A.G. Puranic, X. Qin, A. Dokhanchi, J.V. Deshmukh, H. Ben Amor, G. Fainekos, Specifying and evaluating quality metrics for vision-based perception systems, in: Design, Automation & Test in Europe Conference & Exhibition (DATE), 2019, pp. 1433–1438.
[60] A. Balakrishnan, J. Deshmukh, B. Hoxha, T. Yamaguchi, G. Fainekos, Percemon: online monitoring for perception systems, in: Runtime Verification, Springer, Cham, 2021, pp. 297–308.