Toward a Unified Framework for Verifying and Interpreting Learning-Based Networking Systems

Yangfan Huang*, Yuling Lin†, Haizhou Du‡, Yijian Chen‡, Haohao Song*,
Linghe Kong♢, Qiao Xiang*†, Qiang Li△, Franck Le⋆, Jiwu Shu*†◦,
*Institute of Artificial Intelligence, Xiamen University, †School of Informatics, Xiamen University,
‡School of Computer Science and Technology, Shanghai University of Electric Power,
♢School of Electronic Information and Electrical Engineering, Shanghai Jiao Tong University,
△Alibaba, ⋆IBM Research, ◦Minjiang University

***Abstract*—There has been a growing interest in applying machine learning to real-world tasks. However, due to the black-box nature of machine learning models, it is crucial to (1) verify important properties of a model and (2) understand the reasons behind a model's prediction before deploying them in a production environment. Existing approaches typically handle them as two separate and sometimes orthogonal topics. In this paper, we show that the verification and interpretability of machine learning models are tightly related and can be unified by satisfiability modulo theories (SMT). Our key insight is: not only a wide range of properties of machine learning models can be formulated as SMT problems and verified accordingly, but many commonly studied interpretability questions can also be answered by iteratively checking the satisfiability and related properties of multiple SMT problems. Leveraging this insight, we design *UINT*, a general verification and interpretability framework for learning-based networking systems. *UINT* (1) allows operators to specify verification and interpretability problems as SMT formulas, (2) encodes the target machine learning models into SMT constraints, and (3) automatically solves the corresponding verification and interpretability problems using commodity SMT solvers. We implement a prototype of *UINT* and evaluate it on real-world learning-based networking systems. Results demonstrate the efficiency and efficacy of *UINT* in verifying and interpreting key questions for these systems.**

I. Introduction

Motivated by the success of machine learning in computer vision [1], natural language processing [2], and recommendation systems [3], there has been an emerging trend to apply machine learning in various networking systems, such as bitrate selection [4], congestion control [5], network traffic classification [6], anomaly prediction [7] and so on.

However, due to the black-box nature of these systems, fully understanding their behaviors is difficult. This makes it difficult to deploy them in production networks [8], [9]. For example, in our private conversation with a large cloud provider, it is revealed that their in-house learning-based traffic anomaly detection system only slightly outperforms traditional non-learning-based solutions. Because the decision-making mechanism of the system is not precise, it is hard to judge the rationality of decisions and further improve the system design. Moreover, if a learning-based system makes a wrong decision in the production network, it could lead to severe financial and social losses. Therefore, it is crucial and imperative to study the decision-making and reasoning of the model.

Toward understanding the capability and decision rationales of learning-based networking systems, people recently started to focus on how to verify and interpret the behaviors of learning-based systems automatically [10]–[18]. Specifically, verification focuses on examining whether the machine learning model of the system is robust (*e.g.*, not sensitive to small changes in features). The theoretical foundation used by verification typically includes predicate logic [12], [16] and mixed integer linear programming [11], [17], [18]. In contrast, interpretability typically focuses on providing explanations to specify why the model outputs. The explanations can be either the extracted rules to explain the decisions of a system's learning model [13], [19], [20] (*e.g.*, what features anchor the prediction of a model) or the feature attribution to the decisions through analysis. Despite such progress, these studies are point solutions, and treat the verification and interpretation of learning-based systems separately and orthogonally.

In this paper, we argue that the verification and interpretation of learning-based networking systems are tightly related and therefore should be unified. In particular, verification studies the question of *whether*: given a learning-based system and a set of predicates on its input, examine whether a set of predicates on its output always holds. Interpretation studies the question of *what*: given a learning-based system and a set of predicates on its output, find a set of predicates on its input satisfying certain criteria such that the predicates on the output always hold. In analogy, their relationship is similar to the decision and optimization problems in computational complexity theory: the former requires answering whether a solution exists, and the latter requires answering what the solution is. Given this close tie between verification and interpretation, unifying them is beneficial for learning-based systems. It enables a deeper understanding of the behaviors of these black box systems. Such understanding can provide helpful information to operators on the capability and decision

Haizhou Du and Qiao Xiang are co-corresponding authors.

979-8-3503-9973-8/23/$31.00 © 2023 IEEE

rationale of such systems and hence help refine system design. However, due to the diversity of verification and interpretability issues, there is a key challenge to uniform the question forms and find a general solution tool.

To this end, our key insight is that the verification and interpretation of learning-based networking systems can be unified by satisfiability modulo theories (SMT). Specifically, given a learning-based system, a set of predicates P on system input and a set of predicates Q on system output, motivated by the recent success of computer network configuration verification and synthesis [21], [22], we can verify whether Q always holds by checking whether the SMT problem $P \wedge N \wedge \neg Q$ is unsatisfiable, where N is an SMT formula encoding the machine learning model of the system. Moreover, given N and only the predicates Q on system output, we can find the predicates P satisfying certain criteria and guaranteeing that Q always holds by iteratively checking the satisfiability and related properties of multiple SMT formulas.

Building on this insight, we propose *UINT*, a general verification and interpretability framework for learning-based networking systems. *UINT* (1) allows operators to specify verification and interpretation problems as SMT formulas; (2) encodes the learning model of a given learning-based system as SMT formulas; and (3) automatically computes answers to the specified questions by solving SMT problems using commodity SMT solvers. The **main contributions** of this paper are as follows:

- We study the important and novel problem of how to unify various verification and interpretability problems of learning-based networking systems, and find that they can be unified using SMT. To our knowledge, we are the first to make this finding.
- Leveraging this insight, we propose *UINT*, a general verification and interpretability framework for learning-based networking systems, which can verify and interpret a wide range of common problems about the behaviors of learning-based systems.
- We implement a prototype of *UINT* and evaluate it to verify and interpret real-world learning-based networking systems. Results demonstrate the efficiency and efficacy of *UINT* in interpreting key questions for learning-based systems.

II. Overview

In this section, we first provide a background introduction to the verification and interpretability of learning-based networking systems and discuss their relationship (Section II-A). We next elaborate our key insight on how these two issues can be related using SMT (Section II-B), and introduce the architecture of *UINT*, a general verification and interpretability framework built on top of this insight (Section II-C).

A. Background

Verification of learning-based systems. Given such a system, the verification problem studies whether the (input, output) mapping of a trained model of this system always satisfies certain properties [23]. Properties of interest to verification are usually related to the model quality: the robustness of the model [24], including adversarial perturbation [11], [24], missing features [11], extreme value [12]. Existing tools typically model the verification problem of learning-based networking systems as a mixed-integer-based linear programming problem [25] or a predicate logic model [10], [12]. However, these are point solutions focusing on different properties, and each work proposes its methods applied to the specific properties. There is still a lack of a general tool that can verify a wide range of properties of learning-based networking systems, obtaining results about the model robustness against perturbations.

Interpretability of learning-based systems. Different from verification, the interpretability problem aims to provide operators with a set of simple, deterministic rules that explains a trained model's behaviors [13], [26] or analyze features to provide explanations [27], [28]. Behaviors that people are interested in finding rules for include anchor [19], counterfactual explanations [29], and decision rules [30]. As for feature attribution methods, there are feature importance, sensitivity analysis [8], [28] and so on. However, not only do the different tools focus on answering different interpretation questions, but they also vary in form and complexity. A general tool that can explain a wide range of behaviors of learning-based networking systems is also missing.

Relationship between verification and interpretability. Although both verification and interpretability enable important understanding toward the behaviors of the black box learning-based systems, people typically treat them as two separate and often orthogonal topics. Recently, some studies hint that the verification and interpretability of machine learning models partially overlap [8], [26], [31]. For example, some argue that certain model verification problems (*e.g.*, adversarial perturbations) fall into the scope of deriving reliability-related rules for interpretability [26]. Some regard the more robust the model, the better the explanation produced by the interpretability method, thus connecting between explainability and robustness [24]. However, to the best of our knowledge, there have been no studies that systematically investigate how diverse property verification and various interpretability problems relate to each other.

We observe that the relationship between verification and interpretability of learning-based systems is similar to that between the decision and optimization problems in computational complexity theory. Verification studies whether a given property is always held by a system while interpretability studies what properties (*i.e.*, rules) a system holds. As such, unifying verification and interpretability can enable a deeper understanding toward the behaviors of learning-based networking systems. It can also prevent system errors and irregularities from happening by finding them before such systems are deployed and fulfilling the ethical and legal obligations of deploying learning-based networking systems.

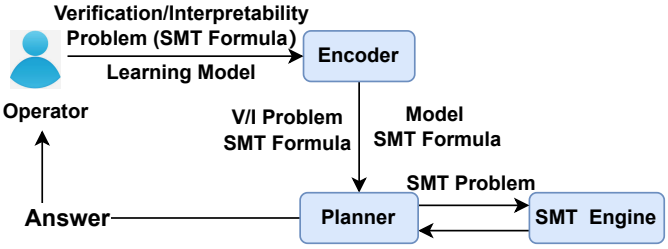

Fig. 1: The architecture and workflow of *UINT*.

B. Key Insight

Our critical insight is that the verification and interpretability of learning-based systems can be unified by SMT. In a nutshell, SMT is a problem that checks whether a formula of first-order logic is satisfiable. It generalizes the Boolean satisfiability (SAT) problems from formulas of Boolean variables to more complex formulas with richer variables (*e.g.*, integers and bit vectors) [32]. Although SMT is NP-hard, many SMT solvers have demonstrated the capability of efficient SMT solving [33], [34]. As such, SMT has been widely used in many areas, including software and hardware verification, program analysis, and mathematical programming.

Given a learning-based networking system, we use N to denote its machine learning model. The verification problem of such a system can be defined as to check whether the SMT formula $P \wedge N \rightarrow Q$ is a tautology, where P is a set of predicates specifying the assumptions of the model input, and Q is a set of predicates specifying the assertion of the model output. This is equivalent to checking whether $P \wedge N \wedge \neg Q$ is unsatisfiable [35]. If it is, for any data input $\mathbf{x}$ that satisfies P, its output $y = N(\mathbf{x})$ always satisfies Q. Otherwise, there exists at least one data input $\mathbf{x}$ satisfies P while $y = N(\mathbf{x})$ does not satisfy Q. For example, the property of adversarial perturbation [11], [36] requires that: for a given input data with m features $\mathbf{x}^* = [x_1^*, \ldots, x_m^*]$ and its output $y^* = N(\mathbf{x}^*)$, if there exists a small perturbation of no more than ϵ on each input feature, the output should not change. This property can be verified by checking the satisfiability of

$$\wedge_{i=1,\ldots,m}, x_i \in (x_i^* - \epsilon, x_i^* + \epsilon) \Rightarrow N(\mathbf{x}) = y^*. \quad (1)$$

Similarly, the interpretability problem of learning-based networking systems can be usually defined as given a learning model N and a set of predicates Q on the model output, find a set of predicates P on the model input such that (1) $P \wedge N \wedge \neg Q$ is unsatisfiable; and (2) P satisfies some additional criteria. Such a problem can be solved by checking the satisfiability and related properties of multiple SMT formulas. For example, the interpretability problem of feature importance asks to find the minimal ϵ such that Equation (1) of adversarial perturbation property is unsatisfiable. Such an ϵ can be found by repeatedly solving Equation (1) using binary search.

C. Architecture of UINT

Leveraging the key insight above, we design *UINT*, a general verification and interpretability framework for learning-based networking systems. Fig. 1 shows the architecture of *UINT*. *UINT* works in four steps. First, given a learning-based networking system, an operator specifies a verification or interpretability problem as an SMT formula, and sends it together with the learning model of this system to the *UINT* encoder. Second, the encoder automatically encodes the received learning model into an SMT formula and sends it together with the received problem to the *UINT* planner. Third, the planner analyzes the received problem and decides the corresponding SMT problems to solve. Fourth, the planner invokes the SMT engine, iteratively if necessary, to solve the received problem, and returns the final verification or interpretability answer to the operator. In the following sections, we will elaborate on how *UINT* solves the most commonly studied verification and interpretability problems specifically.

III. Model Verification

In this section, we introduce how *UINT* verifies a wide range of common properties of learning-based networking systems. We first describe given such a system, how *UINT* encodes its machine learning model N (Section III-A). We then provide details on how operators can specify a wide range of verification properties by specifying P predicates on model input, and Q predicates on model output (Section III-B).

A. Encoding of Model

For ease of exposition, we assume the machine learning model of a learning-based networking system is a deep neural network. Other models, such as decision trees and support vector machine (SVM) can be encoded similarly. Given a deep neural network, *UINT* encodes it into SMT formulas similar to other SMT-based verification tools [11], [12]. Specifically, we divide the model encoding into three phases: input layer encoding, hidden layer encoding, and output layer encoding. For input and output layers, we encode them as predicates on their ranges. As a simple example, the output layer of Pensieve [4] is a classification network layer with six outputs. The encoding is: $\bigwedge_{i=1}^{6} x_i \in [l_i, u_i]$.

Hidden layers are slightly more complicated to encode in that we need to construct relational expressions between input variables and output variables based on the internal logic of the neural network. To this end, *UINT* provides the encodings of fully connected layers, convolutional layers, and activation functions. Given a fully connected layer, assuming the input is an m-dimensional vector $\mathbf{x}$ and the output is an n-dimensional vector $\mathbf{y}$, the weight matrix of this layer is $\mathbf{W}$, b_n denotes bias. The network logic of this layer can be seen as: $\mathbf{y} = A(\mathbf{W}\mathbf{x} + b_n)$, where $A()$ is an activation function. The encoding of this fully connected layer is a set of linear mathematical expressions:

$$\bigwedge_{i=1}^{n} i, \mathbf{y}_i = \sum_{j=1}^{m} \mathbf{W}_{ij} \cdot \mathbf{x}_j + b_i. \quad (2)$$

A convolutional layer is the matrix multiplication of the input and the convolution kernel K, where the latter is a multidimensional vector. We choose the value of the first channel

TABLE 1: Details of robustness properties.

Property	L	P_l	P_r
Adversarial perturbation	F	$x_i \in (x_i^* - \epsilon \cdot k_i, x_i^* + \epsilon \cdot k_i)$	None
Missing features	K	$x_i \in (l_i, r_i)$	$x_j = x_j^*$
Extreme values	K	$x_i \in (l_i^e, r_i^e)$	$x_j \in (l_j, r_j)$

in the convolution kernel, regard this reduced dimensionality matrix as a weight matrix and convert the operation of the convolutional layer into the fully connected layer [11]. For activation functions, we can directly encode the rectified linear unit function $ReLU(\mathbf{x}, y)$ iff $y = max(0, \mathbf{x})$ because it is piecewise linear. For other activation functions (*e.g.*, softmax, sigmoid), we also use ReLU to approximate them as does in other SMT-based tools [11], [23].

B. Encoding of Property

In *UINT*, operators can specify the properties to verify in the form of SMT formulas. We summarized common properties of interest in learning-based networking systems into **robustness properties**: verify the resilience of the systems against perturbation [11], [25], [36], [37]; These properties examine whether a model maintains its original output when the input features deviate from their original values. We provide a generic formulation for robustness properties. Specifically, assume the output of model N on input $\mathbf{x}^*$ is y^*. Let F denote the set of all input features, and L denote the set of input features that can be perturbed. We let P_l denote the predicates on perturbable input features and P_r denote the predicates on other input features. Set the output predicate Q as $N(\mathbf{x}) = y^*$. The unified formulation of robustness is:

$$\forall i \in L, j \in F \backslash L, P_l(x_i) \wedge P_r(x_j) \rightarrow N(\mathbf{x}^*) = y^*. \quad (3)$$

By setting set L and the predicates P_l and P_r differently, Equation (3) can represent different robustness properties (Table 1). Next, we will describe them in detail one by one.

Adversarial perturbation. This property examines given an (input, output) mapping $(\mathbf{x}^*, y^*)$, whether the output remains the same when each input feature may have a slight perturbation [11], [36]. This property can help operators understand how resilient the model is. We can specify it by setting $L = F$, and set P_l based on the attack power. Specifically, let k_i be the difference between the maximum and minimum possible values of x_i. Then the attacker with an attack ability ϵ can change x_i within range $x_i \in (x_i^* - \epsilon \cdot k_i, x_i^* + \epsilon \cdot k_i)$. This property is then specified as:

$$\forall i \in F, x_i \in (x_i^* - \epsilon \cdot k_i, x_i^* + \epsilon \cdot k_i) \rightarrow N(\mathbf{x}) = y^*. \quad (4)$$

If this formula is a tautology, then the system is immune to attackers whose attack power is within ϵ.

Missing features. This property examines given an (input, output) mapping $(\mathbf{x}^*, y^*)$, whether a model maintains the output y^* even if some features are lost [11]. Suppose a set of missing features K. We can specify this property by letting $L = K$ and set P_l and P_r accordingly:

$$\forall i \in K, j \in F \setminus K, (x_i \in (l_i, r_i)) \wedge (x_j = x_j^*) \rightarrow N(\mathbf{x}) = y^*, \quad (5)$$

Algorithm 1: Qualitative anchor.

1: **Input:** The original input $\mathbf{x}^*$ with output y^*, model encoding N
2: **Output:** The minimum dominant feature set F_d
3: Set constraints: $N \wedge N(\mathbf{x}^*) \neq y^*$ as $hard$, $x_1^*, ..., x_m^*$ as $soft$
4: **if** Equation (10) returns $unsat$ **then**
5: L = CoMSS(Equation (10))
6: **end if**
7: Return L

where (l_i, r_i) is the valid range of feature x_i. If the formula is always satisfiable, we can conclude that the model used in the system is resilient when missing a set of features K.

Extreme values. This property examines given an (input, output) mapping $(\mathbf{x}^*, y^*)$, whether a model remains the same when a subset of features is taking extreme values, while other features still fall in their typical ranges. An example of this property is that in the Airborne Collision Avoidance System [12] if the intruder is distant and is significantly slower than the airship, the model agent should directly make a Clear-of-Conflict (COC) advisory decision.

Suppose the set of features that may take extreme values is K. This property can be specified as:

$$\forall i \in K, j \in F \backslash K, (x_i \in (l_i^e, r_i^e)) \wedge (x_j \in (l_j, r_j)) \rightarrow N(\mathbf{x}) = y^*, \quad (6)$$

where (l_i^e, r_i^e) is the range of extreme values of feature x_i.

IV. Model Interpretability

In this section, we introduce how *UINT* finds answers to common interpretability problems by checking the satisfiability of relevant properties and multiple SMT formulas. *UINT* focuses on analyzing machine learning models after training, and answering interpretable questions about the internals of models [8], [26], [31], [38]. In particular, *UINT* focuses on two categories of interpretability problems, which cover the majority of interpretable problems of learning-based systems: *rule explanation*, which aims to find a set of rules that explains why a model makes a certain prediction, and *feature attribution*, which aims to specify which features are important and which are relatively less important. These categories represent the most important and most commonly asked interpretability problems for learning-based networking systems [13].

We provide a generic formulation for these interpretability problems. Assume the output of model N on input $\mathbf{x}^*$ is y^*. Let F denote the set of all input features, and L denote the set of input features that we focus on (be perturbed). We let P_l denote the predicates on input features L, and P_r denote the predicates on other input features. Set the output predicate Q on $N(\mathbf{x})$. The unified formulation is:

$$\forall i \in L, j \in F \backslash L, P_l(x_i) \wedge P_r(x_j) \wedge N \wedge Q(N(\mathbf{x})). \quad (7)$$

By setting set L and the predicates P_l and P_r differently, Equation (7) can represent different interpretability problems (Table 2). Next, we will describe them in detail one by one.

TABLE 2: Details of Interpretability.

Sub-problem	L	P_l	P_r	$Q(N(\mathbf{x}))$
Qualitative anchor	None	None	$x_j = x_j^*$	$N(\mathbf{x}^*) = y^*$
Quantitative anchor	F_d	$x_i \in (x_i^* - \epsilon, x_i^* + \epsilon)$	None	$N(\mathbf{x}^*) \neq y^*$
Counterfactual explanation	F_d	$x_i^c \in (l_i, r_i) \bigwedge d(\mathbf{x}^*, \mathbf{x}^c) \in (l_T, r_T)$	None	$N(\mathbf{x}^c) = y^c$
Decision boundary	x_a, x_b	$x_a \in (l_a, r_a)) \wedge (x_b \in (l_b, r_b)$	$x_j = x_j^*$	$N(\mathbf{x}) = y^*$
Feature importance	F_p	$x_i \in (x_i^* - \epsilon, x_i^* + \epsilon)$	$x_j = x_j^*$	$N(\mathbf{x}) \neq y^*$
Sensitivity analysis	F_p	$x_i \in (x_i^* - \epsilon, x_i^* + \epsilon)$	$x_j = x_j^*$	$N(\mathbf{x})$

TABLE 3: Rule Explanation.

Sub-problem	$P(\mathbf{x})$	$Q(N(\mathbf{x}))$
Qualitative anchor	$\forall i \in F_d, x_i = x_i^*$	$N(\mathbf{x}) = y^*$
Quantitative anchor	$\forall i \in F_d, r(x_i)$	$N(\mathbf{x}) = y^*$
Counterfactual explanation	$\forall i \in F_d, x_i^* = x_i^c$	$N(\mathbf{x}^c) = y^c$
Decision boundary	$\exists \{\mathbf{x}\} \in \mathcal{X}^D, \mathcal{X}^{D^*} \wedge \mathcal{X}^{D^{other}} = \mathcal{X}^D, \mathbf{x} \in \mathcal{X}^{D^*}$	$N(\mathbf{x}) = y^*$

A. Rule Explanation

Given a learning-based system, we use N to denote its machine learning model, and given input data with m features $\mathbf{x}^* = [x_1^*, \ldots, x_m^*]$ and its output $y^* = N(\mathbf{x}^*)$. F denotes the m-dimensional feature set of N. We use these symbols to illustrate all the following problems. Rule explanation aims to find a set of predicates $P = \{r_1, r_2, ..., r_m\}$ that express a logical relationship between the input $\mathbf{x}^*$ and corresponding output $N(\mathbf{x}^*)$:

$$P(\mathbf{x}^*) \rightarrow Q(N(\mathbf{x}^*)). \tag{8}$$

Sub-problems provide different rule explanations with different P and Q, as in Table 3.

Anchors. Given an (input, output) mapping $(\mathbf{x}^*, y^*)$ of learning model N, the problem of anchor aims to find a set of predicates of dominant features such that they are decisive for prediction and all other features are irrelevant for models to make the decision [19].

1) Qualitative anchor: The qualitative anchor aims to find the *minimum* set of dominant features F_d that are decisive for $N(\mathbf{x}^*) = y^*$ such that for any other feature in $F \setminus F_d$, it can take on arbitrary values without changing the output [39]:

$$\forall i \in F_d, x_i = x_i^* \rightarrow N(\mathbf{x}) = y^*. \tag{9}$$

To solve this problem, *UINT* leverages the concept of unsatisfiable core and MaxSAT problem in SMT [35], [40]. To be concrete, an SMT formula can be written as a conjunctive normal form (CNF) of clauses. By marking clauses "hard" or "soft", when this formula is unsatisfiable, a MaxSAT solver can return a *maximum* set of *soft* clauses that can be simultaneously satisfied by an assignment that also satisfies all *hard* clauses. The complement of such a set of maximum soft satisfiability clauses (CoMSS) is a set of soft clauses of minimum cardinality whose removal will make the original formula satisfiable [40]. Leveraging these concepts, we define the following SMT formula based on the given data mapping and Equation (9):

$$\forall i \in F, \underbrace{(x_i^*)}_{\text{soft}} \wedge \underbrace{N \wedge N(\mathbf{x}^*) \neq y^*}_{\text{hard}}. \tag{10}$$

Obviously, the SMT solver will return an $unsat$ for this formula. As such, we can find the minimum set of features F_d by getting the CoMSS of soft constraints. The process is summarized in Algorithm 1.

Algorithm 2: Quantitative anchor.

1: **Input:** The original input $\mathbf{x}^*$ with output y^*, model encoding N, update step a.
2: **Output:** Anchor P_a
3: $P_a \leftarrow \phi$
4: Get F_d from Algorithm 1
5: Set the attack power $\epsilon \leftarrow init$
6: Set constraint C_1 for F_d: $\forall i \in F_d$, $x_i \in (x_i^* - \epsilon, x_i^* + \epsilon)$ as $soft$
7: Set constraint C_2: $N \wedge (N(\mathbf{x}) \neq y^*$ as $hard$
8: **while** (F_d != $null$) **do**
9: **while** Equation (12) returns sat **do**
10: $\epsilon \leftarrow \epsilon$ - a
11: **end while**
12: L = CoMSS(Equation (12))
13: $P_a \leftarrow P_a \cup L$
14: $F_d \leftarrow F_d - L$, update C_1
15: **end while**
16: Return P_a

2) Quantitative anchor: Given an (input, output) mapping $(\mathbf{x}^*, y^*)$ of learning-based model N and its qualitative anchor (*i.e.*, the dominant feature set F_d), the quantitative anchor problem aims to find a set of predicates $P_a = \{r_1, r_2, ..., r_d\}$ for the dominant feature set F_d of $\mathbf{x}^*$, where each r_i is in the form of $x_i \in [v_1, v_2]$ [19], [39], such that:

$$\forall i \in F_d, r(x_i) \rightarrow N(\mathbf{x}) = y^*. \tag{11}$$

In other words, the quantitative anchor aims to find the value range of each dominant feature x_i, where $i \in F_d$ that collaborative guarantees the model output stays as y^*.

We design Algorithm 2 to find the quantitative anchor. Specifically, after finding F_d using Algorithm 1, we perturb the dominant features with an attack power of ϵ. When Equation (12) is satisfiable, indicating there exists an input value whose output is different (the scope of the soft clause is outside the scope where the output is y^*), then reduce ϵ until Equation (12) is not satisfied (line 8 to 11), the corresponding CoMSS L is the feature range that keeps the prediction unchanged and saves these clauses in P_a (line 12 to 13), then remove the features in L from F_d to update F_d and C_1 (line 14), repeat the process until we get complete P_a.

$$\forall i \in F_d, \underbrace{x_i \in (x_i^* - \epsilon, x_i^* + \epsilon)}_{C_1\text{: soft}} \wedge \underbrace{N \wedge N(\mathbf{x}) \neq y^*}_{C_2\text{: hard}}. \tag{12}$$

Counterfactual explanation. Given an (input, output) mapping $(\mathbf{x}^*, y^*)$ of learning-based model N, and the expected prediction y_e, the problem of counterfactual explanation aims to find the closest counterfactual $\mathbf{x}^c$ such that $N(\mathbf{x}^c) = y_e$ [28], [29]. A commonly used distance measure d is the $\mathcal{L}_1$ norm weighted by the inverse median absolute deviation (MAD) [29], expressed as $d(\mathbf{x}^*, \mathbf{x}^c) = \sum_{k \in \mathbf{x}^D} \frac{|\mathbf{x}^* - \mathbf{x}^c|}{MAD_k}$ where

Algorithm 3: Closest counterfactual.

1: **Input:** The original input $\mathbf{x}^*$ with output y^*, model encoding N, expected output y_e, accuracy δ.
2: **Output:** The closest counterfactual $\mathbf{x}^c$
3: $l_d \leftarrow 0, r_d \leftarrow d_{max}$
4: Set constraint: $(\bigwedge_{i=1}^{m} x_i^c \in (l_i, r_i)) \wedge N$ $\wedge(N(\mathbf{x}^c) = y_e)$
5: **while** $r_d - l_d > \delta$ **do**
6: $mid = (r_d + l_d)/2$
7: Set constraint: $d(\mathbf{x}^*, \mathbf{x}^c) < mid$
8: **if** Equation (13) return sat **then**
9: $r_d = mid$
10: **else**
11: $l_d = mid$
12: **end if**
13: **end while**
14: $\mathbf{x}^c$ = Equation (13)
15: Return $\mathbf{x}^c$

$\mathbf{x^D}$ is the neighbors of $\mathbf{x}^*$.

We design Algorithm 3 to find $\mathbf{x}^c$. Specifically, we first set the possible search range of d as (l_d, r_d). We then perform a binary search on this range (line 5 to 13) [41] by iteratively solving the following SMT formula in Equation (13). In each iteration, if the formula is satisfiable, it indicates that the upper bound r_d of d can be reduced (line 9). If the formula is unsatisfiable, it indicates that the range is too small and the lower bound l_d should be increased (line 11). By the end of this search process, we can find a counterfactual $\mathbf{x}^c$ whose distance to $\mathbf{x}^*$ is minimum within an accuracy of δ.

$$\begin{aligned}&\forall i \in F,\\&(x_i^c \in (l_i, r_i)) \wedge (d(\mathbf{x}^*, \mathbf{x}^c) \in (l_d, r_d) \wedge N \wedge N(\mathbf{x}^c) = y_e.\end{aligned} \tag{13}$$

Decision boundary. Given an (input, output) mapping $(\mathbf{x}^*, y^*)$ of learning-based model N, the problem of decision boundary [42], [43] aims to find a set of input variables vectors $\{\mathbf{x}\}$ that divides the feature space $\mathcal{X}^D$ into the corresponding parts $\mathcal{X}^{D^*}$ and $\mathcal{X}^{D^{other}}$. This explanation of decision boundary is expressed as:

$$\begin{aligned}\exists\{\mathbf{x}\} \in \mathcal{X}^D, \mathcal{X}^{D^*} \wedge \mathcal{X}^{D^{other}} = \mathcal{X}^D,\\ \mathbf{x} \in \mathcal{X}^{D^*} \rightarrow N(\mathbf{x}) = y^*.\end{aligned} \tag{14}$$

We focus on finding the decision boundary of two features x_a, x_b of $\mathbf{x}$ in $\mathcal{X}^D$ [11]. We iteratively check the SMT formula in Equation (15). Every time the solver finds it satisfiable, a satisfiable assignment is returned. To avoid SMT returning duplicate solutions, we add solution constraints C_4 to Equation (15) (line 8) to make sure next returned solutions are outside the neighborhood of the known solutions. By solving this SMT formula iteratively (line 9 to 12), we get enough value combinations of x_a, x_b, and we can fit the corresponding decision boundary. See the detailed process in Algorithm 4.

Algorithm 4: Decision boundary.

1: **Input:** The original input $\mathbf{x}^*$ with output y^*, model encoding N, the selected features: x_a, x_b, neighborhood range δ.
2: **Output:** The solution list S (draw the area of decision boundary based on S)
3: Set constraint C_1: $x_a \in (l_a, r_a)$, $x_b \in (l_b, r_b)$;
4: Set constraint C_2: $\forall j \in F\backslash\{a, b\}$, $x_j = x_j^*$
5: Set constraint C_3: $N(\mathbf{x}) = y^*$
6: $S \leftarrow \phi$
7: Add s_1 = Equation (15) to S
8: Set solution constraints C_4: $\forall s_i \in S : \{x_{ai}, x_{bi}\}$, the values of x_a, x_b in returned solution is not in $(x_{ai} - \delta, x_{ai} + \delta)$ and $(x_{bi} - \delta, x_{bi} + \delta)$
9: **while** (Equation (15) $\wedge C_4$) return sat **do**
10: Add s_i to S
11: Update solution constraint C_4
12: **end while**
13: return S

Algorithm 5: Feature importance.

1: **Input:** The original input $\mathbf{x}^*$ with output y^*, model encoding N, perturbed feature set F_p, increase step α, accuracy δ.
2: **Output:** The minimal attack power ϵ_0 of F_p
3: Set $\epsilon \leftarrow init$, $l_\epsilon \leftarrow 0$
4: Set constraint C: $\forall i \in F_p, x_i \in (x_i^* - \epsilon \cdot k_i, x_i^* + \epsilon \cdot k_i)$
5: Set constraint: $(\forall j \in F\backslash F_p, x_j = x_j^*) \wedge N \wedge N(\mathbf{x}) \neq y^*$
6: **while** Equation (16) returns $unsat$ **do**
7: $l_\epsilon \leftarrow \epsilon$, $\epsilon \leftarrow \epsilon \cdot (1 + \alpha)$
8: $\forall i \in F_p$, re-encode C with the updated ϵ
9: **end while**
10: $r_\epsilon \leftarrow \epsilon$
11: **while** $r_\epsilon - l_\epsilon > \delta$ **do**
12: $\epsilon \leftarrow (l_\epsilon + r_\epsilon)/2$
13: $\forall i \in F_p$, re-encode C with the updated ϵ
14: **if** Equation (16) returns sat **then**
15: $r_\epsilon \leftarrow \epsilon$
16: **else**
17: $l_\epsilon \leftarrow \epsilon$
18: **end if**
19: **end while**
20: Return ϵ_0

$$\underbrace{(x_a \in (l_a, r_a)) \wedge (x_b \in (l_b, r_b))}_{C_1} \wedge \underbrace{(\bigwedge_{j=1}^{F\backslash\{a,b\}} x_j = x_j^*)}_{C_2} \wedge N \wedge \underbrace{N(\mathbf{x}) = y^*}_{C_3}. \tag{15}$$

B. Feature Attribution

In addition to finding the rules that each feature needs to satisfy, people are also interested in the feature's contribution to the prediction of the learning-based systems [13], [44], this type of interpretability problem is called "Feature Attribution". We focus on the feature importance and sensitivity analysis, as these are the two most significant problems of Feature Attribution, from which all other sub-problems extend.

Feature importance. This problem aims to get the features importance ranking array M for the prediction. We leverage the perturbation-based metric to get the feature importance based on this fact: the features that contributed more to the original classification will be just those that contributed more to the misclassification [8], [44]. So we rank the features according to how well a feature resists the perturbation.

Given an (input, output) mapping $(\mathbf{x}^*, y^*)$, we can get the importance array M by comparing the minimal attack power on each feature that can change the output. The smaller the minimum attack power, the more important the feature. Algorithm 5 presents the process of finding a minimal attack power. Given the SMT formula in Equation (16), we set the perturbed feature set F_p, set k_i be the difference between the maximum and minimum possible values of x_i, and initialize the attack power ϵ with range (l_ϵ, r_ϵ) according to the actual systems. We perturb the features within the F_p to verify whether the current attack power ϵ can change the output. If the solver returns $unsat$, indicating current ϵ cannot change the output, multiplicative increase with α to update ϵ (line 7) until SMT returns sat, and set the current ϵ as upper bound. Then we can determine the minimal attack power ϵ_0 under the accuracy δ by binary search (lines 11 to 19).

$$(\bigwedge_{i=1}^{F_p} x_i \in (x_i^* - \epsilon \cdot k_i, x_i^* + \epsilon \cdot k_i)) \wedge (\bigwedge_{j=1}^{F \backslash F_p} x_j = x_j^*) \wedge N \wedge N(\mathbf{x}) \neq y^*. \tag{16}$$

Sensitivity analysis. This problem aims to evaluate the uncertainty in the output of a model for different sources of uncertainty in its inputs [27]. Given an (input, output) mapping $(\mathbf{x}^*, y^*)$, we apply random perturbations to the selected feature set to examine the change in the output. We use the degree of deviation of the output as a measure of the sensitivity of the feature F_p: $s(F_p) = N(\mathbf{x}) - N(\mathbf{x}^*)$. For the same value of perturbation, the feature set with the largest value indicates how much potential it has in causing output.

The SMT formula below is used as a constraint, we can get the values of $N(\mathbf{x})$ returned by the SMT solver, and calculate the value of $s(F_p)$ by subtracting the original output.

$$\begin{aligned} &\forall i \in F_p, j \in F \backslash F_p, \\ &x_i = (x_i^* - \epsilon \cdot k_i, x_i^* + \epsilon \cdot k_i) \wedge (x_j = x_j^*) \wedge N \wedge N(\mathbf{x}). \end{aligned} \tag{17}$$

Algorithm 6: Sensitivity analysis.

1: **Input:** The original input $\mathbf{x}^*$ with output y^*, model encoding N, the selected feature set F_p.
2: **Output:** The deviation of output $s(F_p)$
3: Set constraints: $\forall i \in F_p, x_i \in (x_i^* - \epsilon \cdot k_i, x_i^* + \epsilon \cdot k_i)$
4: Set constraints: $\forall j \in F \backslash F_p, (x_j = x_j^*) \wedge N \wedge N(\mathbf{x})$
5: Get $N(\mathbf{x})$ returned by SMT
6: Return $N(\mathbf{x}) - y^*$

V. Performance Evaluation

We implement a prototype of *UINT* and evaluate its performance with extensive experiments on real-world learning-based systems. We aim to answer the following questions: (1) Can our framework verify common properties of different learning-based networking systems, and how efficient is it? (2) Can our framework provide understandable interpretability in different networking systems, and how efficient is it? We first describe our experiment settings and then present the results.

A. Experiment Settings

Real-world learning-based networking system.

As our work innovatively unifies verification and interpretation into one framework, no analogous tools are currently available for comparative analysis. We evaluate *UINT* on two representative systems: Pensieve and Aurora. The source code is available at [45].

Pensieve is an RL-based adaptive video bitrate selection system [4]. It trains a neural network model composed of two fully connected hidden layers with ReLU activation, and both hidden layers have 128 neurons. Pensieve divides its 25 features into six categories: *previous bit rate*, *throughput*, *download time*, *current buffer size*, *next chunk sizes*, and *remain chunks*. The prediction is the download bitrate with the highest probability of the output layer.

Aurora is an RL-based congestion control protocol [5]. It uses a neural network architecture with two hidden layers and we have several options for the number of neurons per layer (*e.g.*, 32*16). There are three features in Aurora: *latency inflation*, *latency ratio*, and *send ratio*. The prediction is the changed ratio of transmission rate over the last action. It makes decisions based on the latest k-step history of all features.

Verification and interpretability. We present some results of verification properties due to the space limit, whose experiment parameters for each system are in Table 4. These parameters are set similarly to other model verification tools [11], [23]. Similarly, we evaluate the interpretability problems of *rule explanation* and *feature attribution* discussed in Section IV and present the results of some of them due to space limitations. The experiment settings are in Table 5.

Environment. All experiments run on a server with 2 * Intel Xeon Silver 4210R @ 2.40GHz CPU and 128G DDR4 Memory. Our prototype uses Z3 [33] as the SMT engine.

B. Results

Verification. As the result of the verification time distribution of Pensieve shown in Table 6, *UINT* performs well when a feature is missing or when the value of a feature fluctuates within a certain range.

TABLE 4: Settings of verification experiments.

System / Property	Pensieve [4]	Aurora [5]
Adversarial perturbation	$\epsilon \in [0.016, 0.32]$ step size = 0.016	$\epsilon \in [2.5 \times 10^{-4}, 5 \times 10^{-3}]$ step size = 2.5×10^{-4}
Missing features	$\|K\| = \{1, 2\}$, In each run, select one feature as the missing feature set K.	$\|K\| = l$. In each run, select one feature and its most recent $l-1$-step histories as the missing feature set K. $l \in \{1, 3, 5, 10\}$
Extreme values	Pre bitrate $\leq$ 1850kBps, current buffer size $= 0.4$ and throughput $\in$ (0.54, 0.541) $\rightarrow$ the output = 1,850kBps	Send ratio $\in (0.99, 1.0) \rightarrow$ send rate increase

TABLE 5: Settings of interpretability experiments.

System / Interpretability	Pensieve [4]	Aurora [5]
Anchor	Find the qualitative anchor for an input whose output (*i.e.*, download bitrate) $= 1,850$ kbps	Find the qualitative anchor for an input whose output (*i.e.*, changed ratio transmission rate) $= 3.0$
Counterfactual explanations	Give an instance with output $= 1,850$ kbps, find an input with expected output $= 1,200$ kbps, $\delta = 0.01$	Give an instance with output $= 3.0$, find an input with expected output $= 2.5$, $\delta = 0.01$
Decision boundary	Find the boundary for *{buffer size, throughput}*, $\delta = 0.01$, history length = 1	Find the boundary for *{latency ratio, latency inflation}*, $\delta = 0.01$, history length = 10
Feature importance	Give an instance with output $= 1,850$ kbps, find the minimal attack power for each feature. History length = 1, $\epsilon_0 = 0.01, \delta = 1 \times 10^{-2}$	Give an instance with output $= 3.0$, find the minimal attack power for each feature, the universal minimal attack power for all features. History length $\in \{5, 10\}$, $\epsilon_0 = 5 \times 10^{-5}$, $\delta = 1 \times 10^{-6}$
Sensitivity analysis	History length = 1, $\delta = 1 \times 10^{-5}$, $\epsilon = 5 \times 10^{-5}$. The result is the difference in the rate change ratio.	History length = 10, $\delta = 1 \times 10^{-5}$, $\epsilon = 5 \times 10^{-5}$. The result is the difference in the probability of original selected bit rate.

TABLE 6: Property verification time distribution for Pensieve

Property	50%	70%	90%
Adversarial Perturbation	168.57s	309.88s	1494.31s
Missing Features	518.82s	1608.37s	3596.76s
Missing Features ($\|K\| = 2$)	4198.37s	6252.11s	45181.94s

In addition to the Aurora model with 32×16 network architecture, we train three Aurora system models with different neural network structures (32×32, 48×16, 48×32) and deploy experiments on those systems to demonstrate the effectiveness and scalability of *UINT*. We verify several robustness properties on Aurora. The verification run time for different properties on Aurora at all scales can be seen in Fig. 2. *UINT*'s running time increases slightly as the scale of the problem increase. The results on different scales of Aurora's neural network demonstrate that our framework *UINT* is generic and scalable in verifying common verification problems in learning-based networking systems of different scales.

Interpretability. We first show the minimal attack power results to get Pensieve's feature importance in Table 7. The *UINT* takes hundreds to tens of thousands of seconds to find the minimal ϵ and get the importance ranking. Considering the running time of many studies using SMT and some other interpretable methods are mainly between 10^3 and 10^4 seconds [10], [12]. Besides, Pensieve is a relatively complex learning-based networking system, and the interpretability is performed before the model is deployed. This overhead is reasonable and acceptable.

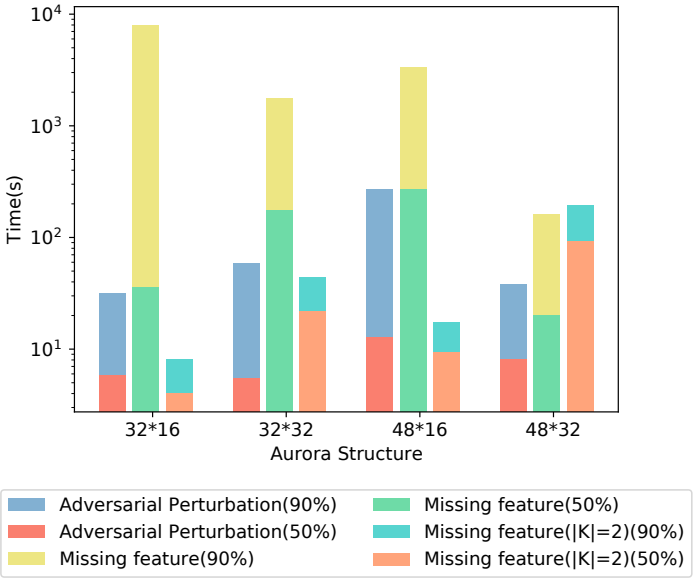

Fig. 2: Property verification time distribution for Aurora

We next show the result of Aurora in Table 8 and Fig. 3. We train four Aurora models with different neural network structures (32×16, 32×32, 48×16, 48×32) and deploy experiments on those systems to demonstrate the effectiveness and scalability of *UINT*. For qualitative anchor, the median time of *UINT* spends finding the answer is less than 100 seconds. Although the median time in Aurora is relatively

TABLE 7: Minimal attack power of Pensieve

Feature	ϵ	Rank	Time(s)
Previous bit rate	0.331	1	833.08
Buffer size	0.334	2	3169.283
Throughput	0.437	3	11064.44
Download time	1.599	4	33138.59

TABLE 8: Minimal attack power of Aurora (32*16)

Feature	ϵ	Rank	Time(s)
Latency ratio (History=5)	0.004	1	98.49
Send ratio (History=5)	0.013	2	120.04
Latency inflation (History=5)	0.034	3	112.46
Latency ratio (History=10)	0.002	1	233.79
Send ratio (History=10)	0.009	2	335.39
Latency inflation (History=10)	0.023	3	233.79

large, most are within 1,000 seconds. This demonstrates the generality and scalability of *UINT* in answering a wide range of interpretability questions.

Fig. 4 shows the decision boundary we found about *latency ratio* and *latency inflation* for Aurora. We can discover a set of feature values dividing the decision space into two regions, forming a decision boundary such that Aurora maintains the original output of 3.57 in the blue region and changes in the red area. This allows operators to intuitively see that model decisions will conform to the expected behavior within a certain range of features. When making judgments near the boundary, operators decide based on previous decisions to avoid abrupt changes in transmission rate, ensuring smoothness in the rate selection process. Besides, it also provides operators with ideas for understanding the model: for example, the model decision in Aurora is highly tolerant of *latency ratio* when *latency inflation* $= -0.2$ compared to -0.4.

Fig. 5 shows the results of sensitivity analysis. While feature importance analyzes the feature contribution under collaborative perturbation of all features, sensitivity analysis focuses on individual features each time. We can find that *send ratio* is the most sensitive feature in Aurora in the current case. Therefore, the feature *send ratio* should be given special attention. Likewise, it can be seen that *previous bit rate* is less responsible for the observed behavior. The remaining features are equally important in contributing to the model, small changes in these features can cause large fluctuations in the model's decision.

Discussion. The current experiments evaluate the performance of basic network structures, but for deeper and wider models, *UINT* may suffer from the limitations of longer time. We leave the optimization to future work.

VI. Conclusion

In this paper, we design a framework *UINT* that unifies the verification and interpretation of learning-based networking systems using SMT. We give the generic formulations to verify robustness properties and solve a wide range of interpretability problems. *UINT* automatically encodes machine learning models as SMT formulas and automatically solves operator-specified verification and interpretation issues using an SMT solver. Extensive experiments on real-world systems demonstrate the efficiency and efficacy of *UINT*.

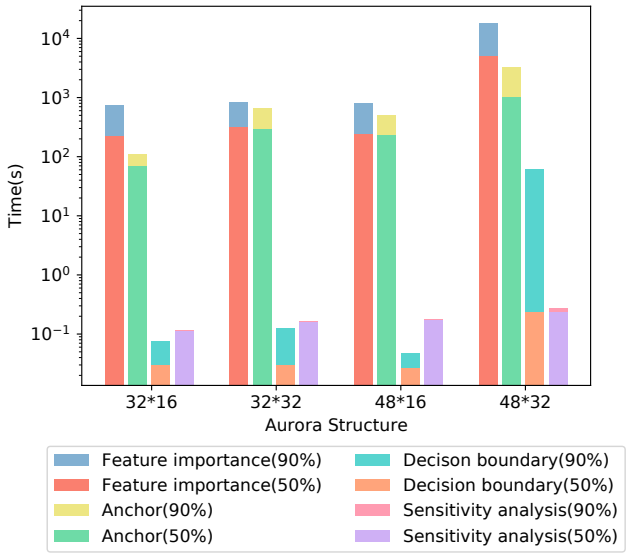

Fig. 3: Interpretability running time distribution for Aurora

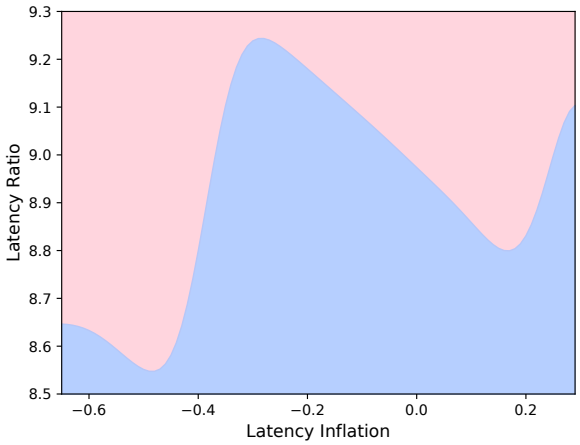

Fig. 4: Decison boundary for Aurora (32*16)

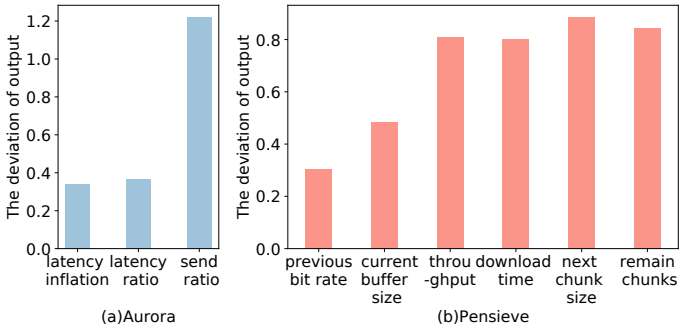

Fig. 5: Sensitivity analysis for Aurora and Pensieve

ACKNOWLEDGMENTS

The authors are extremely grateful for the anonymous IWQoS23 reviewers for their wonderful feedback. The authors thank Li Chen, Xi Chen, Chen Ma, Ruzica Piskac and Yibo Zhu for their suggestions during the preparation of this paper. Qiao Xiang, Yangfan Huang, Yuling Lin, and Haohao Song are supported in part by the National Key R&D Program of China 2022YFB2901502, Open Research Projects of Zhejiang Lab 2022QA0AB05, NSFC Award 62172345, Alibaba Innovative Research Award, MOE China Award 2021FNA02008, NSF-Fujian-China 2022J01004 and IKKEM Award HRTP-2022-34.

REFERENCES

[1] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *CVPR*. IEEE Computer Society, 2016, pp. 770–778.

[2] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to Sequence Learning with Neural Networks," in *NIPS, pages = 3104–3112, year = 2014*.

[3] J. B. Schafer, D. Frankowski, J. L. Herlocker, and S. Sen, "Collaborative Filtering Recommender Systems," ser. Lecture Notes in Computer Science, vol. 4321. Springer, 2007, pp. 291–324.

[4] H. Mao, R. Netravali, and M. Alizadeh, "Neural Adaptive Video Streaming with Pensieve," in *SIGCOMM 2017*.

[5] N. Jay, N. Rotman, B. Godfrey, M. Schapira, and A. Tamar, "A deep reinforcement learning perspective on internet congestion control," in *ICML*. PMLR, 2019, pp. 3050–3059.

[6] R. Li, X. Xiao, S. Ni, H. Zheng, and S. Xia, "Byte segment neural network for network traffic classification," in *2018 IEEE/ACM 26th International Symposium on Quality of Service (IWQoS)*, pp. 1–10.

[7] S. Huang, C. Fung, K. Wang, P. Pei, Z. Luan, and D. Qian, "Using recurrent neural networks toward black-box system anomaly prediction," in *2016 IEEE/ACM 24th International Symposium on Quality of Service (IWQoS)*, 2016, pp. 1–10.

[8] R. Guidotti, A. Monreale, S. Ruggieri, F. Turini, F. Giannotti, and D. Pedreschi, "A survey of methods for explaining black box models," *ACM Computing Surveys*, vol. 51, 8 2018.

[9] L. H. Gilpin, D. Bau, B. Z. Yuan, A. Bajwa, M. Specter, and L. Kagal, "Explaining explanations: An overview of interpretability of machine learning," in *DSAA*. IEEE, 2018, pp. 80–89.

[10] G. Katz, D. A. Huang, D. Ibeling, K. Julian, C. Lazarus, R. Lim, P. Shah, S. Thakoor, H. Wu, A. Zeljic, D. L. Dill, M. J. Kochenderfer, and C. W. Barrett, "The Marabou Framework for Verification and Analysis of Deep Neural Networks," in *CAV*, vol. 11561, 2019, pp. 443–452.

[11] A. Dethise, M. Canini, and N. Narodytska, "Analyzing Learning-Based Networked Systems with Formal Verification," in *INFOCOM*, 2021, pp. 1–10.

[12] G. Katz, C. W. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer, "Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks," in *CAV*, R. Majumdar and V. Kuncak, Eds., vol. 10426, 2017, pp. 97–117.

[13] Z. Meng, M. Wang, J. Bai, M. Xu, H. Mao, and H. Hu, "Interpreting deep learning-based networking systems," in *SIGCOMM*, 2020, pp. 154–171.

[14] L. Pulina and A. Tacchella, "Challenging smt solvers to verify neural networks," *Ai Communications*, vol. 25, no. 2, pp. 117–135, 2012.

[15] F. Leofante, N. Narodytska, L. Pulina, and A. Tacchella, "Automated Verification of Neural Networks: Advances, Challenges and Perspectives," *CoRR*, vol. abs/1805.09938, 2018.

[16] X. Huang, M. Kwiatkowska, S. Wang, and M. Wu, "Safety verification of deep neural networks," in *ICCAV*. Springer, 2017, pp. 3–29.

[17] V. Tjeng and R. Tedrake, "Verifying neural networks with mixed integer programming," *CoRR*, vol. abs/1711.07356, 2017.

[18] O. Bastani, Y. Ioannou, L. Lampropoulos, D. Vytiniotis, A. Nori, and A. Criminisi, "Measuring neural net robustness with constraints," *NIPS*, vol. 29, 2016.

[19] M. T. Ribeiro, S. Singh, and C. Guestrin, "Anchors: High-Precision Model-Agnostic Explanations," in *AAAI*, 2018, pp. 1527–1535.

[20] K. Poularakis, Q. Qin, F. Le, S. Kompella, and L. Tassiulas, "Generalizable and Interpretable Deep Learning for Network Congestion Prediction," in *ICNP*. IEEE, 2021, pp. 1–10.

[21] R. Beckett, A. Gupta, R. Mahajan, and D. Walker, "A general approach to network configuration verification," in *SIGCOMM*. ACM, 2017, pp. 155–168.

[22] A. El-Hassany, P. Tsankov, L. Vanbever, and M. Vechev, "Netcomplete: Practical network-wide configuration synthesis with autocompletion," in *NSDI*, 2018, pp. 579–594.

[23] T. Eliyahu, Y. Kazak, G. Katz, and M. Schapira, "Verifying learning-augmented systems," in *SIGCOMM 2021*, 2021, pp. 305–318.

[24] A. Datta, M. Fredrikson, K. Leino, K. Lu, S. Sen, and Z. Wang, "Machine learning explainability and robustness: connected at the hip," in *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, 2021, pp. 4035–4036.

[25] V. Tjeng, K. Xiao, and R. Tedrake, "Evaluating robustness of neural networks with mixed integer programming," *arXiv preprint arXiv:1711.07356*, 2017.

[26] Y. Zhang, P. Tio, A. Leonardis, and K. Tang, "A Survey on Neural Network Interpretability," *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. 5, no. 5, pp. 726–742, 2021.

[27] A. Dethise, M. Canini, and S. Kandula, "Cracking open the black box: What observations can tell us about reinforcement learning agents," in *Proceedings of the 2019 Workshop on Network Meets AI & ML*, 2019, pp. 29–36.

[28] C. Molnar, *Interpretable Machine Learning*, 2nd ed., 2022. [Online]. Available: https://christophm.github.io/interpretable-ml-book.

[29] S. Wachter, B. Mittelstadt, and C. Russell, "Counterfactual explanations without opening the black box: Automated decisions and the GDPR," *Harv. JL & Tech.*, vol. 31, p. 841, 2017.

[30] R. Guidotti, A. Monreale, S. Ruggieri, D. Pedreschi, F. Turini, and F. Giannotti, "Local rule-based explanations of black box decision systems," *arXiv preprint arXiv:1805.10820*, 2018.

[31] A. B. Arrieta, N. Díaz-Rodríguez, J. Del Ser, A. Bennetot, S. Tabik, A. Barbado, S. García, S. Gil-López, D. Molina, R. Benjamins *et al.*, "Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI," *Information fusion*, vol. 58, pp. 82–115, 2020.

[32] L. De Moura and N. Bjørner, "Satisfiability modulo theories: introduction and applications," *Communications of the ACM*, vol. 54, no. 9, pp. 69–77, 2011.

[33] L. d. Moura and N. Bjørner, "Z3: An efficient SMT solver," in *ICTACAS*. Springer, 2008, pp. 337–340.

[34] E. M. Clarke and R. P. Kurshan, "Computer-aided verification," 1996.

[35] A. Gember-Jacobson, R. Shrestha, and X. Sun, "Localizing router configuration errors using minimal correction sets," *arXiv preprint arXiv:2204.10785*, 2022.

[36] A. Venzke and S. Chatzivasileiadis, "Verification of Neural Network Behaviour: Formal Guarantees for Power System Applications," *IEEE Transactions on Smart Grid*, vol. 12, no. 1, pp. 383–397, 2021.

[37] Y. Kazak, C. Barrett, G. Katz, and M. Schapira, "Verifying deep-RL-driven systems," in *Proceedings of the 2019 workshop on network meets AI & ML*, 2019, pp. 83–89.

[38] Z. C. Lipton, "The mythos of model interpretability: In machine learning, the concept of interpretability is both important and slippery." *Queue*, vol. 16, no. 3, pp. 31–57, 2018.

[39] J. Townsend, T. Chaton, and J. M. Monteiro, "Extracting relational explanations from deep neural networks: A survey from a neural-symbolic perspective," *IEEE transactions on neural networks and learning systems*, vol. 31, no. 9, pp. 3456–3470, 2019.

[40] M. Jose and R. Majumdar, "Cause clue clauses: error localization using maximum satisfiability," *ACM SIGPLAN Notices*, vol. 46, no. 6, pp. 437–446, 2011.

[41] J. L. Bentley, "Multidimensional binary search trees used for associative searching," *Communications of the ACM*, vol. 18, no. 9, pp. 509–517, 1975.

[42] C. Lee and D. Landgrebe, "Decision boundary feature extraction for neural networks," *IEEE Transactions on Neural Networks*, vol. 8, no. 1, pp. 75–83, 1997.

[43] Z. Wang, M. Fredrikson, and A. Datta, "Robust models are more interpretable because attributions look normal," *arXiv preprint arXiv:2103.11257*, 2021.

[44] M. Chapman-Rounds, U. Bhatt, E. Pazos, M.-A. Schulz, and K. Georgatzis, "FIMAP: feature importance by minimal adversarial perturbation," in *AAAI'2021*, vol. 35, no. 13, pp. 11 433–11 441.

[45] UINT Framework. https://github.com/sngroup-xmu/UINT, 2023.