QuantifyML: How good is my machine learning model?

Muhammad Usman
University of Texas at Austin, USA
muhammadusman@utexas.edu

Divya Gopinath
KBR Inc., CMU, Nasa Ames
divya.gopinath@nasa.gov

Corina S. Păsăreanu
KBR Inc., CMU, Nasa Ames
corina.s.pasareanu@nasa.gov

The efficacy of machine learning models is typically determined by computing their accuracy on test data sets. However, this may often be misleading, since the test data may not be representative of the problem that is being studied. With *QuantifyML* we aim to *precisely* quantify the extent to which machine learning models have learned and generalized from the given data. Given a trained model, *QuantifyML* translates it into a C program and feeds it to the CBMC model checker to produce a formula in Conjunctive Normal Form (CNF). The formula is analyzed with off-the-shelf model counters to obtain precise counts with respect to different model behavior. *QuantifyML* enables i) evaluating learnability by comparing the counts for the outputs to ground truth, expressed as logical predicates, ii) comparing the performance of models built with different machine learning algorithms (decision-trees vs. neural networks), and iii) quantifying the safety and robustness of models.

1 Introduction

Recent years have seen a surge in the use of machine learning algorithms in a variety of applications to analyze and learn from huge amounts of data. For instance, decision-trees are a popular class of supervised learning techniques that can learn easily interpretable rules from data. They have found success in areas such as medical diagnosis and credit scoring [32, 6]. Another example is Deep Neural Networks (DNN), which have gained popularity in diverse fields such as banking, health-care, image and speech recognition, as well as perception in self-driving cars [21, 30].

Machine learning models are typically evaluated statistically, by computing their *accuracy* on test datasets, to determine how well a model learned and generalized from the training data. However, this is an imperfect measure, as the train and test sets may not cover well the inputs space. Furthermore, it is often not clear which learning algorithm or trained model is better suited for a particular problem (e.g., neural networks vs. decision trees), and simply comparing the accuracy of different models may lead to misleading results. It may also be the case that well-trained models may be vulnerable to *adversarial inputs* [40, 37, 23] or they may violate desired *safety* properties [36]. However it is unclear how to quantify the *extent* to which these vulnerabilities affect the performance of a model, as evaluating the model on the available test or adversarial datasets may again give imprecise results.

We present an approach, *QuantifyML*, that aims to *precisely quantify* the learnability, safety and robustness of machine learning models.

A trained model is translated into a C program, enabling the application of the CBMC tool [31] to translate it to the Conjunctive Normal Form (CNF), which in turn enables the application of state-of-the-art approximate and exact model counters [19, 8, 18] to obtain precise counts w.r.t different output classes. *QuantifyML* enables i) evaluating the learnability of models by comparing the counts for the outputs to ground truth (expressed as logical predicates, if available), ii) comparing the performance of different models that may be built with different machine learning algorithms (e.g., decision-trees vs. neural networks), and iii) quantifying the safety and robustness of neural network models. Our evaluation demonstrates these applications of *QuantifyML* on decision-tree and neural network classifiers for the

Submitted to:
FMAS 2021

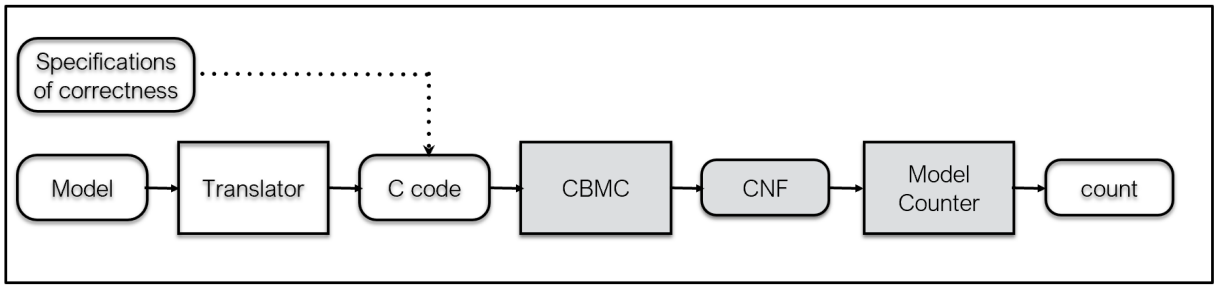

Figure 1: *QuantifyML* Framework

problems of learning relational properties of graphs, image classification and aircraft collision avoidance system.

We derive inspiration from the recent paper [41] which presents *Model Counting meets Machine Learning (MCML)* to evaluate the learnability of binary decision trees. *QuantifyML* generalizes MCML by providing a generic tool-set that can handle more realistic multi-class problems, such as decision trees with non-binary inputs and with more than two output decisions, and also neural networks. Other learning algorithms can be accommodated in our approach provided that the learned models can be translated into C programs. *QuantifyML*'s applications extend beyond MCML and include: (i) comparison of the performance of different models, built with different learning algorithms, (ii) quantification of robustness in image classifiers, and (iii) quantification of safety of neural network models, as we demonstrate in this paper.

2 Approach

Figure 1 shows the overall framework of *QuantifyML*. The input is the trained machine learning (ML) model, i.e., a Decision Tree or a Neural Network. The *ML to C* translator will generate the C program representation of the machine learning model. In applications where the ground-truth or an oracle of correctness is available, it is also encoded as a function in C. The C file also contains assertions encoding various checks involving the outputs of the model and of the ground-truth predicates. The final C program is then converted into CNF form using the CBMC tool. The obtained CNF file is then fed to an off-the-shelf model counter, allowing us to quantify the quality of the model.

2.1 Translation of Decision Trees to C Programs

We illustrate the translation via an example. Figure 3 shows the C program representation of the decision tree which is trained on the reflexive property of graphs, i.e., all nodes in the graph have a self loop (all diagonal elements in the adjacency matrix should be 1). The input is an array of size 25 (since the tree is trained on graphs with 5 nodes – represented using an adjacency matrix of size 25). Function *decisiontree* computes and returns the classification label for the given input. The function returns label 0 if any of the diagonal elements is 0 otherwise it returns label 1. The *ML to C* translator currently supports decision trees trained using the Scikit-Learn [38] machine learning library. We plan to add support for other machine learning libraries in the future.

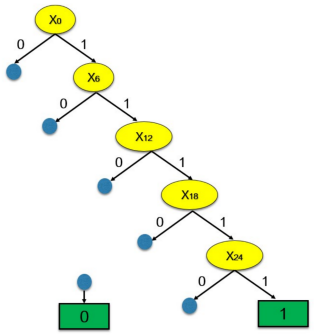

Figure 2: Decision Tree for Reflexive Property

```
1  int decisiontree (bool x[]){
2      if (x[0] == 0){ return (0);}
3      else if (x[6] == 0) { return (0); }
4      else if (x[12] == 0){ return (0); }
5      else if (x[18] == 0){ return (0); }
6      else if (x[24] == 0){ return (0); }
7      else { return (1); }  }
```

Figure 3: C code for the decision tree from Figure 2

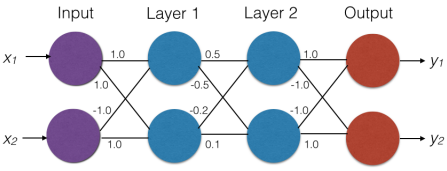

Figure 4: Neural Network

```
1  int neuralnetwork(float x1,float x2){
2     float layer1[2];
3     layer1[0]=(+1.0)*x1+(-1.0)*x2;
4     layer1[1]=(+1.0)*x1+(+1.0)*x2;
5     float layer2[2];
6     layer2[0]=(+0.5)*layer1[0]+(-0.20)*layer1[1];
7     layer2[1]=(-0.5)*layer1[0]+(+0.10)*layer1[1];
8     float output[2];
9     output[0]=(+1.0)*layer2[0]+(-1.0)*layer2[1];
10    output[1]=(-1.0)*layer2[0]+(+1.0)*layer2[1];
11    if(output[0]>output[1]) { return 0;}
12    else { return 1; } }
```

Figure 5: C code for the neural network from Figure 4

2.2 Translation of Neural Networks to C Programs

QuantifyML can handle feed-forward neural networks with dense, convolutional, and pooling layers, with ReLU activations and Softmax functions. Figure 5 shows the C program representation of the example neural network shown in Figure 4. Function *neuralnetwork* computes and returns the classification label for the given input. There are two inputs to the function i.e., x1 and x2. Line 2 declares an array, *layer1*, of dimension 2 (layer 1 has 2 nodes) that will store the values of the nodes at layer 1. Line 3 and 4 computes the value of nodes at layer 1 by multiplying the inputs with the weights on the respective edges. Line 5 declares an array, *layer2*, of dimension 2 (layer 2 has 2 nodes) that will store the values of the nodes at layer 2. Line 6 and 7 computes the value of nodes at layer 2 by multiplying the values of nodes at layer 1 with the weights on the respective edges. Line 8 declares an array, *output*, of dimension 2 (output layer has 2 nodes) that will store the values of the nodes at output layer. Line 9 and 10 computes the value of nodes at output layer by multiplying the values of nodes at layer 2 with the weights on the respective edges. Line 11 - 12 returns label 0 if the value of first node at output layer (y1) is greater than the value of the second node at the output layer (y2), otherwise it returns 1. The *ML to C* translator currently supports neural networks trained in Keras [29] machine learning library. We plan to add support for other machine learning libraries in the future.

2.3 Quantifying the learnability of machine learning models

Given the C program representations of the model and the oracle encoding the ground-truth, *QuantifyML* employs the CBMC tool to generate formulas encoded in the CNF form. The CNF files are then fed to model counters to obtain counts which are in turn used to calculate the following metrics, to quantify the learnability of the models, given finite bounds on the input space.

For each output label l of the classifier, *QuantifyML* computes the following: i) *True Positives (TP)* denotes the portion of the inputs for which the ground truth is the label l and the classifier model correctly predicts the label to be l, ii) *False Positives (FP)* denotes the portion of the inputs for which the ground truth is not the label l and the classifier model incorrectly predicts the label to be l, iii) *True Negatives (TN)* denotes the portion of the inputs for which the ground truth is not the label l and the classifier model correctly predicts a different label than l, and iv) *False Negatives (FN)* denotes the portion of the inputs for which the ground truth is the label l and the classifier model incorrectly predicts a different label than l. It then uses these counts to assess the quality of the model using standard measures such as *Accuracy*, *Precision*, *Recall* and *F1-score* for the model. *Accuracy*=$\frac{TP+TN}{TP+FP+TN+FN}$, Precision=$\frac{TP}{TP+FP}$, *Recall*= $\frac{TP}{TP+FN}$ *and* F1-score= $\frac{2*Precision*Recall}{Precision+Recall}$.

In order to compute these measures for each label l, a predicate function in C is generated ($\phi_l(x)$) which returns 1 if the output of the model is l for a given input x and returns 0 otherwise. Similarly, a predicate function ($\psi_l(x)$) is generated which returns 1 if the ground-truth for the given input x is l and returns 0 otherwise. Given these predicates, four CNF files are generated for each label l, which encode the following formulas corresponding to the counts. N is the scope or bound on the input domain, *CNF* represents a function that translates C program to formulas in the CNF form, and *MC* represents a function that uses the projected model counter to return the number of solutions projected to the input variables.

- *True Positives (TP):* $MC(CNF(\psi_l(x) \wedge \phi_l(x)), N)$
- *False Positives (FP):* $MC(CNF(\neg\psi_l(x) \wedge \phi_l(x)), N)$
- *True Negatives (TN):* $MC(CNF(\neg\psi_l(x) \wedge \neg\phi_l(x)), N)$
- *False Negatives (FN):* $MC(CNF(\psi_l(x) \wedge \neg\phi_l(x)), N)$

CBMC generates the above mentioned set of CNF files which are then passed to an off-the-shelf model counter and final counts are obtained. Performance metrics as explained above are then calculated. CBMC introduces auxiliary variables so only model counters that support projected model counting should be used. We use two model counters, ApproxMC and projMC, for the experiments in this paper.

2.4 Quantifying the safety of machine learning models

Given a property p that a machine learning model (example a neural network) needs to satisfy, we use *QuantifyML* to obtain the following counts; i) $QuantifyML_S$ denoting the portion of the inputs for which the model satisfies the given property, and ii) $QuantifyML_N$ denoting the portion of the inputs for which the model violates the property. These counts are then used to obtain an accuracy metric; $QuantifyML_{Acc}$=$\frac{QuantifyML_S}{QuantifyML_N+QuantifyML_S}$, which is a measure of the extent to which the network satisfies the property.

2.5 Quantifying Local Robustness

As mentioned, *QuantifyML* is more general than *MCML*, which is applicable only to decision trees where both inputs and decisions are binary. *QuantifyML* can be applied to more challenging models, which cannot be described with binary decision trees. The challenge with the analysis of more realistic models is that we typically do not have the ground truth. Image classification is such a problem.

Consider that given an image which contains an object of interest such as a digit, the goal is to classify it with a label that identifies the digit. It is not feasible to define a specification that can automatically generate the correct label or the ground truth for any arbitrary image. Machine learning models are typically trained using a dataset of images that are manually labelled. However, images that are similar to, or are in close proximity (in terms of distance in the input space) to an image with a known label can be expected to have the same label. This property is called *robustness* in the literature. For instance, given an image of a certain digit, all images that can be generated by applying perturbations that are imperceptible to the human eye should represent the same digit. Therefore, we know the ground truths for the inputs in the neighborhood of user labelled inputs.

Evaluating the robustness of image classification models is an active area of research [20, 1, 10]. Current techniques typically search for the existence of an adversarial input (x') within an ε ball surrounding a labelled input (x); e.g., $||x-x'||_\infty \leq \varepsilon$ (here the distance is in terms of the L_∞ metric) such that the output of the model on x and x' is different. When no such input exists, the model is declared robust, however, in the presence of an adversarial input there is no further information available. We propose to use *QuantifyML* to *quantify* robustness of machine learning models, where instead of using a predicate encoding the ground truth, we encode the local robustness requirement that the model should give the same output within the region defined by $||x-x'||_\infty \leq \varepsilon$.

In order to quantify local robustness around a concrete n-dimensional input $x = (x_0, x_1, ..x_n)$, we first define an input region R_ε by constraining the inputs across each dimension to be within $[x_i - \varepsilon, x_i + \varepsilon]$ in the translated C program. We then define $Robustness_\varepsilon$ as $\frac{MC(CNF(\phi_l(x)),R_\varepsilon)}{|R_\varepsilon|}$, where $\phi_l(x)$ is defined as before as a predicate which returns 1 if the output of the model is l and 0 otherwise, R_ε defines the scope for the check, and $|R_\varepsilon|$ quantifies its size. Intuitively, $Robustness_\varepsilon$ quantifies the portion of the input on which the model is robust, within the small region described by R_ε.

3 Evaluation

We seek to address the following research questions in our evaluation.

- **RQ 1:** How does *QuantifyML* perform when applied to the problem of evaluating the true performance of a model in comparison with the ground truth?
- **RQ 2:** Does *QuantifyML* enable a comparison of different models, that are built using different learning techniques (decision-tree vs. neural networks)?
- **RQ 3:** How does *QuantifyML* perform when applied to the problem of quantifying adversarial robustness for image classification models?
- **RQ 4:** How does *QuantifyML* perform when applied to the problem of quantifying the safety of machine learning classification models?

All experiments were performed on Windows 10 with an Intel Core-i7 8750H CPU (2.20 GHz) and 16GB RAM. Appendix along with detailed results are available at the project's GitHub repository[1].

[1] `https://github.com/muhammadusman93/quantifyml`

3.1 RQ 1: Evaluating the true performance of the models.

In the first set of experiments we seek to evaluate the performance of trained models against ground truth, encoded as predicates (as described in Section 2). For this set of experiments we use the same set up as in [41], allowing us to also compare the performance of MCML with *QuantifyML*. We recap that setup here.

Datasets: We consider the problem of learning relational properties of graphs. Alloy [26] was used to create datasets for 11 relational properties of graphs including Antisymmetric, Connex, Equivalence, Irreflexive, NonStrictOrder, PartialOrder, PreOrder, Reflexive, StrictOrder, TotalOrder and Transitive. Alloy generated datasets containing positive solutions i.e., graphs which satisfy a given relational property and negative solutions i.e., graphs which violate the given property. The number of negative solutions is much larger than the number of positive solutions. Table 8 in Appendix tabulates the total number of positive and negative solutions for each property. It is infeasible to enumerate all negative solutions. For example, for the reflexive property (graphs in which all nodes have a self-loop) with a scope of 6, there are only around 1 million graphs (1048576) that satisfy the property out of the 68.7 billion (2^{36}) possible graphs. We use Alloy to enumerate all positive solutions and then create an equal number of negative solutions. To create the set of negative solutions, we first create a random solution and verify if it violates the property using the Alloy evaluator. If the solution is a graph satisfying the property then it is discarded otherwise it is added to the set of negative solutions. Each solution has a feature vector and a binary label (1 for the positive class and 0 for the negative class). The features are represented using an adjacency matrix. For example, if the scope of the reflexive property is 6, then a 36 bit vector is used. The smallest scope was chosen for each property such that there are $\geq$ 90,000 positive solutions.

Although, Alloy was used in the generation of datasets, the generated datasets are not dependent on Alloy. Please note that for positive solutions, all possible solutions were generated. The same set of solutions will be generated by any SAT solver (Representation format can be different). For negative solutions, we randomly created them and confirmed them using the Alloy evaluator which simply evaluates if the given solution is positive or negative. Therefore, there is no bias or dependence in using Alloy.

For the decision trees, the quality of the split (criterion) was measured using *gini* and rest of the hyper-parameters were set to default settings.

Table 1: Quantifying the learnability of Decision Trees on graph properties with *projMC*. Metrics from *QuantifyML* (*QML*), *MCML* and Statistical calculation on test set(*Stat*). "-" indicates a time-out of 5000 seconds.

Property	*Accuracy*			*Precision*			*Recall*			*F1-score*		
	Stat	*MCML*	*QML*	*Stat*	*MCML*	*QML*	*Stat*	*MCML*	*QML*	*Stat*	*MCML*	*QML*
Antisymmetric	0.9997	0.9996	0.9996	0.9996	0.9938	0.9938	0.9998	0.9998	0.9998	0.9997	0.9968	0.9968
Connex	0.9957	0.9934	0.9934	0.9933	0.2106	0.2106	0.9982	0.9984	0.9984	0.9957	0.3478	0.3478
Equivalence	0.9997	-	-	0.9994	-	-	1.0000	1.0000	1.0000	0.9997	-	-
Irreflexive	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
NonStrictOrder	0.9990	0.9984	-	0.9985	0.0012	-	0.9994	0.9995	0.9995	0.9990	0.0024	-
PartialOrder	0.9934	0.9877	0.9877	0.9879	0.2473	0.2473	0.9991	0.9992	0.9992	0.9935	0.3964	0.3964
PreOrder	0.9985	0.9973	-	0.9974	0.0011	-	0.9996	0.9996	0.9996	0.9985	0.0022	-
Reflexive	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
StrictOrder	0.9988	0.9978	-	0.9979	0.0009	-	0.9997	0.9998	0.9998	0.9988	0.0018	-
TotalOrder	0.9999	-	-	0.9997	-	-	1.0000	1.0000	-	0.9999	-	-
Transitive	0.9999	0.9764	0.9764	0.9997	0.1622	0.1622	1.0000	0.9913	0.9913	0.9999	0.2789	0.2789

Results: Table 1 presents the *QuantifyML* and *MCML* metrics (accuracy, precision, recall and F1-score) for the decision-tree models for the respective properties. These were calculated using the projMC

Table 2: Efficiency comparison *QuantifyML* vs. MCML. Number of primary variables (*Prim. Vars*) are same for MCML and *QuantifyML* (*QML*), Total number of *Clauses* for true positive (TP), false negative (FN), false positive (FP), and true negative (TN) in the CNF formulae. Times taken by ApproxMC and projMC model counters. "-" indicates time-out of 5000 seconds

Property	*Prim*	*TP Clauses*		*FN Clauses*		*FP Clauses*		*TN Clauses*		*Time (s) - projMC*		*Time (s) - ApproxMC*	
	Vars	*MCML*	*QML*	*MCML*	*QML*	*MCML*	*QML*	*MCML*	*QML*	*MCML*	*QML*	*MCML*	*QML*
Antisymmetric	25	1015	86364	1031	86540	1185	86427	1201	86603	1.5	121.5	2.1	15.8
Connex	25	69	7643	74	7508	231	4301	236	4236	0.3	4.8	0.8	0.4
Equivalence	100	2591	44644	2582	44617	3349	40562	3340	40546	-	-	34.1	30.1
Irreflexive	25	10	172	6	208	147	142	143	162	0.0	0.0	0.5	0.2
NonStrictOrder	36	507	20336	489	20520	624	14303	606	14424	12.2	-	1.9	4.6
PartialOrder	25	514	23563	435	23674	576	22399	497	22499	0.8	1404.7	1.2	2.9
PreOrder	36	528	22850	516	22892	659	17158	647	17188	34.5	-	2.0	3.8
Reflexive	25	10	227	6	207	98	177	94	165	0.0	0.0	0.5	0.1
StrictOrder	36	480	16924	464	17212	611	12647	595	12782	14.1	-	1.9	2.8
TotalOrder	81	1834	39521	1819	39778	2331	42961	2316	43033	-	-	8.4	8.9
Transitive	25	774	43456	672	43534	862	43544	760	43622	14.9	2377.5	2.0	8.4

Table 3: Quantifying the learnability of Decision Trees on graph properties with *ApproxMC*. *Diff* shows the difference between MCML and *QuantifyML* (*QML*) metrics.

Property	*Accuracy*			*Precision*			*Recall*			*F1-score*		
	MCML	*QML*	*Diff*	*MCML*	*QML*	*Diff*	*MCML*	*QML*	*Diff*	*MCML*	*QML*	*Diff*
Antisymmetric	0.9996	0.9996	0.0000	0.9935	0.9936	-0.0001	0.9998	0.9998	0.0000	0.9967	0.9967	0.0000
Connex	0.9935	0.9936	-0.0001	0.2258	0.2292	-0.0032	0.9985	0.9985	0.0000	0.3683	0.3728	-0.0045
Equivalence	0.9995	0.9995	0.0000	0.0000	0.0000	0.0000	1.0000	1.0000	0.0000	0.0000	0.0000	0.0000
Irreflexive	1.0000	1.0000	0.0000	1.0000	1.0000	0.0000	1.0000	1.0000	0.0000	1.0000	1.0000	0.0000
NonStrictOrder	0.9983	0.9985	-0.0002	0.0011	0.0012	-0.0001	0.9995	0.9995	0.0000	0.0022	0.0024	-0.0002
PartialOrder	0.9864	0.9880	-0.0016	0.2407	0.2535	-0.0128	0.9992	0.9992	0.0000	0.3879	0.4045	-0.0166
PreOrder	0.9972	0.9973	-0.0001	0.0012	0.0012	0.0000	0.9997	0.9997	0.0000	0.0024	0.0024	0.0000
Reflexive	1.0000	1.0000	0.0000	1.0000	1.0000	0.0000	1.0000	1.0000	0.0000	1.0000	1.0000	0.0000
StrictOrder	0.9979	0.9979	0.0000	0.0009	0.0010	-0.0001	0.9998	0.9998	0.0000	0.0019	0.0020	-0.0001
TotalOrder	0.9997	0.9997	0.0000	0.0000	0.0000	0.0000	1.0000	0.0017	0.9983	0.0000	0.0000	0.0000
Transitive	0.9760	0.9773	-0.0013	0.1588	0.1826	-0.0238	0.9902	0.9922	-0.0020	0.2737	0.3085	-0.0348

model counter which is an exact model counter. Please note that we applied both techniques to the same model for each property. As can be seen, the values of the metrics obtained by *QuantifyML* and *MCML* match exactly for all the properties. This is as expected since both the tools seek to obtain precise quantification over the input space. *QuantifyML* is more general and uses a C program representation of any machine learning model and the CBMC tool to perform the translation to CNF form, whereas *MCML* has an implementation that is dedicated to decision-trees. These results serve as a check for the correctness of *QuantifyML* and show that the tool produces exactly the same results as the technique that is dedicated to decision-trees.

However, *QuantifyML* is less efficient than *MCML*. Table 2 shows the times taken by *QuantifyML* and *MCML* using projMC (*Time(s) projMC* column). It can be observed that both *MCML* and *QuantifyML* time out (after 5000 seconds) for TotalOrder and Equivalence properties. *QuantifyML* times out for the following properties as well; NonStrictOrder, StrictOrder and PreOrder. This is because the CNF formulas generated by the CBMC tool after the analysis of the C program representation of the machine learning model is larger than that produced by *MCML*, which has a custom implementation for decision-trees. Table 2 shows the size of the CNF formulas in terms of primary variables and clauses. projMC is an exact model counter and takes a long time to process formulas with large number of clauses.

To alleviate this scalability problem, we experimented with another model counter, namely Ap-

proxMC, which is faster but produces approximate counts. The column *Time(s) ApproxMC* in Table 2 shows the times and Table 3 presents the respective metrics for both *QuantifyML* and *MCML* using ApproxMC. As it can be observed, the analysis times for *QuantifyML* are greatly reduced and we are able to obtain results for all the properties. Due to the approximate nature of the model counter, there are minor differences in the metrics computed with *QuantifyML* vs. *MCML*. However, these differences are minor (as can be seen in the *Diff* columns of the table). In some cases, *QuantifyML* is closer to the true values (presented in table 1), while in others *MCML* is closer, however, the approximate results are within +/- 15% of the true results. [45] empirically determined ApproxMC model counts to be +/- 16% of the true counts.

In order to understand the benefits of *QuantifyML* in assessing the learnability of models, we considered the metrics calculated statistically on a test set. We present the statistical metrics (accuracy, precision, recall and F1-scores) for the decision-tree models in Table 1. It can be seen that for all the properties, the statistical accuracy values are higher than the true values. This difference is more pronounced for the precision metric. For instance, for the Connex, PartialOrder and Transitive properties, while the statistical precision values are greater than 98%, the actual precision of these models are less than 25%. This highlights that statistical metrics could be misleading and give a false impression of good performance. The F1-score indicates the generalizability of the models. It can be seen that specifically for the above mentioned three properties, the true F1-scores are quite poor indicating that the models are possibly over-fitted to the respective train sets. However, this fact would be hidden if we rely only on the statistical metrics. We would like to highlight that even with approximate results obtained using ApproxMC, *QuantifyML* metrics still present a more accurate picture of the accuracy and generalizability of the models in comparison with the statistical metrics. Specifically, for the Equivalence and TotalOrder properties (Table 3), we can see that the number of false positives is very high leading to very low values for the true precision. The true F1-scores for these models are close to 0, however, the respective statistical F1-scores are close to 0.99, giving a false impression of good performance.

Tables 4 and 5 present *QuantifyML* and statistically calculated metrics for decision-tree and neural network models respectively for relational properties of small (4-node) graphs. We describe these models in more detail in the next section. We can observe a similar trend here as well that the statistical metrics indicate that the models have high accuracy and F1-scores for all properties. The decision-tree models (Table 4) for the Antisymmetric, Irreflexive, Reflexive, NonStrictOrder and PreOrder properties have statistical accuracy and the F1-scores of 100%. However, the precise quantification by *QuantifyML* highlights that for the NonStrictOrder and PreOrder properties, the models in fact have less than 100% accuracy and more importantly have poor precision indicating large number of false positives. The decision-trees for StrictOrder property seems to have the lowest accuracy and F1-score when calculated statistically. However, the *QuantifyML* scores highlight that this is mis-leading and in truth the decision-tree for the Connex Property has the lowest accuracy and F1-score. For the neural network models (Table 5), in all cases except Irreflexive and Reflexive properties, the accuracy values calculated using *QuantifyML* highlight that the true performance is mostly worse and in some cases better (PartialOrder, Transitive) than the respective statistical accuracy metric values. The precision and recall metrics seem to suffer greatly when calculated purely statistically (as can be seen from the *Diff* columns in the table 5). This indicates that they give a false impression of good generalizability of the respective models, while in truth the F1-scores are less than 50% for most of the properties (refer *F1-score* column in table 5).

3.2 RQ 2: Comparing the performance of different models.

For this set of experiments, we seek to evaluate *QuantifyML* on different models that are trained on the same problem. Consider the problem of learning relational properties of graphs as explained earlier in *RQ 1*. Although this seems a fairly simple problem with binary decisions and binary input features, it is not immediately apparent which learning algorithm would work best to learn a suitable classifier. The properties range from simple ones such as Reflexive to more complex ones such as StrictOrder and Connex (refer Appendix - section 6.1). Each property translates to a different set of features. While a decision-tree classifier may be faster to learn and easier to interpret, it may be overfitted. On the other hand, employing a deep neural network algorithm which relies on feature extraction may in fact not yield good results for this problem considering the inputs are purely binary.

Table 4: Quantifying the learnability of Decision Trees on graph (4-node) properties with *projMC*. *Diff* shows the difference between Statistical (*Stat*) and *QuantifyML* (*QML*) metrics.

Property	*Accuracy*			*Precision*			*Recall*			*F1-score*		
	Stat	*QML*	*Diff*	*Stat*	*QML*	*Diff*	*Stat*	*QML*	*Diff*	*Stat*	*QML*	*Diff*
Antisymmetric	1.0000	1.0000	0.0000	1.0000	1.0000	0.0000	1.0000	1.0000	0.0000	1.0000	1.0000	0.0000
Connex	0.9932	0.8179	-0.1752	0.9865	0.4219	-0.5646	1.0000	0.0625	-0.9375	0.9932	0.1089	-0.8843
Irreflexive	1.0000	1.0000	0.0000	1.0000	1.0000	0.0000	1.0000	1.0000	0.0000	1.0000	1.0000	0.0000
NonStrictOrder	1.0000	0.9721	-0.0279	1.0000	0.1069	-0.8931	1.0000	1.0000	0.0000	1.0000	0.1932	-0.8068
PartialOrder	0.9957	0.9919	-0.0038	0.9916	0.8690	-0.1226	1.0000	1.0000	0.0000	0.9958	0.9299	-0.0659
PreOrder	1.0000	0.9693	-0.0307	1.0000	0.1499	-0.8501	1.0000	1.0000	0.0000	1.0000	0.2607	-0.7393
Reflexive	1.0000	1.0000	0.0000	1.0000	1.0000	0.0000	1.0000	1.0000	0.0000	1.0000	1.0000	0.0000
StrictOrder	0.9545	0.9721	0.0175	0.9200	0.1069	-0.8131	1.0000	1.0000	0.0000	0.9583	0.1932	-0.7651
Transitive	0.9850	0.9799	-0.0051	0.9810	0.7524	-0.2285	0.9904	0.9990	0.0086	0.9856	0.8583	-0.1273

Table 5: Quantifying the learnability of Neural Network on graph (4-node) properties with *projMC*. *Diff* shows the difference between Statistical (*Stat*) and *QuantifyML* (*QML*) metrics.

Property	*Accuracy*			*Precision*			*Recall*			*F1-score*		
	Stat	*QML*	*Diff*	*Stat*	*QML*	*Diff*	*Stat*	*QML*	*Diff*	*Stat*	*QML*	*Diff*
Antisymmetric	0.8058	0.7614	-0.0445	0.7520	0.4211	-0.3309	0.9095	0.9093	-0.0002	0.8233	0.5756	-0.2476
Connex	0.9658	0.7866	-0.1791	0.9359	0.2326	-0.7033	1.0000	0.0865	-0.9135	0.9669	0.1261	-0.8408
Irreflexive	1.0000	1.0000	0.0000	1.0000	1.0000	0.0000	1.0000	1.0000	0.0000	1.0000	1.0000	0.0000
NonStrictOrder	0.9773	0.9054	-0.0719	0.9583	0.0338	-0.9245	1.0000	0.9909	-0.0091	0.9787	0.0654	-0.9133
PartialOrder	0.7803	0.8303	0.0500	0.8367	0.2002	-0.6364	0.7051	0.7260	0.0210	0.7652	0.3139	-0.4514
PreOrder	0.9577	0.8825	-0.0753	0.9302	0.0433	-0.8870	1.0000	0.9803	-0.0197	0.9639	0.0829	-0.8810
Reflexive	1.0000	1.0000	0.0000	1.0000	1.0000	0.0000	1.0000	1.0000	0.0000	1.0000	1.0000	0.0000
StrictOrder	0.9545	0.9409	-0.0136	0.9200	0.0535	-0.8665	1.0000	1.0000	0.0000	0.9583	0.1016	-0.8567
Transitive	0.7722	0.7903	0.0181	0.8063	0.1864	-0.6198	0.7404	0.7258	-0.0145	0.7719	0.2967	-0.4753

We applied the two different learning algorithms, decision-trees and neural networks, to learn classification models for the same set of properties using the same dataset for training. We were unable to apply *QuantifyML* to analyze neural network models with greater than 16 features due to the limitation in the scalability of the model counters (both ApproxMC and projMC were considered). Therefore we restricted the size of the graphs to have 4 nodes. The model counter times out after 10 hours when considering graphs with more than 4 nodes for all properties. However, with the reduction in the number of nodes in the graph, the number of *positive* graphs become very low for the Equivalence and TotalOrder properties. This hinders the learning of suitable classifiers. Therefore we consider only the remaining 9 properties for this study.

For the decision trees, the quality of the split (criterion) was measured using *gini* and rest of the

hyper-parameters were set to default settings. The neural networks has four layers. The first layer is a dense layer with 3 nodes. The second layer is an activation layer (*linear* activation function) with 3 nodes. The third layer is a dense layer with 2 nodes. The fourth layer is an activation layer (*softmax* activation function) with 2 nodes. The neural networks were trained up to 50 epochs with a batch size of 10. Adam optimizer was used.

Results: We applied *QuantifyML* to obtain the accuracy, precision, recall and F1-score metrics for the models learnt using both the techniques. These metrics were also calculated statistically on a test set (same for both the learning algorithms). Tables 4 and 5 present the results.

The models for the Irreflexive and Reflexive properties have 100% accuracy and F1-score, both when learnt using the decision-tree learning algorithm or as a neural network. These two are indeed very simple graph properties. However, decision-tree models have the ability to learn more complex properties such as Antisymmetric and StrictOrder as well. Overall the decision-tree models seem to have better accuracy and generalizability than the respective neural network models. Note, that while such a comparison can be done using the statistical metrics, their lack of precision may lead to wrong interpretations. For instance, for the StrictOrder property, the statistical accuracy, precision, recall and F1-scores are exactly the same for the neural network and decision-tree models, however, the corresponding *QuantifyML* metrics highlight that for this problem, the decision-tree models are in fact better than neural networks in terms of the true performance.

Albeit using a simple example, this study highlights the benefit of *QuantifyML* in enabling one to compare the true performance of different models, and different learning algorithms, for a given problem. Note that as long as the models being compared are for the same application, they need not have been trained using the same set of data points.

3.3 RQ 3: Quantifying the robustness of image classifiers.

We use the MNIST dataset (Modified National Institute of Standards and Technology dataset), which is a large collection of handwritten digits that is commonly used for training various image processing systems [34]. MNIST has 60,000 training input images, each comprised of 784 pixels and belonging to one of 10 labels (digits 0 through 9). We trained a decision-tree model on this dataset. For the decision trees, the quality of the split (criterion) was measured using *entropy* and rest of the hyper-parameters were set to default settings. The overall accuracy of this model on the test set was 83.64%.

The size of the CNF files for this problem is large (refer Appendix - Table 15). This can be attributed to the 784 integer input variables and the logic for image classification which results in bigger trees. There are 25088 primary variables and around 31 million clauses in the CNF formulas for each label. It was not feasible to use the exact model counter projMC, therefore, we used the ApproxMC model counter for this study. Typically neural networks are used for the problem of image classification. However, both ApproxMC and projMC were not scalable enough to handle the CNF files generated from the neural networks that we trained for this study.

Results: We selected (randomly) an image for each of the 10 labels to evaluate the robustness of the model in the neighborhood of each of these inputs. We considered regions around these inputs for $\varepsilon = 1$; these represent all the inputs that can be generated by altering each pixel of the given image by +/- 1. Column *Total count*$_\varepsilon$ in Table 6 shows the total number of images in the $\varepsilon = 1$ neighborhood of each input. We then employ *QuantifyML* to precisely quantify the number of inputs within the $\varepsilon = 1$ neighborhood that are given the correct label. In order to obtain this count, for every input, we apply *QuantifyML* to quantify the portion of the input region that leads to the same label as the original in-

Table 6: Quantifying robustness for the MNIST model.

Actual Label	*Total* $count_\varepsilon$	*Correctly classified* $count_\varepsilon$	$Robustness_\varepsilon$ %	$Accuracy_\varepsilon$ *100*	$Accuracy_\varepsilon$ *1000*	$Accuracy_\varepsilon$ *10000*	*Accuracy (TestSet)*
0	3.32E+270	2.21E+270	66.67	65.00	67.90	67.05	92.65
1	9.59E+247	3.15E+247	32.81	34.00	32.10	32.66	96.12
2	3.53E+258	8.81E+257	25.00	32.00	25.90	25.30	83.91
3	1.92E+272	1.92E+272	99.99	93.00	93.70	93.62	77.52
4	3.42E+264	3.42E+264	99.99	52.00	61.50	63.58	82.08
5	4.02E+259	4.02E+259	99.99	100.00	100.00	100.00	76.23
6	1.04E+258	5.22E+257	50.00	47.40	47.40	50.21	85.39
7	1.17E+262	1.17E+262	99.99	100.00	100.00	100.00	85.60
8	9.99E+266	9.99E+266	99.99	100.00	100.00	100.00	73.72
9	1.84E+253	6.89E+252	37.50	38.20	38.20	38.12	80.67

put. Column *Correctly classified* $count_\varepsilon$ in Table 6 shows this count for each input. The corresponding $Robustness_\varepsilon$ value shows the accuracy with which the model classifies the inputs in the region to the same label. The results indicate that the robustness of the model is poor or the model is more vulnerable to attacks around the inputs corresponding to labels 1, 2 and 9 respectively. Note also that in principle, *QuantifyML* could be used to provide proofs of robustness (if $Robustness_\varepsilon$*=100%)*; however we could not obtain such proofs here.

We also computed an accuracy metric statistically by perturbing each image within ε to randomly generate sample sets of size 100, 1000 and 10000 images respectively. We then executed the model on each set to determine the corresponding labels and computed the respective accuracies as shown in columns $Accuracy_\varepsilon$*(100)*, $Accuracy_\varepsilon$*(1000)*, and $Accuracy_\varepsilon$*(10000)*. The statistically computed accuracies are close to the $Robustness_\varepsilon$ values for most of the labels. We can see that for labels 5,7 and 8, the statistical accuracies with all sample sets are 100% respectively. This gives a false impression of adversarial robustness around these inputs. The corresponding $Robustness_\varepsilon$ of 99.99% indicates that there are subtle adversarial inputs which get missed when the robustness is determined statistically. We can also observe an interesting case, for the input corresponding to label 4, where the sample sets considered for statistical accuracy seem to include a number of inputs that are incorrectly classified. $Robustness_\varepsilon$ of 99.99% indicates that only 0.01% of the inputs are incorrectly classified. However, these inputs comprise of almost 36% of the samples considered for statistical accuracy. This seems to indicate that this region is more vulnerable to attacks than the inputs corresponding to labels 5 and 8, while in truth, these regions are very similar in terms of robustness. It may be the case that this result for $Robustness_\varepsilon$ is incorrect, which may be due to the approximation in the model counter. We plan to investigate this issue further.

The last column, *Accuracy (TestSet)* in Table 6, shows the accuracy of the model per label when evaluated statistically on the whole MNIST test set. Note that although the model may have high statistical accuracy, it can have low adversarial robustness. Consider the input for label 1, although the statistical targeted accuracy is 96%, the $Robustness_\varepsilon$ around the input is only around 32% indicating poor adversarial robustness. We would like to highlight that although the above discussion is based on the results produced by ApproxMC which produces approximate counts, it has been shown in an earlier study [45] that the outputs are typically within 16% of the outputs from projMC (exact model counter). The results obtained using statistical measures typically have a much higher difference from the exact counts (as was highlighted in RQ1).

3.4 RQ 4: Quantifying the safety of neural network models.

ACAS Xu is a safety-critical collision avoidance system for unmanned aircraft control [36]. It receives sensor information regarding the drone (the *ownship*) and any nearby intruder drones, and then issues horizontal turning advisories aimed at preventing collisions. The input sensor data includes, i) distance from aircraft to intruder; (ii) angle to intruder relative to aircraft heading direction; (iii) heading angle of intruder relative to aircraft heading direction; (iv) speed of aircraft; and (v) speed of intruder. The output of the network is one of 5 labels corresponding to the horizontal turn advisories; Clear-of-Conflict (COC), weak right, strong right, weak left, and strong left. Each advisory is assigned a score, with the lowest score corresponding to the best action. The FAA is exploring an implementation of ACAS Xu that uses 45 deep neural networks. The work in [28] presents 10 properties specified by domain experts that the networks need to satisfy.

Table 7: Quantifying the safety of Neural Networks on ACASXu dataset. "-" shows a timeout of 5000 seconds (*ApproxMC*). Properties 1 - 9 represent properties ϕ_2 to ϕ_{10} from [28].

Property	$Stat_N$	$Stat_S$	$Stat_{Acc}(\%)$	$QuantifyML_N$	$QuantifyML_S$	$QuantifyML_{Acc}(\%)$	$QuantifyML_{Time}$ *(s)*
1	0	228	100.00	9.00E+93	2.50E+94	73.56	3347.1
2	0	0	N/A	5.67E+88	2.79E+88	32.94	4067.5
3	0	0	N/A	3.37E+67	1.32E+65	0.39	2791.8
4	0	0	N/A	1.18E+74	0.00E+00	0.00	2918.4
5	1	4062	99.98	0	2.25E+86	100.00	1005.2
6	5680	140563	96.12	-	6.67E+94	-	-
7	0	218	100.00	0.00E+00	8.15E+90	100.00	1753.5
8	0	1	100.00	4.24E+73	8.62E+74	95.31	2073.3
9	0	62	100.00	0.00E+00	4.17E+79	100.00	812.2

We used a dataset comprising of 324193 inputs and used one of the ACAS Xu network to obtain the labels for them. We used this dataset to train a smaller neural network that is amenable to a quantitative analysis. The neural network has four layers. The first layer is a dense layer with 10 nodes. The second layer is an activation layer (*relu* activation function) with 10 nodes. The third layer is a dense layer with 5 nodes. The fourth layer is an activation layer (*softmax* activation function) with 5 nodes. The neural network was trained for 50 epochs with a batch size of 10. Adam optimizer was used. The overall accuracy of this model on the test set was 96.0%. We selected 9 properties of ACAS Xu (refer to the Appendix section 1.3 for details on the properties) and employed our tool to evaluate the extent to which the smaller neural network model complies to each of them.

Results: Table 7 documents the results of our experiments. We first evaluated the compliance of the model to each property statistically on a test set of size 162096 inputs (randomly selected). Each property defines a pre-condition on the inputs and prescribes a specific output property for the respective input region (such as the output being a certain label). For each property, we select inputs from the test set that belong to the input region as defined in the property ($InpSet_{P\#}$). The column $Stat_N$ shows the subset of inputs in $InpSet_{P\#}$ that violate the property, $Stat_S$ shows the number of inputs in $InpSet_{P\#}$ that satisfies it, and $Stat_{Acc}$ shows the respective statistical accuracy. For each property, we calculate the *QuantifyML* metrics as described in section 2.4. The *QuantifyML* counts represent the precise portion of the entire input space defined by the property for which the property is satisfied or violated.

For properties 2, 3 and 4, there were no inputs in the test set that belonged to the input region as defined in the property, therefore the statistical accuracy could not be calculated, whereas we were able to use *QuantifyML* to evaluate the model on these properties. Results show that the neural network never satisfies property 4. This highlights the benefit of using our technique to obtain precise counts without

being dependent on a dataset of inputs. We can also note that relying on statistical accuracy can give false confidence in the performance of the neural network. For instance, the statistical scores show that the model always satisfies property 1, 7, 8 and 9. However, precise scores based on entire input space (*QuantifyML*) show that this is not the case with properties 1 and 8. In fact, only 73.56% of the inputs in the region define by property 1, satisfy the output prescribed by the property. This study highlights the benefit of *QuantifyML* in enabling the quantification of the true extent to which a given property is satisfied by a neural network model.

4 Related Work

Quantitative analysis of neural networks has typically been performed using statistical methods to obtain probabilistic guarantees. For instance, VeriFair [5] performs probabilistic analysis of fairness properties by leveraging sampling and concentration inequalities. PROVEN [46] is another framework for probabilistic analysis that employs robustness verification tools to derive probabilistic certificates for robustness of neural networks. PROVERO [3] performs quantitative verification to provide certificates for adversarial robustness by performing sampling of the inputs space and treating the model as a black-box. While statistical and sampling based approaches scale to large models, they do not provide precise quantification which is required to accurately assess the learnability, safety and robustness of the models.

The technique in [11] uses symbolic execution and constraint solution space quantification to precisely quantify probabilistic properties of neural networks, thereby enabling analysis of fairness and robustness of the models. FairSquare [2] is another probabilistic framework that implements a custom symbolic-volume-computation algorithm for fairness and bias analysis in neural networks. In contrast, our approach is not specialized to neural networks and can be applied to assess the learnability and robustness of any machine learning model.

Model counting has been applied in machine learning to enable probabilistic reasoning [9, 14, 17, 33]. Existing work that uses model counting technology for quantifying performance has been restricted for a sub-class of neural networks called Binarized Neural Networks (BNNs) [24]. The technique in [4] performs a translation of a BNN to SAT/CNF [35] and uses model-counters to provide quantitative estimates for the satisfaction of logical properties, which enables applications such as providing adversarial robustness guarantees and evaluating the efficacy of trojan attacks. In contrast, *QuantifyML* facilitates applications beyond binarized neural networks, albeit small. Expressing the model as a C program facilitates the use of standard bounded model checking for C programs, and the advancements in this field [13, 27] to perform the analysis.

As already mentioned, the work in [41] (MCML) uses model counting to perform a quantitative assessment of the performance of decision-tree classifier models. *MCML* is limited to binary decision trees and has been used on decision-tree models when used to learn relational properties of graphs. *QuantifyML* goes beyond *MCML* as it enables quantification of the performance of more general machine learning models, that may have non-binary inputs and multi-class outputs. Our evaluation presents applications such as robustness analysis of decision-tree models on an image-classification problem (MNIST), comparison of neural network and decision-tree models for learning relational properties of graphs, and evaluation of safety for collision avoidance, which cannot be achieved with the *MCML* tool.

Many frameworks based on statistical learning theory [44, 39] have been used to analyze the learnability of machine learning models. Popular ones such as the Probably Approximately Correct (PAC) learning framework [42] enable formal analytical characterization of the number of examples needed to train models for binary classification tasks. The work in [7] provides a detailed set of conditions

for learnability in terms of the Vapnik-Chervonenkis (VC) theory [43] dimension. *QuantifyML* complements theoretical frameworks by providing empirical evidence to precisely quantify generalizability both in terms of the accuracy as well as precision, recall and F1-score with respect to the ground truth.

5 Conclusion, Limitations and Future Work

We presented *QuantifyML*, which employs model counting to assess the *learnability*, *safety* and *robustness* of machine learning models. Our evaluation highlighted the benefits of using *QuantifyML* in various application scenarios. The main limitation of *QuantifyML* is scalability, which is due to the limited scalability of the underlying model counting technology. While approximate model counting is more efficient, it is less precise and still could not scale to large models. The analysis of neural network models was particularly challenging. The model counters (both exact and approximate) timed out while analyzing the neural network models for learning graph properties when the size was greater than 4 nodes. For the image classification study, *QuantifyML* could not handle neural network models, while we could only handle a small model for ACAS Xu. For the MNIST network, we attempted to reduce the state space of the model by changing the representation of weights and biases (e.g., from *floats* to *longs*), and considered perturbing only a subset (10 out of 784) pixels. However, although this did reduce the number of clauses and variables in the CNF files, the model counters still timed out. More details on the size of the CNF files for neural network models and the results can be found in the Appendix.

To address the scalability problem we plan to investigate slicing and/or compositional analysis of the C program representation of the models. Identifying important input features (using attribution techniques or decision-tree learning) and slicing and/or decomposing the model counting problem as dictated by these features could potentially lead to smaller problems that can be handled by off-the-shelf tools. We also plan to add support for other decision trees and neural network learning libraries and for other machine learning algorithms such as Adaboost Trees [15], Gradient Boosting Trees [16], Random Forest Trees [22] and SVM [12].

6 Appendix

6.1 Property specifications in Alloy

The Alloy [25] specifications for the 11 relational properties studied in this paper are given below.

```
// set and relation declaration

sig S {
  r: set S
}

// predicates

pred Antisymmetric() {
  all s, t: S | s->t in r and t->s in r implies s = t
}

pred Connex() {
  all s, t: S | s->t in r or t->s in r
}

pred Equivalence() {
  Reflexive[]
  Symmetric[]
  Transitive[]
}

pred Irreflexive() {
  all s, t: S | s->t in r implies s != t
}

pred NonStrictOrder() {
  Reflexive[]
  Antisymmetric[]
  Transitive[]
}

pred PartialOrder() {
  Antisymmetric[]
  Transitive[]
}

pred PreOrder() {
  Reflexive[]
  Transitive[]
}

pred Reflexive() {
  all s: S | s->s in r
}

pred StrictOrder() {
```

```
  Irreflexive[]
  Transitive[]
}

pred Symmetric() {
  all s, t: S | s->t in r implies t->s in r
}

pred TotalOrder() {
  PartialOrder[]
  Connex[]
}

pred Transitive() {
  all s, t, u: S | s->t in r and t->u in r implies s->u in r
}
```

6.2 Property specifications in ACAS Xu

6.2.1 Property ϕ_1.

If the intruder is distant and is significantly slower than the ownship, the score of a COC advisory will always be below a certain fixed threshold. Property ϕ_1 was not checked in our study because the output is an absolute value and it wouldn't match the outputs of the smaller network.

– Input constraints: $\rho \geq 55947.691$, $v_{own} \geq 1145$, $v_{int} \geq 60$.

– Desired output property: the score for COC is at most 1500.

6.2.2 Property ϕ_2 (corresponds to Property 1 in our study)

If the intruder is distant and is significantly slower than the ownship, the score of a COC advisory will never be maximal.

– Input constraints. $\rho \geq 55947.691$, $v_{own} \geq 1145$, $v_{int} \leq 60$.

– Desired output property. the score for COC is not the maximal score.

6.2.3 Property ϕ_3 (corresponds to Property 2 in our study).

If the intruder is directly ahead and is moving towards the ownship, the score for COC will not be minimal.

– Input constraints. $1500 \leq \rho \leq 1800$, $-0.06 \leq \theta \leq 0.06$, $\psi \geq 3.10$, $v_{own} \geq 980$, $v_{int} \geq 960$.

– Desired output property. the score for COC is not the minimal score.

6.2.4 Property ϕ_4 (corresponds to Property 3 in our study).

If the intruder is directly ahead and is moving away from the ownship but at a lower speed than that of the ownship, the score for COC will not be minimal.

- Input constraints. $1500 \leq \rho \leq 1800$, $-0.06 \leq \theta \leq 0.06$, $\psi = 0$, $v_{own} \geq 1000$, $700 \leq v_{int} \leq 800$.

- Desired output property. the score for COC is not the minimal score.

6.2.5 Property ϕ_5 (corresponds to Property 4 in our study).

If the intruder is near and approaching from the left, the network advises "strong right".

- Input constraints. $250 \leq \rho \leq 400$, $0.2 \leq \theta \leq 0.4$, $-3.141592 \leq \psi \leq -3.141592 + 0.005$, $100 \leq v_{own} \leq 400$, $0 \leq v_{int} \leq 400$.

- Desired output property. the score for "strong right" is the minimal score.

6.2.6 Property ϕ_6 (corresponds to Property 5 in our study).

If the intruder is sufficiently far away, the network advises COC.

- Input constraints. $12000 \leq \rho \leq 62000$, $(0.7 \leq \theta \leq 3.141592) \vee (-3.141592 \leq \theta \leq -0.7)$, $-3.141592 \leq \psi \leq -3.141592 + 0.005$, $100 \leq v_{own} \leq 1200$, $0 \leq v_{int} \leq 1200$.

- Desired output property. the score for COC is the minimal score.

6.2.7 Property ϕ_7 (corresponds to Property 6 in our study).

If vertical separation is large, the network will never advise a strong turn.

- Input constraints. $0 \leq \rho \leq 60760$, $-3.141592 \leq \theta \leq 3.141592$, $-3.141592 \leq \psi \leq 3.141592$, $100 \leq v_{own} \leq 1200$, $0 \leq v_{int} \leq 1200$.

- Desired output property. the scores for "strong right" and "strong left" are never the minimal scores.

6.2.8 Property ϕ_8 (corresponds to Property 7 in our study).

For a large vertical separation and a previous "weak left" advisory, the network will either output COC or continue advising "weak left".

- Input constraints. $0 \leq \rho \leq 60760$, $-3.141592 \leq \theta \leq -0.75*3.141592$, $-0.1 \leq \psi \leq 0.1$, $600 \leq v_{own} \leq 1200$, $600 \leq v_{int} \leq 1200$.

- Desired output property. the score for "weak left" is minimal or the score for COC is minimal.

6.2.9 Property ϕ_9 (corresponds to Property 8 in our study).

Even if the previous advisory was "weak right", the presence of a nearby intruder will cause the network to output a "strong left" advisory instead.

- Input constraints. $2000 \leq \rho \leq 7000$, $-0.4 \leq \theta \leq -0.14$, $-3.141592 \leq \psi \leq -3.141592 + 0.01$, $100 \leq v_{own} \leq 150$, $0 \leq v_{int} \leq 150$.

- Desired output property. the score for "strong left" is minimal.

6.2.10 Property ϕ_{10} (corresponds to Property 9 in our study).

For a far away intruder, the network advises COC.

- Input constraints. $36000 \leq \rho \leq 60760$, $0.7 \leq \theta \leq 3.141592$, $-3.141592 \leq \psi \leq -3.141592 + 0.01$, $900 \leq v_{own} \leq 1200$, $600 \leq v_{int} \leq 1200$.

- Desired output property. the score for COC is minimal.

6.3 General Statistics

Table 8: Training Datasets for graph properties. For each property, the number of nodes in the graph (*Scope*), the number of graphs satisfying the property (*Positive*) and the number of graphs violating the property (*Negative*) are shown.

Property	*Scope*	*Positive*	*Negative*
Antisymmetric	5	3.78E+06	2.98E+07
	4	23328	42208
Connex	5	1.18E+05	3.34E+07
	4	1458	64078
Equivalence	10	2.32E+05	1.27E+30
Irreflexive	5	2.10E+06	3.15E+07
	4	8192	57344
NonStrictOrder	6	2.60E+05	6.87E+10
	4	438	65098
PartialOrder	5	2.71E+05	3.33E+07
	4	7008	58528
PreOrder	6	4.19E+05	6.87E+10
	4	710	64826
Reflexive	5	2.10E+06	3.15E+07
	4	8192	57344
StrictOrder	6	2.60E+05	6.87E+10
	4	438	65098
TotalOrder	9	7.25E+05	2.42E+24
Transitive	5	3.09E+05	3.32E+07
	4	7988	57548

Table 9: Table 1 - Model Counts (MCML and QuantifyML) are tabulated for each of the four cases considered (i.e., true positive (TP), false negative (FN), false positive (FP), and true negative (TN). Difference column shows differences between model counts calculated by MCML and QuantifyML. Model Counter is projMC. "-" indicates a time-out of 5000 seconds

Property	*True Positves*			*False Negatives*			*False Positives*			*True Negatives*		
	MCML	*QML*	*Diff(%)*	*MCML*	*QML*	*Diff(%)*	*MCML*	*QML*	*Diff(%)*	*MCML*	*QML*	*Diff(%)*
Antisymmetric	1.89E+06	1.89E+06	0.0	332	332	0.0	1.18E+04	1.18E+04	0.0	3.17E+07	3.17E+07	0.0
Connex	5.90E+04	5.90E+04	0.0	96	96	0.0	2.21E+05	2.21E+05	0.0	3.33E+07	3.33E+07	0.0
Equivalence	1.16E+05	1.16E+05	0.0	0	0	0.0	-	-	-	-	-	-
Irreflexive	1.05E+06	1.05E+06	0.0	0	0	0.0	0	0	0.0	3.25E+07	3.25E+07	0.0
NonStrictOrder	1.30E+05	1.30E+05	0.0	66	66	0.0	1.10E+08	-	-	6.86E+10	-	-
PartialOrder	1.35E+05	1.35E+05	0.0	114	114	0.0	4.11E+05	4.11E+05	0.0	3.30E+07	3.30E+07	0.0
PreOrder	2.09E+05	2.09E+05	0.0	75	75	0.0	1.88E+08	-	-	6.85E+10	-	-
Reflexive	1.05E+06	1.05E+06	0.0	0	0	0.0	0	0	0.0	3.25E+07	3.25E+07	0.0
StrictOrder	1.30E+05	1.30E+05	0.0	31	31	0.0	1.48E+08	-	-	6.86E+10	-	-
TotalOrder	3.63E+05	3.63E+05	0.0	0	-	-	-	-	-	-	-	-
Transitive	1.53E+05	1.53E+05	0.0	1338	1338	0.0	7.90E+05	7.90E+05	0.0	3.26E+07	3.26E+07	0.0

Table 10: Table 3 - Model Counts (MCML and QuantifyML) are tabulated for each of the four cases considered (i.e., true positive (TP), false negative (FN), false positive (FP), and true negative (TN). Difference column shows differences between model counts calculated by MCML and QuantifyML. Model Counter is ApproxMC.

Property	*True Positves*			*False Negatives*			*False Positives*			*True Negatives*		
	MCML	*QML*	*Diff(%)*	*MCML*	*QML*	*Diff(%)*	*MCML*	*QML*	*Diff(%)*	*MCML*	*QML*	*Diff(%)*
Antisymmetric	2.00E+06	1.90E+06	4.9	336	336	0.0	1.31E+04	1.23E+04	6.2	3.15E+07	3.20E+07	-1.5
Connex	6.45E+04	6.45E+04	0.0	98	98	0.0	2.21E+05	2.17E+05	1.8	3.36E+07	3.36E+07	0.0
Equivalence	1.21E+05	1.27E+05	-5.1	0	0	0.0	6.96E+26	6.96E+26	0.0	1.27E+30	1.27E+30	0.0
Irreflexive	1.05E+06	1.05E+06	0.0	0	0	0.0	0	0	0.0	3.25E+07	3.25E+07	0.0
NonStrictOrder	1.29E+05	1.25E+05	3.2	66	66	0.0	1.17E+08	1.05E+08	10.4	6.87E+10	6.87E+10	0.0
PartialOrder	1.45E+05	1.39E+05	4.2	114	116	-1.8	4.59E+05	4.10E+05	10.8	3.30E+07	3.36E+07	-1.7
PreOrder	2.29E+05	2.29E+05	0.0	80	74	7.5	1.93E+08	1.89E+08	2.2	6.87E+10	6.87E+10	0.0
Reflexive	1.05E+06	1.05E+06	0.0	0	0	0.0	0	0	0.0	3.25E+07	3.25E+07	0.0
StrictOrder	1.35E+05	1.43E+05	-6.1	31	31	0.0	1.43E+08	1.43E+08	0.3	6.87E+10	6.87E+10	0.0
TotalOrder	4.01E+05	3.44E+05	14.3	0	2.01E+08	-	7.01E+20	7.38E+20	-5.3	2.42E+24	2.42E+24	0.0
Transitive	1.52E+05	1.72E+05	-13.5	1504	1344	10.6	8.03E+05	7.70E+05	4.1	3.25E+07	3.30E+07	-1.6

Table 11: Table 4 (Table 1 of Short Paper) - Model Counts (Stat and QuantifyML) are tabulated for each of the four cases considered (i.e., true positive (TP), false negative (FN), false positive (FP), and true negative (TN). Model Counter is ApproxMC. "-" indicates a time-out of 5000 seconds

Property	*True Positves*		*False Negatives*		*False Positives*		*True Negatives*	
	Stat	*QML*	*Stat*	*QML*	*Stat*	*QML*	*Stat*	*QML*
Antisymmetric	1160	11664	0	0	0	0	1173	53872
Connex	73	729	0	10935	1	999	72	52873
Irreflexive	401	4096	0	0	0	0	419	61440
NonStrictOrder	23	219	0	0	0	1829	21	63488
PartialOrder	356	3504	0	0	3	528	342	61504
PreOrder	40	355	0	0	0	2013	31	63168
Reflexive	401	4096	0	0	0	0	419	61440
StrictOrder	23	219	0	0	2	1829	19	63488
Transitive	412	3990	4	4	8	1313	375	60229

Table 12: Table 4 (Table 1 of Short Paper) - QuantifyML - Decision Trees - The number of primary variables and total number of clauses are tabulated for each of the four cases considered (i.e., true positive (TP), false negative (FN), false positive (FP), and true negative (TN)).

Property	*Primary Variables*	Clauses			
		True Positives	*False Negatives*	*False Positives*	*True Negatives*
Antisymmetric	16	3731	3825	3743	3837
Connex	16	1231	1188	928	900
Irreflexive	16	132	155	114	128
NonStrictOrder	16	3870	3942	3124	3171
PartialOrder	16	7427	7543	7417	7521
PreOrder	16	4143	4113	3526	3504
Reflexive	16	170	154	140	130
StrictOrder	16	3270	3377	2850	1105
Transitive	16	12588	12574	12651	12637

Table 13: Table 5 (Table 2 of Short Paper) - Model Counts (Stat and QuantifyML) are tabulated for each of the four cases considered (i.e., true positive (TP), false negative (FN), false positive (FP), and true negative (TN). Model Counter is ApproxMC. "-" indicates a time-out of 5000 seconds

Property	*True Positves*		*False Negatives*		*False Positives*		*True Negatives*	
	Stat	*QML*	*Stat*	*QML*	*Stat*	*QML*	*Stat*	*QML*
Antisymmetric	1055	10606	105	1058	348	14581	825	39291
Connex	73	1009	0	10655	5	3329	68	50543
Irreflexive	401	4096	0	0	0	0	419	61440
NonStrictOrder	23	217	0	2	1	6197	20	59120
PartialOrder	251	2544	105	960	49	10162	296	51870
PreOrder	40	348	0	7	3	7695	28	57486
Reflexive	401	4096	0	0	0	0	419	61440
StrictOrder	23	219	0	0	2	3871	19	61446
Transitive	308	2899	108	1095	74	12650	309	48892

Table 14: Table 5 (Table 2 of Short Paper) - QuantifyML - Neural Networks - The number of primary variables and total number of clauses are tabulated for each of the four cases considered (i.e., true positive (TP), false negative (FN), false positive (FP), and true negative (TN)).

Property	*Primary Variables*	Clauses			
		True Positives	*False Negatives*	*False Positives*	*True Negatives*
Antisymmetric	16	979496	979493	979496	979493
Connex	16	979303	979295	979260	979257
Irreflexive	16	977680	977674	977668	977665
NonStrictOrder	16	979498	979490	978862	978859
PartialOrder	16	979021	979017	979449	979446
PreOrder	16	984947	984940	984610	984607
Reflexive	16	976226	976220	976214	976211
StrictOrder	16	983108	983101	982771	982768
Transitive	16	978063	978060	978063	978060

Table 15: Table 6 (Table 3 of Short Paper) - Quantifying robustness for the MNIST model - The number of primary variables and total number of clauses.

Label	*Variables*	Clauses
0	25088	30834437
1	25088	30559298
2	25088	30732685
3	25088	30819983
4	25088	30762196
5	25088	30828172
6	25088	30224216
7	25088	30810223
8	25088	30832412
9	25088	30798957

Table 16: Table 7 (Table 4 of Short Paper) - Quantifying safety for the MNIST model - The number of primary variables and total number of clauses.

Property	*QuantifyML$_N$ Variables*	QuantifyML$_N$ Clauses	*QuantifyML$_S$ Variables*	QuantifyML$_S$ Clauses
2	321	2853129	321	2853114
3	321	2850722	321	2850722
4	321	2850938	321	2851252
5	321	2851993	321	2851993
6	321	2852357	321	2852357
7	321	2851367	321	2851367
8	321	2851683	321	2851683
9	321	2851836	321	2851836
10	321	2851490	321	2851490

References

[1] Mahdieh Abbasi, Arezoo Rajabi, Christian Gagné & Rakesh B. Bobba (2020): *Toward Adversarial Robustness by Diversity in an Ensemble of Specialized Deep Neural Networks*. In Cyril Goutte & Xiaodan Zhu, editors: *Advances in Artificial Intelligence*, Springer International Publishing, Cham, pp. 1–14.

[2] Aws Albarghouthi, Loris D'Antoni, Samuel Drews & Aditya V. Nori (2017): *FairSquare: probabilistic verification of program fairness*. *PACMPL* 1(OOPSLA), pp. 80:1–80:30.

[3] Teodora Baluta, Zheng Leong Chua, Kuldeep S. Meel & Prateek Saxena (2020): *Scalable Quantitative Verification For Deep Neural Networks*. *CoRR*.

[4] Teodora Baluta, Shiqi Shen, Shweta Shinde, Kuldeep S. Meel & Prateek Saxena (2019): *Quantitative Verification of Neural Networks and Its Security Applications*. In: *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019*, pp. 1249–1264.

[5] Osbert Bastani, Xin Zhang & Armando Solar-Lezama: *Probabilistic verification of fairness properties via concentration*. *PACMPL* 3(OOPSLA).

[6] Joao Bastos (2007): *Credit scoring with boosted decision trees*.

[7] Anselm Blumer, A. Ehrenfeucht, David Haussler & Manfred K. Warmuth (1989): *Learnability and the Vapnik-Chervonenkis Dimension*. *JACM* 36(4).

[8] B. Bonakdarpour & S. S. Kulkarni (2007): *Exploiting Symbolic Techniques in Automated Synthesis of Distributed Programs with Large State Space*. In: *ICDCS*.

[9] Mark Chavira & Adnan Darwiche (2008): *On probabilistic inference by weighted model counting*. *JAI* 172(6-7).

[10] Jeremy Cohen, Elan Rosenfeld & Zico Kolter (2019): *Certified Adversarial Robustness via Randomized Smoothing*. In Kamalika Chaudhuri & Ruslan Salakhutdinov, editors: *Proceedings of the 36th International Conference on Machine Learning*, *Proceedings of Machine Learning Research* 97, PMLR, pp. 1310–1320. Available at `http://proceedings.mlr.press/v97/cohen19c.html`.

[11] H. Converse, A. Filieri, D. Gopinath & C. S. Păsăreanu (2020): *Probabilistic Symbolic Analysis of Neural Networks*. In: *2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE)*.

[12] Corinna Cortes & Vladimir Vapnik (1995): *Support-vector networks*. *Machine Learning* 20(3), pp. 273–297, doi:10.1007/BF00994018. Available at `https://doi.org/10.1007/BF00994018`.

[13] S. Falke, F. Merz & C. Sinz (2013): *The bounded model checker LLBMC*. In: *2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE)*.

[14] Daan Fierens, Guy Van den Broeck, Ingo Thon, Bernd Gutmann & Luc De Raedt (2012): *Inference in Probabilistic Logic Programs using Weighted CNF's*. *CoRR* abs/1202.3719.

[15] Yoav Freund & Robert E Schapire (1997): *A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting*. *Journal of Computer and System Sciences* 55(1), pp. 119 – 139, doi:https://doi.org/10.1006/jcss.1997.1504. Available at `http://www.sciencedirect.com/science/article/pii/S002200009791504X`.

[16] Jerome H. Friedman (2001): *Greedy function approximation: A gradient boosting machine.* *Ann. Statist.* 29(5), pp. 1189–1232, doi:10.1214/aos/1013203451. Available at `https://doi.org/10.1214/aos/1013203451`.

[17] Robert Gens & Pedro Domingos (2013): *Learning the Structure of Sum-product Networks*. In: *ICML*.

[18] Patrice Godefroid & Sarfraz Khurshid (2002): *Exploring Very Large State Spaces Using Genetic Algorithms*. In Joost-Pieter Katoen & Perdita Stevens, editors: *Tools and Algorithms for the Construction and Analysis of Systems*, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 266–280.

[19] Carla P Gomes, Ashish Sabharwal & Bart Selman (2008): *Model counting*.

[20] Divya Gopinath, Guy Katz, Corina S Pasareanu & Clark Barrett (2017): *DeepSafe: A data-driven approach for checking adversarial robustness in neural networks*. arXiv preprint arXiv:1710.00486.

[21] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath & B. Kingsbury (2012): *Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups*. IEEE Signal Processing Magazine 29(6), pp. 82–97.

[22] Tin Kam Ho (1995): *Random Decision Forests*. In: Third International Conference on Document Analysis and Recognition (Volume 1) - Volume 1.

[23] Xiaowei Huang, Daniel Kroening, Wenjie Ruan, James Sharp, Youcheng Sun, Emese Thamo, Min Wu & Xinping Yi (2020): *A survey of safety and trustworthiness of deep neural networks: Verification, testing, adversarial attack and defence, and interpretability*. Computer Science Review 37, p. 100270.

[24] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv & Yoshua Bengio (2016): *Binarized neural networks*. In: NIPS.

[25] Daniel Jackson (2002): *Alloy: A Lightweight Object Modeling Notation*. TOSEM 11(2).

[26] Daniel Jackson (2002): *Alloy: a lightweight object modelling notation*. ACM Trans. Softw. Eng. Methodol. 11(2).

[27] Anushri Jana, Uday P. Khedker, Advaita Datar, R Venkatesh & C Niyas (2016): *Scaling Bounded Model Checking By Transforming Programs With Arrays*.

[28] Guy Katz, Clark W. Barrett, David L. Dill, Kyle Julian & Mykel J. Kochenderfer (2017): *Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks*. In: Computer Aided Verification - 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part I, pp. 97–117, doi:10.1007/978-3-319-63387-9_5.

[29] Nikhil Ketkar (2017): *Introduction to keras*. In: Deep learning with Python, Springer, pp. 97–111.

[30] Alex Krizhevsky, Ilya Sutskever & Geoffrey E Hinton (2012): *ImageNet Classification with Deep Convolutional Neural Networks*. In F. Pereira, C. J. C. Burges, L. Bottou & K. Q. Weinberger, editors: Advances in Neural Information Processing Systems, 25, Curran Associates, Inc., pp. 1097–1105.

[31] Daniel Kroening & Michael Tautschnig (2014): *CBMC–C bounded model checker*. In: International Conference on Tools and Algorithms for the Construction and Analysis of Systems, Springer, pp. 389–391.

[32] Wen-Jia Kuo, Ruey-Feng Chang, Dar-Ren Chen & Cheng Chun Lee (2001): *Data mining with decision trees for diagnosis of breast tumor in medical ultrasonic images*. Breast cancer research and treatment 66(1), pp. 51–57.

[33] Yitao Liang, Jessa Bekker & Guy Van den Broeck (2017): *Learning the Structure of Probabilistic Sentential Decision Diagrams*. In: UAI.

[34] *The MNIST Database of handwritten digits Home Page*. `http://yann.lecun.com/exdb/mnist/`.

[35] Nina Narodytska, Shiva Prasad Kasiviswanathan, Leonid Ryzhyk, Mooly Sagiv & Toby Walsh (2018): *Verifying Properties of Binarized Deep Neural Networks*. In: AAAI.

[36] M. P. Owen, A. Panken, R. Moss, L. Alvarez & C. Leeper (2019): *ACAS Xu: Integrated Collision Avoidance and Detect and Avoid Capability for UAS*. In: 2019 IEEE/AIAA 38th Digital Avionics Systems Conference (DASC), pp. 1–10, doi:10.1109/DASC43569.2019.9081758.

[37] Nicolas Papernot, Patrick D. McDaniel, Somesh Jha, Matt Fredrikson, Z. Berkay Celik & Ananthram Swami (2016): *The Limitations of Deep Learning in Adversarial Settings*. In: EuroS&P.

[38] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg et al. (2011): *Scikit-learn: Machine learning in Python*. the Journal of machine Learning research 12, pp. 2825–2830.

[39] Shai Shalev-Shwartz, Ohad Shamir, Nathan Srebro & Karthik Sridharan (2010): *Learnability, Stability and Uniform Convergence*. JMLR 11. Available at `http://dl.acm.org/citation.cfm?id=1756006.1953019`.

[40] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow & R. Fergus (2013): *Intriguing Properties of Neural Networks*. Technical Report. http://arxiv.org/abs/1312.6199.

[41] Muhammad Usman, Wenxi Wang, Marko Vasic, Kaiyuan Wang, Haris Vikalo & Sarfraz Khurshid (2020): *A Study of the Learnability of Relational Properties: Model Counting Meets Machine Learning (MCML)*. PLDI 2020, Association for Computing Machinery, New York, NY, USA, p. 1098–1111, doi:10.1145/3385412.3386015. Available at https://doi.org/10.1145/3385412.3386015.

[42] L. G. Valiant (1984): *A Theory of the Learnable*. *CACM* 27(11), doi:10.1145/1968.1972. Available at http://doi.acm.org/10.1145/1968.1972.

[43] V. N. Vapnik & A Ya. Chervonenkis (1971): *On the Uniform Convergence of Relative Frequencies of Events to Their Probabilities*. *Theory of Probabibility and its Applicactions.*

[44] Vladimir N. Vapnik (1995): *The Nature of Statistical Learning Theory*.

[45] Wenxi Wang, Muhammad Usman, Alyas Almaawi, Kaiyuan Wang, Kuldeep S Meel & Sarfraz Khurshid (2020): *A Study of Symmetry Breaking Predicates and Model Counting*. In: *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, Springer, pp. 115–134.

[46] Lily Weng, Pin-Yu Chen, Lam M. Nguyen, Mark S. Squillante, Akhilan Boopathy, Ivan V. Oseledets & Luca Daniel (2019): *PROVEN: Verifying Robustness of Neural Networks with a Probabilistic Approach*. In: *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, pp. 6727–6736. Available at http://proceedings.mlr.press/v97/weng19a.html.