2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)

Fairify: Fairness Verification of Neural Networks

Sumon Biswas
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA, USA
sumonb@cs.cmu.edu

Hridesh Rajan
Dept. of Computer Science
Iowa State University
Ames, IA, USA
hridesh@iastate.edu

***Abstract*—Fairness of machine learning (ML) software has become a major concern in the recent past. Although recent research on testing and improving fairness have demonstrated impact on real-world software, providing fairness guarantee in practice is still lacking. Certification of ML models is challenging because of the complex decision-making process of the models. In this paper, we proposed *Fairify*, an SMT-based approach to verify individual fairness property in neural network (NN) models. Individual fairness ensures that any two similar individuals get similar treatment irrespective of their protected attributes e.g., race, sex, age. Verifying this fairness property is hard because of the global checking and non-linear computation nodes in NN. We proposed sound approach to make individual fairness verification tractable for the developers. The key idea is that many neurons in the NN always remain inactive when a smaller part of the input domain is considered. So, Fairify leverages white-box access to the models in production and then apply formal analysis based pruning. Our approach adopts input partitioning and then prunes the NN for each partition to provide fairness certification or counterexample. We leveraged interval arithmetic and activation heuristic of the neurons to perform the pruning as necessary. We evaluated Fairify on 25 real-world neural networks collected from four different sources, and demonstrated the effectiveness, scalability and performance over baseline and closely related work. Fairify is also configurable based on the domain and size of the NN. Our novel formulation of the problem can answer targeted verification queries with relaxations and counterexamples, which have practical implications.**

***Index Terms*—fairness, verification, machine learning**

I. Introduction

Artificial intelligence (AI) based software are increasingly being used in critical decision making such as criminal sentencing, hiring employees, approving loans, etc. Algorithmic fairness of these software raised significant concern in the recent past [1–8]. Several studies have been conducted to measure and mitigate algorithmic fairness in software [9–18]. However, providing formal guarantee of fairness properties in practice is still lacking. Fairness verification in ML models is difficult given the complex decision making process of the algorithms and the specification of the fairness properties [19–22]. Our goal in this paper is to enable the verification in real-world development and guarantee fairness in critical domains.

Albarghouthi et al. and Bastani et al. proposed probabilistic techniques to verify *group fairness* [20, 22]. Group fairness property ensures that the protected groups (e.g., male-vs-female, young-vs-old, etc.) get similar treatment in the prediction. On the other hand, *individual fairness* states that any two similar individuals who differ only in their protected attribute get similar treatment [21, 23]. Galhotra et al. argued that group fairness property might not detect bias in scenarios when same amount of discrimination is made for any two groups [23], which led to the usage of individual fairness property in many recent works [1, 8, 23–25]. John et al. proposed individual fairness verification for two ML classifiers, i.e., linear classifier and 2) kernelized classifier e.g., support vector machine [21], which is not applicable to neural networks.

We propose *Fairify*, the technique to verify individual fairness of NN models in production, i.e., already trained models. Both abstract interpretation [26–30] and satisfiability modulo theories (SMT) based techniques [31–33] have shown success in verifying different properties of NN such as robustness. However, fairness verification of NN on real-world models has received little attention. We adopted SMT based verification since it enables practical benefits, e.g., solving arbitrary verification query and providing counterexamples. Urban et al. proposed Libra, an abstract interpretation based *dependency fairness* certification for NN [27]. The approach can not check relaxation of fairness queries and does not provide counterexamples in case of a violation. Fairify on the other hand provides configurable options to the developers enabling fairness verification of NN in practice.

Verifying a property in NN is challenging mainly because of the presence of non-linear computation nodes i.e., activation functions [27, 32]. With the size of the NN, the verification task becomes harder and often untractable [34, 35]. Many studies have been conducted to verify NN for different local robustness properties [29, 31, 36]. However, the individual fairness property requires global checking which makes the verification task even harder and existing local property verifiers can not be used [37, 38]. To that end, we propose a novel technique that can verify fairness, i.e., provide satisfiability (SAT) with counterexample, or show UNSAT in a tractable time and available computational resource. Fairify takes a trained NN and the verification query as input. If the verifier can verify within the given timeout period, the output should be SAT (violation) or UNSAT (certification). When the verifier is unable to show any proof within the timeout, the result is UNK (unknown). Our evaluation shows that using state-of-the-art SMT solver, we cannot verify the fairness property of the NN models in days; however, Fairify can verify most of the verification queries within an hour. Fairify makes the

1558-1225/23/$31.00 ©2023 IEEE
DOI 10.1109/ICSE48619.2023.00134

verification task tractable and scalable by combining input partitioning and pruning approach.

Our key insight is that the activation patterns of the NN used in practice are sparse when we consider only a smaller part of the input region. Therefore, we perform **input partitioning** to a certain point and divide the problem into multiple sub-problems which result into split-queries. To show the problem as UNSAT, all the split-queries have to be UNSAT. On the other hand, if any of the split-queries gives SAT, the whole problem becomes SAT. Because of this construction of the problem, Fairify is able to provide certification or counterexample for each partition which has further value in fairness defect localization.

The input partitioning allowed us to perform static analysis on the network for the given partition and identify inactive neurons. We applied interval arithmetic to compute bounds of the neurons. In addition, we run individual verification query on each neuron to identify the ones that are always inactive. Since the inactive neurons do not impact the decision, removing those neurons gives a pruned version of the network that can be verified for the given partition. This **sound pruning** is lightweight, sound, and achieves high pruning ratio, which makes the verification task tractable.

We further improve the efficiency of the approach by analyzing activation heuristics. We conduct lightweight simulation to profile the network and find candidate neurons that remain *almost* always inactive but could not be removed through sound pruning. Thereby, we propose layer-wise **heuristics** that suggest inactive nodes; if necessary given the time budget. Although this pruning is based on heuristics, our evaluation shows that a conservative approach provides much improvement in the verification with negligible loss of accuracy. Another novelty in this idea is that the developer can choose to deploy the pruned (and verified) version of the NN. Thus, the pruned NN would provide sound fairness guarantee with little loss of accuracy.

After partitioning and reducing the complexity of the problem, we leverage a constraint solver to verify the split-queries on the pruned networks. Then we accumulate the results for each partition to provide verification for the original query. We evaluated Fairify on 25 different NN models collected from four different sources. We collected appropriate real-world NNs from Kaggle [39] which are built for three popular fairness-critical tasks and took the NNs used in three prior works in the area [8, 24, 27]. Our results show significant improvement over the baseline with respect to utility, scalability, and performance. The main contributions of our work are as follows.

1) Fairify is the first to solve the individual fairness verification problem for already trained NN using SMT based technique.
2) Our formulation of the problem enables verification of different relaxation of the fairness property. Then we proposed two novel NN pruning methods designed to solve those queries effectively.
3) The approach can be integrated in the development pipeline and provides practical benefits for the developers, i.e., certification or counterexample for each input partition and targeted fairness certification.
4) We implemented Fairify using Python and openly available constraint solver Z3. We also created a benchmark of NN models for fairness verification. The code, models, benchmark datasets, and results are available in the self-contained GitHub repository [40] [1] that can be leveraged by future research.

The rest of the paper is organized as follows: §II describes the background and §III introduces fairness verification problem in NN. §IV provides detailed description of the approach. §V describes the results and answers the research questions. Finally, §VII describes the related work, §VI discusses threats to validity, and §VIII concludes.

II. Preliminaries

Neural Networks (NN). We consider NN as a directed acyclic graph (DAG), where the nodes hold numeric values that are computed using some functions, and edges are the data-flow relations. The nodes are grouped into layers: one input layer, one or more hidden layers, and one output layer. More formally, a NN model $M : \mathbb{R}^n \rightarrow \mathbb{R}^m$ is a DAG with k layers: $L_1, L_2, \ldots, L_k$ where L_1 is the input layer and L_k is the output layer. The size of each layer L_i is denoted by s_i and layer L_i has the nodes $v_i^1, v_i^2, \ldots, v_i^{s_i}$. Therefore, $s_1 = n$ (number of inputs) and $s_k = m$ (number of output classes). In this paper, we consider the fully connected networks because of its success in real-world tasks [8, 41, 42], where the value of a node is given by $v_i^j = \mathsf{NL}(\Sigma_t w_{i-1,t} \cdot v_{i-1}^t + b_i^j)$, where a node v_i^j is computed by the weighted sum $\mathsf{WS} = \Sigma_t w_{i-1,t} \cdot v_{i-1}^t + b_i^j$, and then applying a non-linear activation function NL. WS is computed from the neurons of preceding layer, associated weights of incoming edges $w_{i-1,t}$, and bias b_i^j. The weights and biases are constant real values in a trained NN, which are learned in the training phase. In practice, the ReLU activation is widely used as the NL function because of better convergence, efficient computation, and performance. Therefore, following the prior works in the area [27, 32, 41, 42], we also considered ReLU based NN. ReLU is given by Eq. (1) which is a piecewise-linear (PL) function.

$$\mathsf{ReLU}(x) = \begin{cases} 0 & \text{if } x \leq 0 \text{ ; inactive neuron} \\ x & \text{if } x > 0 \text{ ; active neuron} \end{cases} \quad (1)$$

The neurons in the input layer accept data input values, which are passed to the following layers through the edges. Each neuron in the hidden and output layer computes its value by applying the weighted sum (WS) function and then the PL function. The output neurons may have different PL functions (e.g., Sigmoid, Softmax) that computes the predictive classes in classification problem.

III. Fairness Verification

In this section, we define the individual fairness verification problem and compare with closely related NN verifications.

NN Verification. A property $\phi(M)$ of the NN model M defines a set of input constraints as precondition ϕ_x, and output

[1] https://github.com/sumonbis/Fairify

constraints as postcondition ϕ_y. Several safety properties of NN have been investigated in the literature [43, 44]. One of the most studied property is adversarial robustness [31, 45]. Given a model M and input x_0, adversarial robustness ensures that a minimum perturbation to x_0 does not change the label predicted by the model, i.e., $\forall x'. ||x_0 - x'|| \leq \delta \Rightarrow M(x_0) = M(x')$. The property ensures *local* safety as it searches for a violation only in the neighborhood of x_0 with a distance δ. Different approaches have been proposed to guarantee safe region around the given input [44, 46] or generate targeted attacks that violate the property [43, 44, 47]. Unlike the robustness property, individual fairness requires *global* safety guarantee.

A. Individual Fairness

The group fairness property considers average-case fairness, where some notion of parity is maintained between protected groups. Hence, this property fails to detect bias if same amount of discrimination is made between two groups [23]. Individual fairness considers worst-case fairness, i.e., all similar input pairs get similar outcome [21, 23]. Our goal is to provide fairness guarantee which makes individual fairness property ideal for checking.

Suppose, $\mathcal{D}$ is a dataset containing n data instances, where each instance $x = (x_1, \ldots, x_t)$ is a tuple with t attribute values. The set of attributes is denoted by $A = \{A_1, \ldots, A_t\}$ with its domain I, where $A_i = \{a \,|\, a \in I_i\}$. The set of protected attributes is denoted by P where $P \subset A$. So, individual fairness is defined as follows, which is widely used in the literature [1, 8, 24, 25].

Definition 1: **(Individual Fairness)** The model M is individually fair if there is no such pair of data instances (x, x') in the input domain such that: ❶ $x_i = x'_i$, $\forall i \in A \setminus P$, ❷ $x_j \neq x'_j$, $\exists j \in P$, and ❸ $M(x) \neq M(x')$.

Intuitively, individual fairness ensures that any two individuals who have same attribute values except the protected attributes, get the same prediction. If there exists such pair, the property is violated and (x, x') is considered as an individual bias instance. For example, suppose while predicting income of individuals ($> \$50$K or not), *race* is considered as the protected attribute. Definition 1 suggests that if two individuals (x, x') with different *race* but exact same non-protected attributes, e.g., *occupation*, *age*, *marital-status*, are predicted to the different class, then the model is unfair.

The above definition is similar to the global robustness property introduced by [32, 38]. In global property checking, both the inputs x and x' can obtain any value within the domain and thereby is significantly harder to check. Whereas in local robustness an input is fixed and checking in the neighborhood is sufficient [37, 38]. To check such global property, two copies of the same NN are encoded in the postcondition to check $M(x) \neq M(x')$. Hence, the approaches to verify local robustness properties (e.g., Reluplex) can not verify individual fairness [37, 38]. Gopinath et al. proposed a data-driven approach to assess global robustness [37]. The approach requires labeled training data to cluster inputs and then casts the problem as local robustness checking. However, our goal is to verify individual fairness statically, i.e., we do not require training data or an oracle of correct labels.

In practice, definition 1 can be a weaker constraint, and it can be *relaxed* to ensure further fairness of the model. The attributes of x and x' are said to be relaxed when they are not equal. The constraint ❶ above is relaxed on the non-protected attributes so that instead of equality, a small perturbation is allowed [21]. For example, we can still consider (x, x') as bias instance even if a non-protected attribute (e.g. *age*) differs in a small amount (e.g., 5 years). Note that if a model is fair with respect to Definition 2, it is also fair with respect to Definition 1 but the opposite is not true. Therefore, from the verification perspective, the relaxed query requires stronger certification and subsumes the basic fairness requirement.

Definition 2: **(ϵ-Fairness)** The model M is individually fair if no two data instances (x, x') in the input domain satisfy: ❶ $|x_i - x'_i| \leq \epsilon_i$, $\forall i \in A \setminus P$, ❷ $x_j \neq x'_j$ $\exists j \in P$, and ❸ $M(x) \neq M(x')$, where ϵ is a small perturbation that limits the similarity.

For further practical benefit, we introduce the notion of targeted fairness, which imposes an arbitrary additional constraint on the inputs. For example, the developer might be interested in verifying whether the NN is fair in giving loans to individuals who have at least *high-school* education, works more than *50 hours-per-week*, and occupation is *sales*.

Definition 3: **(Targeted Fairness)** A target T is a set of arbitrary linear constraints on the inputs of NN, i.e., $\{(l_i, u_i) \,|\, \forall i \in A,\, l_i \leq x_i \leq u_i\}$. Targeted fairness ensures that all valid inputs in the given target satisfy individual fairness.

Holstein et al. identified a common difficulty in fair software development which is to diagnose and audit problems [48]. The following developer's response [48] conveys the utility of targeted verification and counterexamples, which was not addressed in prior works.

> *"If an oracle was able to tell me, 'look, this is a severe problem and I can give you a hundred examples [of this problem],' [...] then it's much easier internally to get enough people to accept this and to solve it. So having a process which gives you more data points where you mess up [in this way] would be really helpful."*

IV. Approach

In this section, we formulate the fairness verification problem and describe our approach to verify the property.

A. Problem Formulation

The verification problem that we are interested in is, given the precondition on the input x and the NN function M, the output $y = M(x)$ satisfies some postcondition. The preconditions and postconditions are designed to verify the fairness defined in §III on a trained model, where we have white-box access to the NN. In the definitions, ❶ and ❷ are the fairness preconditions, and ❸ is the fairness postcondition. Suppose, each input feature x_i is bound by some lower bound lb_i and upper bound ub_i that is obtained from the domain knowledge. In this case, we obtain it from training data, e.g., while predicting income on

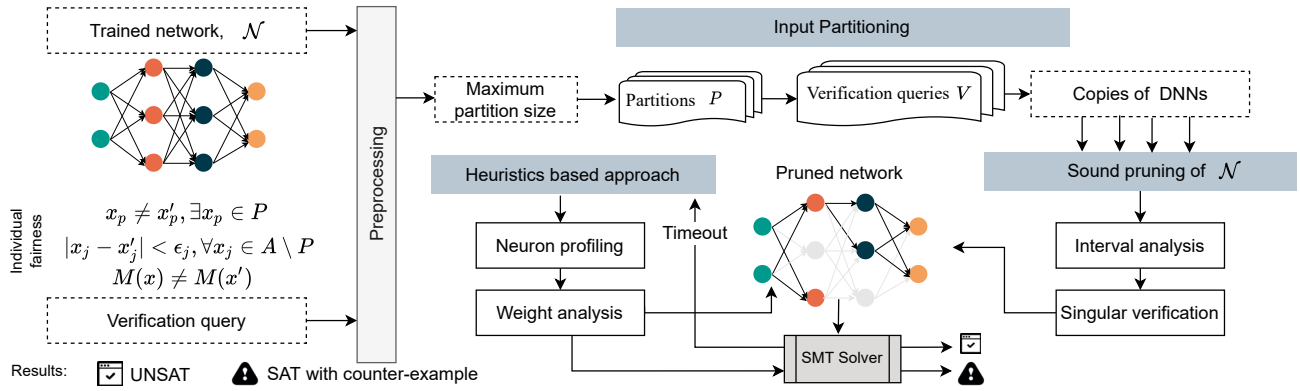

Fig. 1: The overview of our approach for the fairness verification of neural networks

Adult Census data, the *work-hours* of individuals is $[1, 100]$. Let, M be a classification network with one neuron in the output layer and suppose, Sigmoid function is applied on the output node y to predict the classes. Sigmoid function is given by the equation: $\text{Sig}(y) = 1/(1 + e^{-y})$. Therefore, for any two inputs x, x', the postcondition that needs to be satisfied for fairness is: $M(x) = M(x')$. Here, we encode the negation of the postcondition in the verification query so that the solver provides counterexample with SAT when there is a violation. Next, we compute the weakest precondition (WP) of the postcondition. A WP function in this case gives the minimum requirements need to be satisfied to assert the postcondition [49]. The WP computation is shown below, which transfers the fairness postcondition to the neurons of the output layer before applying the activation function, Sigmoid.

$$WP(M(x) \neq M(x')) \equiv \text{Sig}(y) \neq \text{Sig}(y')$$
$$WP(\text{Sig}(y) \neq \text{Sig}(y')) \equiv (y < 0 \wedge y' > 0) \vee (y > 0 \wedge y' < 0)$$

Similarly, for other non-linear activation functions, we can compute the WP. Another WP transfer used in our evaluation is Softmax function: $f(x_i) = e^{x_i}/\Sigma_j e^{x_j}$. The WP of Softmax for binary classification tasks is: $(y_0 > y_1 \Rightarrow y'_0 < y'_1) \wedge (y_0 < y_1 \Rightarrow y'_0 > y'_1)$. Overall, this step makes the verification query simpler as well as the postcondition is reasoned on the network output layer. Thus, after reducing the postconditions, we have to verify the following constraint according to Definition 1.

$$\overbrace{\Big(\bigwedge_{\forall i \in A} lb_i \leq x_i, x'_i \leq ub_i\Big)}^{\text{domain constraint, } \phi_x^d} \wedge \overbrace{\Big(\bigwedge_{\forall j \in A \setminus P} x_j = x'_j\Big) \wedge \Big(\bigwedge_{\forall k \in P} x_k \neq x'_k\Big)}^{\text{fairness precondition, } \phi_x^f} \wedge$$
$$\underbrace{y = M(x), y' = M(x')}_{\text{outputs}} \Longrightarrow \underbrace{(y < 0 \wedge y' > 0) \vee (y > 0 \wedge y' < 0)}_{\text{fairness postcondition, } \phi_y}$$

The above verification formula is defined using two copies of the same NN, its input constraints (x), and output constraints (y). The above constraint also support the Definition 2 and Definition 3. For Definition 2, we update the equality in ϕ_x^f with $|x_j - x'_j| \leq \epsilon_j$. For Definition 3, we use the bounds from target T in ϕ_x^d instead of the original domains constraints.

Unlike local robustness [37, 38], the above fairness constraint requires twice as many input variables (x_i and x'_i) and two copies of networks to obtain outputs (y and y'). Other than that, the complexity of the constraint depends on the number of neurons and the structure of M. In each neuron of the hidden layers, the solver divides the query into two branches: if the weighted sum is non-negative or else, as shown in Eq. 1. Thus, for n neurons the query divides into 2^n branches, which can not be parallelized [50]. In our approach, we identify the neurons that always remain inactive and remove them to reduce the complexity. Furthermore, we formulate the verification as an SMT based problem so that we can provide counterexample along the certification and verify ϵ-fairness and targeted fairness. Existing abstract interpretation based techniques [27] can not be used towards that goal [49, 51].

B. Solution Overview

Fairify takes two inputs: trained neural network and fairness verification query. Then it performs three main steps: 1) input partitioning, 2) sound pruning, and 3) heuristic based pruning. An overview of the solution is depicted in Figure 1. Before going to step 1, Fairify preprocesses the verification query. It computes the WP of the postcondition to reduce the complexity of the verification formula.

After the preprocessing step, we have precondition defined on the neurons of input layer and postcondition defined on the neurons of the output layer. Then Fairify performs the input partitioning method. The main objective is two-fold: 1) the verification query becomes simpler, i.e., now the solver has to check less input region to prove the constraint; 2) given a smaller partition, the NN exhibits certain activation pattern so that we can prune the network.

After the input partitioning, Fairify first attempts the sound pruning approach where we use the tightened input bounds for the partitions to compute bound for each neuron. Here, we use the white-box access to the network weights and biases. Then we perform another step of verification on each neuron to prove its activation. This process is applied layer-wise, only one layer at a time. Hence, the checking takes very little time and identifies additional neurons that are inactive. Then inactive neurons are removed from the NN to reduce its size. Removing

neurons from the NN largely reduces the complexity of the verification. Finally, Fairify leverages an SMT solver to solve the query on the pruned version of the network. Fairify takes a soft-timeout as input parameter. When the soft-timeout is reached without any result, Fairify takes the heuristic based pruning approach and attempts to solve the verification query.

In this pruning approach, Fairify uses the network heuristics, e.g., weight magnitudes and their distribution among the neurons. We conduct a simulation on the NN to profile the neurons. If a neuron is never activated, it is a candidate for removal. Then we compare the candidates' distribution of magnitude with that of non-candidates. Thus, we identify additional neurons that are inactive in the given partition. Since this step may prune neurons which are rarely activated, the pruned version can lose small accuracy. Fairify takes a conservative approach which result in very little to no loss of accuracy. The approach design also allows **partial verification** for a subset of the partitions. The goal of verification is to provide SAT/UNSAT for as many partitions as possible within given time budget. Next, we describe three main components of Fairify in detail.

C. Input Partitioning

Fairify takes the parameter MS, which is the maximum size of an attribute A_i. Based on MS, Fairify automatically partitions the input domain into m regions following the Algorithm 1. We first divide each attribute into $\lceil (ub - lb)/MS \rceil$ partitions, and then taking each partition from each attribute, we get the regions. For each region, we assign a copy of the NN so that different pruning can be applied for each region as well as the query processing can be parallelized. The verification results for the partitions are accumulated as follows.

- If one partition is SAT, the whole problem becomes SAT. The counterexample shows a violation for the fairness query.
- If one partition provides UNSAT, the whole problem is not necessarily UNSAT. However, the verification provides guarantee that there are no two inputs possible in the given partition that violates the property. This provides partial certification which has benefits in provable repair.
- To prove that the whole problem is UNSAT, all the partitions required to be UNSAT.

Finally, after the partitioning, we shuffle the partitions to check different parts of the regions in the given timeout.

D. Sound Pruning

This step receives a number of verification problems from input partitioning. Each problem is associated with a copy of NN and the query ϕ. The query is updated from the original query by tightening the bound of each attribute, given the smaller region. Now, we attempt to prune the network by removing the neurons that do not impact the prediction of the current copy of NN.

Static analysis and Pruning of NN. Before deployment, a trained NN can be assessed for certain properties. We obtain the weights and biases of the network to analyze its behavior.

Algorithm 1 Input partitioning based on domain constraints

Input: Attributes A, Input domain $\mathbb{I}$, Max-size MS, Query Q
Output: Region set R

```
1: procedure INPUT_PARTITIONER(A, 𝕀, MS, 𝒩)
2:     for each attribute A_i ∈ A do
3:         PT_{A_i} = []
4:         if |A_i| > MS then
5:             low = LB(𝕀_i)                    ▷ lower bound of attribute
6:             high = UP(𝕀_i)                   ▷ upper bound of attribute
7:             if Q contains |x − x| ≤ ε_i |x, x' ∈ A_i then
8:                                              ▷ A_i is relaxed
9:                 part = [low, high]           ▷ no partitioning
10:            else
11:                while low ≤ high do
12:                    high = low + MS
13:                    part = [low, high]
14:                    Add part to PT_{A_i}
15:                end while
16:            end if
17:        end if
18:    end for
19:    R = {(part) | part ∈ PT_{A_i}, ∀A_i ∈ A}
20:    return R
21: end procedure
```

Especially, each neuron does not contribute to the decision. The value of certain neurons depends on the incoming values, associated weights and bias. Because of the ReLU activation function, many nodes become inactive, i.e., gets a zero valuation whenever the WS is negative. In our approach, we analyze activation pattern to remove those neurons which are always inactive. We perform such analysis using specific bounds on the input values. For example, if we can assert $v_3^3 = 0$ for certain bounds on inputs $lb_1 < v_1^1 < ub_1$, $lb_2 < v_1^2 < ub_2$, then removing v_3^3 and the associated edges does not impact the value of v_4^1 and v_4^2. After obtaining the weights and bias, we translate the NN into imperative program representation [51] so that it could be executed symbolically and constraint checker can assert first-order formulas. We leverage Numpy arrays and matrix operations to enable tracking the NN structure (e.g., layers, neurons) and perform network pruning. We take two steps to find such neurons: 1) interval analysis and 2) individual verification. The complete sound pruning algorithm is presented in Algorithm 2.

1) Interval analysis: We perform bound computation for the neurons in each hidden layer by using the bounds from its preceding layer, as shown in line-18 in Algorithm 2. Here, we used interval arithmetic to compute the bounds. First, we separate the positive and negative incoming weights for a neuron v_i^j. Then we compute its maximum value by multiplying the upper bounds of the incoming neurons with positive weights and lower bounds with negative weights. We do the opposite to get the minimum value of v_i^j. The bounds are calculated without considering the activation functions but only the weighted sum. If the weighted sum is always ≤ 0, we can remove the neuron from the NN with the edges connecting to it. Since the interval arithmetic on multiplication does not lose any accuracy, the bounds always hold for the neurons. When we find that the

Algorithm 2 Sound pruning

Input: Network $\mathcal{N}$, Weights W, Biases B, Region R
Output: Pruned network $\mathcal{N}'$

procedure SOUND_PRUNING($\mathcal{N}, W, B$)
  candidates = [] ▷ The candidate neurons for removal
  for hidden neuron v_i^j *in* Layer L_i **do**
    lb, ub = NEURON_BOUND(v_i^j, W, B)
    if ub < 0 **then**
      candidates.add(v_i^j)
    else if lb > 0 **then**
      $\mathcal{N}'$ = Update($\mathcal{N}$) ▷ Remove ReLU function from v_i^j
    else
      if INDIVIDUAL_VERIFICATION(v_i^j, L_{i-1}) **then**
        candidates.add(v_i^j)
      end if
    end if
  end for
  $\mathcal{N}'$ = Update($\mathcal{N}$) ▷ Remove neurons in candidates
  return $\mathcal{N}'$
end procedure
procedure NEURON_BOUND(v_i^j, W, B)
  p_w: non-negative incoming weights
  n_w: negative incoming weights
  $p_w = \{w_{i-1,j} \mid w_{i-1,j} \geq 0, \forall j \in \{1, ..., |L_{i-1}|\}$
  $n_w = \{w_{i-1,j} \mid w_{i-1,j} \leq 0, \forall j \in \{1, ..., |L_{i-1}|\}$
  $UB(v_i^j) = \Sigma_{w \in p_w} w * UB(v_{i-1}^j) + \Sigma_{w \in n_w} w * LB(v_{i-1}^j) + b_i^j$
  $LB(v_i^j) = \Sigma_{w \in p_w} w * LB(v_{i-1}^j) + \Sigma_{w \in n_w} w * UB(v_{i-1}^j) + b_i^j$
  return $(UB(v_i^j), LB(v_i^j))$
end procedure
procedure INDIVIDUAL_VERIFICATION(v_i^j, L_{i-1})
  precondition $pre = \{LB \leq v_{i-1}^j \leq UB, \forall j \in \{1, ..., |L_{i-1}|\}\}$
  postcondition $post = v_i^j < 0$
  singular_verification = SMT-Solver($pre, post$)
  if singular_verification = UNSAT **then**
    return $true$
  end if
  return $false$
end procedure

upper bound of v_i^j is ≤ 0 we mark v_i^j for removal. The key intuition for this step is that since we have tightened bounds for the input layer (after partitioning), the bounds of hidden neurons are also tighter. Therefore, we can remove more neurons by applying this step on the partitioned regions. However, the bounds of many neurons, especially in deep layers, may not be tight enough to prove it is inactive. So, we apply the next step of individual verification on each neuron.

2) Individual verification: We formulate verification query for each hidden neuron to further prove whether it is inactive. The precondition for the verification query for neuron v_i^j consists of the bounds v_{i-1}^j, and the postcondition is $v_i^j < 0$. Then we leverage the SMT solver to prove that the neuron is inactive. We run these individual verification queries on neurons in the increasing order of the layers. So, all the neurons of one layer are verified before going to the next layer. Similar to the bound analysis, the verification query does not include any non-linear activation functions i.e., ReLU. In addition, the query is designed with the precondition that contains values of only the immediate preceding layer. Furthermore, we apply individual verification query only on the neurons that are not already pruned by interval analysis. Therefore, the individual verification queries are faster and can find more inactive neurons that were not found in the previous step.

E. Heuristic Based Pruning

We investigated the real-world models used in the fairness problems on structured data, and we found that the datasets are far sparse from problem that involve high dimensional data such as image classification or natural language processing. For example, each instance in the MNIST dataset is (28, 28) Numpy array, whereas a popular Adult Census dataset contains data instances each with the shape (1, 13). It causes the trained NN to have many neurons that are never activated or only activated for a specific region. Applying the methods in the previous stage, we could detect some provably guaranteed inactive neurons in the NN. But there can be further such neurons which are not activated for the given region in practice. How can we detect those neurons?

A variety of network heuristics has been studied for different purposes such as quantization, efficiency, accuracy, etc. [52–54], which can be performed in different stages of the development e.g., before or during training. Here, we proposed a novel conservative approach of NN pruning for fairness verification based on heuristics. Our goal is to reduce the network size without losing accuracy to the extent possible.

The algorithm for heuristic based pruning is shown in Algorithm 3. For learning the heuristics of the network, we run the neuron profiler. The objective is to get the distribution of magnitudes of each neuron. First, we create a simulated dataset $\mathcal{D}$ of size S by uniformly generating valid inputs from attribute domain. Then the network M is run S times to record the values for each neuron. In practice, we used $S = 1000$, which allows to separate most of the candidates and non-candidates. Initially, all the neurons are selected as candidates for removal. If a neuron gets non-zero value for at least one execution of the simulation, we remove that from the candidate list. Thus, after recording the values of both the candidates and non-candidates, we confirm that whether the distribution of positive magnitude differs significantly. The intuition is that even if the neuron in the candidate is inactive for all the executions, it might get activated for some valid input data. However, the magnitude distribution of the neurons suggests whether it is active or not. Our evaluation shows that there is a significant difference in their distribution. For example, we observed that in the first hidden layer of the NNs, the mean and median of the positive magnitude of non-candidates is at least 10 times larger.

The above heuristic allows to compare the magnitude of each candidate neuron and conservatively select it for removal. Fairify takes a tolerance level as input for this comparison. In our evaluation, we used 5% as the tolerance level. Fairify selects the candidates which has a magnitude less than the 5-percentile of the non-candidate neurons. The weight magnitude difference is more prominent in deep layers than the shallow ones. So, we do layer-wise comparison and candidate selection based on the heuristics.

Algorithm 3 Heuristics based pruning

Input: Network $\mathcal{N}$, Domain $\mathbb{I}$, Simulation size S, Tolerance T
Output: Pruned network $\mathcal{N}'$

procedure HEURISTIC-PRUNE($\mathcal{N}$, $\mathbb{I}$, S)
  $candidates, Mag^+, Mag^-$ = NEURON-PROFILER($\mathcal{N}, S$)
  $Dist_{candidate}$ ▷ Distribution of the candidates
  $Dist_{noncandidate}$ ▷ Distribution of the non-candidates
  for v_i^j in hidden layer L_i **do**
    check candidates and non-candidates distribution differs
    if $Mag^+(v_i^j) < T$-level of $Dist_{noncandidate}$ **then**
      Remove v_i^j from M
    end if
  end for
end procedure
procedure NEURON-PROFILER($\mathbb{I}$, S)
  $\mathcal{D} = \{X | X_i \in \mathbb{I}$ and $|X| = S\}$
  $candidates = \{v_i^j \mid v_i^j \in L\}$ ▷ L is a hidden layer in M
  for $X_i \in \mathcal{D}$ **do**
    Run $\mathcal{N}(X_I)$ to record heuristic
    for hidden neurons v_i^j in $\mathcal{N}$ **do**
      if v_i^j is active **then**
        $candidates.remove(v_i^j)$
      end if
      if $v_i^j \geq 0$ **then** ▷ Positive magnitude
        $Mag^+ = |v_i^j|$
      else ▷ Negative magnitude
        $Mag^- = |v_i^j|$
      end if
    end for
  end for
  return $candidates, Mag^+, Mag^-$
end procedure

One might argue that since this pruning does not provably guarantee the same outcome as the original NN, the verification result can be inaccurate. However, the idea here is to use the pruned version of the NN in production as opposed to the original NN. If we certify the pruned NN and observe little or no accuracy decrease, then the pruned and verified model itself can be used in the production. Thus, we can preserve the sanity of the verification results. Our evaluation shows that it is possible to carefully select the heuristic so that NN is pruned conservatively, which affects accuracy negligibly but enables faster verification.

V. EVALUATION

In this section, first, we discuss the experimental details and then answer three research questions regarding the utility, scalability, and performance of our approach.

A. Experiment

1) Benchmarks: The verification benchmark is a crucial part of the evaluation. We undertook several design considerations to enable fairness verification in development pipeline. Therefore, unlike prior works we created comprehensive real-world benchmark of NN models from theory and practice.

We evaluated Fairify on three popular fairness datasets i.e., Bank Marketing (BM), Adult Census (AC), and German Credit (GC) [1, 3, 8, 23]. The benchmark models (Table I) are collected from four different sources. First, we followed the methodology of Biswas and Rajan to collect real-world NNs from Kaggle [2]. We searched all the notebooks under the three datasets in Kaggle and found 16 different NNs. Second, Zhang et al. used 3 NNs trained on the aforementioned datasets [8]. Third, Udeshi et al. [24] evaluated fairness testing on one NN architecture, which is further used by Aggarwal et al. [1] on the three datasets. We found that two of these three models are also implemented in Kaggle notebooks. Finally, NN models AC8-12 were created by [27, 28] for dependency fairness certification. Thus, we created a fairness benchmark of 25 NNs. The networks used for these fairness problems are fully connected with ReLU activation functions, which is also observed by many prior works [8, 20, 27, 28]. The models and datasets are placed into our replication package to make the tool self-contained. The details of the datasets are as follows:

TABLE I: The NN benchmark for fairness verification

Dataset	Model	Source	#Layers	#Neurons	Acc %
Bank Marketing	BM1	Kaggle	4	97	89.20
	BM2	Kaggle	4	65	88.76
	BM3	Kaggle, [1, 24]	3	117	88.22
	BM4	Kaggle	5	318	89.55
	BM5	Kaggle	4	49	88.90
	BM6	Kaggle	4	35	88.94
	BM7	Kaggle	4	145	88.70
	BM8	[8]	7	141	89.20
German Credit	GC1	Kaggle	3	64	72.67
	GC2	[1, 24]	3	114	74.67
	GC3	Kaggle	3	23	75.33
	GC4	Kaggle	4	24	70.67
	GC5	[8]	7	138	69.33
Adult Census	AC1	Kaggle	4	45	85.24
	AC2	Kaggle, [1, 24]	3	121	84.70
	AC3	Kaggle	3	71	84.52
	AC4	Kaggle	4	221	84.86
	AC5	Kaggle	4	149	85.19
	AC6	Kaggle	4	45	84.77
	AC7	[8]	7	145	84.85
	AC8	[27, 28]	4	10	82.15
	AC9	[27, 28]	6	12	81.22
	AC10	[27, 28]	6	20	78.56
	AC11	[27, 28]	6	40	79.25
	AC12	[27, 28]	11	45	81.46

Bank Marketing dataset contains marketing data of a Portuguese bank which is used to classify whether a client will subscribe to the term deposit [55]. It has 45,000 data instances with 16 attributes, and age is considered the protected attribute. *German Credit* dataset contains 1000 data instances of individuals with 20 attributes who take credit from a bank [56]. The task is to classify the credit risk of a person (good vs bad). *Adult Census* dataset contains United States census data of 32,561 individuals with 13 attributes [57]. The task is to predict whether the person earns over $50,000 in a year.

2) Experiment setup: Fairify is implemented in Python and the models are trained using Keras APIs [27]. Following the prior works in the area [20], we used Z3 [58] as the off-the-shelf SMT solver for manipulating the first-order formulas. However, other SMT solvers can also be used, since our technique of input partitioning and network pruning can work independent of any SMT solver. We used SMT solver because they can solve arbitrary first-order constraints and enable solving both

ϵ-fairness and targeted fairness queries. The experiments are executed on a 4.2 GHz Quad-Core Intel Core i7 processor with 32 GB memory.

Input. Fairify takes the trained NN model, input domain, the verification query, maximum partition size, and timeout as inputs. The trained models are saved as `h5` files, from which Fairify extracts the necessary information e.g., weight, bias, structure. The query includes the name of the protected attributes and relaxation information of the other attributes.

Output. Fairify provides verification result for each partition. The results include verification (SAT/UNSAT/UNKNOWN), counterexample (if SAT), and pruned NN for the partitions.

B. Results

We answered three broad research questions for evaluation:

- RQ1: What is the **utility** of our approach?
- RQ2: Is Fairify **scalable** to relaxed fairness queries?
- RQ3: What is the **performance** with respect to time and accuracy of the approach?

1) Utility: First, we evaluated the baseline fairness verification results and then compared with our approach, which is shown in Table II. Since GC and AC contained multiple protected attributes (PA), we setup multiple verification for each of those NNs. However, the verification results are consistent over different PAs. Hence, we showed result for one PA for each dataset in Table II. The whole result is also available in our supplementary material [59]. In this RQ, we evaluated the fairness defined in Eq. (1). We discussed the verification of relaxed fairness constraints Eq. (2) and (3) in RQ2.

For the baseline verification, we encoded fairness property into satisfiability constraints, and then passed the *original NN* and constraints to the SMT solver. In addition, we attempted to verify the original NN with input partitioning to understand the contribution of partitioning alone. On the other hand, Fairify used the method of input partitioning and NN pruning together to demonstrate the improvement over the baseline. For the baseline, we set a timeout of 30 hours for the solver and run the verification for the models in our benchmark. Only one model (AC6) could be verified within the timeout. For the other models, the solver reported UNK i.e., the model could not be verified within the time limit. Furthermore, we try the baseline verification with input partitioning and run the queries on the NNs but we get the same verification result as the baseline. The main takeaway is that the difficulty in verifying fairness lies in the complex structure of the NNs. Only input partitioning reduces the input space to be verified, but that does not reduce the complexity of NN. Fairify combines partitioning and pruning to enable network complexity reduction, which made the verification feasible.

How effective is our approach to verify fairness of NN? We presented the verification results of 25 NNs in Table II. We found that Fairify produces verification results very quickly compared to the baseline. For each model, we run the verification task for 30 minutes (hard-timeout). Since we have divided the single verification into multiple partitions, we set 100 seconds as the soft-timeout for the SMT solver, which means that the solver gets at most 100 seconds to verify. When the result of the sound verification is UNK, then Fairify attempts the heuristic based pruning and runs the SMT solver for another 100 seconds. The goal of using a short soft-timeout is to show the effectiveness of our approach over baseline.

The results show that 19 out of 25 models were verified within the 30 minutes timeout. The models that could not be verified in that time period were considered again with scaled experiment setup in RQ2. Fairify takes the maximum size of an attribute (MS) as an input to automatically partition the input region. The user can select MS based on the range of the attributes in the dataset. The timeout and MS can be configured based on the budget of the user. For BM and GC models, we used 100 as MS, and for AC models we used 10 as MS. To that end, Fairify divided input region into 510, 201, and 16000 partitions using Algorithm 1. Here is an example verification task from our evaluation:

Example: While verifying AC3 (*race* as PA), Fairify takes the following partition as a sub-problem. First, it attempts sound pruning and achieves 86.27% compression. Then it runs verification query for 100 seconds and reports SAT with the counterexamples **C1** and **C2** in 21.47 seconds. Note that, these are two inputs for which the NN is not fair. Here, the two individuals had the same attributes except for *race* but were classified as *bad* and *good* credit-class, respectively.

workclass: [0, 6], marital-status: [0, 6], relationship: [0, 5], race: [0, 4], sex: [0, 1], age: [80, 89], education: [0, 9], education-num: [11, 16], occupation: [0, 9], capital-gain: [10, 19], capital-loss: [0, 9], hours-per-week: [51, 60], native-country: [30, 39]

C1: $[89, 6, 9, 14, 1, 0, 0, 0, 1, 14, 8, 59, 39]$
C2: $[89, 6, 9, 14, 1, 0, 0, 0, 0, 14, 8, 59, 39]$

Depending on the complexity of each NN, Fairify could complete verification for a certain number of partitions in the given timeout period. Whenever we get SAT for at least one partition, the whole verification is SAT. Fairify also reports the counterexample when it reports SAT. We included the detailed results for each partition including all generated counterexamples in our replication package [40].

Is the NN pruning effective for verifying fairness? The baseline models cannot be verified in a tractable amount of time. Also, the partitioning alone on the baseline can not improve the verification. After being able to prune the models significantly, we could verify them in a short period. We computed the amount of pruning applied to the models. For the original M and pruned version M', compression ratio is calculated using the following formula: $1 - |M'|/|M|$, where $|M|$ is the number of neurons in M. The average compression percentage in Table II shows that Fairify could reduce the size of NN highly in all the models. Heuristic based pruning is only applied when Fairify cannot get verification result within the soft-timeout. Furthermore, the individual contribution (compression) of sound and heuristic based pruning is showed in Table II. Heuristic based pruning is applied on the already pruned version of the NN. Therefore, the compression ratio for heuristics based pruning is lower. We found that 13.98% times Fairify attempted

TABLE II: Fairness verification results for neural network models

PA	M	Ver	#P	Coverage%	#sat	#unsat	#unk	HP-A	HP-S	C(S)	C(H)	SV time	HV time	Total time
Age	BM1	SAT	75	13.14	9	58	8	9	1	.90	.01	14.04	11.00	25.40
	BM2	SAT	141	26.08	30	103	8	9	1	.85	.01	7.76	5.83	13.84
	BM3	SAT	139	26.47	27	108	4	6	2	.96	.00	9.78	3.42	13.49
	BM4	SAT	37	6.08	1	30	6	6	0	.91	.03	17.10	16.29	49.98
	BM5	SAT	510	99.61	114	394	2	7	5	.83	.00	2.29	0.54	2.96
	BM6	SAT	510	100.00	156	354	0	3	3	.76	.00	1.17	0.08	1.36
	BM7	SAT	124	23.33	62	57	5	9	4	.92	.01	8.31	5.10	14.89
	BM8	UNK	23	3.14	0	16	7	10	3	.79	.04	46.23	31.64	79.08
Sex	GC1	SAT	31	13.43	27	0	4	6	2	.76	.01	41.78	15.98	58.18
	GC2	SAT	11	1.99	4	0	7	9	2	.78	.04	97.18	75.67	174.18
	GC3	SAT	201	100.00	195	6	0	1	1	.69	.00	1.29	0.24	1.63
	GC4	SAT	201	100.00	2	199	0	1	1	.63	.00	0.73	0.14	0.97
	GC5	UNK	12	1.49	0	3	9	9	0	.59	.03	75.27	75.43	151.53
Race	AC1	SAT	29	0.14	8	15	6	8	2	.64	.04	40.73	23.50	64.43
	AC2	SAT	15	0.04	4	3	8	10	2	.82	.04	72.62	54.86	128.45
	AC3	SAT	23	0.11	16	1	6	10	4	.75	.02	52.75	32.33	85.40
	AC4	UNK	8	0.00	0	0	8	8	0	.67	.20	100.57	100.08	241.70
	AC5	SAT	10	0.02	3	0	7	9	2	.71	.13	98.14	74.06	180.41
	AC6	SAT	19	0.07	6	5	8	9	1	.49	.05	55.70	44.43	100.37
	AC7	UNK	14	0.04	0	7	7	10	3	.59	.11	78.22	51.09	132.28
	AC8	SAT	101	0.63	82	19	0	1	1	.22	.00	17.59	0.13	17.84
	AC9	SAT	741	4.63	399	342	0	4	4	.19	.00	2.29	0.02	2.44
	AC10	SAT	20	0.09	6	8	6	10	4	.27	.05	62.07	32.82	95.07
	AC11	UNK	9	0.00	0	0	9	9	0	.11	.02	100.13	100.12	200.66
	AC12	UNK	16	0.04	0	7	9	9	0	.29	.01	57.56	56.35	114.19

Experiment setup: soft-timeout 100s, hard-timeout 30m, max-partition size: 100 (BM, GC), 10 (AC).
M: models, Ver: Verification, #P: number of partitions, HP-A: # times **H**euristic **P**runing **A**ttempted, HP-S: # times **H**euristic **P**runing **S**ucceeded,
C: average **C**ompression ratio-(S): sound pruning, (H): heuristic pruning, (Average) times are in seconds – SV: sound verification, HV: heuristic verification.

heuristic pruning, 19.38% of those times Fairify provided SAT or UNSAT result, meaning that the additional pruning of Fairify helped to complete the verification. In the example showed above, Fairify did not attempt a heuristic based pruning, since it got the result in sound pruning step.

Can Fairify be used to localize fairness defects? For ML models, it is difficult to reason or find defects since the model learns from data. In fairness problems, oftentimes the model learns from biased data or augments the bias during training. If we can filter the input domain where the model is unfair, then it would guide fairness repair. For example, in model BM3, Fairify provides SAT for 27 partitions and UNSAT for 108 partitions. The developer can leverage the verification result to further improve the training data and retrain the network. Another novelty is that since Fairify verifies multiple copies of pruned NN, the developer may choose to deploy those pruned versions into production. When an input comes to the system, the software can choose to use the verified copies of NN. Qualitatively, Fairify provides the following two main utilities:

a) Tractability and speedup: The results showed that the verification for the NN becomes tractable when our approach is applied. Furthermore, the partition size and timeout can be tuned based on the complexity of the NN. In our evaluation, we used a short timeout of 30 minutes [41]. The verification has been possible for most of the models in this time because of successful NN pruning. In addition, as soon as the first SAT is found, the developer may choose to stop running verification.

b) Partial verification: Even after getting SAT for one partition, our evaluation continued verification for other partitions, which essentially provides partial verification. For example, partitions with UNSAT imply that the NN is fair for that specific input region. Similarly, SAT with the counterexamples for a partition can be used towards repairing the NN, which is a potential future work.

TABLE III: Verification with scaled experiment setup

M	Ver	#P	#sat	#unsat	#unk	HPA	HPS	C(S)	C(H)	SV ⏱	HV ⏱	Total ⏱
BM8	SAT	48	4	37	7	10	3	.85	.02	47.64	33.46	81.68
GC5	UNK	12	0	3	9	9	0	.60	.03	150.25	150.55	301.83
AC4	SAT	24	15	4	5	9	4	.92	.02	96.65	56.17	160.26
AC7	UNK	21	0	15	6	9	3	.77	.02	111.11	65.64	178.12
AC11	UNK	9	0	0	9	9	0	.21	.02	200.23	200.37	401.18
AC12	SAT	35	3	26	6	9	3	.41	.01	63.47	43.93	107.71

Soft-timeout 200s, hard-timeout 60m, max-partition: 10 (for BM, GC), and 6 (for AC)

2) Scalability: In RQ1, we have set a small timeout and could verify 19 out of 25 models. In this RQ, we set a scaled experiment setup to further verify the remaining 6 models. We noticed that these 6 NNs are complex because of more number of layers and neurons. This time we used a hard-timeout of 1 hour and soft-timeout of 200 seconds, essentially doubling the timeouts. We also reduced the maximum size of the partition (MS). The results are shown in Table III. We have verified the 3 out of the 6 models within that 1 hour. The results demonstrate the scalability of our approach for more complex NNs. We found that for some partitions, the compression ratio is more, and hence the SMT solver could verify quickly. So, it would be an interesting future work to prioritize the partitions to verify for efficiency. Next, we showed whether our approach can verify complex verification queries for all the models.

Can Fairify verify relaxed and targeted verification queries? To answer this question, we created relaxed and targeted fairness queries according to Definition 2 and Definition 3 respectively. The verification results for the relaxed queries are presented in Table IV. First, for the **relaxation** of individual fairness, we define small perturbation (ϵ) on the non-protected attributes so that two individuals are considered similar even if they are not equal in any non-protected attributes. Those two individuals still have to be classified to the same class to ensure fairness. Therefore, the relaxation on the queries impose stricter fairness requirement. We created six different

TABLE IV: Verification of neural networks for relaxed and targeted fairness queries

	Result	ϕ	BM1	BM2	BM3	BM4	BM5	BM6	BM7	BM8	ϕ	GC1	GC2	GC3	GC4	GC5	ϕ	AC1	AC2	AC3	AC4	AC5	AC6	AC7	AC8	AC9	AC10	AC11	AC12
Relaxed	Ver		SAT	SAT	SAT	SAT	SAT	SAT	SAT	SAT		SAT	SAT	SAT	SAT	UNK		SAT	SAT	SAT	SAT	SAT	SAT	UNK	SAT	SAT	SAT	UNK	UNK
	#P		333	566	1600	132	4915	8671	590	112		111	32	2697	9695	9		216	75	129	41	37	177	66	639	3991	216	31	123
	#sat	ϕ_{r11}	40	65	233	10	731	1484	168	11	ϕ_{r12}	107	24	2026	185	0	ϕ_{r13}	9	8	34	5	3	28	0	218	483	19	0	0
	#unsat		279	493	1358	111	4180	7187	411	87		0	2	671	9510	0		193	55	86	22	18	133	53	418	3508	192	13	107
	#unk		14	8	9	11	4	0	11	14		4	6	0	0	9		14	12	9	14	16	16	13	3	0	5	18	16
	Ver		SAT	SAT	SAT	SAT	SAT	SAT	SAT	SAT		SAT	SAT	SAT	SAT	UNK		SAT	SAT	SAT	SAT	SAT	SAT	UNK	SAT	SAT	SAT	UNK	SAT
	#P		968	1514	2768	157	7601	13779	1340	188		116	30	3175	8361	20		210	74	124	30	27	117	44	402	4811	164	18	53
	#sat	ϕ_{r21}	65	110	208	10	583	1200	187	6	ϕ_{r22}	106	16	2343	79	0	ϕ_{r23}	47	20	84	13	11	47	0	182	1294	34	0	1
	#unsat		890	1396	2550	143	7014	12579	1151	167		1	2	832	8282	2		157	45	35	5	3	56	31	220	3517	128	0	36
	#unk		13	8	10	4	4	0	2	15		9	12	0	0	18		6	9	5	12	13	14	13	0	0	2	18	16
Targeted	Ver		SAT	SAT	SAT	SAT	SAT	SAT	SAT	SAT		SAT	SAT	SAT	SAT	UNK		SAT	SAT	SAT	SAT	SAT	SAT	UNK	SAT	SAT	SAT	UNK	UNK
	#P		261	546	1541	92	5937	12233	544	90		162	32	3810	11067	18		223	67	115	25	23	111	38	448	6800	108	18	68
	#sat	ϕ_{t11}	34	61	208	9	881	1939	157	8	ϕ_{t12}	153	17	2786	96	0	ϕ_{t13}	55	16	75	7	8	52	0	179	1624	23	0	0
	#unsat		216	479	1322	72	5050	10294	374	67		0	2	1023	10971	0		160	37	32	3	1	47	26	266	5176	81	0	50
	#unk		11	6	11	11	6	0	13	15		9	13	1	0	18		8	14	8	15	14	12	12	3	0	4	18	18
	Ver		SAT	SAT	SAT	SAT	SAT	SAT	SAT	SAT		SAT	SAT	SAT	SAT	UNK		SAT	SAT	SAT	SAT	SAT	SAT	UNK	SAT	SAT	SAT	UNK	SAT
	#P		432	683	1987	108	6474	13855	986	89		187	46	5957	13426	18		281	85	119	35	33	119	43	677	6781	164	18	59
	#sat	ϕ_{t21}	57	76	270	10	904	2224	285	11	ϕ_{t22}	175	28	3534	63	0	ϕ_{t23}	61	18	79	14	16	51	0	309	1765	36	0	1
	#unsat		361	602	1705	87	5569	11631	693	64		4	7	2422	13363	0		214	58	33	8	4	57	26	368	5016	125	0	41
	#unk		14	5	12	11	1	0	8	14		8	11	1	0	18		6	9	7	13	13	11	17	0	0	3	18	17

Relaxed queries: ϕ_{r11}: duration < 5, ϕ_{r12}: job $< \infty$, ϕ_{r21}: credit-amount < 100, ϕ_{r22}: foreign-worker $< \infty$, ϕ_{r31}: marital-status $< \infty$, ϕ_{r32}: age < 5,
Targeted queries: ϕ_{t11}: personal-loan & profession:entrepreneur, ϕ_{t12}: previous-marketing: yes, ϕ_{t21}: # credits=2, ϕ_{t22}: foreign-worker & credit-purpose: education,
ϕ_{t31}: 30$\leq$age$\leq$35, ϕ_{t32}: education: bachelor or doctorate

such queries run Fairify on all the models. In each query, we selected an additional non-protected attribute, which was relaxed. For example, a verification query for AC is (ϕ_{r31}): *Is the NN fair with respect to race, where any two people are similar irrespective of their marital status?* We found that 21/25 for ϕ_{r1} and 22/25 for ϕ_{r2} were verified within one hour. Compared to Table II, the number of SAT found is more since ϵ-fairness is a stricter requirement and creates possibility of more counterexamples. Second, for the **targeted** verification, we created six other fairness queries that targets a specific population. For example, ϕ_{t32} verifies that *whether the NN is fair for people who have bachelor or doctorate education*. The results show that Fairify can verify most of the models in a quick time. Similar to the scaled experiment setup in Table III, we used a smaller partition size in this experiment. Although the queries are more complex, Fairify could provide counterexample or certification for many partitions.

3) Performance: ***How quickly the verification is done and what is the overhead?*** Depending on the verification query and the model, the verification time varies. Table II presents the average time taken for the partitions of each model. We showed the time taken by the SMT solver to output a result in the sound verification phase, and in heuristic verification phase. We also calculated the total time taken to complete verification, which includes the partitioning and pruning discussed in §IV. The results show that the additional time taken for the three steps in our approach is negligible with respect to the time taken by the SMT solver. The partitioning of inputs is a one time step. For each partition, we apply pruning once or twice. However, pruning is static operation without any complex constraint solving. The only part of our pruning steps that take more time compared to other steps is the individual verification of each neuron which uses the SMT solver. But in that verification, only one layer is considered at a time, activation functions are excluded, and the number of constraints is at most the number of hidden neurons in a layer. Therefore, excluding the time taken by the SMT solvers to solve the final constraints, Fairify did not take more than 10 seconds for any model in that step.

What is the performance of Fairify compared to the related work? As described in §III, individual fairness verification of NN can not be done with existing robustness checkers [37, 38]. Shriver et al. proposed a framework called DNNV [60] that incorporates the state-of-the-art NN verifiers, e.g., ReluPlex, Marabou, etc. We tried verifying the fairness queries using those DNNV verifiers, however, they can verify queries with a single network input variable only [60, 61]. Other than that, we verified the models (AC8-12) from Libra [27, 28] shown in Table II and Table IV. Fairify could verify the models for all the queries except AC11-12 for two out of four queries. The overall coverage is less than that reported by [28] because Fairify verifies a different property, the configuration is lower, and experiment setup is different. Libra computes abstract domain and projects into the input space to find biased region. So, the precision of Libra depends on the chosen abstract domain. On the other hand, Fairify can be configured for arbitrary queries, partial verification, and additionally we provide counterexample and pruned NN as output. Therefore, Fairify can be more appropriate for defect localization or repair. One limitation of Fairify is that when the NN is deep and wide at the same time, the pruning ratio is less, and the SMT solver may return UNK in the given timeout. However, developers can configure less conservative approach in heuristic pruning to circumvent the problem. Thus, dynamical tuning of configuration could be a potential future work for Fairify.

What is the accuracy loss of heuristic-based pruning? We calculated the accuracy of the pruned NNs for each partition. When no heuristic based pruning is done, there is no accuracy loss. However, when Fairify applies heuristic based pruning there might be accuracy loss compared to the original NN. We took a conservative approach for the heuristic based pruning. Therefore, there was no accuracy loss for the pruned NN. Note that even if there is a small accuracy reduction for a heuristic based approach, the developer may choose to deploy the pruned version of the NN as opposed to the original one.

VI. Threats to Validity

Construct validity. We leveraged the fairness definitions from prior work and then encoded using well studied weakest precondition and constraint based methodologies. In §III-A, we argued why individual fairness is useful for verification. However, as mentioned by Corbett-Davies and Goel [62], the property can fall short in bias quantification. Our evaluation shows that based on the number of certified partitions, future work can be done to quantify individual fairness. In addition, the proposed sound-pruning strategy leverages interval arithmetic which is sound by construction; as shown by [47] i.e., NNs have well-defined transformers (addition, subtraction, scaling) and hence interval analysis provides sound approximation. For individual verification, we used SMT based constraint solving which is always sound. Additionally, input partitioning (Algorithm 1) creates disjoint partitions of attributes and covers the complete domain. Furthermore, to be able to verify symbolically, we converted the models into Python functions using the weights and biases extracted from the actual model. We followed the same approach taken by prior work for such conversion [20].

External validity. We evaluated Fairify on the popular structured datasets extensively used in prior works [1, 3–8, 20–24, 27]. As pointed out by [21, 27], verifying ML models trained on unstructured data including image or natural-language would require different approach. Yet, because of the wide usage in real-world (loan approval, criminal sentencing, etc.), this line of work only considered such structured data. Further work is needed to scale the approach for high-dimensional datasets. In addition, followed by [1, 8, 24, 27, 28], we considered the ReLU based NNs. To further demonstrate the applicability, we collected top-rated models from practice, which are state-of-the-art accurate models for respective tasks. Verifying fairness of other classes of NNs such as CNN or LSTM could be potential future improvements. Finally, Fairify is built on top of the popular open source libraries and SMT solver Z3 so that other works can leverage the tool. Since Fairify in built independent of specific constraint solver, other solvers could also be used in future.

VII. Related Works

Fairness Testing and Verification. With the increasing need to ensure fairness of AI based systems [2–5, 7, 63], many recent works focused on the fairness testing and verification of ML models [1, 8, 20–25, 64]. While many of the prior works focused on testing ML models [1, 23, 24], more recent works proposed individual fairness testing on NN [8, 25]. While input test generation has been helpful to find fairness violations, verification is more difficult since it proves the property. Although some fairness testing works [1, 64] leverages constraint solvers for test generation, they can not guarantee the absence of violation.

Probabilistic verification techniques have been proposed to verify group fairness property [20, 22]. Recently, John et al. proposed individual fairness verification approach for two different kinds of ML models [21], which do not apply for NN. Urban et al. proposed Libra to provide certification of another property called dependency fairness using abstract interpretation [27]. Mazzucato and Urban extended Libra with one abstract domain [28]. Another class of works proposed methods of individually fair learning by enforcing it during model training [65, 66]. However, our focus in this paper is to verify the already trained models in ML pipeline [67].

NN Verification. Verification of neural network has been studied for various application domains and property of interest. The robustness of NN has gained a lot of attention for safety-critical applications e.g., autonomous vehicles [43, 44, 68]. Research showed that NN can be fooled by applying small perturbations to data instances i.e., adversarial inputs [43, 44, 46, 47, 69]. Algorithms have been proposed to detect adversarial inputs and satisfy local robustness property [38, 45, 46, 70, 71]. §III further describes NN verification and how it compares with the fairness property. Katz et al. proposed efficient SMT solving algorithms to provide robustness guarantee in NN [32, 33]. In addition, some verification algorithms leverage off-the-shelve SMT solver [31, 36]. Research has also been conducted to compute bounds of the neurons and provide probabilistic guarantees for some properties [30, 34, 35]. With the extensive use of NN, many recent works focused on new types of properties using both static and dynamic analysis techniques [41, 46, 72–74].

VIII. Conclusion and Future Work

In this work, we addressed the fairness verification problem of neural networks. Our technique, Fairify can verify individual fairness and its relaxations on real-world NN models, which has not been tackled before. We proposed lightweight techniques to reduce the problem into multiple sub-problems and prune the networks to reduce the verification complexity. Fairify applies static interval analysis and individual verification to provably prune the neurons. In addition, we conducted neuron profiling to observe their heuristic and prune further. While many prior works focused on individual fairness testing and improving, Fairify provides formal guarantee of fairness. Our work also bridges the gap between the theoretical formal analysis and its usage in real-world, as Fairify provides several practical benefits for the developers, e.g., provide counterexample, targeted fairness. The result of Fairify can be leveraged in fairness testing for guided test case generation. Also, the counterexamples can be used to repair the NN in interactive verification setting. Extending Fairify for other activation functions would also be potential future work. Finally, novel analysis can be proposed to prioritize the verification of partitions and dynamically allocate time, which will guide provable design of fairness-aware software.

Acknowledgment

This work was supported in part by US NSF grants CCF-19-34884, CCF-22-23812, and CNS-21-20448. We also thank the anonymous reviewers for their insightful comments. All opinions are of the authors and do not reflect the view of sponsors.

REFERENCES

[1] A. Aggarwal, P. Lohia, S. Nagar, K. Dey, and D. Saha, "Black box fairness testing of machine learning models," in *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2019, pp. 625–635.

[2] S. Biswas and H. Rajan, "Do the machine learning models on a crowd sourced platform exhibit bias? an empirical study on model fairness," in *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2020, p. 642–653. [Online]. Available: https://doi.org/10.1145/3368089.3409704

[3] S. Biswas and H. Rajan, "Fair preprocessing: Towards understanding compositional fairness of data transformers in machine learning pipeline," in *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2021, p. 981–993. [Online]. Available: https://doi.org/10.1145/3468264.3468536

[4] J. Chakraborty, S. Majumder, Z. Yu, and T. Menzies, "Fairway: A way to build fair ml software," in *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2020, pp. 654–665.

[5] M. Hort, J. M. Zhang, F. Sarro, and M. Harman, "Fairea: A model behaviour mutation approach to benchmarking bias mitigation methods," in *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2021.

[6] J. Chakraborty, S. Majumder, and T. Menzies, "Bias in machine learning software: Why? how? what to do?" ser. ESEC/FSE 2021. New York, NY, USA: Association for Computing Machinery, 2021, p. 429–440. [Online]. Available: https://doi.org/10.1145/3468264.3468537

[7] J. M. Zhang and M. Harman, "Ignorance and Prejudice in Software Fairness," in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 2021, pp. 1436–1447.

[8] P. Zhang, J. Wang, J. Sun, G. Dong, X. Wang, X. Wang, J. S. Dong, and T. Dai, "White-box fairness testing through adversarial sampling," in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, 2020, pp. 949–960.

[9] T. Calders and S. Verwer, "Three naive bayes approaches for discrimination-free classification," *Data Mining and Knowledge Discovery*, vol. 21, no. 2, pp. 277–292, 2010.

[10] A. Chouldechova, "Fair prediction with disparate impact: A study of bias in recidivism prediction instruments," *Big data*, vol. 5, no. 2, pp. 153–163, 2017.

[11] M. Feldman, S. A. Friedler, J. Moeller, C. Scheidegger, and S. Venkatasubramanian, "Certifying and removing disparate impact," in *proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*, 2015, pp. 259–268.

[12] M. Hardt, E. Price, and N. Srebro, "Equality of opportunity in supervised learning," in *Advances in neural information processing systems*, 2016, pp. 3315–3323.

[13] C. Dwork, M. Hardt, T. Pitassi, O. Reingold, and R. Zemel, "Fairness through awareness," in *Proceedings of the 3rd innovations in theoretical computer science conference*, 2012, pp. 214–226.

[14] T. Speicher, H. Heidari, N. Grgic-Hlaca, K. P. Gummadi, A. Singla, A. Weller, and M. B. Zafar, "A unified approach to quantifying algorithmic unfairness: Measuring individual &group unfairness via inequality indices," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2018, pp. 2239–2248.

[15] M. B. Zafar, I. Valera, M. G. Rogriguez, and K. P. Gummadi, "Fairness constraints: Mechanisms for fair classification," in *Artificial Intelligence and Statistics*. PMLR, 2017, pp. 962–970.

[16] B. H. Zhang, B. Lemoine, and M. Mitchell, "Mitigating unwanted biases with adversarial learning," in *Proceedings of the 2018 AAAI/ACM Conference on AI, Ethics, and Society*, 2018, pp. 335–340.

[17] G. Pleiss, M. Raghavan, F. Wu, J. Kleinberg, and K. Q. Weinberger, "On fairness and calibration," in *Advances in Neural Information Processing Systems*, 2017, pp. 5680–5689.

[18] R. Zemel, Y. Wu, K. Swersky, T. Pitassi, and C. Dwork, "Learning fair representations," in *International Conference on Machine Learning*, 2013, pp. 325–333.

[19] Y. Brun and A. Meliou, "Software fairness," in *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2018, pp. 754–759.

[20] A. Albarghouthi, L. D'Antoni, S. Drews, and A. V. Nori, "Fairsquare: probabilistic verification of program fairness," *Proceedings of the ACM on Programming Languages*, vol. 1, no. OOPSLA, pp. 1–30, 2017.

[21] P. G. John, D. Vijaykeerthy, and D. Saha, "Verifying individual fairness in machine learning models," in *Conference on Uncertainty in Artificial Intelligence*. PMLR, 2020, pp. 749–758.

[22] O. Bastani, X. Zhang, and A. Solar-Lezama, "Probabilistic verification of fairness properties via concentration," *Proceedings of the ACM on Programming Languages*, vol. 3, no. OOPSLA, pp. 1–27, 2019.

[23] S. Galhotra, Y. Brun, and A. Meliou, "Fairness testing: testing software for discrimination," in *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, 2017, pp. 498–510.

[24] S. Udeshi, P. Arora, and S. Chattopadhyay, "Automated directed fairness testing," in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, 2018, pp. 98–108.

[25] H. Zheng, Z. Chen, T. Du, X. Zhang, Y. Cheng, S. Ji, J. Wang, Y. Yu, and J. Chen, "Neuronfair: Interpretable white-box fairness testing through biased neuron identification," May 21-May 29 2022.

[26] L. Pulina and A. Tacchella, "An abstraction-refinement approach to verification of artificial neural networks," in *International Conference on Computer Aided Verification*. Springer, 2010, pp. 243–257.

[27] C. Urban, M. Christakis, V. Wüstholz, and F. Zhang, "Perfectly parallel fairness certification of neural networks," *Proceedings of the ACM on Programming Languages*, vol. 4, no. OOPSLA, pp. 1–30, 2020.

[28] D. Mazzucato and C. Urban, "Reduced products of abstract domains for fairness certification of neural networks," in *International Static Analysis Symposium*. Springer, 2021, pp. 308–322.

[29] T. Gehr, M. Mirman, D. Drachsler-Cohen, P. Tsankov, S. Chaudhuri, and M. Vechev, "Ai2: Safety and robustness certification of neural networks with abstract interpretation," in *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2018, pp. 3–18.

[30] G. Singh, T. Gehr, M. Püschel, and M. Vechev, "An abstract domain for certifying neural networks," *Proceedings of the ACM on Programming Languages*, vol. 3, no. POPL, pp. 1–30, 2019.

[31] X. Huang, M. Kwiatkowska, S. Wang, and M. Wu, "Safety verification of deep neural networks," in *International conference on computer aided verification*. Springer, 2017, pp. 3–29.

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

[33] G. Katz, D. A. Huang, D. Ibeling, K. Julian, C. Lazarus, R. Lim, P. Shah, S. Thakoor, H. Wu, A. Zeljić *et al.*, "The marabou framework for verification and analysis of deep neural networks," in *International Conference on Computer Aided Verification*. Springer, 2019, pp. 443–452.

[34] Y. Sun, M. Wu, W. Ruan, X. Huang, M. Kwiatkowska, and D. Kroening, "Concolic testing for deep neural networks," in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, 2018, pp. 109–119.

[35] S. Wang, K. Pei, J. Whitehouse, J. Yang, and S. Jana, "Efficient formal safety analysis of neural networks," *Advances in Neural Information Processing Systems*, vol. 31, 2018.

[36] R. Ehlers, "Formal verification of piece-wise linear feed-forward neural networks," in *International Symposium on Automated Technology for Verification and Analysis*. Springer, 2017, pp. 269–286.

[37] D. Gopinath, G. Katz, C. S. Păsăreanu, and C. Barrett, "Deepsafe: A data-driven approach for assessing robustness of neural networks," in *International symposium on automated technology for verification and analysis*. Springer, 2018, pp. 3–19.

[38] G. Katz, C. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer, "Towards proving the adversarial robustness of deep neural networks," *arXiv preprint arXiv:1709.02802*, 2017.

[39] Kaggle, "The world's largest data science community with powerful tools and resources to help you achieve your data science goals." 2010, www.kaggle.com.

[40] S. Biswas and H. Rajan, "Replication package for Fairify," 2022. [Online]. Available: https://github.com/sumonbis/Fairify

[41] B. Paulsen, J. Wang, and C. Wang, "Reludiff: Differential verification of deep neural networks," in *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*. IEEE, 2020, pp. 714–726.

[42] Y. Xiao, I. Beschastnikh, D. S. Rosenblum, C. Sun, S. Elbaum, Y. Lin, and J. S. Dong, "Self-checking deep neural networks in deployment," in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 2021, pp. 372–384.

[43] A. Nguyen, J. Yosinski, and J. Clune, "Deep neural networks are easily fooled: High confidence predictions for unrecognizable images," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 427–436.

[44] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," *arXiv preprint arXiv:1412.6572*, 2014.

[45] R. Huang, B. Xu, D. Schuurmans, and C. Szepesvári, "Learning with a strong adversary," *arXiv preprint arXiv:1511.03034*, 2015.

[46] N. Carlini and D. Wagner, "Towards evaluating the robustness of neural networks," in *2017 ieee symposium on security and privacy (sp)*. IEEE, 2017, pp. 39–57.

[47] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," 2014.

[48] K. Holstein, J. Wortman Vaughan, H. Daumé III, M. Dudik, and H. Wallach, "Improving fairness in machine learning systems: What do industry practitioners need?" in *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, 2019, pp. 1–16.

[49] D. Gopinath, H. Converse, C. Pasareanu, and A. Taly, "Property inference for deep neural networks," in *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2019, pp. 797–809.

[50] Y. Y. Elboher, J. Gottschlich, and G. Katz, "An abstraction-based framework for neural network verification," in *International Conference on Computer Aided Verification*. Springer, 2020, pp. 43–65.

[51] D. Gopinath, M. Zhang, K. Wang, I. B. Kadron, C. Pasareanu, and S. Khurshid, "Symbolic execution for importance analysis and adversarial generation in neural networks," in *2019 IEEE 30th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 2019, pp. 313–322.

[52] X. Dong, S. Chen, and S. J. Pan, "Learning to prune deep neural networks via layer-wise optimal brain surgeon," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, 2017, pp. 4860–4874.

[53] Y. Guo, A. Yao, and Y. Chen, "Dynamic network surgery for efficient dnns," *arXiv preprint arXiv:1608.04493*, 2016.

[54] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," *Advances in Neural Information Processing Systems*, vol. 28, 2015.

[55] S. Moro, P. Cortez, and P. Rita, "A data-driven approach to predict the success of bank telemarketing," *Decision Support Systems*, vol. 62, pp. 22–31, 2014. [Online]. Available: https://archive.ics.uci.edu/ml/datasets/Bank+Marketing

[56] D. H. Hofmann, "German credit dataset: UCI machine learning repository," 1994. [Online]. Available: https://archive.ics.uci.edu/ml/datasets/Statlog+(German+Credit+Data)

[57] R. Kohavi, "Scaling up the accuracy of naive-bayes classifiers: A decision-tree hybrid." in *KDD*, vol. 96, 1996, pp. 202–207. [Online]. Available: https://archive.ics.uci.edu/ml/datasets/adult

[58] L. De Moura and N. Bjørner, "Z3: An efficient SMT solver," in *International conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2008, pp. 337–340.

[59] S. Biswas and H. Rajan, "Supplementary results in the replication package for Fairify," 2022. [Online]. Available: https://github.com/sumonbis/Fairify/blob/main/Appendix-Result.pdf

[60] D. Shriver, S. Elbaum, and M. B. Dwyer, "Dnnv: A framework for deep neural network verification," in *International Conference on Computer Aided Verification*. Springer, 2021, pp. 137–150.

[61] D. Shriver, S. Elbaum, and M. B. Dwyer, "DNNV Documentation," 2021, https://docs.dnnv.org/en/stable/dnnp/introduction.html.

[62] S. Corbett-Davies and S. Goel, "The measure and mismeasure of fairness: A critical review of fair machine learning," *arXiv preprint arXiv:1808.00023*, 2018.

[63] U. Gohar, S. Biswas, and H. Rajan, "Towards understanding fairness and its composition in ensemble machine learning," in *ICSE'23: The 45th International Conference on Software Engineering*, May 14-May 20 2023.

[64] A. Sharma and H. Wehrheim, "Automatic fairness testing of machine learning models," in *Testing Software and Systems: 32nd IFIP WG 6.1 International Conference, ICTSS 2020, Naples, Italy, December 9–11, 2020, Proceedings 32*. Springer, 2020, pp. 255–271.

[65] A. Ruoss, M. Balunovic, M. Fischer, and M. Vechev, "Learning certified individually fair representations," *Advances in Neural Information Processing Systems 33 pre-proceedings (NeurIPS 2020)*, 2020.

[66] M. Yurochkin, A. Bower, and Y. Sun, "Training individually fair ml models with sensitive subspace robustness," in *International Conference on Learning Representations*, 2019.

[67] S. Biswas, M. Wardat, and H. Rajan, "The art and practice of data science pipelines: A comprehensive study of data science pipelines in theory, in-the-small, and in-the-large," in *Proceedings of the 44th International Conference on Software Engineering*, ser. ICSE '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 2091–2103. [Online]. Available: https://doi.org/10.1145/3510003.3510057

[68] S. Chaudhuri, S. Gulwani, and R. Lublinerman, "Continuity and robustness of programs," *Communications of the ACM*, vol. 55, no. 8, pp. 107–115, 2012.

[69] F. Zhang, S. P. Chowdhury, and M. Christakis, "Deepsearch: A simple and effective blackbox attack for deep neural networks," in *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2020, pp. 800–812.

[70] A. Athalye, N. Carlini, and D. Wagner, "Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples," in *International conference on machine learning*. PMLR, 2018, pp. 274–283.

[71] L. Engstrom, A. Ilyas, and A. Athalye, "Evaluating and understanding the robustness of adversarial logit pairing," *arXiv preprint arXiv:1807.10272*, 2018.

[72] S. Ma, Y. Liu, W.-C. Lee, X. Zhang, and A. Grama, "Mode: automated neural network model debugging via state differential analysis and input selection," in *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2018, pp. 175–186.

[73] Y. Tian, K. Pei, S. Jana, and B. Ray, "Deeptest: Automated testing of deep-neural-network-driven autonomous cars," in *Proceedings of the 40th international conference on software engineering*, 2018, pp. 303–314.

[74] M. Huchard, C. Kästner, and G. Fraser, "Proceedings of the 33rd acm/ieee international conference on automated software engineering (ASE 2018)," in *ASE: Automated Software Engineering*. ACM Press, 2018.