Perfectly Parallel Fairness Certification of Neural Networks

Caterina Urban
INRIA
DIENS, École Normale Supérieure, CNRS, PSL University
Paris, France
caterina.urban@inria.fr

Maria Christakis
MPI-SWS
Germany
maria@mpi-sws.org

Valentin Wüstholz
ConsenSys Diligence
Germany
valentin.wustholz@consensys.net

Fuyuan Zhang
MPI-SWS
Germany
fuyuan@mpi-sws.org

Abstract
Recently, there is growing concern that machine-learning models, which currently assist or even automate decision making, reproduce, and in the worst case reinforce, bias of the training data. The development of tools and techniques for certifying fairness of these models or describing their biased behavior is, therefore, critical. In this paper, we propose a perfectly parallel static analysis for certifying causal fairness of feed-forward neural networks used for classification tasks. When certification succeeds, our approach provides definite guarantees, otherwise, it describes and quantifies the biased behavior. We design the analysis to be sound, in practice also exact, and configurable in terms of scalability and precision, thereby enabling pay-as-you-go certification. We implement our approach in an open-source tool and demonstrate its effectiveness on models trained with popular datasets.

1 Introduction
Due to the tremendous advances in machine learning and the vast amounts of available data, software systems, and neural networks in particular, are of ever-increasing importance in our everyday decisions, whether by assisting them or by autonomously making them. We are already witnessing the wide adoption and societal impact of such software in criminal justice, health care, and social welfare, to name a few examples. It is, therefore, not far-fetched to imagine a future where most of the decision making is automated.

However, several studies have recently raised concerns about the fairness of such systems. For instance, consider a commercial recidivism-risk assessment algorithm that was found racially biased [39]. Similarly, a commercial algorithm that is widely used in the U.S. health care system falsely determined that Black patients were healthier than other equally sick patients by using health costs to represent health needs [52]. There is also empirical evidence of gender bias in image searches, for instance, there are fewer results depicting women when searching for certain occupations, such as CEO [36]. Commercial facial recognition algorithms, which are increasingly used in law enforcement, are less effective for women and darker skin types [8].

In other words, machine-learning software may reproduce, or even reinforce, bias that is directly or indirectly present in the training data. This awareness will most definitely lead to regulations and strict audits in the future. It is, therefore, critical to develop tools and techniques for certifying fairness of neural networks and understanding the circumstances of their potentially biased behavior.

Causal fairness. To meet these needs, we have designed a static analysis framework for certifying causal fairness [38] of feed-forward neural networks. Specifically, given input features that are sensitive to bias, like race or gender, a neural network is causally fair if the output classification is not affected by different values of the sensitive features. Of course, the most obvious approach to avoid such bias is to remove any sensitive feature from the training data. However, this does not work for three main reasons. First, neural networks learn from latent variables (e.g., [41, 63]). For instance, a credit-screening algorithm might not use gender as an explicit input but still be biased with respect to it, say, because most individuals whose first name ends in ‘a’ are denied credit in the training data. Second, the training data is only a relatively small sample of the entire input space, on portions of which the neural network might end up being inaccurate. For example, if Asians are underrepresented in the training data, facial recognition is less likely to be accurate for these people. Third, the information provided by a sensitive feature might be necessary, for instance, to introduce intended bias in a certain input region. Assume a credit-screening algorithm that should not discriminate with respect to age unless it is above a particular threshold. Above this age threshold, the higher the requested credit amount, the lower the chances of receiving it. In such cases, removing the sensitive feature is not even possible.

Our approach. Our approach certifies causal fairness of neural networks used for classification by employing a combination of a forward and a backward static analysis. On
a high level, the forward pass aims to reduce the overall analysis effort. At its core, it divides the input space of the network into independent partitions. The backward analysis then attempts to certify fairness of the classification within each partition (in a perfectly parallel fashion) with respect to a chosen (set of) sensitive feature(s).

In the end, our approach reports for which regions of the input space the neural network is proved fair and for which there is bias. Note that we do not necessarily need to analyze the entire input space; our technique is also able to answer specific bias queries about a fraction of the input space. For instance, are Hispanics over 45 years old discriminated against with respect to gender?

The scalability-vs-precision tradeoff of our approach is configurable. Partitions that do not satisfy the given configuration are excluded from the analysis and may be resumed later, with a more flexible configuration. This enables usage scenarios in which our approach adapts to the available resources, e.g., time or CPUs, and is run incrementally. In other words, we designed a pay-as-you-go certification approach that the more resources it is given, the larger the region of the input space it analyzes.

Related work. In the literature, most work on verifying fairness of machine-learning models has focused on providing probabilistic guarantees (e.g., [2, 7]). In contrast, our approach gives definite guarantees for those input partitions that satisfy the analysis configuration. Moreover, our approach is exact for these partitions. In other words, the tradeoff in comparison to related work is that we might exclude partitions for which our analysis is not exact. In this paper, we investigate how far we can push such an exact analysis in the context of fairness certification of neural networks.

Contributions. We make the following contributions:

1. We propose a perfectly parallel static analysis approach for certifying causal fairness of feed-forward neural networks. If certification fails, our approach can describe and quantify the biased input space region(s).
2. We show that our approach is sound and, in practice, exact for the analyzed regions of the input space.
3. We discuss the configurable scalability-vs-precision tradeoff of our approach that enables pay-as-you-go certification.
4. We implement our approach in an open-source tool calledumba and evaluate it on neural networks trained with popular datasets. We show the effectiveness of our approach in detecting injected bias and answering bias queries. We also experiment with the precision and scalability of the analysis and discuss the tradeoffs.

2 Overview

In this section, we give an overview of our approach using a small constructed example, which is shown in Figure 1.

---

**Example.** The figure depicts a feed-forward neural network for credit approval. There are two inputs $x_{0,1}$ and $x_{0,2}$ (shown in purple). Input $x_{0,1}$ denotes the requested credit amount and $x_{0,2}$ denotes age. Both inputs have continuous values in the range $[0, 1]$. Output $x_{3,2}$ (shown in green) denotes that the credit request is approved, whereas $x_{3,1}$ (in red) denotes that it is denied. The neural network also consists of two hidden layers with two nodes each (in gray).

Now, let us assume that this neural network is trained to deny requests for large credit amounts from older people. Otherwise, the network does not discriminate with respect to age for small credit amounts. There is also no bias for younger people with respect to the requested credit. When choosing age as the sensitive input, our approach can certify fairness with respect to different age groups for small credit amounts. Our approach is also able to find (as well as quantify) bias with respect to the age for large credit amounts. Note that this bias may be intended or accidental — our analysis does not aim to address this question. Below, we present on a high level how our approach achieves these results.

**Naïve approach.** In theory, the simplest way to certify fairness with respect to a given sensitive input is to first analyze the neural network backwards starting from each output node, in our case $x_{3,1}$ and $x_{3,2}$. This allows us to determine the regions of the input space (i.e., age and requested credit amount) for which credit is approved and denied. For example, assume that we find that requests are denied for credit amounts larger than 10 000 (i.e., 10 000 < $x_{0,1}$) and age greater than 60 (i.e., 60 < $x_{0,2}$), while they are approved for $x_{0,1}$ ≤ 10 000 and 60 < $x_{0,2}$ or for $x_{0,2}$ ≤ 60.

The second step is to forget the value of the sensitive input (i.e., age) or, in other words, to project these regions over the credit amount. In our example, after projection we have that credit requests are denied for 10 000 < $x_{0,1}$ and approved for any value of $x_{0,1}$. A non-empty intersection between the projected input regions indicates bias with respect to the sensitive input. In our example, the intersection is non-empty for 10 000 < $x_{0,1}$: there exist people that differ in age but request the same credit amount (smaller than 10 000), some of whom receive the credit while others do not.

This approach, however, is not practical. Specifically, for a neural network using the popular ReLU activation functions
(see Section 3 for more details, other activation functions are discussed in Section 9), each hidden node effectively represents a disjunction between two activation statuses (active and inactive). In our example, there are $2^4$ possible activation patterns for the 4 hidden nodes. To retain maximum precision, a backward analysis would have to explore all of them, which does not scale in practice.

**Our approach.** Our analysis is based on the observation that there might exist many activation patterns that do not correspond to a region of the input space [31]. Such patterns can, therefore, be ignored during the analysis. We push this idea further by defining abstract activation patterns, which fix the activation status of only certain nodes and thus represent sets of (concrete) activation patterns. Typically, a relatively small number of abstract activation patterns is sufficient for covering the entire input space, without necessarily representing and exploring all possible concrete patterns.

Identifying those patterns that definitely correspond to a region of the input space is only possible with a forward analysis. Hence, we combine a forward pre-analysis with a backward analysis. The pre-analysis partitions the input space into independent partitions corresponding to abstract activation patterns. Then, the backward analysis tries to prove fairness of the neural network for each such partition.

More specifically, we set an upper bound $U$ on the number of tolerated disjunctions (i.e., on the number of nodes with an unknown activation status) per abstract activation pattern. Our forward pre-analysis uses a cheap abstract domain (e.g., the boxes domain [17]) to iteratively partition the input space along the non-sensitive input dimensions to obtain fair input partitions (i.e., boxes). Each partition satisfies one of the following conditions: (a) its classification is already fair because only one network output is reachable for all inputs in the region, (b) it has an abstract activation pattern with at most $U$ unknown nodes, or (c) it needs to be partitioned further. We call partitions that satisfy condition (b) feasible.

In our example, let $U = 2$. At first, the analysis considers the entire input space, that is, $X_{0,1} : [0,1]$ (credit amount) and $X_{0,2} : [0,1]$ (age). The abstract activation pattern corresponding to this initial partition $I$ is $\epsilon$ (i.e., no hidden nodes have fixed activation status) and, thus, the number of disjunctions would be four, which is greater than $U$. Therefore, I needs to be divided into $I_1$ ($X_{0,1} : [0,0.5], X_{0,2} : [0,1]$) and $I_2$ ($X_{0,1} : [0.5,1], X_{0,2} : [0,1]$). Note that the input space is not split with respect to $X_{0,2}$, which is the sensitive input. Now $I_1$ is feasible since its abstract activation pattern is $X_{1,2}X_{2,1}X_{2,2}$ (i.e., all three nodes are always active), while $I_2$ must be divided further since its abstract activation pattern is $\epsilon$.

To control the number of partitions, we impose a lower bound $L$ on the size of each of their dimensions. Partitions that require a dimension of a smaller size are excluded. In other words, they are not considered until more analysis budget becomes available, that is, a larger $U$ or a smaller $L$.

In our example, let $L = 0.25$. The forward pre-analysis further divides $I_2$ into $I_{2,1}$ ($X_{0,1} : [0.5,0.75], X_{0,2} : [0,1]$) and $I_{2,2}$ ($X_{0,1} : [0.75,1], X_{0,2} : [0,1]$). Now $I_{2,1}$ is feasible, with abstract pattern $X_{1,2}X_{2,1}$, while $I_{2,2}$ still is not. However, $I_{2,2}$ may not be split further because the size of the only non-sensitive dimension $X_{0,1}$ has already reached the lower bound $L$. As a result, $I_{2,2}$ is excluded, and only the remaining 75% of the input space is considered for the analysis.

Feasible input partitions (within bounds $L$ and $U$) are then grouped by abstract activation patterns. In our example, the pattern corresponding to $I_1$, namely $X_{1,2}X_{2,1}$, is subsumed by the (more abstract) pattern of $I_{2,1}$, namely $X_{1,2}X_{2,1}$. Consequently, we group $I_1$ and $I_{2,1}$ under the pattern $X_{1,2}X_{2,1}$.

The backward analysis is then run in parallel for each representative abstract activation pattern, in our example $X_{1,2}X_{2,1}$. This analysis determines the region of the input space (within a given partition group) for which each output of the neural network is returned, e.g., credit is approved for $c_1 \leq X_{0,1} \leq c_2$ and $d_1 \leq X_{0,2} \leq d_2$. To achieve this, the analysis uses an expensive abstract domain, for instance, disjunctive or powerset polyhedra [19, 20], and leverages abstract activation patterns to avoid disjunctions. For instance, pattern $X_{1,2}X_{2,1}$ only requires reasoning about two disjunctions from the remaining hidden nodes $X_{1,2}$ and $X_{2,2}$.

Finally, fairness is checked for each partition in the same way that it is done by the naïve approach for the entire input space. In our example, we prove that the classification within $I_1$ is fair and determine that within $I_{2,1}$ the classification is biased. Concretely, our approach determines that bias occurs for $0.54 \leq X_{0,1} \leq 0.75$, which corresponds to 21% of the entire input space. In other words, the network returns different outputs for people that request the same credit in the above range but differ in age. Recall that partition $I_{2,2}$, where $0.75 \leq X_{0,1} \leq 1$, was excluded from analysis, and therefore, we cannot draw any conclusions about whether there is any bias for people requesting credit in this range.

## 3 Feed-Forward Deep Neural Networks

Formally, a feed-forward deep neural network consists of an input layer ($I_0$), an output layer ($I_n$), and a number of hidden layers ($I_1, \ldots, I_{n-1}$) in between. Each layer $I_i$ contains $|I_i|$ nodes and, with the exception of the input layer, is associated to a $|I_i| \times |I_{i-1}|$-matrix $W_i$ of weight coefficients and a vector $b_i$ of $|I_i|$ bias coefficients. In the following, we use $X$ to denote the set of all nodes, $X_i$ to denote the set of nodes of the $i$th layer, and $x_{i,j}$ to denote the $j$th node of the $i$th layer of a neural network. We focus here on neural networks used for classification tasks. Thus, $|I_n|$ is the number of target classes (e.g., two classes in Figure 1).

The value of the input nodes is given by the input data: continuous data is represented by one input node (e.g., $X_{0,1}$ or $X_{0,2}$ in Figure 1), while categorical data is represented by multiple input nodes via one-hot encoding. In the following,
we use K to denote the subset of input nodes considered sensitive to bias (e.g., \( x_{0,2} \) in Figure 1) and \( \mathcal{K} \) to denote the input nodes not deemed sensitive to bias.

The value of each hidden and output node \( x_{i,j} \) is computed by an activation function \( f \) applied to a linear combination of the values of all nodes in the preceding layer [27], i.e., \( x_{i,j} = f \left( \sum_{k} w_{i,j}^{k} \cdot x_{i-1,k} + b_{i,j} \right) \), where \( w_{i,j}^{k} \) and \( b_{i,j} \) are weight and bias coefficients in \( W_{i} \) and \( B_{i} \), respectively. In a fully-connected neural network, all \( w_{i,j}^{k} \) are non-zero. Weights and biases are adjusted during the training phase of the neural network. In what follows, we focus on already trained neural networks, which we call neural-network models.

Nowadays, the most commonly used activation for hidden nodes is the Rectified Linear Unit (ReLU) [50]: ReLU(\( x \)) = max(\( x \), 0). In this case, the activation used for output nodes is the identity function. The output values are then normalized into a probability distribution on the target classes [27]. We discuss other activation functions in Section 9.

4 Trace Semantics

The semantics of a neural-network model is a mathematical characterization of its behavior when executed for all possible input data. We model the operational semantics of a feed-forward neural-network model \( M \) as a transition system \((\Sigma, \tau)\), where \( \Sigma \) is a (potentially infinite) set of states and the acyclic transition relation \( \tau \subseteq \Sigma \times \Sigma \) describes the possible transitions between states [16, 18].

More specifically, a state \( s \in \Sigma \) maps neural-network nodes to their values. Here, for simplicity, we assume that nodes have real values, i.e., \( x : \Sigma \rightarrow \mathbb{R} \). (We discuss floating-point values in Section 9.) In the following, we often only care about the values of a subset of the neural-network nodes in certain states. Thus, let \( \Sigma|_{Y} \) be the restriction of \( \Sigma \) to a domain of interest \( Y \). Sets \( \Sigma_{X_{0}} \) and \( \Sigma_{X_{n}} \) denote restrictions of \( \Sigma \) to the network nodes in the input and output layer, respectively. With a slight abuse of notation, let \( x_{i,j} \) denote \( \Sigma|_{\{s_{i,j}\}} \), i.e., the restriction of \( \Sigma \) to the singleton set containing \( x_{i,j} \). Transitions happen between states with different values for consecutive nodes in the same layer, i.e., \( \tau \subseteq X_{i,j} \times X_{i,j+1} \), or between states with different values for the last and first node of consecutive layers of the network, i.e., \( \tau \subseteq X_{i,|L_{i}|} \times X_{i+1,0} \). The set \( \Omega \) is the set of all possible value choices for all sensitive input nodes in \( K \), e.g., for \( (x_{0,1}, x_{0,2}) \) one-hot encoding, say, gender information, \( \Omega = \{(1,0),(0,1)\} \); for \( x_{0,k} \) encoding continuous data, say, in the range \([0,1]\), a possibility is \( \Omega = \{0,0.25],[0.25,0.75],[0.75,1]\} \). In the following, given a trace \( \sigma \in \mathcal{P}(\Sigma^{+}) \), we write \( \sigma_{0} \) and \( \sigma_{o} \) to denote its initial and final state, respectively. We also write \( \sigma_{0} \equiv \sigma_{o}^{'} \) to indicate that the states \( \sigma_{0} \) and \( \sigma_{o}^{'} \) agree on all values of all non-sensitive input nodes, and \( \sigma_{o} \equiv \sigma_{o}^{'} \) to indicate that \( \sigma \) and \( \sigma^{'} \) have the same output \( O \in \mathcal{O} \). We can now formally define when the sensitive input nodes in
Perfectly Parallel Certification of Neural Network Fairness

K are unused with respect to a set of traces $T \in \mathcal{P}(\Sigma^*)$ \footnote{Note that here and in the following, for convenience, we abuse notation and reuse the order symbol $\subseteq$, defined over sets of sets of traces, instead of its abstraction, defined over sets of sets of pairs of states.}.

$$\text{UNUSED}_K(T) \overset{\text{def}}{=} \forall \sigma \in T, \forall \nu \in \nu : \sigma_0(K) \neq \nu \Rightarrow \exists \sigma' \in T : \sigma_0 = \sigma'_0 \land \sigma'_0(\nu) = \nu \land \sigma_\alpha \equiv \sigma'_\alpha,$$

where $\sigma_0(K) \overset{\text{def}}{=} \{\sigma_0(x) | x \in K\}$ is the image of $K$ under $\sigma_0$. Intuitively, the sensitive input nodes in $K$ are unused if any possible outcome in $T$ (i.e., any outcome $\sigma_\alpha$ of any trace $\sigma$ in $T$) is possible from all possible value choices for $K$ (i.e., there exists a trace $\sigma'$ in $T$ for each value choice for $K$ with the same outcome as $\sigma$). In other words, each outcome is independent of the value choice for $K$.

**Example 5.1.** Let us consider again our example in Figure 1. We write $(c, a) \leadsto o$ for a trace starting in a state with $x_{0,1} = c$ and $x_{0,2} = a$ and ending in a state where $o$ is the node with the highest value (i.e., the output class). The sensitive input $x_{0,2}$ (age) is unused in $T = \{ (0,5, a) \leadsto x_{3,2} | 0 \leq a \leq 1 \}$. It is instead used in $T' = \{ (0,75, a) \leadsto x_{3,2} | 0 \leq a < 0,51 \} \cup \{ (0,75, a) \leadsto x_{3,1} | 0,51 \leq a \leq 1 \}$.

The causal-fairness property $\mathcal{F}_K$ can now be defined as

$$\mathcal{F}_K[Y] \overset{\text{def}}{=} \{ [M]^Y | \text{UNUSED}_K([M]^Y) \},$$

where $Y \in \mathcal{P}(\Sigma)$ is a set of initial states of interest and the restriction $T^Y \overset{\text{def}}{=} \{ \sigma \in T | \sigma_0 \in Y \}$ only contains traces of $T \in \mathcal{P}(\Sigma^*)$ that start with a state in $Y$. Similarly, in the rest of the paper, we write $S^Y \overset{\text{def}}{=} \{ T^Y | T \in S \}$ for the set of sets of traces restricted to initial states in $Y$. Thus, from Equation 3, we have the following:

**Theorem 5.2.** $M \models \mathcal{F}_K[Y] \equiv [M]^Y \subseteq \mathcal{F}_K[Y]$

**Proof.** The proof follows trivially from Equation 3 and the definition of $\mathcal{F}_K[Y]$ (cf. Equation 5) and $[M]^Y$.

### 6 Dependency Semantics

We now use abstract interpretation to systematically derive, by successive abstractions of the collecting semantics $\Lambda$, a sound and complete semantics $\Lambda_{\leftarrow}$ that contains only and exactly the information needed to reason about $\mathcal{F}_K[Y]$.

#### 6.1 Outcome Semantics

Let $T_Z \overset{\text{def}}{=} \{ \sigma \in T | \sigma_\alpha \in Z \}$ be the set of traces of $T \in \mathcal{P}(\Sigma^*)$ that end with a state in $Z \in \mathcal{P}(\Sigma)$. As before, we write $S_Z \overset{\text{def}}{=} \{ T \in S \}$ for the set of sets of traces restricted to final states in $Z$. From the definition of $\mathcal{F}_K[Y]$ (and in particular, from the definition of $\text{UNUSED}_K$, cf. Equation 4), we have the following result:

**Lemma 6.1.** $[M]^Y \subseteq \mathcal{F}_K[Y] \Rightarrow \forall \sigma \in \Sigma : [M]^Y_\sigma \subseteq \mathcal{F}_K[Y]$

**Proof.** Let $[M]^Y \subseteq \mathcal{F}_K[Y]$. From the definition of $[M]^Y$ (cf. Equation 2), we have that $[M]^Y \subseteq \mathcal{F}_K[Y]$. Thus, from the definition of $\mathcal{F}_K[Y]$ (cf. Equation 5), we have $\text{UNUSED}_K([M]^Y)$. Now, from the definition of $\text{UNUSED}_K$ (cf. Equation 4), we equivalently have $\forall \sigma \in \Sigma : \text{UNUSED}_K([M]^Y_\sigma)$. Thus, we can conclude that $\forall \sigma \in \Sigma : (M)^Y_\sigma \subseteq \mathcal{F}_K[Y]$.

In particular, this means that in order to determine whether a neural-network model $M$ satisfies causal fairness, we can independently verify, for each of its possible target classes $\sigma \in \Sigma$, that the values of its sensitive input nodes are unused.

We use this insight to abstract the collecting semantics $\Lambda$ by partitioning. More specifically, let $\bullet \overset{\text{def}}{=} \{ \sigma_\alpha \in \Sigma | \sigma_\alpha \in \sigma \}$ be a trace partition with respect to outcome. We have the following Galois connection

$$\langle \mathcal{P}(\mathcal{P}(\Sigma^*)), \subseteq \rangle \overset{\text{def}}{=} \langle \mathcal{P}(\mathcal{P}(\Sigma^*)), \subseteq \rangle,$$

where $a_\sigma(S) \overset{\text{def}}{=} \{ T_0 \in T \in S \}$. The order $\subseteq$ is the pointwise ordering between sets of traces with the same outcome, i.e., $A \subseteq B \overset{\text{def}}{=} \forall \sigma \subseteq O \in A \subseteq B$, where $S_Z$ denotes the only non-empty set of traces in $S_Z$. We can now define the outcome semantics $\Lambda_{\leftarrow} \in \mathcal{P}(\mathcal{P}(\Sigma^*))$ by abstraction of $\Lambda$:

$$\Lambda_{\leftarrow} \overset{\text{def}}{=} \Lambda_{\leftarrow} = \{ T_0 \in T \in \mathcal{F}_K \}$$

In the rest of the paper, we write $[M]_{\leftarrow}$ to denote the outcome semantics of a particular neural-network model $M$.

#### 6.2 Dependency Semantics

We observe that, to reason about causal fairness, we do not need to consider all intermediate computations between the initial and final states of a trace. Thus, we can further abstract the outcome semantics into a set of dependencies between initial states and outcomes of traces.

To this end, we define the following Galois connection

$$\langle \mathcal{P}(\mathcal{P}(\Sigma^*)), \subseteq \rangle \overset{\text{def}}{=} \langle \mathcal{P}(\mathcal{P}(\Sigma^*)), \subseteq \rangle,$$

where $a_{\sigma}(S) \overset{\text{def}}{=} \{ \{ \sigma_0(\sigma_\alpha) \in T_0 \} | T \in S \}$ [64] abstracts away all intermediate states of any trace. We finally derive the dependency semantics $\Lambda_{\leftarrow} \in \mathcal{P}(\Sigma \times \Sigma)$:

$$\Lambda_{\leftarrow} \overset{\text{def}}{=} \{ \sigma_0(\sigma_\alpha) \in \sigma_0(\sigma_\alpha) \in \sigma_0 \in \sigma_0 \in \sigma_0 \}, \sigma_0 \in \sigma_0 \}, \sigma_0 \in \sigma_0 \}, \sigma_0 \in \sigma_0 \}, \sigma_0 \in \sigma_0 \}, \sigma_0 \in \sigma_0 \}.$$
While it is too naïve to be practical, it is still useful for building the dependency semantics. Let \( M \models \mathcal{F}_K[Y] \). Thus, from the Galois connections in Equation 6 and 8, we have \( a \cdashdash \cdashdash (\alpha \circ ([M]^Y_{\neg})) \subseteq a \cdashdash \cdashdash (\alpha \circ (\mathcal{F}_K[Y])) \). From the definition of \( ([M]^Y_{\neg}) \) (cf. Equation 9), we can then conclude that \( ([M]^Y_{\neg}) \subseteq a \cdashdash \cdashdash (\mathcal{F}_K[Y]) \).

**Corollary 6.3.** M \( \models \mathcal{F}_K[Y] \iff ([M]^Y_{\neg}) \subseteq a \cdashdash \cdashdash (\mathcal{F}_K[Y]) \)

**Proof.** The proofs follows trivially from the definition of \( \subseteq \) (cf. Equation 6 and 8) and Lemma 6.1.

Furthermore, we observe that partitioning with respect to outcome induces a partition of the space of values of the input nodes used for classification. For instance, partitioning \( T' \) in Example 5.1 induces a partition on the values of (the indeed unused node) \( x_{a_2} \). Thus, we can equivalently verify whether \( ([M]_{\neg}) \subseteq a \cdashdash \cdashdash (\mathcal{F}_K[Y]) \) (cf. Equation 5). In particular, from the definition of \( a \cdashdash \cdashdash \) and \( \mathcal{F}_K[Y] \) (cf. Equation 5), we have that \( \text{unused}_K([M]^Y_{\neg}) \) for each \( O \in \mathcal{O} \). From the definition of \( \text{unused}_K \) (cf. Equation 4), for each pair of non-empty \( [M]_{\neg, O_1} \) and \( [M]_{\neg, O_2} \) for different \( O_1, O_2 \in \mathcal{O} \) (the case in which one or both are empty is trivial), it must necessarily be the case of the non-sensitive input nodes in \( K \) that causes the different outcome \( O_1 \), or \( O_2 \). We can thus conclude that \( \forall A, B \in [M]^Y_{\neg} : (A_{\neg} \neq B_{\neg} \Rightarrow A_{\neg, K} \cap B_{\neg, K} = \emptyset) \).

**Lemma 6.4.** \( M \models \mathcal{F}_K[Y] \iff \forall A, B \in [M]_{\neg, K} : (A_{\neg} \neq B_{\neg} \Rightarrow A_{\neg, K} \cap B_{\neg, K} = \emptyset) \)

**Proof.** Let \( M \models \mathcal{F}_K[Y] \). From Corollary 6.3, we have that \( ([M]^Y_{\neg}) \subseteq a \cdashdash \cdashdash (\mathcal{F}_K[Y]) \). Thus, from the definition of \( [M]_{\neg} \) (cf. Equation 9), we have \( \forall O \in \mathcal{O} : a \cdashdash \cdashdash ([M]_{\neg, O_1}) \subseteq a \cdashdash \cdashdash (\mathcal{F}_K[Y]) \). In particular, from the definition of \( a \cdashdash \cdashdash \) and \( \mathcal{F}_K[Y] \) (cf. Equation 5), we have that \( \text{unused}_K([M]^Y_{\neg}) \) for each \( O \in \mathcal{O} \). From the definition of \( \text{unused}_K \) (cf. Equation 4), for each pair of non-empty \( [M]_{\neg, O_1} \) and \( [M]_{\neg, O_2} \) for different \( O_1, O_2 \in \mathcal{O} \) (the case in which one or both are empty is trivial), it must necessarily be the case of the non-sensitive input nodes in \( K \) that causes the different outcome \( O_1 \), or \( O_2 \). We can thus conclude that \( \forall A, B \in [M]_{\neg, K} : (A_{\neg} \neq B_{\neg} \Rightarrow A_{\neg, K} \cap B_{\neg, K} = \emptyset) \).

**7 Naïve Causal-Fairness Analysis**

In this section, we present a first static analysis for causal fairness that computes a sound over-approximation \( \Lambda^b_{\neg} \) of the dependency semantics \( \Lambda_{\neg} \), i.e., \( \Lambda_{\neg} \subseteq \Lambda^b_{\neg} \). This analysis corresponds to the naïve approach we discussed in Section 2. While it is too naïve to be practical, it is still useful for building upon later in the paper.

For simplicity, we consider ReLU activation functions. (We discuss extensions to other activation functions in Section 9.) The naïve static analysis is described in Algorithm 1. It takes as input (cf. Line 14) a neural-network model \( M \), a set of sensitive input nodes \( K \) of \( M \), a (representation of a) set of initial states of interest \( Y \), and an abstract domain \( A \) to be used for the analysis. The analysis proceeds backward for each outcome (i.e., each target class \( x_{k_j} \)) of \( M \) (cf. Line 17) in order to determine an over-approximation of the initial states that satisfy \( Y \) and lead to \( x_{k_j} \) (cf. Line 18).

**Algorithm 1 : A Naïve Backward Analysis**

```plaintext
1: function backward(M, A, x)
2:     a ← outcome_A[x](new_A)
3:     for i ← N − 1 down to 0 do
4:         for j ← |I_i| down to 0 do
5:             a ← assign_A[x,j](relu_A[x,j]a)
6:     return a
7: function check(O)
8:     B ← 0         ⇒ B: biased
9:     for all 0_1, a_1 ∈ O do
10:         for all 0_2 ≠ 0_1, a_2 ∈ O do
11:             if a_1 ∩ a_2 ≠ a_1 then
12:                 B ← B ∪ {a_1 ∩ a_2}
13:     return B
14: function analyze(M, K, Y, A)
15:     O ← 0
16:     for j ← 0 up to |I_o| do
17:         a ← backward(M, A, x_{k_j})
18:         O ← O ∪ {x_{k,j} ← (assume_A[Y]a|K)}
19:     B ← check(O)
20: return B = 0, B         ⇒ fair: B = 0, maybe biased: B ≠ 0
```

More specifically, the transfer function \( \text{outcome}_A[x] \) (cf. Line 2) modifies a given abstract-domain element to assume the given outcome \( x \), that is, to assume that \( \max X_{k} = x \). The transfer functions \( \text{relu}_A[x,j] \) and \( \text{assign}_A[x,j] \) (cf. Line 5) respectively consider a ReLU operation and replace \( x_{k_j} \) with the corresponding linear combination of nodes in the preceding layer (see Section 3).

Finally, the analysis checks whether the computed over-approximations satisfy causal fairness with respect to \( K \) (cf. Line 19). In particular, it checks whether they induce a partition of \( Y_K \) as observed for Lemma 6.4 (cf. Lines 7-13). If so, we have proved that \( M \) satisfies causal fairness. If not, the analysis returns a set \( B \) of abstract-domain elements over-approximating the input regions in which bias might occur.

**Theorem 7.1.** If \( \text{analyze}(M, K, Y, A) \) of Algorithm 1 returns \( \text{true}, 0 \) then \( M \) satisfies \( \mathcal{F}_K[Y] \).

**Proof (Sketch).** \( \text{analyze}(M, K, Y, A) \) in Algorithm 1 computes an over-approximation \( a \) of the regions of the input space that yield each target class \( x_{k_j} \) (cf. Line 17). Thus, it actually computes an over-approximation \( [M]_{\neg} \) of the dependency semantics \( [M]_{\neg} \), i.e., \( [M]_{\neg} \subseteq [M]_{\neg} \). Thus, if \( [M]_{\neg} \) satisfies \( \mathcal{F}_K[Y] \), i.e., \( \forall A, B \in [M]_{\neg} : (A_{\neg} \neq B_{\neg} \Rightarrow A_{\neg, K} \cap B_{\neg, K} = \emptyset) \) (according to Lemma 6.4, cf. Line 19), then by transitivity we can conclude that also \( [M]_{\neg} \) necessarily satisfies \( \mathcal{F}_K[Y] \).
Perfectly Parallel Certification of Neural Network Fairness

existing numerical abstract domains that are less expressive than polyhedra [20] would make for a rather fast but too imprecise analysis. This is because they are not able to precisely handle constraints like $\max X = x$, which are introduced by $\text{outcome}_A[x]$ to partition with respect to outcome.

Furthermore, even polyhedra would not be precise enough in general. Indeed, each $\text{ReLU}^\|_{\{x_i\}}$ would over-approximate what effectively is a conditional branch. Let $|M| = |I_1| + \cdots + |I_{m-1}|$ denote the number of hidden nodes (i.e., the number of ReLUs) in a model M. On the other side of the spectrum, one could use a disjunctive completion [19] of polyhedra, thus keeping a separate polyhedron for each branch of a ReLU. This would yield a precise (in fact, exact) but extremely slow analysis: even with parallelization (cf. Lines 16), each of the $|I_{m-1}|$ processes would have to effectively explore $2^{|M|}$ paths!

In the rest of the paper, we improve on this na"ive analysis and show how far we can go all the while remaining exact by using disjunctive polyhedra.

8 Parallel Semantics

We first have to take a step back and return to reasoning at the concrete-semantics level. At the end of Section 6, we observed that the dependency semantics of a neural-network model M satisfying $\mathcal{F}_K[Y]$ effectively induces a partition of $Y_{\|}$.

More formally, given a set $Y$ of initial states of interest, we say that an input partition $\xi$ of $Y$ is fair if all value choices $\forall$ for the sensitive input nodes K of M are possible in all elements of the partitions: $\forall I \in \mathcal{I}, \forall V \in \mathcal{V} : \exists I \in I \colon s(K) = V$. For instance, $\mathcal{I} = \{ T_0, T_0' \}$, with $T$ and $T'$ in Example 5.1 is a fair input partition of $Y = \{ s \mid s(x_0) = 0.5 \vee s(x_1) = 0.75 \}$.

Given a fair input partition $\mathcal{I}$ of $Y$, the following result shows that we can verify whether a model M satisfies $\mathcal{F}_K[Y]$ for each element $I$ of $\mathcal{I}$, independently.

Lemma 8.1. $M \models \mathcal{F}_K[Y] \iff \forall I \in \mathcal{I} : \forall A, B \in \langle M \rangle_{\|} : (A_{\|} \neq B_{\|} \implies A_{\|K} \cap B_{\|K} = \emptyset)$

Proof. The proof follows trivially from Lemma 6.4 and the fact that $I$ is a fair partition. \hfill $\square$

We use this new insight to further abstract the dependency semantics $\Lambda_{\|}$. We have the following Galois connection

$$\langle \mathcal{P}(\mathcal{P}(\Sigma \times \Sigma)), \mathcal{G} \rangle \sqsubseteq \langle \mathcal{P}(\mathcal{P}(\Sigma \times \Sigma)), \mathcal{G}_I \rangle,$$

where $\mathcal{G}_I$ is the pointwise ordering between sets of pairs of restricted to first elements in the same $I \in \mathcal{I}$, i.e., $A \sqsubseteq B \iff \forall I \in \mathcal{I} A_I \subseteq B_I$, where $S^T$ denotes the only non-empty set of pairs in $S^T$. We can now derive the parallel semantics $\Pi_{\|}$:

$$\Pi_{\|} = \alpha(L_{\|}) = \{ \langle \sigma_0, \sigma_\omega \rangle \mid \sigma \in Y_{\|} \} \iff I \in \mathcal{I} \land \emptyset \in \emptyset$$

In fact, we derive a hierarchy of semantics, as depicted in Figure 2. We write $\langle M \rangle_{\|}^I$ to denote the parallel semantics of a particular neural-network model M. It remains to show soundness and completeness for $\Pi_{\|}$.

Theorem 8.2. $M \models \mathcal{F}_K[Y] \implies \langle M \rangle_{\|}^I \subseteq \alpha(\Lambda_{\|})(\mathcal{F}_K[Y])$

Proof. Let $M \models \mathcal{F}_K[Y]$. From Theorem 6.2, we have that $\langle M \rangle_{\|} \subseteq \alpha(\mathcal{F}_K[Y])$. Thus, from the Galois connections in Equation 10, we have $\alpha(\langle M \rangle_{\|}) \subseteq \alpha(\mathcal{F}_K[Y]))$. From the definition of $\langle M \rangle_{\|}$ (cf. Equation 11), we can then conclude that $\langle M \rangle_{\|} \subseteq \alpha(\mathcal{F}_K[Y]))$. \hfill $\square$

Corollary 8.3. $M \models \mathcal{F}_K[Y] \implies \langle M \rangle_{\|}^I \subseteq \alpha(\Lambda_{\|})(\mathcal{F}_K[Y])$

Proof. The proofs follows trivially from the definition of $\mathcal{G}_I$ (cf. Equation 6 and 8 and 10) and Lemma 6.1 and 8.1. \hfill $\square$

Finally, from Lemma 8.1, we have that we can equivalently verify whether $\langle M \rangle_{\|} \subseteq \alpha(\mathcal{F}_K[Y])$ by checking if the parallel semantics $\langle M \rangle_{\|}$ induces a partition of each $\Sigma^I$.

Lemma 8.4. $M \models \mathcal{F}_K[Y] \iff \forall I \in \mathcal{I} : \forall A, B \in \langle M \rangle_{\|} : (A_{\|} \neq B_{\|} \implies A_{\|K} \cap B_{\|K} = \emptyset)$

Proof. The proof follows trivially from Lemma 8.1. \hfill $\square$

9 Parallel Causal-Fairness Analysis

In this section, we build on the parallel semantics to design our novel perfectly parallel static analysis for causal fairness, which automatically finds a fair partition I and computes a sound over-approximation $\Pi_{\|}^I$, of $\Pi_{\|}$, i.e., $\Pi_{\|}^I \subseteq \Pi_{\|}$.

ReLU activation functions. We again only consider ReLU activation functions for now and postpone the discussion of other activation functions to the end of the section. The analysis is described in Algorithm 2. It combines a forward pre-analysis (Lines 15–24) with a backward analysis (Lines 28–38). The forward pre-analysis uses an abstract domain $A_1$ and builds partition $\mathcal{I}$, while the backward analysis uses an abstract domain $A_2$ and performs the actual causal-fairness analysis of a neural-network model M with respect to its
Algorithm 2: Our Analysis Based on Activation Patterns

1: function forward(M, A, I)
2:   a, p ← assumeA(∥I∥)(newA), ε
3: for i ← 1 up to n do
4:    for j ← 0 up to |I_i| do
5:       a, p ← relu_p^[A][x_{i,j}](assign^[A][x_{i,j}][a])
6: return a, p

7: function backward(M, A, O, p)
8: a ← outcome^[A][∥O∥](newA)
9: for i ← n − 1 down to 0 do
10:   for j ← |I_i| down to 0 do
11:      a ← assign^[A][x_{i,j}](relu^[A][x_{i,j}][a])
12: return a

13: function analyze(M, K, Y, A1, A2, L, U)
14: F, E, C ← ∅, ∅, ∅ // F: feasible, E: excluded, C: completed
15: I ← {Y}
16: while I ≠ ∅ do ▷ perfectly parallelizable
17:   I ← I_GET() ▷ I is already fair
18:   a, p ← forward(M, A1, I)
19: if uniquely-classified(a) then ▷ I must be partitioned further
20:   C ← C ∪ {I}
21: else if |M| − |p| ≤ U then ▷ I is feasible
22:   F ← F ∪ {∥p∥ → I}
23: else if ∥I∥ ≤ L then ▷ I is excluded
24:   E ← E ∪ {∥p∥ → I}
25:   else ▷ I is set aside and excluded from the analysis
26:      I ← I ∪ partition^[K](I)
27:   B ← ∅ ▷ B: biased
28: for all p, I ∈ F do ▷ perfectly parallelizable
29:   O ← ∅ ▷ O: output
30: for j ← 0 up to |K| do
31:   a ← backward(M, A2, p, x_{K,j})
32:   O ← O ∪ {x_{K,j} ↦ a}
33: for all l ∈ I do
34:   O' ← ∅
35:   for all o, a ∈ O do
36:      O' ← O' ∪ {0 ↦ (assign^[A][∥I∥][a])^[K]}
37: B ← B ∪ check(O')
38: C ← C ∪ {I}

sensitive input nodes K and a (representation of a) set of initial states Y (cf. Line 13).

More specifically, the forward pre-analysis bounds the number of paths that the backward analysis has to explore. Indeed, not all of the 2^{|M|} paths of a model M are necessarily viable starting from its input space.

In the rest of this section, we represent each path by an activation pattern, which determines the activation status of every ReLU operation in M. More precisely, an activation pattern is a sequence of flags. Each flag p_{i,j} represents the activation status of the ReLU operation used to compute the value of hidden node x_{i,j}. If p_{i,j} is x_{i,j}, the ReLU is always active, otherwise the ReLU is always inactive and p_{i,j} is x_{i,j}.

An abstract activation pattern gives the activation status of only a subset of the ReLUs of M, and thus, represents a set of activation patterns. ReLUs whose corresponding flag does not appear in an abstract activation pattern have an unknown (i.e., not fixed) activation status. Typically, only a relatively small number of abstract activation patterns is sufficient for covering the entire input space of a neural-network model. The design of our analysis builds on this key observation.

We set an analysis budget by providing an upper bound U (cf. Line 13) on the number of tolerated ReLUs with an unknown activation status for each element I of I, i.e., on the number of paths that are to be explored by the backward analysis in each I. The forward pre-analysis starts with the trivial partition I = {Y} (cf. Line 15). It proceeds forward for each element I in I (cf. Lines 17-18). The transfer function relu^[A][x_{i,j}] considers a ReLU operation and additionally builds an abstract activation pattern p for I (cf. Line 5) starting from the empty pattern ε (cf. Line 2).

If I leads to a unique outcome (cf. Line 19), then causal fairness is already proved for I, and there is no need for a backward analysis; I is added to the set of completed partitions (cf. Line 20). Instead, if abstract activation pattern p fixes the activation status of enough ReLUs (cf. Line 21), we say that the backward analysis for I is feasible. In this case, the pair of p and I is inserted into a map F from abstract activation patterns to feasible partitions (cf. Line 22). The insertion takes care of merging abstract activation patterns that are subsumed by other (more) abstract patterns. In other words, it groups partitions whose abstract activation patterns fix more ReLUs with partitions whose patterns fix fewer ReLUs, and therefore, represent a superset of (concrete) patterns.

Otherwise, I needs to be partitioned further, with respect to K (cf. Line 25). Partitioning may continue until the size of I is smaller than the given lower bound L (cf. Lines 13 and 23). At this point, I is set aside and excluded from the analysis until more resources (a larger upper bound U or a smaller lower bound L) become available (cf. Line 24).

Note that the forward pre-analysis lends itself to choosing a relatively cheap abstract domain A1 since it does not need to precisely handle polyhedral constraints (like max x_{K} = x, needed to partition with respect to outcome, cf. Section 7).

The analysis then proceeds backwards, independently for each abstract activation pattern p and associated group of partitions I (cf. Lines 28 and 31). The transfer function relu^[A][x_{i,j}] uses p to choose which path(s) to explore at each ReLU operation, i.e., only the active (resp. inactive) path if x_{i,j} (resp. x_{i,j}) appears in p, or both if the activation status of the ReLU corresponding to the hidden node x_{i,j} is unknown. The (as we have seen, necessarily) expensive backward analysis only
We implemented our causal-fairness analysis described in the previous section in a tool called libra. The implementation is written in python and is open-source\(^2\).

**Tool inputs.** libra takes as input a neural-network model \(M\) expressed as a python program (cf. Section 3), a specification of the input layer \(I_0\) of \(M\), an abstract domain for the forward pre-analysis, and budget constraints \(L\) and \(U\).

\(^2\)https://github.com/caterinaurban/Libra

Finally, the analysis checks causal fairness of each element \(i\) associated to \(p\) (cf. Line 37). The analysis returns the set of input-space regions \(C\) that have been completed and a set \(B\) of abstract-domain elements over-approximating the regions in which bias might occur (cf. Line 39). If \(B\) is empty, then the given neural-network model \(M\) satisfies causal fairness with respect to \(K\) and \(Y\) over \(C\).

**Theorem 9.1.** If function \(\text{analyze}(M, K, Y, A_1, A_2, U)\) in Algorithm 2 returns \(C, \text{true}, \emptyset\), then \(M\) satisfies \(\mathcal{F}_K[Y]\) over the input-space fraction \(C\).

**Proof (Sketch).** \(\text{analyze}(M, K, Y, A_1, A_2, U)\) in Algorithm 2 first computes the abstract activation patterns that cover a fraction \(C\) of the input space in which the analysis is feasible (Lines 15-24). Then, it computes an over-approximation \(\mathcal{A}\) of the regions of \(C\) that yield each target class \(X_{s_i}\) (cf. Line 31). Thus, it actually computes an over-approximation \(\mathcal{A}\) of the parallel semantics \(\{M\}_\mathcal{A}\), i.e., \(\{M\}_\mathcal{A} \subseteq \mathcal{A}\). Thus, if \(\{M\}_\mathcal{A}\) satisfies \(\mathcal{F}_K[Y]\), i.e., \(\forall i \in I: \forall A, B \in \{M\}_\mathcal{A}: (A_0 \neq B_0 \Rightarrow A_0 \cap B_0 = \emptyset)\) (according to Lemma 8.4, cf. Lines 33-37), then by transitivity we can conclude that also \(\{M\}_\mathcal{A}\) necessarily satisfies \(\mathcal{F}_K[Y]\).

**Remark.** Recall that we assumed neural-network nodes to have real values (cf. Section 4). Thus, Theorem 9.1 is true for all choices of classical numerical abstract domains [17, 20, 25, 47, etc.] for \(A_1\) and \(A_2\). If we were to consider floating-point values instead, the only sound choices would be floating-point abstract domains [13, 45, 57].

**Other activation functions.** Let us discuss how activation functions other than ReLUs would be handled. The only difference in Algorithm 2 would be the transfer functions \(\text{ReLU}_{\mathcal{D}}^p(x_{s_i})\) (cf. Line 5) and \(\text{ReLU}_{\mathcal{D}}^p(x_{s_i})\) (cf. Line 11), which would have to be replaced with the transfer functions corresponding to the considered activation function. Piecewise-linear activation functions, like Leaky ReLU \((x) = \max(x, k \cdot x)\) or Hard TanH \((x) = \max(-1, \min(x, 1))\), can be treated analogously to ReLUs. Other functions, e.g., Sigmoid \((x) = \frac{1}{1 + e^{-x}}\), can be soundly over-approximated [57].

11 Experimental Evaluation

In this section, we evaluate our approach by focusing on the following research questions:

- **RQ1:** Can our analysis detect seeded (i.e., injected) bias?
- **RQ2:** Is our analysis able to answer specific bias queries?
- **RQ3:** How does the model structure affect the scalability of the analysis?
- **RQ4:** How does the analysis budget affect the scalability-vs-precision tradeoff?
- **RQ5:** Can our analysis effectively leverage multiple CPUs?

11.1 Data

For our evaluation, we used public datasets from the UCI Machine Learning Repository and ProPublica (see below for more details) to train several neural-network models. We primarily focused on datasets discussed in the literature [44] or used by related techniques (e.g., [1–3, 7, 21, 23, 62, 63]).

We pre-processed these datasets both to make them fair with respect to a certain sensitive input feature as well as to seed bias. We describe how we seeded bias in each particular dataset later in this section.
Our methodology for making the data fair was common across datasets. In particular, given an original dataset and a sensitive feature (say, race), we selected the largest population with a particular value for this feature (say, Caucasian) from the dataset (and discarded all others). We removed any duplicate or inconsistent entries from this population. We then duplicated the population for every other value of the sensitive feature (say, Asian and Hispanic). For example, assuming the largest population was 500 Caucasians, we created 500 Asians and 500 Hispanics, and any two of these populations differ only in the value of race. Consequently, the new dataset is causally fair because there do not exist two inputs $k$ and $k'$ that differ only in the value of the sensitive feature for which the classification outcomes are different.

We define the \textit{causal-unfairness score} of a dataset as the percentage of inputs $k$ in the dataset for which there exists another input $k'$ that differs from $k$ only in the value of the sensitive feature and the classification outcome. Our fair datasets have an unfairness score of 0%.

11.2 Setup

Since neural-network training is non-deterministic, we typically train eight neural networks (with four hidden layers with five nodes) on each dataset, unless stated otherwise. We performed all experiments on a 12-core Intel® Xeon® X5650 CPU @ 2.67GHz machine with 48GB of memory, running Debian GNU/Linux 9.6 (stretch). All datasets and models we used are also open-source as part of LIBRA.

11.3 Results

In the following, we present our experimental results for each of the above research questions.

**RQ1: Detecting seeded bias.** This research question focuses on detecting seeded bias by comparing the analysis results for models trained with fair versus biased data.

For this experiment, we used the German Credit Data\(^3\). This dataset classifies creditworthiness into two categories, “good” and “bad”. An input feature is age, which we consider sensitive to bias. We seeded bias in the fair dataset by randomly assigning a bad credit score to people of age 60 and above who request a credit amount of more than EUR 1 000 until we reached a 20% causal-unfairness score of the dataset. The median classification accuracy of the models trained on fair and biased data was 71% and 65%, respectively.

Table 1 shows the analysis results for all models. For the forward pre-analysis, we used the BOXES+SymbOlMIC domain. We set $L = 0$ to be sure to complete the analysis on 100% of the input space. The drawback with this is that the pre-analysis might end up splitting input partitions endlessly. To counteract, for each model, we chose the smallest upper bound for each model. Column $C$ shows

\(^3\)https://archive.ics.uci.edu/ml/datasets/Statlog+(German+Credit+Data)

| CREDIT | \begin{tabular}{c|c|c|c|c} \hline \textbf{FAIR DATA} \hline \textbf{BIASED DATA} \hline \end{tabular} |
|--------|----------------------------------|
| \(\leq 1000\) | \begin{tabular}{c|c|c|c|c} \hline \textbf{U} & \textbf{BIAS} & \textbf{|C|} & \textbf{|F|} & \textbf{TIME} \\ \hline 12 & 0.33% & 138 & 32 & 32 & 52s \\ 12 & 0.17% & 165 & 23 & 23 & 4m 16s \\ 12 & 0.09% & 140 & 10 & 10 & 29s \\ 12 & 0.15% & 159 & 22 & 22 & 2m 5s \\ 12 & 0.23% & 157 & 25 & 25 & 1m 56s \\ 12 & 0.30% & 166 & 32 & 32 & 1m 11s \\ 12 & 0.20% & 135 & 25 & 25 & 1m 4s \\ 12 & 0.16% & 168 & 14 & 14 & 17s \\ \hline \end{tabular} |
| \(> 1000\) | \begin{tabular}{c|c|c|c|c} \hline \textbf{MIN} & 0.09% & 17s & 0.00% & 22s \\ \textbf{MEDIAN} & 0.19% & 1m 6s & 0.41% & 2m 19s \\ \textbf{MAX} & 0.33% & 4m 16s & 0.90% & 14m 27s \\ \hline \end{tabular} |

Table 1. Analysis of Neural Networks Trained on Fair and \{Age, Credit > 1000\}-Biased Data (German Credit Data) — Full Table.
the total number of analyzed (i.e., completed) input space partitions. Column \(|F|\) shows the total number of abstract activation patterns (left) and feasible input partitions (right) that the backward analysis had to explore. The difference between \(|C|\) and the number of partitions shown in \(|F|\) are the input partitions that the pre-analysis found to be already active partitions. Column \(\mathit{bias}\) and \(\mathit{TIME}\) show the detected bias (in percentage of the entire input space) and the analysis running time, respectively. In particular, the table shows whether the models are biased with respect to age for credit requests of 1 000 or less as well as for credit requests of over 1 000. We also report minimum, median, and maximum values for both bias and analysis running time.

We observe that, for models trained on fair data, age bias for credit amounts \(\leq 1\ 000\) is very small in comparison to larger amounts. This is because small credit amounts correspond to a mere 4% of the input space. When only considering the input space of amounts \(\leq 1\ 000\), the median bias is 0.19% / 4% = 4.75%, whereas when only considering larger amounts, the median bias is 6.72% / 96% = 7%. This shows that the models contain bias that does not necessarily depend on the credit amount. The bias is introduced by the training process itself (as explained in the Introduction) and is not due to imprecision of our analysis. Recall that our approach is exact, and imprecision is only introduced when estimating the bias percentage (cf. Section 10).

For the models trained on biased data, the analysis finds significantly more bias for larger credit amounts in comparison to the models trained on the fair dataset. As expected, it also finds similar bias across the different models for smaller credit amounts. This demonstrates that our approach is able to effectively detect seeded bias. It is interesting to point out that the model on the fourth row of the table does not pick up the bias introduced in the dataset, which of course only corresponds to a small sample of the input space.

**RQ2: Answering specific bias queries.** To further evaluate the precision of our approach, we created queries concerning
bias within specific groups of people, each corresponding to a subset of the entire input space.

We used the compas dataset† from ProPublica for this experiment. The data assigns a recidivism-risk score (high, medium, and low) indicating whether criminals are likely to re-offend. The data includes both personal attributes (e.g., age and race) as well as criminal history (e.g., number of priors and violent crimes). As for RQ1, we trained models both on fair and biased data. Here, we considered race as the sensitive feature. We seeded bias in the fair data by randomly assigning high recidivism risk to African Americans until we reached a 20% causal-unfairness score of the dataset. The median classification accuracy of the models trained on fair and biased data was 55% and 56%, respectively.

To analyze these models, we used the boxes+symbolic domain for the forward pre-analysis, a lower bound L of 0, and an upper bound U between 10 and 19. Table 2 summarizes the results of our analysis (i.e., all columns are shown as in Table 1) for three queries:

Q_A: Is there race-bias for people younger than 25?
Q_B: Is there age-bias for males?
Q_C: Is there number-of-priors-bias for Caucasians?

The analysis is able to complete 100% of the input space for each query. For Q_A, the analysis detects only a small percentage of bias in the fair models, but as expected, the bias is found to be significantly higher (ca. 3X) for the biased models. In contrast, for Q_B, the analysis is able to verify that there is no bias for males in both sets of models. Finally, for Q_C, the analysis detects significant bias with respect to the number of priors. Note that the bias percentages are always with respect to the entire input space; not just with respect to Caucasians (for Q_C) representing 1/6 of the input space. Also, note that we did not introduce any bias with respect to the number of priors, so this bias is intended and present in the original data. As one would expect, recidivism risk differs for different numbers of priors. Overall, these results demonstrate the effectiveness of our analysis in answering specific bias queries by detecting bias or verifying its absence.

RQ3: Effect of model structure on scalability. This research question evaluates the effect of the model structure on the scalability of our analysis. To answer it, we trained models on the Adult Census Data‡ by varying the number of layers and nodes per layer. The dataset assigns a yearly income (> or ≤ USD 50K) based on personal attributes such as gender, race, and occupation. We trained all models on a fair dataset with respect to gender and ensured that each model reached a minimum classification accuracy of 78%.

Table 3 summarizes the results for all models. The first column shows the total number of hidden nodes and introduces the marker symbols used in the scatter plot of Figure 3 (the left symbol refers to the boxes domain, whereas the right one refers to the boxes+symbolic domain used by the forward pre-analysis). The models use the following number

Table 3. Comparison of Different Model Structures (Adult Census Data) — Full Table

<table>
<thead>
<tr>
<th></th>
<th>M</th>
<th>U</th>
<th>BOXES</th>
<th>BOXES+SYMBOLIC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>INPUT</td>
<td>C</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>86.81%</td>
<td>1447</td>
<td>230</td>
<td>1142</td>
</tr>
<tr>
<td>6</td>
<td>99.51%</td>
<td>786</td>
<td>255</td>
<td>739</td>
</tr>
<tr>
<td>8</td>
<td>100.00%</td>
<td>152</td>
<td>118</td>
<td>143</td>
</tr>
<tr>
<td>10</td>
<td>100.00%</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>49.76%</td>
<td>712</td>
<td>26</td>
<td>334</td>
</tr>
<tr>
<td>6</td>
<td>72.67%</td>
<td>1191</td>
<td>60</td>
<td>926</td>
</tr>
<tr>
<td>8</td>
<td>98.68%</td>
<td>342</td>
<td>56</td>
<td>284</td>
</tr>
<tr>
<td>10</td>
<td>99.06%</td>
<td>313</td>
<td>65</td>
<td>260</td>
</tr>
<tr>
<td>4</td>
<td>22.01%</td>
<td>625</td>
<td>24</td>
<td>39</td>
</tr>
<tr>
<td>6</td>
<td>45.24%</td>
<td>1111</td>
<td>123</td>
<td>260</td>
</tr>
<tr>
<td>8</td>
<td>64.17%</td>
<td>1108</td>
<td>299</td>
<td>795</td>
</tr>
<tr>
<td>10</td>
<td>85.87%</td>
<td>1376</td>
<td>387</td>
<td>1329</td>
</tr>
<tr>
<td>4</td>
<td>0.00%</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>0.00%</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>0.14%</td>
<td>4</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>10</td>
<td>0.63%</td>
<td>18</td>
<td>12</td>
<td>13</td>
</tr>
<tr>
<td>4</td>
<td>0.00%</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>0.83%</td>
<td>24</td>
<td>3</td>
<td>22</td>
</tr>
<tr>
<td>8</td>
<td>9.41%</td>
<td>270</td>
<td>58</td>
<td>234</td>
</tr>
<tr>
<td>10</td>
<td>18.68%</td>
<td>522</td>
<td>150</td>
<td>488</td>
</tr>
</tbody>
</table>

‡https://archive.ics.uci.edu/ml/datasets/adult
Perfectly Parallel Certification of Neural Network Fairness

![Figure 3. Comparison of Different Model Structures (Adult Census Data)](image)

![Figure 4. Comparison of Different Analysis Configurations (Japanese Credit Screening)](image)

of hidden layers and nodes per layer (from top to bottom): 2 and 5; 4 and 3; 4 and 5; 4 and 10; 9 and 5.

Column U shows the upper bound chosen for each model, while the input and time columns show the input-space coverage (i.e., the percentage of the input space that was completed by the analysis) and the running time. As before, column [C] shows the number of completed input space partitions, and [F] shows the number of abstract activation patterns (left) and feasible input partitions (right) explored by the backward analysis. We used a lower bound L of 0.5 and a total-time limit of 13h.

The scatter plot of Figure 3 visualizes the input coverage and analysis running time. Overall, coverage decreases for more complex model structures and the more precise (but expensive) boxes+symbolic domain results in a significant coverage boost, especially for more complex structures.

Increasing the upper bound U tends to increase coverage independently of the specific model structure. However, interestingly, this does not always come at the expense of an increased running time. In fact, as we will explain in RQ4, such a change tends to help the forward pre-analysis in already proving certain partitions fair. This results in decreasing the number of partitions that the expensive backward analysis needs to analyze as well as the overall running time.

**RQ4: Scalability-vs-precision tradeoff.** To evaluate the effect of the analysis budget (bounds L and U), we analyzed a model using different budget configurations. For this experiment, we used the Japanese Credit Screening6 dataset, which we made fair with respect to gender. Our model had a classification accuracy of 86%.

Table 4 shows the results for different analysis configurations and domains of the forward pre-analysis. Note that the symbol next to each domain introduces the marker used in the scatter plot of Figure 4, which visualizes the coverage and running time.

6https://archive.ics.uci.edu/ml/datasets/Japanese+Credit+Screening

Overall, we observe that the more precise boxes+symbolic domain boosts input coverage (most noticeably for configurations with a larger L). Surprisingly, this additional precision does not always result in longer running times. In fact, for long-running analyses, boxes+symbolic typically reduces the running time. This is because the classification within more partitions is proved fair already by the pre-analysis without requiring the backward analysis.

As expected, a larger U or a smaller L increase precision. Increasing U or L typically reduces the number of partitions. Consequently, partitions tend to be more complex, requiring both forward and backward analyses. Since the backward analysis tends to dominate the running time, more partitions generally increase the running time (when comparing configurations with similar coverage). Based on our experience, the optimal budget largely depends on the analyzed model.

**RQ5: Leveraging multiple CPU cores.** To evaluate the effect of parallelizing the analysis using multiple cores, we re-ran the analyses of RQ4 on 4 CPU cores instead of 12. Table 5 shows these results. For the boxes domain, we observe a significant increase in running time for 4 cores, especially for configurations that achieve high coverage. For instance, the running time increases by a factor of 3.4 for L = 0 and U = 10. On the other hand, for the boxes+symbolic domain, the running time with 4 cores typically increases less drastically. This is again explained by the increased precision of the forward analysis; fewer partitions require a backward pass, where parallelization is most effective.

Finally, Table 6 shows the same experiment on 24 vCPUs.

**12 Related Work**

Significant progress has been made on testing and verifying machine-learning models. We focus on fairness, safety, and robustness properties in the following, especially of deep neural networks.
Table 4. Comparison of Different Analysis Configurations (Japanese Credit Screening) — 12 CPUs

<table>
<thead>
<tr>
<th>L</th>
<th>U</th>
<th>INPUT</th>
<th>BOXES</th>
<th>BOXES+SYMBOLIC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>[C]</td>
<td>[F]</td>
<td>TIME</td>
</tr>
<tr>
<td>0.5</td>
<td>4</td>
<td>2.08%</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>11.11%</td>
<td>28</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>51.39%</td>
<td>98</td>
<td>55</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>79.86%</td>
<td>83</td>
<td>67</td>
</tr>
<tr>
<td>0.25</td>
<td>4</td>
<td>41.91%</td>
<td>1225</td>
<td>41</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>79.77%</td>
<td>1470</td>
<td>214</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>95.92%</td>
<td>1159</td>
<td>476</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>99.54%</td>
<td>437</td>
<td>294</td>
</tr>
<tr>
<td>0.125</td>
<td>4</td>
<td>94.68%</td>
<td>16348</td>
<td>671</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>99.74%</td>
<td>6219</td>
<td>951</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>99.98%</td>
<td>1775</td>
<td>786</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>100.00%</td>
<td>399</td>
<td>287</td>
</tr>
<tr>
<td>0</td>
<td>4</td>
<td>94.68%</td>
<td>47380</td>
<td>1133</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>99.74%</td>
<td>5369</td>
<td>938</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>99.98%</td>
<td>1531</td>
<td>751</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>100.00%</td>
<td>512</td>
<td>354</td>
</tr>
</tbody>
</table>

Table 5. Comparison of Different Analysis Configurations (Japanese Credit Screening) — 4 CPUs

<table>
<thead>
<tr>
<th>L</th>
<th>U</th>
<th>INPUT</th>
<th>BOXES</th>
<th>BOXES+SYMBOLIC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>[C]</td>
<td>[F]</td>
<td>TIME</td>
</tr>
<tr>
<td>0.5</td>
<td>4</td>
<td>2.08%</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>11.11%</td>
<td>30</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>51.39%</td>
<td>90</td>
<td>57</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>79.86%</td>
<td>128</td>
<td>108</td>
</tr>
<tr>
<td>0.25</td>
<td>4</td>
<td>41.91%</td>
<td>1159</td>
<td>42</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>79.77%</td>
<td>1456</td>
<td>210</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>95.92%</td>
<td>926</td>
<td>407</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>99.54%</td>
<td>519</td>
<td>342</td>
</tr>
<tr>
<td>0.125</td>
<td>4</td>
<td>94.68%</td>
<td>15993</td>
<td>681</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>99.74%</td>
<td>4951</td>
<td>851</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>99.98%</td>
<td>1548</td>
<td>745</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>100.00%</td>
<td>506</td>
<td>344</td>
</tr>
<tr>
<td>0</td>
<td>4</td>
<td>94.68%</td>
<td>36165</td>
<td>1076</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>99.74%</td>
<td>5802</td>
<td>955</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>99.98%</td>
<td>1552</td>
<td>751</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>100.00%</td>
<td>528</td>
<td>373</td>
</tr>
</tbody>
</table>

Testing and verifying fairness. Galhotra et al. [23] proposed an approach, Themis, that allows efficient fairness testing of software. Udeshi et al. [63] designed an automated and directed testing technique to generate discriminatory inputs for machine-learning models. Tramer et al. [62] introduced the unwarranted-associations framework and instantiated it in FairTest. In contrast, our technique provides formal fairness guarantees.

Bastani et al. [7] used adaptive concentration inequalities to design a scalable technique for verifying fairness of machine-learning models. Albarghouthi et al. [2] encoded fairness problems as probabilistic program properties and developed an SMT-based technique for verifying fairness of decision-making programs. For certain biased decision-making programs, the program repair technique proposed by Albarghouthi et al. [1] can be used to repair their bias. Albarghouthi et al. [3] further introduced fairness-aware programming, where programmers can specify fairness properties in their code for runtime checking. As mentioned in the Introduction, our approach differs in that it gives definite (instead of probabilistic) guarantees. However, it might exclude partitions for which the analysis is not exact.
Robustness of deep neural networks. Robustness is a desirable property for traditional software [12, 30, 43], especially control systems. Deep neural networks are also expected to be robust. However, research has shown that deep neural networks are not robust to small perturbations of their inputs [59] and can even be easily fooled [51]. Subtle imperceptible perturbations of inputs, known as adversarial examples, can change their prediction results. Various algorithms [11, 28, 42, 60, 66] have been proposed that can effectively find adversarial examples. Research on developing defense mechanisms against adversarial examples [4, 9–11, 15, 22, 28, 32, 48, 49] is also active. Causal fairness of neural networks is a special form of robustness in the sense that neural networks are expected to be globally robust with respect to their sensitive features.

Testing deep learning systems. Multiple frameworks have been proposed to test the robustness of deep learning systems. Pei et al. [54] proposed the first whitebox framework for testing such systems. They used neuron coverage to measure the adequacy of test inputs. Sun et al. [58] presented the first concolic-testing [26, 56] approach for neural networks. Tian et al. [61] and Zhang et al. [67] proposed frameworks for testing autonomous driving systems. Gopinath et al. [29] used symbolic execution [14, 37]. Odena et al. [55] were the first to develop coverage-guided fuzzing for neural networks. Zhang et al. [66] proposed a blackbox-fuzzing technique to test their robustness.

Formal verification of deep neural networks. Formal verification of deep neural networks has mainly focused on safety properties. However, the scalability of such techniques for verifying large real-world neural networks is limited. Early work [55] applied abstract interpretation to verify a neural network with six neurons. Recent work [24, 33, 35, 57, 65] significantly improves scalability. Huang et al. [33] proposed a framework that can verify local robustness of neural networks based on SMT techniques [5]. Katz et al. [35] developed an efficient SMT solver for neural networks with ReLU activation functions. Gehr et al. [24] traded precision for scalability and proposed a sound abstract interpreter that can prove local robustness of realistic deep neural networks. Singh et al. [57] proposed a new abstract domain for certifying robustness of neural networks. Their abstract domain could also be used in our setting to certify fairness properties. Wang et al. [65] are the first to use symbolic interval arithmetic to prove security properties of neural networks.

### 13 Conclusion and Future Work

We have presented an automated, perfectly parallel analysis for certifying fairness of neural networks. The analysis is configurable to support a wide range of use cases throughout the development lifecycle of neural networks: ranging from short sanity checks during development to formal fairness audits before deployments. In future work, we plan to extend our technique in various ways. For instance, by automatically tuning parameters (such as the upper bound U) during the analysis or by feeding analysis results to other tools. Such tools may be used to provide probabilistic fairness guarantees for partitions that could not be certified or repair networks by eliminating bias.

### References
