# Reduced Products of Abstract Domains for Fairness Certification of Neural Networks

Denis Mazzucato[0000−0002−3613−2035] and Caterina Urban[0000−0002−8127−9642]

Inria & École Normale Supérieure | Université PSL
{denis.mazzucato,caterina.urban}@inria.fr

**Abstract.** We present Libra, an open-source abstract interpretation-based static analyzer for certifying fairness of ReLU neural network classifiers for tabular data. Libra combines a sound forward pre-analysis with an exact backward analysis that leverages the polyhedra abstract domain to provide definite fairness guarantees when possible, and to otherwise quantify and describe the biased input space regions. The analysis is configurable in terms of scalability and precision. We equipped Libra with new abstract domains to use in the pre-analysis, including a generic reduced product domain construction, as well as search heuristics to find the best analysis configuration. We additionally set up the backward analysis to allow further parallelization. Our experimental evaluation demonstrates the effectiveness of the approach on neural networks trained on a popular dataset in the fairness literature.

**Keywords:** Fairness · Neural Networks · Reduced Abstract Domain Products · Abstract Interpretation · Static Analysis


## 1 Introduction

Nowadays, machine learning software has an ever increasing societal impact by assisting or even automating decision making in fields such as social welfare, criminal justice, and even health care. At the same time, a number of recent cases have shown that such software may reproduce, or even reinforce, bias directly or indirectly present in the training data [3,16,17,23]. In April 2021, the European Commission proposed a first legal framework on machine learning software – the Artificial Intelligence Act [10] — which imposes strict requirements to minimize the risk of discriminatory outcomes. In this context, methods and tools for certifying fairness or otherwise detecting bias are extremely valuable.

In this paper we present LIBRA, an open-source static analyzer based on abstract interpretation [5] for certifying fairness of neural networks. LIBRA currently supports neural networks with ReLU activations [21], trained for classification of tabular data (e.g., stored in Excel files or relational databases). It is designed to certify that the classification is independent of the values of the inputs that are considered (directly or indirectly) sensitive to bias [12]. This fairness notion is *global*, relative to the entire input space (or a targeted subset of it), and our analysis is able to *quantify* the detected bias. The choice of the sensitive inputs is up to the user of the tool.

The static analysis run by LIBRA combines a cheap and sound forward pre-analysis with an expensive and exact backward analysis. The pre-analysis iteratively partitions the input space of the neural network into independent partitions that satisfy the configured resource requirements. Then, the backward analysis attempts to certify fairness for each of these partitions, and otherwise quantifies and reports their biased (sub)regions.

The pre-analysis can be configured to use any of the abstract domains implementations that LIBRA is equipped with. A preliminary version of LIBRA developed by Urban et al. [25] was equipped with the BOXES [4], SYMBOLIC [18], and the DEEPPOLY [24] abstract domains. In our tool, we additionally implemented the NEURIFY [26] abstract domain, and a generic reduced product domain construction [6] to combine any of these domains together. To the best of our knowledge, we are the first to explore and demonstrate the merits of reduced products of abstract domains for analyzing neural networks.

LIBRA can be further configured in terms of scalability and precision to adapt to the available resources (e.g., computation time or CPUs). We have additionally equipped LIBRA with a configuration auto-tuning mechanism to find the best analysis configuration according to a given search strategy. Finally, we set up the backward analysis to allow further parallelization and thus reduce idle times that were hindering the effective exploitation of multi-core architectures.

In our experimental evaluation we evaluate LIBRA on neural networks trained on a popular dataset and we demonstrate its effectiveness. In particular, we show that LIBRA (configured to use the product domain) outperforms its preliminary version [25] in terms of both precision and running time.

## 2 Tool Architecture

LIBRA is written in PYTHON. Its codebase is open source on GitHub[1].

Figure 1 shows an overview of LIBRA's architecture. The tool takes as input a neural network and a specification of its input space and fairness requirements (cf. Section 2.1). The front-end (cf. Section 2.2) takes care of parsing the neural network and its specification, building an equivalent control flow graph structure, and passing it to the analysis engine (cf. Section 2.3). The analysis can be configured to use different (combinations of) abstract domains (cf. Section 2.4), and

[1] https://github.com/caterinaurban/Libra.git

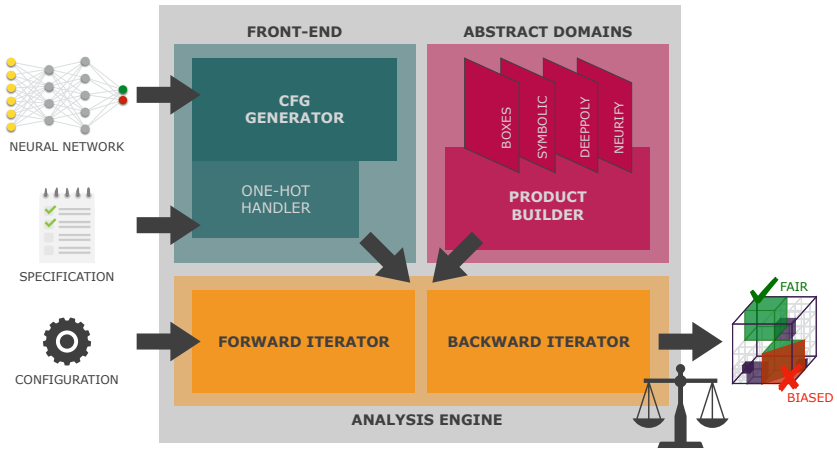


Fig. 1: Overview of Libra.

to be run incrementally to adapt to the available resources (e.g., computation time or CPUs). The tool outputs a partition of the neural network input space into regions that are certified to be fair, regions that are biased, and regions that could not be analyzed (if any) because the analysis exceeded the available resources (cf. Section 2.5). In the rest of the section, we provide more details on each tool component and configuration options.

### 2.1 Tool Input

Libra expects as input a feed-forward neural network with ReLU activation functions (i.e., `ReLU(x)` = `max`$(0, \texttt{x})$ [21]) trained for classification of tabular data. The neural network should be written as a Python program: affine layer transformations are modeled by variable assignments, and ReLU activations are modeled by calls to a `ReLU` function (i.e., a call `ReLU(x)` models the ReLU activation applied to the neuron represented by the variable `x`). Figure 2 depicts a toy network expressed in Python syntax. Specifically, the network is composed by two input neurons $x_{0,1}$ and $x_{0,2}$, two output neurons $x_{3,1}$ and $x_{3,2}$ (one for each class in the output classification), and two hidden layers in between — each one with two hidden neurons. Lines $1, 2, 5, 6, 9$, and 10 show affine computations, while $3, 4, 7$, and 8 apply the activation functions. The output class of the network is determined by the output neuron with the maximum value. The codebase of Libra contains a script to automatically generate such input format from neural networks trained using the Keras framework (`https://keras.io`).

In addition, Libra requires a specification of the neural network input space and fairness requirements. Libra supports both continuous input data features as well as one-hot encoded categorical data features. Thus, the input specification

```
1  x11 = -0.31*x01 + 0.99*x02 - 0.63
2  x12 = -1.25*x01 - 0.64*x02 + 1.88
3  ReLU(x11)
4  ReLU(x12)
5  x21 = 0.40*x11 + 1.21*x12 + 0.00
6  x22 = 0.64*x11 + 0.69*x12 - 0.39
7  ReLU(x21)
8  ReLU(x22)
9  x31 = 0.26*x21 + 0.33*x22 + 0.45
10 x32 = 1.42*x21 + 0.40*x22 - 0.45
```

Fig. 2: Toy Neural Network.

should define which input variables correspond to continuous and categorical data. Additionally, it should indicate which inputs should be considered sensitive to bias by the analysis. In our example in Figure 2, we consider both inputs as continuous, i.e., $x_{0,1}, x_{0,2} \in [0, 1]$, and $x_{0,2}$ as sensitive to bias.

### 2.2 Front-End

The front-end of Libra parses the neural network and its specification and generates a control flow graph (CFG) structure to be given to the analysis engine. More specifically, the *CFG Generator* builds an acyclic graph which is essentially a sequence of nodes alternating between nodes of type affine and nodes of type ReLU, i.e., nodes grouping the affine transformations performed by a neural network layer, or nodes grouping the ReLU activations applied to a neural network layer. The entry node of the CFG is annotated with assumptions restricting the range of values of the input features (i.e., by default, features are assumed to be normalized in the range $[0, 1]$). For categorical features, the *One-Hot Handler* imposes additional constraints restricting the values of the corresponding individual inputs to be either 0 or 1, and their sum to be 1. Figure 3 shows the control flow graph corresponding to the toy network in Figure 2. The caption of each node shows which line of code it represents.

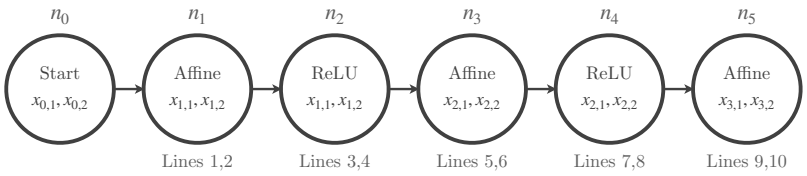


Fig. 3: Control Flow Graph for the Toy Network in Figure 2.

### 2.3 Analysis Engine

LIBRA's analysis engine walks over the CFG in two phases: a forward pre-analysis, starting from the entry node of the CFG, followed by a backward analysis starting from the exit node of the CFG. Both analysis phases use a standard worklist algorithm [22] implemented using a FIFO queue. At each step, a CFG node is extracted from the worklist and its associated instructions are interpreted in an abstract domain (cf. Section 2.4) to update the current value of the analysis. All successor or predecessor nodes — depending on the analysis direction — are then put into the worklist. Each node is explored exactly once each iteration. The analysis terminates once the worklist is empty. In our example, cf. Figure 3, the forward analysis visits the CFG nodes in the order from $n_0$, $n_1$, ..., up to $n_5$, while the backward analysis visits the node in the reverse order, i.e., from $n_5$, $n_4$, ..., down to $n_0$.

The forward pre-analysis is performed by the *Forward Iterator*. The pre-analysis begins with a value representing the entire neural network input space in a chosen **abstract domain**. This value is then propagated through the neural network. If the resulting output value implies that the network always produces a unique classification outcome, then fairness is trivially guaranteed as there is no way to discriminate between input data. Otherwise, there are two possibilities depending on how many ReLUs of the neural network are found to not have a fixed activation status (i.e., `ReLU(x)` is always active when $\texttt{x} \geq 0$ and always inactive when $\texttt{x} < 0$). If this number exceeds a chosen **upper bound** U, the pre-analysis bisects the input space along *any* of the non-sensitive dimensions (randomly chosen) and proceeds again on the resulting partitions. Instead, if the number of non-fixed ReLUs does not exceed U, the input space (partition) is deemed *feasible* and passed to the backward analysis along with its associated ReLU activation pattern. In our example, let the abstract domain be the boxes domain [4], which simply tracks the interval of possible values for each neuron in the neural network, and let $\mathrm{U} = 2$. At first, the pre-analysis starts from the entire input space $I$, i.e., $x_{0,1}, x_{0,2} \in [0, 1]$. By propagating these interval values through the CFG, the analysis finds that the ReLUs at $x_{1,1}, x_{1,2}$, and $x_{2,2}$ are non-fixed while $x_{2,1}$ is always active. Since the number of non-fixed ReLUs exceed the upper bound U, the analysis bisects the input space along the only non-sensitive dimension $x_{0,1}$, yielding two partitions $I_1$ ($x_{0,1} \in [0, 0.5]$ and $x_{0,2} \in [0, 1]$) and $I_2$ ($x_{0,1} \in [0.5, 1]$ and $x_{0,2} \in [0, 1]$). By running the pre-analysis from $I_1$ and $I_2$, we find that $I_1$ is feasible since only the ReLU at $x_{1,1}$ is non-fixed (and all other activations are always active), while $I_2$ must be divided further.

To ensure termination, bisection may continue until the partition size becomes smaller than a chosen **lower bound** L. In such a case, the partition is *excluded* by the analysis as it exceeds the available resources. Continuing our example, let $\mathrm{L} = 0.25$. The forward pre-analysis splits $I_2$ into $I_{2,1}$ ($x_{0,1} \in [0.5, 0.75]$ and $x_{0,2} \in [0, 1]$) and $I_{2,2}$ ($x_{0,1} \in [0.75, 1]$ and $x_{0,2} \in [0, 1]$). Now, the pre-analysis concludes that $I_{2,1}$ is feasible, with only the ReLUs at $x_{1,1}$ and $x_{2,2}$ being non-fixed. Instead $I_{2,2}$ is excluded, since only the ReLU at $x_{1,2}$ is fixed and the par-

tition size (along the non-sensitive dimension $x_{0,1}$) has reached the lower bound L. Thus, only 75% of the input space is considered by the backward analysis.

The configuration of the pre-analysis (i.e., choices of an abstract domain, lower bound L, and upper bound U) allows trading-off between precision and scalability of the approach (cf. Table 2 in Section 3). Ultimately however, the optimal configuration largely depends on the analyzed neural network [25]. For this reason, we have equipped LIBRA with a *configuration auto-tuning mechanism*, which dynamically updates the lower bound and upper bound configuration according to a chosen **search heuristic**. By default, whenever an input partition exceeds the current configuration, the pre-analysis alternates between increasing the upper bound by one, up to a maximum upper bound $\mathrm{U}_{\max}$, and halving the lower bound, down to a minimum lower bound $\mathrm{L}_{\min}$. Other bound update patterns are configurable (e.g., by updating both bounds at the same time, or performing multiple increments to the upper bound before halving the lower bound, etc.). In our example, let $\mathrm{U}_{\max} = 3$, the pre-analysis can thus further increase the upper bound to $\mathrm{U} = 3$. Therefore, also $I_{2,2}$ becomes feasible (with the ReLUs at $x_{1,1}, x_{1,2}$, and $x_{2,2}$ non-fixed).

The *Backward Iterator* takes care of performing the backward analysis independently for each feasible partition and associated ReLU activation pattern. Specifically, the backward analysis starts with different polyhedra abstract domain values [7], each representing a possible classification outcome of the neural network. In our example, the possible classification outcomes[2] are represented by the polyhedra $x_{32} < x_{31}$ and $x_{31} < x_{32}$. These values are then propagated backwards through the network, taking the current activation pattern into account to prune away unfeasible execution paths, and otherwise splitting polyhedra into two at each non-fixed ReLU in order to retain maximum precision (by analyzing their possible activations separately). Ultimately, for each partition, this yields a disjunction of polyhedra covering the inputs that lead to each possible output classification. We can then project away the value of the sensitive inputs and check for intersections between polyhedra leading to different classifications: any non-empty intersection is a region of the input space in which bias is definitely present, as all points in the region represent data that only differ in the values of the sensitive inputs and lead to different classification outcomes. Otherwise, if no intersection can be found, the input space partition is certified to be fair. In our example, the analysis concludes that the classification within $I_1$ is fair, while it is biased within $I_{2,1}$ and $I_{2,2}$. Inside the biased intersections, the neural network returns different output classes for inputs that only differ in the sensitive features (i.e., the have the same value for $x_{0,1}$ and different values for $x_{0,2}$).

In the preliminary version of LIBRA [25], feasible partitions are first grouped by activation pattern, i.e., activation patterns that fix more ReLUs are merged with those that fix fewer ReLUs. This way, in principle, the amount of work that the backward analysis has to do is reduced: it only needs to run once for each activation pattern, and can then perform all the checks for bias on each feasible partition. In practice, the implementation prevents the parallelization of all bias

[2] For simplicity, we ignore ties as they can always be broken arbitrarily.

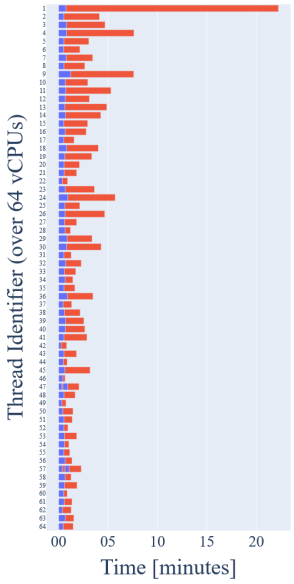


Fig. 4: Former Task Scheduling

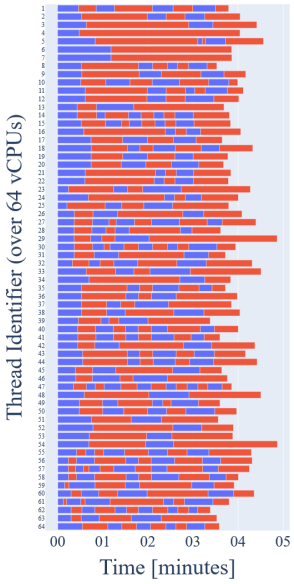


Fig. 5: Current Task Scheduling

checks[3], which are thus run sequentially. This hinders the preliminary version of Libra from exploiting multi-core architectures effectively. In the current version of Libra, we optimize the backward analysis in order to possibly repeat the analysis for the same activation pattern but allowing it to parallelize the bias checks. Figures 4 and 5 compare the previous and current backward analysis task scheduling on the same analysis instance. Each row in the Gantt diagrams shows computations of the same thread. Blue bars stand for activation pattern computations, while red bars indicate bias check computations. As shown in Figure 4, the running time was determined almost completely by the task with the most associated bias checks, leaving all the other threads idle from the very beginning. The diagram in Figure 5 is more compact, meaning that threads are always running jobs uniformly. Consequently, the backward analysis running time decreases from about 22 to only 5 minutes.

### 2.4 Abstract Domains

Different abstract domains can be used by Libra's forward pre-analysis. A preliminary version of Libra [25] was equipped with the boxes [4], symbolic [18,27], and the deeppoly [24] abstract domains. We additionally implemented

[3] This is solely for technical reasons as the serialization of abstract domain elements is not available for the polyhedra domain implementation that Libra relies on. We plan to address this shortcoming as part of our future work.

the NEURIFY [26] abstract domain, and a generic reduced product domain construction [6] to combine any of these domains together.

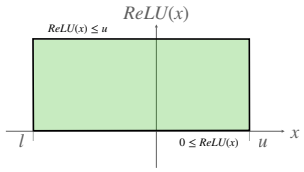


Fig. 6: Naive Convex Approximation of a ReLU Activation.

The BOXES domain simply uses interval arithmetic [13] to compute concrete lower and upper bound estimations $l$ and $u$ for the value of each neuron `x` in the neural network. The SYMBOLIC domain combines BOXES with symbolic constant propagation [20]: in addition to being bounded by concrete lower and upper bounds, the value of each neuron `x` is represented symbolically as a linear combination of the input neurons and the value of the non-fixed ReLUs in previous layers. Specifically, given `x` bounded by $l < 0$ and $u > 0$, `ReLU(x)` is represented by a fresh symbolic variable bounded by 0 and $u$ (cf. Figure 6). By retaining variable dependencies, symbolic representations yield a tighter over-approximation of the value of each neuron in the network.

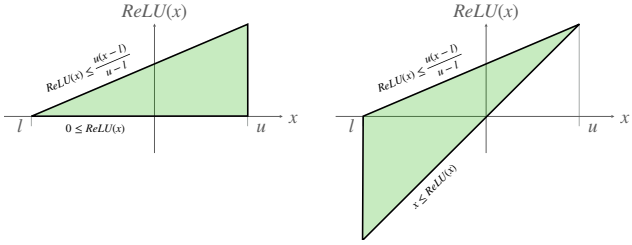


Fig. 7: DEEPPOLY's Convex Approximations of a ReLU Activation.

The DEEPPOLY domain associates to each neuron $x$ of a neural network concrete lower and upper bounds $l$ and $u$ as well as symbolic bounds expressed as linear combinations of neurons in the preceding layer of the network. The concrete bounds are computed by back-substitution of the symbolic bounds up to the input layer. Non-fixed ReLUs are over-approximated by partially retaining dependencies with preceding neurons using the tighter convex approximation between those shown in Figure 7 (i.e., the approximation shown on the left when $u \leq -l$, and the approximation shown on the right otherwise). The NEURIFY domain similarly maintains symbolic lower and upper bounds *low* and *up* for each neuron $x$ of neural network. Unlike DEEPPOLY, concrete lower and upper bounds are computed for *each* symbolic bound: $l_{low}$ and $u_{low}$ for the symbolic lower bound, and $l_{up}$ and $u_{up}$ for the symbolic upper bound. The over-approximation of non-fixed ReLUs is done *independently* for each symbolic bound, i.e., for the *low* bound if $l_{low} < 0 < u_{low}$, and for the *up* bound if $l_{up} < 0 < u_{up}$. Figure 8 shows the approximation

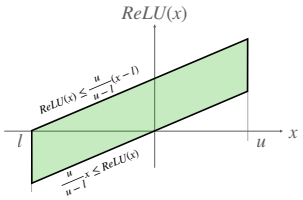


Fig. 8: NEURIFY's Approximation of a ReLU Activation.

for $l = l_{low} = l_{up}$ and $u = u_{low} = u_{up}$. In general, the slope of the symbolic constraints will differ through successive approximation steps.

Finally, the *Product Builder* provides a parametric interface for constructing the product of any of the above domains. The reduction function consists in an exchange of concrete bounds between domains. In particular, this allows determining tighter lower and upper bound estimations for each neuron in the network and thus reducing the over-approximation error introduced by the ReLUs. New abstract domains only need to implement the interface to share bounds information to enable their combination with other domains by the Product Builder.

For the backward analysis, as mentioned, Libra uses the disjunctive polyhedra domain [7]. Its implementation relies on the Apron domain library [14].

### 2.5 Tool Output

Libra outputs which partitions of the input space could be analyzed and which were excluded because they exceeded the configuration of the pre-analysis. For all partitions that could be analyzed, it reports which (sub)regions could be certified to be fair and which were found to be biased. Libra also reports the percentage of the input space that was analyzed and (an estimate of) the percentage that was found biased. To obtain the latter, we simply use the size of a box wrapped around each biased region. More precise but also costlier solutions exist [1].

In our example, the analysis could analyze the entire input space, certifying partition $I_1$ to be fair and finding bias within $I_{2,1}$ and $I_{2,2}$. In particular, the analysis determines that bias occurs for $0.53 < x_{0,1} \leq 0.75$ within $I_{2,1}$ and for $0.75 \leq x_{0,1} < 1$ within $I_{2,2}$, which amounts to 45.76% of the entire input space.

## 3 Experimental Evaluation

To demonstrate the effectiveness of Libra, we evaluated it on neural networks trained on the Adult dataset[4] from the UCI Machine Learning Repository. The dataset assigns to individuals a yearly income greater or smaller than $50k based on personal attributes such as education and occupation but also gender, marital status, or race. We set Libra to use gender as sensitive input feature.

We show below the experimental results on the smaller neural networks used by Urban et al. [25], which better demonstrate the benefits of our implementation compared to its preliminary version. In practice, Urban et al. [25] have already shown that the approach can scale to much larger networks with sizes on par with the literature on fairness certification, e.g., [19,28]. The neural networks were trained with Keras for 50 iterations, using the RMSprop optimizer with the default learning rate, and categorical cross-entropy as the loss function. All networks are open source as part of Libra.

The experiments were conducted on the Inria Paris cleps infrastructure, on a machine with two 16-core Intel® Xeon® 5218 CPU @ 2.4GHz, 192GB of RAM,

[4] https://archive.ics.uci.edu/ml/datasets/adult

Table 1: Comparison of Different Neural Networks

| \|M\| | BOXES | SYMBOLIC | DEEPPOLY | NEURIFY | PRODUCT | |
|---|---|---|---|---|---|---|
| 10 | 96.81% | 98.72% | 98.37% | 98.51% | 99.44% | INPUT |
| | 6m 32s | 4m 52s | 3m 23s | 4m 27s | 4m 40s | TIME |
| 12 | 69.10% | 76.70% | 66.39% | 64.58% | 77.29% | INPUT |
| | 4m 53s | 2m 27s | 2m 0s | 1m 31s | 2m 30s | TIME |
| 20 | 41.01% | 56.11% | 56.10% | 53.06% | 68.23% | INPUT |
| | 4m 8s | 9m 7s | 3m 43s | 3m 53s | 8m 9s | TIME |
| 40 | 0.35% | 34.72% | 38.69% | 41.22% | 51.18% | INPUT |
| | 1m 3s | 7m 2s | 37m 16s | 10m 33s | 38m 27s | TIME |
| 45 | 1.74% | 43.78% | 51.21% | 50.59% | 55.53% | INPUT |
| | 50s | 3m 42s | 5m 14s | 5m 10s | 6m 22s | TIME |

and running CentOS 7.7. with linux kernel 3.10.0. For each experiment, we report the average results of five executions to account for the effect of randomness in the input space partitioning done by the forward pre-analysis (cf. Section 2).

### 3.1 Effect of Neural Network Structure on Precision and Scalability.

The precision and scalability of LIBRA's analysis depend on the analyzed neural network. Table 1 shows the result of running LIBRA on different neural networks with different choices for the abstract domain used by the pre-analysis. Column |M| refers to the analyzed neural network by the number of its ReLU activations. From top to bottom, the neural networks have the following number of hidden layers and nodes per layer: 2 and 5, 4 and 3, 4 and 5, 4 and 10, and 9 and 5. We configured the pre-analysis with lower bound L = 0.5 and upper bound U = 5. Each column shows the chosen abstract domain. We show here the results for BOXES, SYMBOLIC, DEEPPOLY, NEURIFY, and the reduced product DEEPPOLY+NEURIFY+SYMBOLIC (i.e., PRODUCT in the Table 1). The INPUT rows show the average input-space coverage, that is, the average percentage of the input space that LIBRA was able to analyze with the chosen pre-analysis configuration. The TIME rows show the average running time.

For all neural networks, PRODUCT achieves the highest input-space coverage, an improvement of up to 12.49% over the best coverage obtained with only the abstract domains available in the preliminary version of LIBRA [25] (i.e., with respect to the DEEPPOLY domain for $|M| = 40$). Interestingly, such an improvement comes at the cost of a very modest increase in running time (i.e., just over 1 minute). Indeed, using a more precise abstract domain for the pre-analysis generally results in fewer input space partitions being passed to the backward analysis and, in turn, this reduces the overall running time.

For the smallest neural networks (i.e., $|M| \in \{10, 12, 20\}$), the SYMBOLIC abstract domain is the second best choice in terms of input-space coverage. This is likely due to the convex ReLU approximations of DEEPPOLY and NEURIFY which in some case produce a negative lower bound (cf. Figure 7 and 8), while SYMBOLIC always sets the lower bound to zero (cf. Figure 6).

Table 2: Comparison of Different Pre-Analysis Configurations

| L | U | BOXES | SYMBOLIC | DEEPPOLY | NEURIFY | PRODUCT | |
|---|---|---|---|---|---|---|---|
| 0.5 | 3 | 37.88% | 48.78% | 49.01% | 46.49% | 59.20% | INPUT |
| | | 36s | 42s | 1m 35s | 32s | 1m 58s | TIME |
| | 5 | 41.01% | 56.11% | 56.15% | 53.06% | 68.23% | INPUT |
| | | 4m 8s | 9m 10s | 3m 47s | 3m 57s | 8m 16s | TIME |
| 0.25 | 3 | 70.62% | 83.63% | 81.82% | 81.40% | 87.04% | INPUT |
| | | 5m 49s | 5m 55s | 5m 20s | 5m 20s | 7m 12s | TIME |
| | 5 | 83.06% | 91.67% | 91.58% | 92.33% | 95.48% | INPUT |
| | | 26m 43s | 21m 8s | 22m 8s | 25m 54s | 21m 58s | TIME |

Finally, for the largest neural networks (i.e., $|\text{M}| \in \{40, 45\}$), it is the structure of the network (rather than its number of ReLU activations) that impacts the precision and scalability of the analysis: for the deep but narrow network (i.e., $|\text{M}| = 45$), LIBRA achieves a higher input-space coverage in a shorter running time than for the shallow but wide network (i.e., $|\text{M}| = 40$).

### 3.2 Precision-vs-Scalability Tradeoff.

The configuration of LIBRA's pre-analysis allows trading-off between precision and scalability. Table 2 shows the average results of running LIBRA on the neural network with 20 ReLUs with different lower and upper bound configurations, and different choices for the abstract domain used by the pre-analysis. Columns L and U show the configured lower and upper bounds. We tried $\text{L} \in \{0.5, 0.25\}$ and $\text{U} \in \{3, 5\}$. We again show the results for BOXES, SYMBOLIC, DEEPPOLY, NEURIFY, and the reduced product DEEPPOLY+NEURIFY+SYMBOLIC (i.e., PRODUCT in Table 2).

As expected, decreasing the lower bound L or increasing the upper bound U improves the input-space coverage (INPUT rows) and increases the running time (TIME rows). We obtain an improvement of up to 12.44% by increasing U from 3 to 5 (with $\text{L} = 0.25$ and BOXES), and up to 42.05% by decreasing L from 0.5 to 0.25 (with $\text{U} = 5$ and BOXES). The smaller is L and the larger is U, the higher is the impact on the running time. Once again, for all lower and upper bound configurations, DEEPPOLY+NEURIFY+SYMBOLIC achieves the highest input-space coverage, improving up to 12.08% over the best coverage obtained with only the abstract domains available in the preliminary version of LIBRA (i.e., with respect to DEEPPOLY with $\text{L} = 0.5$ and $\text{U} = 5$). The improvement is more important for configurations with larger lower bounds.

Notably, Table 2 shows that *none among the* SYMBOLIC*,* DEEPPOLY*, and* NEURIFY *abstract domains is always more precise than the others.* There are cases where even SYMBOLIC (implemented by [27]) outperforms NEURIFY (implemented by [26] which is the successor of [27] and is believed to be strictly superior to its predecessor), e.g., configuration $\text{L} = 0.5$ and $\text{U} = 5$. We thus argue for using reduced products of abstract domains also in other contexts beyond fairness certification, e.g., verifying local robustness [18,24, etc.] or verifying

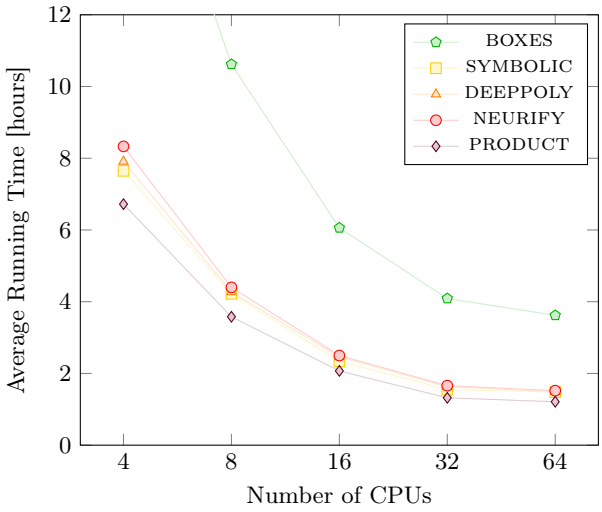


Fig. 9: Comparison of Running Times for Different Number of CPUs

functional properties of neural networks [15].

### 3.3 Leveraging Multiple CPUs.

The optimal pre-analysis configuration in terms of precision or scalability depends on the analyzed neural network. In order to push LIBRA to its limits and obtain 100% input-space coverage on the neural network with 20 ReLUs, we used the new configuration auto-tuning mechanism starting with $L = 1$ and $U = 0$ (i.e., the most restrictive lower and upper bound configuration) and setting $L_{min} = 0$ and $U_{max} = 20$ (i.e., the most permissive configuration). For all choices of abstract domains, the pre-analysis eventually stabilizes with lower bound $L = 0.015625$ and upper bound $U = 6$.

Figure 9 compares the average running times for BOXES, SYMBOLIC, DEEPPOLY, NEURIFY, and the reduced product DEEPPOLY+NEURIFY+SYMBOLIC (i.e., PRODUCT) as a function of the number of available CPUs. With PRODUCT we obtained a running time improvement of 14.39% over SYMBOLIC, i.e., the fastest domain available in the preliminary version of LIBRA (a minimum improvement of 11.54% with 16 CPUs, and a maximum improvement of 18.24% with 64 vCPUs). As expected, adding more CPUs always improves LIBRA running time. The most limited improvement in running time that occurs between 32 CPUs and 64 vCPUs is likely due to the use of hyperthreading as context switches between processes running intense numeric computations produce more overhead.

Table 3 additionally shows the estimated percentage of bias detected with each abstract domain, i.e., LIBRA is able to certify fairness for about 95% of the neural network input space. Note that, the bias estimate depends on the

Table 3: Comparison of Different Number of CPUs

| \|CPU\| | BOXES | SYMBOLIC | DEEPPOLY | NEURIFY | PRODUCT | |
|---|---|---|---|---|---|---|
| 4 | 100% | 100% | 100% | 100% | 100% | INPUT |
| | 4.55% | 5.23% | 5.20% | 5.11% | 5.42% | BIAS |
| | 19h 20m 0s | 7h 38m 43s | 7h 54m 35s | 8h 19m 36s | 6h 43m 28s | TIME |
| 8 | 100% | 100% | 100% | 100% | 100% | INPUT |
| | 4.41% | 5.16% | 5.12% | 5.18% | 5.46% | BIAS |
| | 10h 37m 28s | 4h 13m 27s | 4h 16m 13s | 4h 24m 13s | 3h 34m 38s | TIME |
| 16 | 100% | 100% | 100% | 100% | 100% | INPUT |
| | 4.56% | 5.19% | 5.12% | 5.20% | 5.34% | BIAS |
| | 6h 3m 23s | 2h 20m 37s | 2h 27m 31s | 2h 30m 4s | 2h 4m 9s | TIME |
| 32 | 100% | 100% | 100% | 100% | 100% | INPUT |
| | 4.50% | 5.11% | 5.10% | 5.10% | 5.37% | BIAS |
| | 4h 5m 16s | 1h 33m 16s | 1h 37m 40s | 1h 39m 19s | 1h 19m 23s | TIME |
| 64 | 100% | 100% | 100% | 100% | 100% | INPUT |
| | 4.51% | 5.11% | 5.20% | 5.16% | 5.37% | BIAS |
| | 3h 37m 9s | 1h 28m 38s | 1h 29m 26s | 1h 31m 28s | 1h 12m 21s | TIME |

partitioning of the input space computed by the pre-analysis, cf. Section 2. This explains the different percentages found even by runs with the same abstract domain. Within the same column, the difference is at most 0.14% on average.

Finally, we remark that *the new auto-tuning mechanisms is essential for scalability*. We tried repeating this experiment by directly running Libra with the configuration at which auto-tuning stabilizes, i.e., $L = 0.015625$ and $U = 6$. After six days it still had not completed and we had to interrupt it.

## 4 Conclusion and Future Work

In this paper, we presented our static analyzer Libra for certifying that ReLU-based neural network classifiers are independent of their input values that are sensitive to bias. In particular, we focused on the new release features of Libra: new abstract domains, including a generic reduced product domain construction, a configuration auto-tuning mechanism for finding the optimal configuration for Libra's forward pre-analysis, and a tasks scheduling optimization to leverage all the available CPUs for Libra's backward analysis. With our experimental evaluation, we showed that Libra outperforms its preliminary version [25] in precision as well as, for equal precision, in running time.

It remains for future work to implement support for other activation functions than ReLUs. It would also be straightforward to adapt Libra to support other fairness notions such as individual fairness [9]. Moreover, we plan to design and equip Libra with a smarter reduced product between domains, able to also exchange symbolic bounds along with the concrete bounds. Finally, we intend to extend our approach to other machine learning models, such as support vector machines [8] or decision tree ensembles [2,11].

*Acknowledgements.* The authors are grateful to the anonymous reviewers for their constructive comments and advice, and to the cleps infrastructure from the Inria of Paris for providing resources and support.

## References


1. Barvinok, A.I.: A Polynomial Time Algorithm for Counting Integral Points in Polyhedra When the Dimension is Fixed. Mathematics of Operations Research **19**(4), 769–779 (1994). https://doi.org/10.1287/moor.19.4.769, https://doi.org/10.1287/moor.19.4.769
2. Breiman, L.: Random forests. Machine Learning **45**(1), 5–32 (2001). https://doi.org/10.1023/A:1010933404324, https://doi.org/10.1023/A:1010933404324
3. Buolamwini, J., Gebru, T.: Gender shades: Intersectional accuracy disparities in commercial gender classification. In: FAT. PMLR, vol. 81, pp. 77–91. PMLR (2018)
4. Cousot, P., Cousot, R.: Static Determination of Dynamic Properties of Programs. In: Second International Symposium on Programming. pp. 106–130 (1976)
5. Cousot, P., Cousot, R.: Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints. In: POPL. pp. 238–252 (1977). https://doi.org/10.1145/512950.512973, https://doi.org/10.1145/512950.512973
6. Cousot, P., Cousot, R.: Systematic Design of Program Analysis Frameworks. In: POPL. pp. 269–282 (1979). https://doi.org/10.1145/567752.567778, https://doi.org/10.1145/567752.567778
7. Cousot, P., Halbwachs, N.: Automatic Discovery of Linear Restraints Among Variables of a Program. In: POPL. pp. 84–96 (1978). https://doi.org/10.1145/512760.512770, https://doi.org/10.1145/512760.512770
8. Cristianini, N., Shawe-Taylor, J.: An Introduction to Support Vector Machines and Other Kernel-based Learning Methods. Cambridge University Press (2000). https://doi.org/10.1017/CBO9780511801389
9. Dwork, C., Hardt, M., Pitassi, T., Reingold, O., Zemel, R.S.: Fairness through awareness. In: ITCS. pp. 214–226. ACM (2012)
10. European Commission: Proposal for a Regulation Laying Down Harmonised Rules on Artificial Intelligence (Artificial Intelligence Act). https://digital-strategy.ec.europa.eu/en/library/proposal-regulation-laying-down-harmonised-rules-artificial-intelligence-artificial-intelligence (April 2021)
11. Friedman, J.H.: Greedy function approximation: A gradient boosting machine. Annals of Statistics **29**(5), 1189–1232 (10 2001). https://doi.org/10.1214/aos/1013203451
12. Galhotra, S., Brun, Y., Meliou, A.: Fairness Testing: Testing Software for Discrimination. In: FSE. pp. 498–510 (2017). https://doi.org/10.1145/3106237.3106277, https://doi.org/10.1145/3106237.3106277
13. Hickey, T.J., Ju, Q., van Emden, M.H.: Interval Arithmetic: From Principles to Implementation. Journal of the ACM **48**(5), 1038–1068 (2001)
14. Jeannet, B., Miné, A.: APRON: A Library of Numerical Abstract Domains for Static Analysis. In: CAV. pp. 661–667 (2009). https://doi.org/10.1007/978-3-642-02658-4_52, https://doi.org/10.1007/978-3-642-02658-4_52
15. Katz, G., Barrett, C.W., Dill, D.L., Julian, K., Kochenderfer, M.J.: Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks. In: CAV. pp. 97–117 (2017). https://doi.org/10.1007/978-3-319-63387-9_5, https://doi.org/10.1007/978-3-319-63387-9_5

16. Kay, M., Matuszek, C., Munson, S.A.: Unequal representation and gender stereotypes in image search results for occupations. In: CHI. pp. 3819–3828. ACM (2015)
17. Larson, J., Mattu, S., Kirchner, L., Angwin, J.: How we analyzed the COMPAS recidivism algorithm (2016), https://www.propublica.org/article/how-we-analyzed-the-compas-recidivism-algorithm
18. Li, J., Liu, J., Yang, P., Chen, L., Huang, X., Zhang, L.: Analyzing Deep Neural Networks with Symbolic Propagation: Towards Higher Precision and Faster Verification. In: SAS. pp. 296–319 (2019). https://doi.org/10.1007/978-3-030-32304-2_15, https://doi.org/10.1007/978-3-030-32304-2_15
19. Manisha, P., Gujar, S.: FNNC: Achieving Fairness Through Neural Networks. In: IJCAI. pp. 2277–2283 (2020)
20. Miné, A.: Symbolic Methods to Enhance the Precision of Numerical Abstract Domains. In: VMCAI. pp. 348–363 (2006). https://doi.org/10.1007/11609773_23, https://doi.org/10.1007/11609773_23
21. Nair, V., Hinton, G.E.: Rectified Linear Units Improve Restricted Boltzmann Machines. In: ICML. pp. 807–814 (2010)
22. Nielson, F., Nielson, H.R., Hankin, C.: Principles of Program Analysis. Springer (1999)
23. Obermeyer, Z., Powers, B., Vogeli, C., Mullainathan, S.: Dissecting racial bias in an algorithm used to manage the health of populations. Science **366**, 447–453 (2019)
24. Singh, G., Gehr, T., Püschel, M., Vechev, M.T.: An Abstract Domain for Certifying Neural Networks. PACMPL **3**(POPL), 41:1–41:30 (2019). https://doi.org/10.1145/3290354, https://doi.org/10.1145/3290354
25. Urban, C., Christakis, M., Wüstholz, V., Zhang, F.: Perfectly Parallel Fairness Certification of Neural Networks. Proceedings of the ACM on Programming Languages **4**(OOPSLA), 185:1–185:30 (2020). https://doi.org/10.1145/3428253, https://doi.org/10.1145/3428253
26. Wang, S., Pei, K., Whitehouse, J., Yang, J., Jana, S.: Efficient formal safety analysis of neural networks. In: NeurIPS 2018. pp. 6369–6379 (2018)
27. Wang, S., Pei, K., Whitehouse, J., Yang, J., Jana, S.: Formal security analysis of neural networks using symbolic intervals. In: Security. pp. 1599–1614. USENIX (2018)
28. Yurochkin, M., Bower, A., Sun, Y.: Training Individually Fair ML Models with Sensitive Subspace Robustness. In: ICLR (2020)