

Guarding the Privacy of Label-Only Access to Neural Network Classifiers via iDP Verification

ANAN KABAHA, Technion, Israel

DANA DRACHSLER COHEN, Technion, Israel

Neural networks are susceptible to privacy attacks that can extract private information of the training set. To cope, several training algorithms guarantee differential privacy (DP) by adding noise to their computation. However, DP requires to add noise considering *every possible* training set. This leads to a significant decrease in the network's accuracy. Individual DP (iDP) restricts DP to a *given* training set. We observe that some inputs deterministically satisfy iDP *without any noise*. By identifying them, we can provide iDP label-only access to the network with a minor decrease to its accuracy. However, identifying the inputs that satisfy iDP without any noise is highly challenging. Our key idea is to compute the *iDP deterministic bound* (iDP-DB), which overapproximates the set of inputs that do not satisfy iDP, and add noise only to their predicted labels. To compute the tightest iDP-DB, which enables to guard the label-only access with minimal accuracy decrease, we propose LUCID, which leverages several formal verification techniques. First, it encodes the problem as a mixed-integer linear program, defined over a network and over every network trained identically but without a unique data point. Second, it abstracts a set of networks using a *hyper-network*. Third, it eliminates the overapproximation error via a novel branch-and-bound technique. Fourth, it bounds the differences of matching neurons in the network and the hyper-network, encodes them as linear constraints to prune the search space, and employs linear relaxation if they are small. We evaluate LUCID on fully-connected and convolutional networks for four datasets and compare the results to existing DP training algorithms, which in particular provide iDP guarantees. We show that LUCID can provide classifiers with a perfect individuals' privacy guarantee (0-iDP) – which is infeasible for DP training algorithms – with an accuracy decrease of 1.4%. For more relaxed ϵ -iDP guarantees, LUCID has an accuracy decrease of 1.2%. In contrast, existing DP training algorithms that obtain ϵ -DP guarantees, and in particular ϵ -iDP guarantees, reduce the accuracy by 12.7%.

CCS Concepts: • **Theory of computation** → **Program analysis; Program verification**; • **Software and its engineering** → **Formal methods**; • **Computing methodologies** → **Neural networks**.

Additional Key Words and Phrases: Neural Network Verification, Neural Network Privacy, Constrained Optimization

ACM Reference Format:

Anan Kabaha and Dana Drachler Cohen. 2025. Guarding the Privacy of Label-Only Access to Neural Network Classifiers via iDP Verification. *Proc. ACM Program. Lang.* 9, OOPSLA1, Article 122 (April 2025), 31 pages. <https://doi.org/10.1145/3720480>

1 Introduction

Deep neural networks achieve remarkable success in a wide range of tasks. However, this success is challenged by their vulnerability to privacy leakage [Al-Rubaie and Chang 2019; Jere et al. 2021; Papernot 2018], a critical concern in the era of data-driven algorithms. Privacy leakage refers to the unintended disclosure of sensitive information of the training set. Several kinds of attacks

Authors' Contact Information: Anan Kabaha, Technion, Haifa, Israel, anan.kabaha@campus.technion.ac.il; Dana Drachler Cohen, Technion, Haifa, Israel, ddana@ee.technion.ac.il.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

© 2025 Copyright held by the owner/author(s). Publication rights licensed to ACM.

ACM 2475-1421/2025/4-ART122
<https://doi.org/10.1145/3720480>

demonstrate this vulnerability: membership inference attacks [Choquette-Choo et al. 2021; Ko et al. 2023; Shokri et al. 2017; Song et al. 2019], reconstruction attacks [Balle et al. 2022; Geiping et al. 2020; Zhu et al. 2019], and model extraction attacks [Hu and Pang 2021; Liang et al. 2024; Xie et al. 2024]. Some of these attacks can succeed even in the most challenging scenario, where the attacker has only label-only access to the network [Choquette-Choo et al. 2021; Kahla et al. 2022; Li and Zhang 2021]. This label-only access setting is common in machine-learning-as-a-service platforms (e.g., Google’s Vertex AI [Google Cloud 2024] or Amazon’s ML on AWS [Amazon Web Services 2024]). Thus, it is essential to develop privacy-preserving mechanisms.

There are different approaches for mitigating privacy leakage, including privacy-aware searches for a network architecture [Cheng et al. 2022; Wang et al. 2022c], integration of regularization terms during training to prevent overfitting [Hayes et al. 2019; Melis et al. 2019], or federated learning over local datasets [Chen et al. 2024; Xu et al. 2022]. However, these approaches typically do not have a formal privacy guarantee. One of the most popular formal guarantees for privacy is differential privacy (DP) [Dwork 2006]. An algorithm taking as input a dataset is DP if, for any dataset, the presence or absence of a single data point does not significantly change its output. Several training algorithms add noise to guarantee DP [Abadi et al. 2016; Chamikara et al. 2020; Chen et al. 2020; Esmaili et al. 2021; Ghazi et al. 2021, 2023; Wang et al. 2020; Ye and Shokri 2022]. Many of them rely on *DP-SGD* [Abadi et al. 2016; Esmaili et al. 2021; Ghazi et al. 2021, 2023], where noise is added during training to the parameters’ gradients. However, they have several disadvantages. Their main disadvantage is that to satisfy DP, they add noise calibrated to protect the privacy of *any* training set. To this end, they add noise protecting the *maximal* privacy leakage, which tends to be high and thus decreases the resulting network’s accuracy. In many scenarios, network designers care about protecting the privacy of a *given* dataset. For example, a network designer wishing to protect the privacy of a patients’ dataset may not care that the training algorithm protects the privacy of other datasets (e.g., image datasets). However, even if they do not care about these datasets, ensuring DP forces the training algorithm to add significantly higher noise and, consequently, the accuracy of the network that the designer cares about can be substantially reduced. Another disadvantage is that DP-SGD algorithms add noise to a large number of computations, thus the amount of added noise is often higher than necessary. This is because the noise is a function of the user-defined allowed privacy leakage and the algorithm’s actual privacy leakage which is commonly overapproximated (e.g., using composition theorems [Bagdasaryan et al. 2019; Kairouz et al. 2015; Ye and Shokri 2022]). Also, DP-SGD algorithms are specific algorithms and cannot easily be combined with other training techniques to leverage parallel advances in training. This raises the question: *can we provide privacy guarantees for individuals in a given dataset used by a training algorithm while introducing a minor accuracy decrease to the label predictions of the resulting network?*

We propose label-only access to a network that guarantees *Individual Differential Privacy* (iDP) [Soria-Comas et al. 2017; Wang 2019] with a minor accuracy decrease. iDP enforces the DP requirement only for individuals in a *given* dataset. Thanks to this relaxation, it enables the algorithm to introduce a lower noise and thus reduce the accuracy decrease. Although several other relaxations of DP have been proposed in order to reduce its accuracy decrease, they target specialized settings, which pose additional requirements. For example, individualized privacy assignment [Boenisch et al. 2023a,b] requires to assign different privacy budgets to every individual in the training set, depending on how much the individual wishes to protect their privacy, which is not always given or easy to quantify. Local differential privacy [Bhaila et al. 2024; Chamikara et al. 2020] requires to add randomization to the individuals’ data before being released to form the training set, which assumes that individuals’ data is collected by a certain algorithm. Renyi differential privacy [Jiang et al. 2023; Mironov 2017] changes the DP requirement based on the Renyi divergence. Homomorphic-encryption approaches [Ogilvie 2024; Sébert 2023] require to

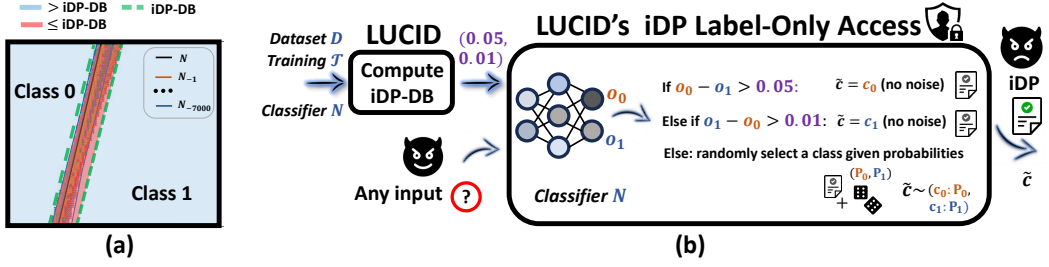


Fig. 1. (a) Illustration of the subspaces of inputs with confidence over or below the iDP-DB on a 2D synthetic training set comprising 7,000 data points. (b) An overview of LUCID: given a dataset D , a training algorithm $\mathcal{T}$, and a classifier N , it computes the iDP-DB of every label (in this example, $\text{iDP-DB}_{c_0} = 0.05$, $\text{iDP-DB}_{c_1} = 0.01$) and returns iDP label-only access to N . Given an input x , this access computes $N(x)$ to identify the predicted label c and the confidence $\mathcal{C}$ in c . If $\mathcal{C} > \text{iDP-DB}_c$, it returns c ; otherwise, it employs a noise mechanism. In this example, if a user submits to the iDP label-only access x where $N(x) = (0.6, 0.4)$, then $c = c_0$ and $\mathcal{C} = 0.6 - 0.4 > 0.05$ and thus c_0 is returned. If $N(x) = (0.52, 0.48)$, then $c = c_0$ and $\mathcal{C} = 0.52 - 0.48 \leq 0.05$ and thus a noise mechanism is used to select a class $\tilde{c} \in \{c_0, c_1\}$ given a probability distribution (P_0, P_1) .

encrypt the data. In contrast, iDP assumes the exact setting as DP, except that it limits the guarantee to the individuals in a given dataset. Additionally, iDP has similar properties to DP (e.g., parallel and sequential composition) and common DP noise mechanisms naturally extend to iDP. To the best of our knowledge, we are the first to enforce iDP for neural networks.

To naively obtain iDP label-only access to a network, we could add noise that depends on the maximum privacy leakage of any possible input. However, this would significantly decrease the accuracy. Instead, we observe that not all inputs to the network lead to an iDP privacy leakage. By identifying these inputs, we can avoid adding noise to their predicted labels and thus avoid unnecessary decrease in accuracy. However, determining whether an input leads to an iDP privacy leakage is challenging since it requires to obtain its classification as determined by a large number of networks: the network trained on the full training set and every network trained on the full training set except for a unique data point. Instead, we overapproximate the set of privacy leaking inputs with the *individual differential privacy deterministic bound* (iDP-DB). The iDP-DB is the maximal classification confidence of inputs that violate iDP. Any input that the network classifies with confidence over this bound does not lead to an iDP violation. Namely, such inputs are classified the same regardless if the classifier is trained on the full training set or if any data point is omitted. Given this bound, we can design iDP label-only access to the network by adding noise only to the predicted labels of inputs that the network classifies with confidence smaller or equal to the iDP-DB. Figure 1(a) visualizes the iDP-DB. In this example, we consider a binary network classifier trained on a 2D synthetic training set comprising 7,000 data points. The figure shows the decision boundary of the classifier and the decision boundaries of every classifier trained identically except that its training set excludes a unique data point (the colored lines in Figure 1(a)). Label-only access to this classifier is iDP if all classifiers classify every input the same. However, in practice, these classifiers are different. Namely, their decision boundaries are close but not identical. Because the decision boundaries are close, there is a large subspace of inputs that are classified the same by all classifiers (shown by the light blue background). These inputs' classification confidence is higher than the iDP-DB (shown by the dashed green lines), and for them, iDP holds *without added noise*.

Building on this idea, we design LUCID (for **L**abel-**G**uarded Classifier by **I**ndividual **D**ifferential Privacy), a system that computes the iDP-DB of every label and then returns iDP label-only access

that wraps a trained network. Given an input, LUCID's label-only access passes the input through the network. If the classification confidence is over the iDP-DB, it returns the predicted label. If not, it selects a label to return using a noise mechanism. Figure 1(b) visualizes LUCID. The main benefits of our iDP label-only access are: (1) it is applicable to any training algorithm, (2) it identifies inputs that cannot lead to privacy leakage and does not add noise to their predicted labels, which leads to a minor decrease in the network's accuracy, (3) its noise mechanism can be easily configured with the desired level of privacy guarantee, providing a privacy-accuracy trade-off, (4) its privacy guarantee is not tied to a specific privacy attack, and (5) it is fully automatic.

Computing the exact iDP-DB is highly challenging for several reasons. First, it is defined over a very large number of classifiers (commonly, several thousand): the classifier trained with the full training set and every classifier trained on the full training set except for a unique data point. Second, the iDP-DB is a global property requiring to determine for *any* input if it may be classified differently by any of these classifiers and thus violate iDP. Third, it requires to determine the *maximal* classification confidence of any input that violates iDP.

To compute the tightest iDP-DB, LUCID relies on several formal verification techniques. First, it encodes the problem as a mixed-integer linear program (MILP), defined over the network and every network trained on the same training set except for a unique data point. Second, to scale the analysis over the immense number of networks, LUCID abstracts a set of networks using a *hyper-network*. It then restates the MILP to compute the iDP-DB of the network and a hyper-network. This enables LUCID a simultaneous analysis of multiple, closely related networks. Third, to eliminate the abstraction's overapproximation error, LUCID relies on a novel branch-and-bound (BaB) technique. Our BaB branches by refining a hyper-network into several hyper-networks, each abstracting close networks that are identified by clustering. Our BaB bounds by identifying hyper-networks whose iDP-DB can bound the iDP-DB of other hyper-networks. To keep the number of analyzed hyper-networks minimal, LUCID further orders the branching of hyper-networks to increase the chances of bounding as many hyper-networks as possible. Fourth, to prune the MILP's search space, LUCID bounds the differences of matching neurons in the network and the hyper-network and adds them as linear constraints. Fifth, neurons in the hyper-network whose differences are small are overapproximated by linear relaxation, to reduce the MILP's complexity. While this step introduces an overapproximation error, it is small in practice and it is configurable.

We evaluate LUCID over four data-sensitive datasets (Cryptojacking, Twitter Spam Accounts, Adult Census, and Default of Credit Card Clients) and over fully-connected and convolutional classifiers. We compare it to two DP-SGD techniques: the original one [Abadi et al. 2016], providing a DP guarantee for all the classifier's parameters, and ALIBI [Esmaeili et al. 2021], providing a DP guarantee for the classifier's output. Our results indicate that thanks to our formal verification analysis, LUCID can provide a perfect iDP guarantee ($\epsilon = 0$) for label-only access with 1.4% accuracy decrease on average. For more relaxed iDP guarantees (higher ϵ), LUCID provides iDP label-only access with 1.2% accuracy decrease on average. In contrast, the DP baselines decrease the accuracy on average by 12.7% to obtain an ϵ -DP guarantee, which in particular provides ϵ -iDP guarantee.

In summary, our main contributions are: (1) formalizing iDP for label-only access to networks, enabling a minor accuracy decrease, (2) defining the iDP-DB that overapproximates inputs violating iDP, (3) designing a system called LUCID for computing the iDP-DB, relying on several formal verification techniques to scale (constraint solving, hyper-networks, branch-and-bound, pruning by bounding the differences of matching neurons, and linear relaxation) (4) an extensive evaluation showing that LUCID provides iDP label-only access with less than 1.4% accuracy decrease.

2 Preliminaries

This section provides background on neural network classifiers and individual differential privacy.

Neural network classifiers. A classifier maps an input into a score vector over a set of labels (also called classes) C , i.e., it is a function: $N : [0, 1]^d \rightarrow \mathbb{R}^{|C|}$. Given an input x , its classification is the label with the maximal score: $c = \operatorname{argmax}(N(x))$. We focus on classifiers implemented as neural networks. A neural network consists of an input layer followed by L layers, where the last layer is the output layer. Layers consist of neurons, each connected to some or all neurons in the next layer. A neuron computes a function, which is typically a linear combination of its input neurons followed by a non-linear activation function. The output layer consists of $|C|$ neurons, each outputs the score of one of the labels. We denote by $z_{m,k}$ the k^{th} neuron in layer m , for $m \in \{0, \dots, L\}$ (where 0 denotes the input layer) and $k \in [k_m] = \{1, \dots, k_m\}$. The input layer z_0 passes the input x to the next layer (i.e., $z_{0,k} = x_k$ for all $k \in [d]$). The neurons compute a function determined by the layer type. Our definitions focus on fully-connected layers, but our implementation also supports convolutional layers and it is easily extensible to other layers, such as max-pooling layers [LeCun et al. 1998] and residual layers [He et al. 2016]. In a fully-connected layer, a neuron $z_{m,k}$ gets as input the outputs of all neurons in the preceding layer. It has a weight for each input $w_{m,k,k'}$ and a bias $b_{m,k}$. Its function is the weighted sum of its inputs and bias followed by an activation function. We focus on the ReLU activation function. Formally, the function of a non-input neuron is $z_{m,k} = \operatorname{ReLU}(\hat{z}_{m,k}) = \max(0, \hat{z}_{m,k})$, where $\hat{z}_{m,k}$ is the weighted sum: $\hat{z}_{m,k} = b_{m,k} + \sum_{k'=1}^{k_{m-1}} w_{m,k,k'} \cdot z_{m-1,k'}$. The network's parameters – the weights and biases – are determined by a training algorithm. We focus on supervised learning, where the training algorithm $\mathcal{T}$ is provided with an initial network $\tilde{N}$ (e.g., with random parameter values), called the network architecture, and a dataset $D \subseteq [0, 1]^d \times C$ of labeled data points (x_D, y_D) . In the following, we abbreviate a pair (x_D, y_D) by x_D . The exact computation of the training algorithm $\mathcal{T}$ is irrelevant to our approach. In particular, it may run a long series of iterations, each consisting of a large number of computations (e.g., over the parameters' gradients). The important point in our context is that given a network architecture $\tilde{N}$ and a random seed (in case $\mathcal{T}$ has randomized choices), the training algorithm is a deterministic function mapping a training set D into a neural network.

Differential privacy. Differential privacy (DP) [Dwork 2006] is a popular privacy property designed to protect the privacy of individual data points in a dataset, which is being accessed by one or more functions f . The original DP definition assumes that f is invoked interactively by the user. To avoid privacy leakage, f is protected by a mechanism $\mathcal{A}$, which is a randomized version of f adding random noise to its computations. The added noise ensures that the inclusion or exclusion of any single data point in the dataset does not significantly affect the function's output. Formally:

Definition 2.1 (Differential Privacy). A randomized mechanism $\mathcal{A}$, over a domain of datasets $\mathcal{D}$, is ϵ -differentially private if for any two adjacent datasets $D, D' \in \mathcal{D}$ (differing by one data point), for all sets of observed outputs $\mathcal{O} \subseteq \operatorname{Range}(\mathcal{A})$ we have: $\Pr[\mathcal{A}(D) \in \mathcal{O}] \leq e^\epsilon \cdot \Pr[\mathcal{A}(D') \in \mathcal{O}]$.

In this definition, ϵ is a small non-negative number, bounding the privacy loss. A common approach to create a randomized version of f is by adding noise that depends on f 's *global sensitivity* $S(f)$ [Abadi et al. 2016; Dwork 2006; Esmaeili et al. 2021; Ghazi et al. 2021, 2023]. The global sensitivity is the maximum difference of f over any two adjacent datasets in $\mathcal{D}$. Formally:

Definition 2.2 (Global Sensitivity). The global sensitivity of a function $f : \mathcal{D} \rightarrow \mathbb{R}^n$ is $S(f) = \max_{D, D' \in \mathcal{D}, |D - D'| = 1} \|f(D) - f(D')\|$.

Differential privacy and global sensitivity are defined with respect to any two adjacent datasets over the dataset domain $\mathcal{D}$. While such definitions are useful in many real-world scenarios, in the context of machine learning, it may be too restrictive. While there may be scenarios where network designers wish to design a DP training algorithm, protecting the privacy of any dataset, in other

scenarios, network designers may only care about protecting the privacy of data points in a *given* dataset. For example, a network designer that has collected private information of patients towards designing a network predicting whether a new person has a certain disease may not care whether their training algorithm does not leak private information of the MNIST image dataset [LeCun et al. 1998] (or any other dataset for this matter). However, even if they do not care about these datasets, ensuring DP may force them to add a significantly higher noise because these irrelevant datasets may increase the global sensitivity of the training algorithm. Consequently, the accuracy of the network that the designer cares about can be substantially reduced. In fact, as we demonstrate in Section 7, in some cases, the network's classification accuracy can be reduced to that of a random classifier. In such scenarios, it may be better to relax DP and ensure the privacy of data points in a *given* dataset. This property is called *individual differential privacy*.

Individual differential privacy. Individual Differential Privacy (iDP) [Soria-Comas et al. 2017] is a relaxation of DP, designed to protect the privacy of individual data points in a *given* dataset. Similarly to DP, a randomized mechanism satisfies iDP if its output is not significantly affected by the inclusion or exclusion of any single data point in the given dataset. Formally:

Definition 2.3 (Individual Differential Privacy). Given a dataset $D \in \mathcal{D}$, a randomized mechanism $\mathcal{A}$, over a domain of datasets $\mathcal{D}$, is ϵ -individually differentially private if for any dataset D' adjacent to D (differing by one data point), for all sets of observed outputs $\mathcal{O} \subseteq \text{Range}(\mathcal{A})$ we have: $e^{-\epsilon} \cdot \Pr[\mathcal{A}(D') \in \mathcal{O}] \leq \Pr[\mathcal{A}(D) \in \mathcal{O}] \leq e^{\epsilon} \cdot \Pr[\mathcal{A}(D') \in \mathcal{O}]$.

Similarly to DP, a randomized mechanism for f can be obtained by adding noise that depends on f 's *local sensitivity* $S_L(f)$. The local sensitivity is the maximum difference of f with respect to a given dataset D and any of its adjacent datasets. Formally:

Definition 2.4 (Local Sensitivity). Given a dataset $D \in \mathcal{D}$, the local sensitivity of a function $f : \mathcal{D} \rightarrow \mathbb{R}^n$ is $S_L(f) = \max_{D' \in \mathcal{D}, |D-D'|=1} \|f(D) - f(D')\|$.

iDP has been shown to have several important properties that hold for DP, including parallel and sequential composition [Soria-Comas et al. 2017]. Additionally, several DP noise mechanisms naturally extend to iDP by replacing the global sensitivity with the local sensitivity (e.g., the Laplace noise mechanism [Soria-Comas et al. 2017]).

3 Problem Definition: Guarding Label-Only Access to Networks

In this section, we define the problem of protecting label-only access to neural network classifiers using individual differential privacy. We begin by defining the functions that may leak private information in a label-only setting. We then define the problem of creating a randomized mechanism for them that introduces a minor accuracy decrease. Lastly, we describe two naive approaches.

3.1 Label-Only Queries

In this section, we formalize the functions that access the dataset (in our setting, it is also called the training set), thus may leak private information, in our label-only access setting.

Label-only access to a neural network N maps an input $x \in [0, 1]^d$ to a label $c \in \mathcal{C}$, where N is the network trained by a training algorithm $\mathcal{T}$ given a dataset D and an architecture $\bar{N}$. One might consider the training algorithm as the function that leaks private information, since it accesses the dataset to compute the network. However, this definition is too coarse for our label-only access setting, where users (including attackers) indirectly access the dataset by obtaining the network's predicted label of any input and do not have access to the network's internal parameters. Further, making the training algorithm private requires adding random noise to its computations,

regardless of the inputs that are later used for querying the network. Our formalization removes this separation of the network's training and its querying. It defines the functions that access the dataset as an infinite set of functions, one for each possible input $x \in [0, 1]^d$ (not only inputs in the dataset). This refined definition enables us later to add noise only to the functions that potentially leak private information. Formally, given a training algorithm $\mathcal{T}$ and an architecture $\tilde{N}$, the set of *label-only queries* is: $\mathcal{F}_{\mathcal{T}, \tilde{N}} = \{f_{x, \mathcal{T}, \tilde{N}} \mid x \in [0, 1]^d\}$, where for every $x \in [0, 1]^d$ the function $f_{x, \mathcal{T}, \tilde{N}}(D)$ maps a dataset D to the label predicted for x by the classifier trained by $\mathcal{T}$ on $\tilde{N}$ and D . To illustrate, consider stochastic gradient descent as the training algorithm $\mathcal{T}_{\text{SGD}}$ and as a network architecture $\tilde{N}_{3,5,2}$ a fully-connected network with three layers, consisting of three neurons, five neurons, and two neurons, respectively. Then, for example the function $f_{(1,0.2,0.1)} \in \mathcal{F}_{\mathcal{T}_{\text{SGD}}, \tilde{N}_{3,5,2}}$ maps a dataset D to one of the two labels by (1) training the network $N = \mathcal{T}_{\text{SGD}}(D, \tilde{N}_{3,5,2})$, (2) computing $N((1, 0.2, 0.1))$ and (3) returning the label with the maximal score $c = \text{argmax}(N(1, 0.2, 0.1))$.

Our definition may raise two questions: (1) is it equivalent to the common practice where a neural network is computed once and then queried? and (2) is it useful in practice if it (supposedly) requires to retrain the network from scratch upon every new input? The answer to both questions is yes. First, it is equivalent because given a dataset D and an architecture $\tilde{N}$, the training algorithm $\mathcal{T}$ is *deterministic*. We note that if a training algorithm $\mathcal{T}$ makes random choices, we assume we are given its random seed, which makes $\mathcal{T}$ a deterministic function. Namely, every function $f_{x, \mathcal{T}, \tilde{N}}(D)$ computes the exact same network given $\mathcal{T}$, D , and $\tilde{N}$ and thus our definition is equivalent to the common practice. Second, in practice, we do not retrain the network from scratch for every input, we define these functions only for the sake of mathematically characterizing the functions that may leak private information, but eventually the network is trained once (as we explain later).

3.2 Problem Definition: iDP Label-Only Queries with a Minor Accuracy Decrease

In this section, we present our problem of creating a randomized mechanism for our label-only queries, while minimally decreasing their accuracy.

As explained in Section 2, creating a randomized mechanism for our label-only queries is possible by adding noise determined by the local sensitivity. We next define the local sensitivity of a label-only query. As defined in Definition 2.4, given a function $f_{x, \mathcal{T}, \tilde{N}}$ and a dataset D , the local sensitivity is the maximum difference of $f_{x, \mathcal{T}, \tilde{N}}$ with respect to D and any of its adjacent datasets. Since a label-only query returns a label, we define the difference over their one-hot encoding. That is, if $f_{x, \mathcal{T}, \tilde{N}}(D) = c$, then the one-hot encoding is a vector of dimension $|C|$ where the entry of c is one and the other entries are zeros. By this definition, the local sensitivity is zero if for *every* adjacent dataset D' , the same label is returned: $f_{x, \mathcal{T}, \tilde{N}}(D) = f_{x, \mathcal{T}, \tilde{N}}(D')$. In this case, $f_{x, \mathcal{T}, \tilde{N}}$ is 0-iDP. It is not zero if there exists an adjacent dataset D' that returns a different label: $f_{x, \mathcal{T}, \tilde{N}}(D) \neq f_{x, \mathcal{T}, \tilde{N}}(D')$.

To illustrate the label-only queries with sensitivity zero, consider Figure 1(a). As described, in this example, we focus on a binary network classifier trained on a 2D synthetic training set of size 7,000. The figure shows the decision boundaries of this classifier along with the decision boundaries of the networks trained on its adjacent datasets. The figure shows that the decision boundaries are not identical but close. Thus, many inputs are classified the same by all classifiers (shown in light blue background). The respective label-only queries of these inputs have local sensitivity of zero. The other inputs are classified differently by at least one network trained on an adjacent dataset. Namely, the local sensitivity of their respective label-only queries is not zero.

While we could design an iDP randomized version of the label-only queries by adding noise that assumes the maximum local sensitivity, it would add unnecessary noise to queries with local sensitivity zero and unnecessarily decrease their accuracy. For example, in Figure 1(a) the local

sensitivity of most label-only queries is zero, and thus these queries are 0-iDP without any noise addition. By adding noise only to queries with non-zero local sensitivity, we can obtain an iDP label-only access to the network with overall a minor decrease in its accuracy. This is our problem:

Definition 3.1 (Problem Definition: Minimal Noise iDP Label-Only Queries). Given a dataset D , a training algorithm $\mathcal{T}$, a network architecture $\tilde{N}$, and a privacy budget ε , our goal is to compute for every label-only query $f_{x,\mathcal{T},\tilde{N}}$, for $x \in [0, 1]^d$, the minimal noise required to make it ε -iDP.

Note that we have access to D , $\mathcal{T}$, and $\tilde{N}$; only the users have label-only access to the trained network (as common in machine-learning-as-a-service platforms). In the following, for brevity, we say that an input x satisfies/violates iDP if its label-only query $f_{x,\mathcal{T},\tilde{N}}(D)$ satisfies/violates iDP. Similarly, the local sensitivity of x is the local sensitivity of its label-only query $f_{x,\mathcal{T},\tilde{N}}(D)$. We call an input x whose local sensitivity is not zero a *leaking input*, since its label-only query is not 0-iDP.

3.3 Naive Algorithms

In this section, we describe two naive algorithms that solve our problem but are inefficient: either in terms of accuracy or in terms of time and space.

The naive noise algorithm. The first naive algorithm invokes the training algorithm $\mathcal{T}$ on the given dataset D and the network architecture $\tilde{N}$ to compute N . Its label-only access is defined as follows. Given an input $x \in [0, 1]^d$, it computes $f_{x,\mathcal{T},\tilde{N}}(D)$ by passing x through N . Then, it ensures ε -iDP by employing the *exponential mechanism* with the maximum local sensitivity. We next provide background on the exponential mechanism and explain how this algorithm uses it.

The exponential mechanism is a popular approach to provide DP guarantees for non-numerical algorithms, such as classification algorithms [Dwork 2006; Gursoy et al. 2017; Ju et al. 2024]. In these scenarios, typically, the output range is discrete and denoted $\mathcal{R}$. The exponential mechanism defines the *global* sensitivity with respect to a given *utility* function $u : \mathcal{D} \times \mathcal{R} \rightarrow \mathbb{R}$, mapping a dataset and an output to a score. The global sensitivity is: $\Delta u = \max_{r \in \mathcal{R}} \max_{D, D'.s.t. ||D-D'|| \leq 1} |u(D, r) - u(D', r)|$. To ensure ε -DP, the exponential mechanism returns an output $r \in \mathcal{R}$, with a probability proportional to $\exp(\varepsilon \cdot u(D, r) / (2\Delta u))$ (we refer to this process as adding noise to the output r).

Given a query $f_{x,\mathcal{T},\tilde{N}} : \mathcal{D} \rightarrow \mathcal{C}$ (mapping a dataset to a class), we define a utility function $u_{x,\mathcal{T},\tilde{N}} : \mathcal{D} \times \mathcal{C} \rightarrow \{0, 1\}$ (we sometimes abbreviate by u to simplify notation). The utility function u maps a dataset D and a class c to 1 if $f_{x,\mathcal{T},\tilde{N}}(D) = c$ and to 0 otherwise. For example, if $f_{x,\mathcal{T},\tilde{N}}(D) = c_0$, then $u_{x,\mathcal{T},\tilde{N}}(D, c_0) = 1$ and $u_{x,\mathcal{T},\tilde{N}}(D, c) = 0$, for $c \in \mathcal{C} \setminus \{c_0\}$. Given a dataset D , the local sensitivity of $f_{x,\mathcal{T},\tilde{N}}$ is $\Delta u = \max_{c \in \mathcal{C}} \max_{D'.s.t. ||D-D'|| \leq 1} |u(D, c) - u(D', c)|$. Note that $\Delta u \in \{0, 1\}$. If our algorithms identify $\Delta u = 0$, they do not invoke the exponential mechanism. That is, they always invoke it with $\Delta u = 1$. Thus, our exponential mechanism returns a class $c \in \mathcal{C}$ with a probability proportional to $\exp(\varepsilon \cdot u(D, c) / 2)$. Our next lemma states that this mechanism guarantees ε -iDP:

LEMMA 3.2. *Given a dataset D , a query $f_{x,\mathcal{T},\tilde{N}}$, and a privacy budget ε , the exponential mechanism with our utility function $u_{x,\mathcal{T},\tilde{N}}$ is ε -iDP.*

The proof (Appendix A) is similar to the proof showing that this mechanism is DP (with respect to the global sensitivity). This lemma implies that the naive noise algorithm solves Definition 3.1. However, it introduces a very high noise since for any input it invokes the exponential mechanism with the maximum local sensitivity. As we show in Section 7, its accuracy decrease is even higher than DP training algorithms. In fact, in this case, the maximum local sensitivity is equal to the maximum global sensitivity, thus the naive noise algorithm is DP.

The naive iDP algorithm. This naive algorithm improves the accuracy of the previous algorithm by identifying the local sensitivity of each query, instead of bounding it by the maximum local

sensitivity. It begins by training the set of classifiers for D and all its adjacent datasets. That is, it computes $\mathcal{N} = \{N\} \cup \{N_{-x_D} \mid x_D \in D\}$, where $N = \mathcal{T}(D, \tilde{N})$ and $N_{-x_D} = \mathcal{T}(D \setminus \{x_D\}, \tilde{N})$. Its label-only access is defined as follows. Given an input $x \in [0, 1]^d$, to compute $f_{x, \mathcal{T}, \tilde{N}}(D)$ it passes x through all networks in $\mathcal{N}$. If all networks return the same label, it returns the label as is. Otherwise, it employs the exponential mechanism, exactly as the naive noise algorithm does. Namely, it employs the exponential mechanism only for inputs whose local sensitivity is not zero and thus obtains ϵ -iDP with minimum accuracy decrease. However, as we show in Section 7, its inference time is high, because every input passes through $|D| + 1$ classifiers. Also it poses a significant space overhead, because it requires to store in the memory all $|D| + 1$ classifiers throughout its execution.

4 Insight: The Individual Differential Privacy Deterministic Bound

In this section, we present our main insight: identifying inputs with local sensitivity zero using the *network's individual differential privacy deterministic bound* (iDP-DB). The iDP-DB overapproximates the leaking inputs (whose local sensitivity is not zero) using the network's classification confidence. Every input that the network classifies with confidence over this bound satisfies 0-iDP. We next provide the definitions, illustrate the iDP-DB, and discuss the challenges in computing it.

Leaking inputs. The set of leaking inputs can be defined by the decision boundaries of all classifiers in $\{N\} \cup \{N_{-x_D} \mid x_D \in D\}$, where N is the classifier trained on the dataset D and N_{-x_D} is the classifier trained on $D \setminus \{x_D\}$. The decision boundaries of a classifier partition the input space into subspaces of inputs classified to the same class. The leaking inputs are the inputs that for N are contained in a subspace labeled as some class and for some N_{-x_D} , where $x_D \in D$, are contained in a subspace labeled as another class. We formalize this definition using *classification confidences*. Given a classifier N , an input x , and a label c , the classification confidence is the difference between the score of c and the highest score of the other classes: $\mathcal{C}_N^c(x) = N(x)_c - \max_{c' \neq c} N(x)_{c'}$. If $\mathcal{C}_N^c(x)$ is positive, then N classifies x as c . Otherwise, N does not classify x as c . Given this notation, the set of leaking inputs of class c contains all inputs classified by N as c and by one of the other classifiers not as c : $\{x \in [0, 1]^d \mid \mathcal{C}_N^c(x) > 0 \wedge \bigvee_{x_D \in D} \mathcal{C}_{N_{-x_D}}^c(x) \leq 0\}$. Computing this set requires computing the decision boundaries, which is highly complex. Instead, we overapproximate this set.

The iDP-DB. We overapproximate the set of leaking inputs by the maximal classification confidence β^* of any leaking input. That is, any input whose classification confidence is higher than β^* satisfies 0-iDP *without added noise*. Our overapproximation is sound: it does not miss any leaking input. However, it may be imprecise: there may be inputs that are not leaking whose classification confidence is smaller or equal to β^* . Importantly, adding noise based on our overapproximation enables LUCID to satisfy iDP: every leaking input will be noised. Due to the overapproximation, LUCID may add noise to inputs that are not leaking, however, as we show in Section 7, this approach enables LUCID to obtain label-only access to a network with a small accuracy decrease. We call β^* the *individual differential privacy deterministic bound*, since inputs whose confidence is above β^* deterministically satisfy iDP, without adding random noise. We next formally define it.

Definition 4.1 (Individual Differential Privacy Deterministic Bound (iDP-DB)). Given a dataset D , a training algorithm $\mathcal{T}$, and a network architecture $\tilde{N}$, the iDP-DB is:

$$\beta^* = \operatorname{argmax}_{\beta} \exists x \in [0, 1]^d \exists c \in C \left(\mathcal{C}_N^c(x) \geq \beta \wedge \bigvee_{x_D \in D} \mathcal{C}_{N_{-x_D}}^c(x) \leq 0 \right)$$

where $N = \mathcal{T}(D, \tilde{N})$ and $N_{-x_D} = \mathcal{T}(D \setminus \{x_D\}, \tilde{N})$, for $x_D \in D$.

We note the following. First, the iDP-DB exists for every network, since there is a maximum confidence for every label c . In the worst case scenario, this bound is equal to the maximum confidence over all labels, in which case the set of leaking inputs is overapproximated by all inputs. Consequently, every input will be noised by LUCID. In the (highly unlikely) best case scenario, this bound is zero, indicating that all classifiers classify the same every input. Second, every higher classification confidence $\beta > \beta^*$ also overapproximates the set of leaking inputs. However, it adds a higher than necessary overapproximation error, which would lead LUCID to add unnecessary noise.

To reduce the overapproximation error, we define the iDP-DB for every class $c \in C$:

$$\beta_c^* = \operatorname{argmax}_{\beta} \exists x \in [0, 1]^d \left(\mathbb{C}_N^c(x) \geq \beta \wedge \bigvee_{x_D \in D} \mathbb{C}_{N-x_D}^c(x) \leq 0 \right) \quad (1)$$

Computing the iDP-DB for every class c enables LUCID to lower the decrease in the network's accuracy: a joint bound for all classes $\beta^* = \max_c \beta_c^*$ would lead LUCID to add unnecessary noise to inputs that the network classifies as c with confidence in $(\beta_c^*, \beta^*]$.

Illustration. We illustrate the iDP-DB on the classifier considered in Figure 1(a), trained on a 2D synthetic training set comprising 7,000 data points. A labeled data point (x_D, y_D) is $x_D = (x_1, x_2) \in [0, 1]^2$ and $y_D \in \{0, 1\}$. Its iDP-DB-s are $\beta_0^* = 6.6$ and $\beta_1^* = 15$ and thus $\beta^* = 15$. Our idea is to make label-only access to a network N iDP by adding noise to every input x whose confidence is at most β_c^* , where c is the class N predicts for x . To visualize the set of inputs overapproximated by the iDP-DB (which will be noised by LUCID), we consider an extended definition of the classification confidence: $\mathbb{C}_N^{c,\beta}(x) = \max(0, \mathbb{C}_N^c(x) - \beta)$. That is, classification confidence up to β is set to zero so inputs classified with such confidence are on the extended decision boundary. Figure 2 visualizes the extended classification confidence for $\beta \in \{0, \beta_0^*, \beta_1^*, 30\}$. For each β , it shows the extended classification confidence as a function of the input (x_1, x_2) , along with the decision boundaries of the 7,000 classifiers in $\{N_{-x_D} \mid x_D \in D\}$ (in colored lines). For $\beta = 0$, the decision boundary of N does not cover all the other classifiers' decision boundaries. Namely, there are leaking inputs not on the extended boundary and thus $\beta = 0$ is an underapproximation of the set of leaking inputs. Adding noise only to these inputs will not make the label-only access to the network iDP. For $\beta = \beta_0^*$, all the leaking inputs classified as 0 are on the extended decision boundary, but there are leaking inputs classified as 1 not on it. For β_1^* , the extended decision boundary covers all decision boundaries (i.e., it covers all classification differences) and thus β_1^* overapproximates the set of leaking inputs. It is also the minimal bound covering all decision boundaries. Adding noise to these inputs will make the label-only access to the network iDP (in fact, for inputs that the network classifies as 0, adding noise is required only if their confidence is at most β_0^*). For $\beta = 30$, all decision boundaries are covered by the extended decision boundary and it also provides an overapproximation of the leaking inputs. While adding noise to all these inputs will make the label-only access to the network iDP, it results in an unnecessary decrease in accuracy. Conceptually, our iDP-DB partitions the input space into two parts: a subspace of inputs that deterministically satisfy iDP (if the attacker only queries them, the label-only access is 0-iDP) and a subspace that overapproximates the leaking inputs. To ensure iDP in this subspace, we employ the exponential mechanism.

Challenges. Computing the iDP-DB is very challenging. First, it is a global property, requiring to analyze the network's computation for *every* input (not confined to a particular dataset). This analysis has shown to be highly complex [Katz et al. 2017, 2019], as it involves precise modeling of a network classifier (a highly non-convex function) over a very large space of inputs. Second, iDP-DB is defined over a very large number of classifiers, $|D| + 1$, where D is the dataset, which tends to be large (several thousand). This amplifies the complexity of the previous challenge. Third, iDP-DB

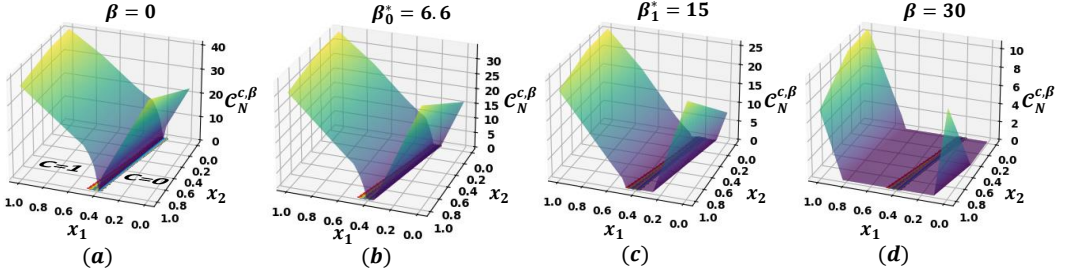


Fig. 2. The conceptually extended decision boundaries as a function of the input $(x_1, x_2) \in [0, 1]^2$, for different confidences, including the iDP-DB.

is the *maximal* bound β satisfying $\exists x \in [0, 1]^d \exists c \in C \left(\mathcal{C}_N^c(x) \geq \beta \wedge \forall x_D \in D. \mathcal{C}_{N-x_D}^c(x) \leq 0 \right)$, where β is a real number, which increases the problem's complexity.

5 Overview on Computing the iDP-DB

In this section, we present our ideas to efficiently compute the iDP-DB. First, we formalize iDP-DB as a constrained problem (MILP), which can be solved by existing solvers but its complexity is very high. To cope, we abstract the analyzed networks using a *hyper-network*. To mitigate the abstraction's overapproximation error, we introduce a branch-and-bound technique that refines a hyper-network into multiple hyper-networks, each abstracts a disjoint set of networks. To further reduce the complexity, we bound the differences of matching neurons in a network and a hyper-network and add them as linear constraints or, if they are small, employ linear relaxation.

5.1 Encoding iDP-DB as a MILP

We express the problem of computing the iDP-DB, which is a constrained optimization, as a mixed-integer linear program (MILP). MILP has been employed by many verifiers for solving constrained optimizations, e.g., local robustness [Müller et al. 2022; Singh et al. 2019; Tjeng et al. 2019], global robustness properties [Kabaha and Drachler-Cohen 2024; Wang et al. 2022a,b], and privacy in local neighborhoods [Reshef et al. 2024]. Compared to these verifiers, computing iDP-DB is much more complex: it is a global property (pertaining to any input) *and* over a very large number of networks (the size of the training set plus one), which is the reason we rely on additional ideas to scale the computation. An advantage of existing MILP optimizers is that they are anytime algorithms. That is, the optimizer can return at any point an interval bounding the value of the optimal solution (given enough time, this interval contains only the optimal value).

Recall that for $c \in C$, the iDP-DB is the maximal β_c satisfying $\mathcal{C}_N^c(x) \geq \beta_c \wedge \forall x_D \in D. \mathcal{C}_{N-x_D}^c(x) \leq 0$ (Equation (1)). A straightforward MILP leverages prior work for encoding the classification confidence [Tjeng et al. 2019] and encodes the disjunction (i.e., $\vee$) using the Big M method [Vanderbei 1996; Winston 1991]. However, this MILP has an exponential complexity in $|D|$ and the multiplication of the number of non-input neurons in N and $|D| + 1$. This is because a MILP's complexity is exponential in the number of boolean variables and this encoding introduces $|D|$ boolean variables for the disjunction and a unique boolean variable for every non-input neuron in every network.

A naive approach to cope with this complexity is to express the iDP-DB as $|D|$ separate MILPs $\{P_{x_D} \mid x_D \in D\}$, where P_{x_D} is $\beta_{c, \{x_D\}}^* = \arg\max_{\beta} \exists x \left(\mathcal{C}_N^c(x) \geq \beta \wedge \mathcal{C}_{N-x_D}^c(x) \leq 0 \right)$. Each MILP is submitted to the solver and the iDP-DB is the maximum: $\beta_c^* = \max_{x_D \in D} \beta_{c, \{x_D\}}^*$. The complexity of this naive approach is $|D|$ times the complexity of each MILP P_{x_D} , which is exponential in the

multiplication of the number of non-input neurons in N and two. However, the naive approach is impractical because it requires solving a large number of MILPs. Additionally, since it requires the bound of all $|D|$ problems, obtaining an anytime solution requires obtaining an anytime solution to all $|D|$ problems, which is computationally expensive.

5.2 Hyper-Networks

To define a MILP with a lower complexity, we rely on *hyper-networks*. A hyper-network abstracts a set of networks by associating the network parameters' *intervals*, defined by the minimum and maximum values over all networks. Formally:

Definition 5.1 (A Hyper-Network). Given a set of networks $\mathcal{N} = \{N_n \mid n \in [K]\}$ of the same architecture, each with weights $\mathcal{W}_n = \{w_{1,1,1}^n, \dots, w_{L,k_L,k_{L-1}}^n\}$ and biases $\mathcal{B}_n = \{b_{1,1}^n, \dots, b_{L,k_L}^n\}$, a *hyper-network* $N^\#$ is a network of the same architecture such that $w_{m,k,k'}^\# = [\min_{n \in [K]}(w_{m,k,k'}^n), \max_{n \in [K]}(w_{m,k,k'}^n)]$ and $b_{m,k}^\# = [\min_{n \in [K]}(b_{m,k}^n), \max_{n \in [K]}(b_{m,k}^n)]$.

Figure 3 shows an example of four networks $N_{-1}, \dots, N_{-4}$ and their hyper-network $N_{1,2,3,4}^\#$. For example, since the values of $z_{1,1}$'s bias in the networks are 0.1, 0, 0.1, 0.3, its interval in the hyper-network is $[0, 0.3]$. A hyper-network introduces an overapproximation error. This is because it abstracts every network whose parameters are contained in their respective hyper-network's intervals. Formally, a hyper-network $N^\#$ abstracts every network N' whose weights satisfy $w_{m,k,k'} \in w_{m,k,k'}^\#$ and biases satisfy $b_{m,k} \in b_{m,k}^\#$. For example, the hyper-network $N_{1,2,3,4}^\#$ in Figure 3 abstracts the network that is identical to N_{-1} except that the bias of $z_{1,1}$ is 0.2, even though this network is not one of the four networks used to define this hyper-network. We note that hyper-networks have been proposed by Reshef et al. [2024], however, their focus is different: they predict a hyper-network from a subset of networks.

Given a subset of the dataset $S \subseteq D$, we define the set of networks of S as $\mathcal{N}_S = \{N_{-x_D} \mid x_D \in S\}$ and denote the hyper-network abstracting the set of networks $\mathcal{N}_S$ by $N_S^\#$. The iDP-DB of S is defined as the following constrained optimization over its hyper-network $N_S^\#$:

$$\beta_{c,S}^* = \operatorname{argmax}_\beta \exists x \left(\mathcal{C}_N^c(x) \geq \beta \wedge \mathcal{C}_{N_S^\#}^c(x) \leq 0 \right) \quad (2)$$

Similarly to Section 5.1, we can encode this problem as a MILP, where the neurons' weighted sums depend on the hyper-network's intervals. By solving this MILP for $S = D$, we can overapproximate the true iDP-DB (Equation (1)) by a greater or equal bound. This follows since $\beta_c^* = \max_{x_D \in D} \beta_{c,\{x_D\}}^*$ and $\forall x_D \in D. \beta_{c,\{x_D\}}^* \leq \beta_{c,D}^*$. This MILP's complexity is exponential in $2 \cdot |N|$, where $|N|$ is the number of non-input neurons. This complexity is significantly lower than the complexity of the approaches of Section 5.1. Also, by expressing the problem as a single MILP (unlike the naive approach), we can benefit from the solver's anytime solution. However, the overapproximation error is high: $\beta_{c,D}^*$ is significantly larger than the true iDP-DB, because it considers *every* network abstracted by the hyper-network, even networks that are not used to define the hyper-network. Having a significantly larger bound than the true iDP-DB would lead LUCID to add much more noise than necessary to make the label-only access to the network iDP, and thus significantly decrease its accuracy. We next explain how to compute the true iDP-DB with hyper-networks.

5.3 A Branch-and-Bound for Hyper-Networks

In this section, we present a novel branch-and-bound technique that enables us to compute the true iDP-DB using hyper-networks.

At a high-level, a branch-and-bound (BaB) technique is defined over a verification analysis of abstract objects and it eliminates the analysis' overapproximation error. Given an abstract object, it

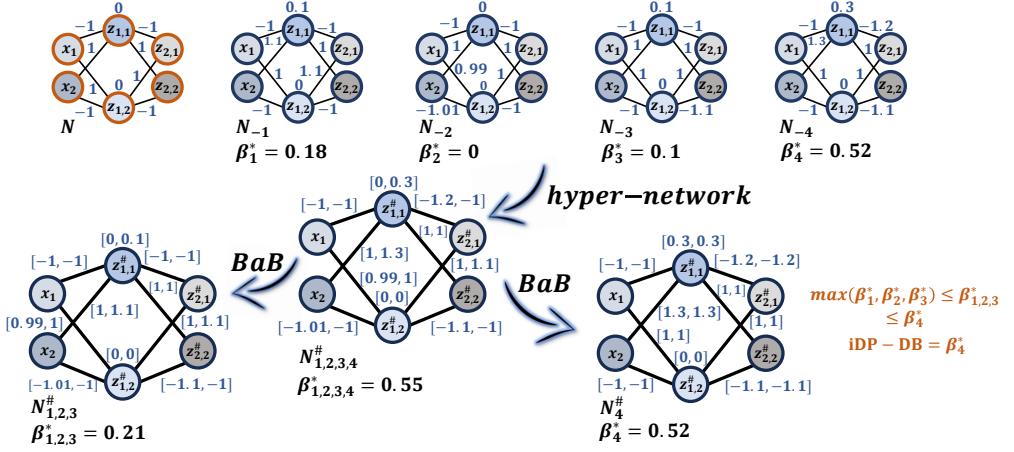


Fig. 3. An example of a hyper-network and our branch-and-bound.

runs the analysis, which returns a value. If this value is smaller or equal to a certain bound (the object is *bounded*), it terminates. If this value is greater than the bound, the object is refined to several abstract objects (the object is *branched*) and this operation continues for each abstract object. BaB terminates when there are no more objects to branch. Its main advantages are: (1) it attempts to reduce the overall execution time of the analysis by analyzing abstract objects instead of analyzing independently a large (or even infinite) set of concrete objects and (2) it does not lose precision despite of analyzing abstract objects, since if the analysis returns a value above the bound, the abstract object is refined. Thanks to these advantages, several BaB techniques have been proposed for local robustness verification [Bunel et al. 2020, 2018; Henriksen and Lomuscio 2021; Lu and Kumar 2020; Palma and et al. 2021; Wu and et al. 2020]. However, determining local robustness is simpler than computing the iDP-DB, since it requires analyzing the predicted labels of a single network for a set of inputs. In contrast, computing the iDP-DB requires analyzing the predicted labels of a *large set of networks* for *any input*.

We propose a new BaB technique to compute the precise iDP-DB. Its verification analysis takes as input a hyper-network abstracting the set of networks of a set S and it computes the iDP-DB of S (Equation (2)). For branching, our BaB refines a hyper-network $N_S^{\#}$ into K hyper-networks $\{N_{S_1}^{\#}, \dots, N_{S_K}^{\#}\}$, such that the sets $S_1, \dots, S_K$ partition S . Our BaB defines the partitioning by clustering the networks in N_S based on their parameters' similarity. The motivation for this clustering is that classifiers whose parameters have closer values are likely to have closer iDP-DB. Consequently, their hyper-network's iDP-DB is likely to have a lower overapproximation error, which increases the likelihood to succeed bounding it and avoid further branching.

For bounding, we rely on the following observation. Given a set S , if its iDP-DB $\beta_{c,S}^*$ is *smaller or equal to* $\beta_{c,\{\hat{x}_D\}}^*$ for some $\hat{x}_D \in D$, then $N_S^{\#}$ need not be branched. This is because the true iDP-DB is $\beta_c^* = \max_{x_D \in D} \beta_{c,\{x_D\}}^*$ and $\forall x_D' \in S. \beta_{c,\{x_D'\}}^* \leq \beta_{c,S}^*$. By transitivity, $\forall x_D' \in S. \beta_{c,\{x_D'\}}^* \leq \beta_{c,\{\hat{x}_D\}}^* \leq \beta_c^*$. Namely, our BaB bounds hyper-networks by the iDP-DB of hyper-networks that abstract a single network (in which case there is no overapproximation error).

Our BaB has several advantages. First, it computes the precise iDP-DB. Second, it can provide an anytime solution: if it early stops, it returns an overapproximation of the iDP-DB. Third, it relies on MILPs over hyper-networks, which reduces the number of boolean variables compared to the straightforward MILP encoding. Fourth, it dynamically identifies a minimal number of MILPs for

computing the iDP-DB. Fifth, it orders the analysis of the hyper-networks to increase the chances of bounding them (described in Section 6.1).

Figure 3 exemplifies our BaB for computing the iDP-DB of $c = 0$ for a classifier N (top left) and four networks $\{N_{-1}, N_{-2}, N_{-3}, N_{-4}\}$ (top). To simplify notation, we omit the subscript $c = 0$ from the iDP-DB. The naive approach (Section 5.1) solves four optimization problems P_{x_D} for $x_D \in [4]$ and returns their maximum (obtained for P_4 in this example): $\beta^* = \beta_4^* = 0.52$. In contrast, our BaB obtains the same bound by solving only three MILPs of the same complexity as P_{x_D} . Our BaB begins by computing the iDP-DB of $\{1, 2, 3, 4\}$, which is $\beta_{1,\dots,4}^* = 0.55$ (bottom center). Although we know upfront that this iDP-DB is not tight, our BaB computes it to provide an anytime solution. Since this hyper-network is not bounded by any iDP-DB of a single network, it is branched. To this end, our BaB clusters the networks based on their similarity. This results in the partitioning $\mathcal{N}_{1,2,3}$ and $\mathcal{N}_4$. Accordingly, two hyper-networks are constructed (bottom left and right). For each, our BaB solves the respective MILP, returning $\beta_{1,\dots,3}^* = 0.21$ and $\beta_4^* = 0.52$. Since $\beta_{1,\dots,3}^* \leq \beta_4^*$, it bounds the hyper-network $N_{1,2,3}^\#$. Similarly, it bounds $N_4^\#$ by β_4^* . Then, our BaB terminates and returns 0.52.

5.4 Matching Dependencies and Relax-If-Similar

In this section, we describe two techniques to further reduce the complexity of the MILP of Equation (2). Both techniques rely on computing bounds for the differences of matching neurons in the network and the hyper-network. The first technique, *matching dependencies*, encodes these differences as linear constraints, for every pair of matching neurons, to prune the search space. The second technique, *relax-if-similar*, overapproximates neurons in the hyper-network whose difference is very small by linear relaxation. It reduces the complexity's exponent by one for every overapproximated neuron. While overapproximation reduces the precision, it is employed when the difference is small and combined with the matching dependencies, the precision loss is small.

These techniques rely on bounding the differences of matching neurons. A pair of matching neurons consists of a neuron $z_{m,k}$ in the network N and its corresponding neuron in the hyper-network $N^\#$, denoted $z_{m,k}^\#$, whose output is a real-valued interval (since the weights and biases of a hyper-network are intervals). We bound the difference of $z_{m,k}^\#$ and $z_{m,k}$ in an interval $[\Delta_{m,k}^l, \Delta_{m,k}^u] \in \mathbb{R}^2$ overapproximating the expression $z_{m,k}^\# - [z_{m,k}, z_{m,k}]$. The difference intervals enable significant pruning to our search space, since in our setting the outputs of matching neurons are highly dependent. This is because the network and hyper-network accept the same input and since their respective weights and biases are very close, because N and the networks used to define $N^\#$ are trained by the same training algorithm and their training sets only slightly differ. To compute the difference intervals, we rewrite the interval of every weight (and bias) in the hyper-network $[\underline{w}_{m,k,k'}, \overline{w}_{m,k,k'}]$ in terms of the weight in N plus a difference: $[w_{m,k,k'} - (w_{m,k,k'} - \underline{w}_{m,k,k'}), w_{m,k,k'} + (\overline{w}_{m,k,k'} - w_{m,k,k'})]$. Then, we employ bound propagation (formalized in Section 6.2) to overapproximate $z_{m,k}^\# - [z_{m,k}, z_{m,k}]$, for all m and k .

The matching dependencies technique adds to Equation (2) the difference intervals. For every non-input neuron $z_{m,k}$ and its difference interval $[\Delta_{m,k}^l, \Delta_{m,k}^u]$, it adds the constraint: $z_{m,k} + \Delta_{m,k}^l \leq z_{m,k}^\# \leq z_{m,k} + \Delta_{m,k}^u$. While these constraints can be added to any pair of networks, they are more effective for pruning the search space when $\Delta_{m,k}^u - \Delta_{m,k}^l$ is small, which is the case in our setting.

The relax-if-similar technique overapproximates the computation of neurons in the hyper-network if their difference interval is small. We remind that our MILP encoding of Equation (2) leverages prior work for encoding the classification confidence [Tjeng et al. 2019]. This encoding introduces a unique boolean variable for every non-input neuron in the network and the hyper-network. As described, the complexity of a MILP is exponential in the number of boolean variables.

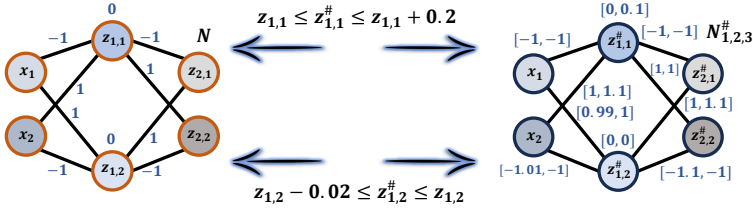


Fig. 4. An example of the matching dependencies and relax-if-similar.

To reduce the exponential complexity, prior work [Kabaha and Drachsler-Cohen 2024; Müller et al. 2022; Wang et al. 2021, 2022a,b] eliminates boolean variables by: (1) computing tight lower and upper bounds to identify neurons whose output is non-positive or non-negative, in which case their ReLU is stable and their boolean variable can be removed, and/or (2) overapproximating the ReLU computations using linear constraints without boolean variables. Although in general overapproximation leads to precision loss, we observe that if the difference interval of a neuron $z_{m,k}^{\#}$ is small, we can overapproximate its computation without losing too much precision. This follows because its matching neuron $z_{m,k}$ is precisely encoded (with a boolean variable) and because the matching dependency of $z_{m,k}$ forces the value of $z_{m,k}^{\#}$ to remain close to $z_{m,k}$. Based on this insight, our relax-if-similar technique eliminates the boolean variable of $z_{m,k}^{\#}$ and replaces its ReLU constraints by linear relaxation constraints. These constraints capture the minimal triangle bounding the piecewise linear function of ReLU using three linear constraints [Ehlers 2017].

Figure 4 exemplifies these techniques on the network N and the hyper-network $N_{1,2,3}^{\#}$ shown in Figure 3. We have $\hat{z}_{1,1} = -x_1 + x_2$ and $\hat{z}_{1,1}^{\#} = -x_1 + [1, 1.1] \cdot x_2 + [0, 0.1]$. We can write $\hat{z}_{1,1}^{\#}$ as a function of $\hat{z}_{1,1}$ and obtain: $\hat{z}_{1,1}^{\#} = z_{1,1} + [0, 0.1] \cdot x_2 + [0, 0.1]$. These values pass through ReLU. Since the input x_2 ranges over $[0, 1]$, we get that $z_{1,1}^{\#}$ is between $z_{1,1}$ and $z_{1,1} + 0.2$. Thus, the matching dependency of $z_{1,1}^{\#}$ is: $z_{1,1} \leq z_{1,1}^{\#} \leq z_{1,1} + 0.2$. Similarly, the matching dependency of $z_{1,2}^{\#}$ is: $z_{1,2} - 0.02 \leq z_{1,2}^{\#} \leq z_{1,2}$. If we precisely encoded all neurons using a boolean variable for each non-input neuron, the computed iDP-DB would be $\beta_{1,2,3}^* = 0.21$. If we employed linear relaxation for $z_{1,1}^{\#}$ and $z_{1,2}^{\#}$, it would eliminate two boolean variables (thereby reducing the problem's complexity) but would result in an overapproximating bound 0.24. To balance, relax-if-similar employs linear relaxation only to similar neurons. In this example, $z_{1,2}^{\#}$ is similar to $z_{1,2}$ (since $\Delta_{1,2}^l = -0.02$ and $\Delta_{1,2}^u = 0$) while $z_{1,1}^{\#}$ is not similar to $z_{1,1}$ ($\Delta_{1,1}^l = 0$ and $\Delta_{1,1}^u = 0.2$). Namely, relax-if-similar eliminates one boolean variable (of $z_{1,2}^{\#}$) and enables to compute the precise bound of $\beta_{1,2,3}^* = 0.21$.

6 Our System: LUCID

In this section, we describe our system LUCID, consisting of: (1) LUCID-Compute, computing the iDP-DB of every class using the verification techniques described in Section 5, and (2) LUCID-Inference, creating iDP label-only access to the network, given the iDP-DB of every class.

6.1 A System for Computing the iDP-DB

In this section, we describe LUCID-Compute, our system for computing the iDP-DB. Algorithm 1 provides its pseudo-code and Figure 5 illustrates it. Its inputs are a training set D , a training algorithm $\mathcal{T}$, a network architecture $\tilde{N}$, and a class c . It returns the iDP-DB of c (Equation (1)).

LUCID-Compute first trains all networks for every possibility to omit up to one data point (Line 1–Line 2). Then, it performs initializations. First, it initializes the anytime overapproximating bound of the iDP-DB to ∞ (Line 3). Next, it initializes a priority queue Q consisting of the sets to branch

Algorithm 1: LUCID-Compute ($D, \mathcal{T}, \tilde{N}, c$)**Input:** A dataset D , a training algorithm $\mathcal{T}$, a network architecture $\tilde{N}$, and a class c .**Output:** The iDP-DB bound β_c .

```

1  $N = \mathcal{T}(D, \tilde{N})$  // The network given the full dataset
2  $N_D = \{(x_D, \mathcal{T}(D \setminus \{x_D\}, \tilde{N})) \mid x_D \in D\}$  // The networks given adjacent datasets
3  $\beta_c = \infty$  // The iDP-DB anytime bound
4  $Q = []$  // A priority queue
5  $L = \{D\}$  // A list of sets to compute their iDP-DB
6 while  $True$  do
7   for  $S \in L$  do
8      $N_S^\# = \text{defineHyperNetwork}(\{(x_D, N_{-x_D}) \in N_D \mid x_D \in S\})$ 
9      $e = \text{encodeMILP}(N, N_S^\#, c)$ 
10     $I = \text{computeDifferenceIntervals}(N, N_S^\#)$ 
11     $e = \text{addMatchingDependencies}(e, I)$ 
12     $e = \text{relaxIfSimilar}(e, \{[\Delta_{m,k}^l, \Delta_{m,k}^u] \in I \mid \Delta_{m,k}^u - \Delta_{m,k}^l \leq \tau\})$ 
13     $\beta_{c,S} = \text{MILPSolver}(e)$ 
14     $\text{push}(Q, (S, \beta_{c,S}))$ 
15   $S, \beta_{c,S} = \text{pop}(Q)$ 
16   $\beta_c = \beta_{c,S}$ 
17  if  $|S| == 1$  then break
18   $L = \text{partition}(k\text{-means\_elbow}, \{(x_D, N_{-x_D}) \in N_D \mid x_D \in S\})$ 
19 return  $\beta_c$ 

```

or bound to an empty queue (Line 4). Lastly, it initializes a list L consisting of the sets whose iDP-DB is next computed to a list containing the dataset D (Line 5). It then runs our BaB (described in Section 5.3). Our BaB begins by running the verification analysis for all sets in L . For every set S in L , the analysis constructs the hyper-network $N_S^\#$ over the networks in S (Line 8). Then, it encodes Equation (2) as a MILP (Line 9), which we define in Section 6.2. It then computes the difference intervals (Line 10), described in Section 5.4 and defined in Section 6.2. Accordingly, it adds the matching dependencies (Line 11) and relaxes neurons whose difference interval is small (Line 12). It then submits the MILP to a MILP solver (Line 13), which returns a bound $\beta_{c,S}$. The set S and its bound $\beta_{c,S}$ are pushed to the priority queue Q (Line 14). We next describe the role of Q .

As described, after computing $\beta_{c,S}$, the problem of S (Equation (2)) is either branched or bounded. S is bounded if our BaB computes $\beta_{c,S'}$ where $\beta_{c,S} \leq \beta_{c,S'}$ and $|S'| = 1$. To reduce the number of branched problems, LUCID-Compute *lazily* branches. In each iteration, it branches the problem S that must be branched, which is the one with the maximal bound $\beta_{c,S}$. To identify this problem, LUCID-Compute prioritizes in Q the pairs by their bound, such that the next pair that is popped is the one with the maximal bound. If LUCID-Compute pops from Q a pair $(S, \beta_{c,S})$ whose S is a singleton $\{x_D\}$, it bounds this problem along with the problems of *all* pairs in the queue (Line 17). At this point, all problems are handled and LUCID-Compute returns $\beta_{c,S}$ (Line 19).

If LUCID-Compute pops from Q a pair $(S, \beta_{c,S})$ such that $|S| > 1$, it partitions S into several sets and stores them in L to analyze them in the next iteration (Line 18). LUCID-Compute partitions based on the closeness of the networks' weights and biases. Such partitioning is effective for two reasons. First, the closer the parameters are, the closer the networks' functions and their iDP-DB.

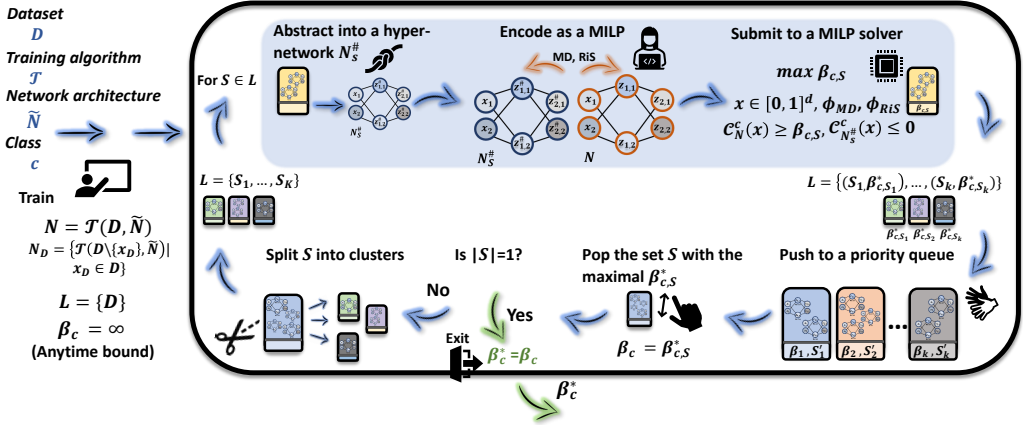


Fig. 5. LUCID's component for computing the iDP-DB.

Consequently, the hyper-network's iDP-DB is expected to be tighter (i.e., smaller), which increases the chances that it will be bounded and not branched. Second, it reduces the overapproximation error that stems from the abstraction of a hyper-network. To partition, LUCID employs the k -means clustering [Lloyd 1982] using the L_2 norm computed over all the networks' parameters. It dynamically determines the optimal number of clusters k using the elbow method.

LUCID-Compute is an anytime algorithm: at any point, it can return an overapproximation of the iDP-DB. Initially, the anytime bound is ∞ (Line 3) and after a pair is popped from the priority queue, the anytime bound is the bound of the last popped pair (Line 16).

Example. Consider the example of computing the iDP-DB of $c = 0$ for the classifier N shown in Figure 3 and $N_D = \{N_{-1}, N_{-2}, N_{-3}, N_{-4}\}$. Initially, $\beta_0 = \infty$, $Q = []$, and $L = \{\{4\}\}$. In the first iteration of Algorithm 1, the hyper-network of $S = [4]$ is constructed (Figure 3, bottom center) and its bound $\beta_{1,2,3,4}^* = 0.55$ is computed. Then, the pair $([4], 0.55)$ is pushed into the queue: $Q = [([4], 0.55)]$. Next, this pair is popped from Q and β_0 is reduced to 0.55. Since $[4]$ is not a singleton, it is partitioned by clustering. This results in $L = \{\{1, 2, 3\}, \{4\}\}$. Then, another iteration of Algorithm 1 begins. This iteration first constructs the hyper-network for $S = \{1, 2, 3\}$ (Figure 3, bottom left), computes its bound $\beta_{1,2,3}^* = 0.21$, and updates the queue: $Q = [(\{1, 2, 3\}, 0.21)]$. Then, this iteration constructs the hyper-network for $S = \{4\}$ (Figure 3, bottom right), computes its bound $\beta_4^* = 0.52$, and updates the queue: $Q = [(\{4\}, 0.52), (\{1, 2, 3\}, 0.21)]$. Note that since $0.52 > 0.21$, the new pair is pushed to the top of the queue. Next, the pair $(\{4\}, 0.52)$ is popped from Q and β_0 is reduced to 0.52. Since $\{4\}$ is a singleton, it bounds *all* pairs in Q and Algorithm 1 returns $\beta_0 = 0.52$.

6.2 The MILP Encoding

In this section, we describe our MILP encoding for computing the iDP-DB of a network and a hyper-network (Equation (2)). We adapt the MILP encoding from Tjeng et al. [2019], originally designed to analyze a network's local robustness, to our property defined over any input and over a network and a hyper-network. We begin with background and then present our adaptations.

Background. The MILP verifier of Tjeng et al. [2019] takes as inputs a classifier N , an input x classified as c , and a neighborhood $\mathcal{I}(x)$, defined by intervals for each input entry (i.e., $x' \in \mathcal{I}(x)$ if and only if $\forall k. x'_k \in [l_k, u_k]$). The verifier determines whether N is locally robust at $\mathcal{I}(x)$, i.e.,

whether it classifies all inputs in $\mathcal{J}(x)$ as c . For the input neurons, the verifier assigns each a real-valued variable $z_{0,k}$, constrained by its interval: $\forall k. l_k \leq z_{0,k} \leq u_k$. For the non-input neurons, it assigns two variables: $\hat{z}_{m,k}$ for the weighted sum (as defined in Section 2) and $z_{m,k}$ for the ReLU application. It also introduces concrete bounds $l_{m,k}, u_{m,k} \in \mathbb{R}$ for $\hat{z}_{m,k}$. The verifier encodes $\hat{z}_{m,k}$ as is using one equality constraint, and it encodes ReLU using four constraints, defined over $\hat{z}_{m,k}$ and over a boolean variable $a_{m,k}$ indicating whether the ReLU is in its inactive state (i.e., outputs zero) or active state (i.e., outputs $\hat{z}_{m,k}$). To determine local robustness, the verifier adds the constraint $z_{L,c} - \max_{c' \neq c} z_{L,c'} \leq 0$. If this MILP has no solution, N is locally robust at $\mathcal{J}(x)$; otherwise, it is not.

Our MILP encoding. We next present our MILP encoding for the iDP-DB of a set S (Equation (3)), which relies on several adaptations. First, since iDP-DB is defined over a network and a hyper-network, it has two copies of the variables: for N and for the hyper-network $N_S^\#$. For N , it has the same constraints for the non-input neurons (Eq. (3d) and (3f)). For $N_S^\#$, it has the same constraints for the ReLU computations (Eq. (3g)). For the weighted sums, since the hyper-network's parameters are intervals and not real numbers, it replaces the equality constraint by two inequality constraints (Eq. (3e)). The correctness of these constraints follows from interval arithmetic and since the input entries and the ReLU's outputs are non-negative. Second, to encode that the network and hyper-network accept the same input, it introduces a variable x and defines $z_{0,k} = x_k$ and $z_{0,k}^\# = x_k$, for all k (Eq. (3b)). Third, since iDP-DB is a global property, there is no neighborhood and each input entry x_k is bounded in the domain $[0, 1]$ (Eq. (3b)). Fourth, it introduces a bound $\beta_{c,S}$ and adds the objective $\max \beta_{c,S}$ (Eq. (3a)) as well as the constraints $z_{L,c} - z_{L,c'} \geq \beta_{c,S}$ for all $c' \neq c$, for encoding $\mathcal{C}_N^c(x) \geq \beta_{c,S}$, and $z_{L,c}^\# - \max_{c' \neq c} z_{L,c'}^\# \leq 0$, for encoding $\mathcal{C}_{N_S^\#}^c(x) \leq 0$ (Eq. (3c)). It expresses $\max_{c' \neq c} z_{L,c'}^\#$ by the Big M method [Winston 1991], which relies on a large constant M and boolean variables $a_{c'}$ for every $c' \neq c$. Fifth, it adds the matching dependencies ϕ_{MD} and linear relaxation constraints ϕ_{RIS} (Eq. (3b)), for neurons in the hyper-network whose difference interval is small (described shortly). Overall, given a classifier N , a hyper-network $N_S^\#$ with parameters $w_{m,k,k'}^\# = [\underline{w}_{m,k,k'}, \overline{w}_{m,k,k'}]$ and $b_{m,k}^\# = [\underline{b}_{m,k}, \overline{b}_{m,k}]$, and a class c , our encoding is given in Equation (3) below:

$$\max \beta_{c,S} \quad \text{subject to} \quad (3a)$$

$$\phi_{MD}; \quad \phi_{RIS}; \quad x \in [0, 1]^d; \quad \forall k : \quad z_{0,k} = x_k; \quad z_{0,k}^\# = x_k \quad (3b)$$

$$\forall c' \neq c. \quad z_{L,c} - z_{L,c'} \geq \beta_{c,S}; \quad \forall c' \neq c. \quad z_{L,c}^\# - z_{L,c'}^\# \leq M \cdot (1 - a_{c'}); \quad \sum_{c' \neq c} a_{c'} \geq 1 \quad (3c)$$

$$\forall m > 0, \forall k : \quad \hat{z}_{m,k} = b_{m,k} + \sum_{k'=1}^{k_{m-1}} w_{m,k,k'} \cdot z_{m-1,k'} \quad (3d)$$

$$\hat{z}_{m,k}^\# \geq \underline{b}_{m,k}^\# + \sum_{k'=1}^{k_{m-1}} \underline{w}_{m,k,k'}^\# \cdot z_{m-1,k'}^\#; \quad \hat{z}_{m,k}^\# \leq \overline{b}_{m,k}^\# + \sum_{k'=1}^{k_{m-1}} \overline{w}_{m,k,k'}^\# \cdot z_{m-1,k'}^\# \quad (3e)$$

$$z_{m,k} \geq 0; \quad z_{m,k} \geq \hat{z}_{m,k}; \quad z_{m,k} \leq u_{m,k} \cdot a_{m,k}; \quad z_{m,k} \leq \hat{z}_{m,k} - l_{m,k}(1 - a_{m,k}) \quad (3f)$$

$$z_{m,k}^\# \geq 0; \quad z_{m,k}^\# \geq \hat{z}_{m,k}^\#; \quad z_{m,k}^\# \leq u_{m,k}^\# \cdot a_{m,k}^\#; \quad z_{m,k}^\# \leq \hat{z}_{m,k}^\# - l_{m,k}^\#(1 - a_{m,k}^\#) \quad (3g)$$

Difference intervals. The matching dependencies and relax-if-similar techniques rely on the difference intervals. A difference interval $I_{z_{m,k}} = [\Delta_{z_{m,k}}^l, \Delta_{z_{m,k}}^u] \in \mathbb{R}^2$ is defined for every matching neurons and is computed by bound propagation and interval arithmetic. To formally define it, we define for each weight $w_{m,k,k'}$ an interval capturing the difference: $I_{w_{m,k,k'}} = w_{m,k,k'}^\# - [w_{m,k,k'}, w_{m,k,k'}]$, where $w_{m,k,k'}$ is the real-valued weight in N and $w_{m,k,k'}^\#$ is the weight interval in $N_S^\#$. Similarly,

$I_{b_{m,k}} = b_{m,k}^\# - [b_{m,k}, b_{m,k}]$. We define the difference intervals inductively on m . For the input layer, $\forall k \in [d]$. $I_{z_{0,k}} = [0, 0]$. For the other layers, for all $m > 0$ and $k \in [k_m]$:

$$\begin{aligned} I_{\hat{z}_{m,k}} &= \left(b_{m,k}^\# + \sum_{k'=1}^{k_{m-1}} w_{m,k,k'}^\# \cdot z_{m-1,k'}^\# \right) - \hat{z}_{m,k} = \\ & b_{m,k} + I_{b_{m,k}} + \sum_{k'=1}^{k_{m-1}} (w_{m,k,k'} + I_{w_{m,k,k'}}) \cdot (z_{m-1,k'} + I_{z_{m-1,k'}}) - \hat{z}_{m,k} \\ &= I_{b_{m,k}} + \sum_{k'=1}^{k_{m-1}} w_{m,k,k'} \cdot I_{z_{m-1,k'}} + I_{w_{m,k,k'}} \cdot (z_{m-1,k'} + I_{z_{m-1,k'}}) \end{aligned}$$

Here, $+$ and $\cdot$ are the standard addition and multiplication of two intervals (when writing a variable z , we mean the interval $[z, z]$). For the ReLU computation, we bound the difference as follows:

$$\begin{aligned} \text{ReLU}(\hat{z}_{m,k}^\#) - \text{ReLU}(\hat{z}_{m,k}) &= \text{ReLU}(\hat{z}_{m,k} + I_{\hat{z}_{m,k}}) - \text{ReLU}(\hat{z}_{m,k}) \leq \\ \text{ReLU}(\hat{z}_{m,k}) + \text{ReLU}(\Delta_{\hat{z}_{m,k}}^u) - \text{ReLU}(\hat{z}_{m,k}) &= \max(0, \Delta_{\hat{z}_{m,k}}^u) \end{aligned}$$

Similarly,

$$\begin{aligned} \text{ReLU}(\hat{z}_{m,k}^\#) - \text{ReLU}(\hat{z}_{m,k}) &= \text{ReLU}(\hat{z}_{m,k} + I_{\hat{z}_{m,k}}) - \text{ReLU}(\hat{z}_{m,k}) \geq \text{ReLU}(\hat{z}_{m,k} - (-\Delta_{\hat{z}_{m,k}}^l)) - \\ \text{ReLU}(\hat{z}_{m,k}) &\geq \text{ReLU}(\hat{z}_{m,k}) - \text{ReLU}(-\Delta_{\hat{z}_{m,k}}^l) - \text{ReLU}(\hat{z}_{m,k}) = -\max(0, -\Delta_{\hat{z}_{m,k}}^l) \end{aligned}$$

The inequality transitions follow from the triangle inequalities for ReLU. Overall, $I_{z_{m,k}} = [-\max(0, -\Delta_{\hat{z}_{m,k}}^l), \max(0, \Delta_{\hat{z}_{m,k}}^u)]$. It is possible to compute a tighter difference interval [Paulsen et al. 2020a,b], but with a longer computation, and in practice it is not required in our setting.

Matching dependencies. The matching dependencies are linear constraints encoding the difference intervals to prune the MILP's search space. Given the intervals, the combined constraint is:

$$\phi_{MD} = \bigwedge_{m \in [L], k \in [k_m]} \left(z_{m,k}^\# \geq z_{m,k} + \Delta_{\hat{z}_{m,k}}^l \wedge z_{m,k}^\# \leq z_{m,k} + \Delta_{\hat{z}_{m,k}}^u \right)$$

Relax-if-similar. To reduce the MILP's complexity, relax-if-similar overapproximates neurons in the hyper-network whose difference interval is small (its size is below a small threshold τ). The overapproximation replaces their ReLU constraints defined over a boolean variable by linear constraints overapproximating the ReLU computation as proposed by Ehlers [2017]. Namely, it replaces the constraints of $z_{m,k}^\#$ for these neurons (Eq. (3g)) by:

$$\phi_{Ris} = \bigwedge_{\Delta_{\hat{z}_{m,k}}^u - \Delta_{\hat{z}_{m,k}}^l \leq \tau} \left(z_{m,k}^\# \geq 0 \wedge z_{m,k}^\# \geq \hat{z}_{m,k} \wedge z_{m,k}^\# \leq \frac{u_{m,k}^\#}{u_{m,k}^\# - l_{m,k}^\#} (\hat{z}_{m,k} - l_{m,k}^\#) \right)$$

6.3 Our iDP Label-Only Access to the Network Classifier

In this section, we describe LUCID-Inference, our system for creating label-only access to a neural network classifier that is individually differentially private (iDP). LUCID-Inference (Algorithm 2) takes as inputs a dataset D , a training algorithm $\mathcal{T}$, a network architecture $\tilde{N}$, and a privacy budget ϵ . It first trains the network N and runs LUCID-Compute for every class c to obtain all iDP-DB (Line 1–Line 2). It then initializes a dictionary M of inputs and their labels, for inputs whose labels are selected by the exponential mechanism (Line 3). This is used in case the attacker repeatedly submits the same input. Then, given an input x , the iDP access is defined as follows. If M contains x ,

Algorithm 2: LUCID-Inference ($D, \mathcal{T}, \tilde{N}, \varepsilon$)

```

1  $N = \mathcal{T}(D, \tilde{N})$ 
2  $\beta\text{Set} = \{\text{LUCID-Compute}(D, \mathcal{T}, \tilde{N}, c) \mid c \in C\}$ 
3  $M = []$  // A dictionary of inputs for which the exponential mechanism ran
4 Function  $\text{Access}(x)$ 
    Input: The input to the network  $x$ .
    Output: A label, such that the label-only query satisfies  $\varepsilon$ -iDP.
5   if  $M[x] \neq \perp$  then return  $M[x]$ 
6    $\text{scores} = N(x)$ 
7    $c = \text{argmax}_{c \in C} \text{scores}$ 
8   if  $\text{scores}[c] - \max_{c' \neq c} \text{scores}[c'] > \beta\text{Set}[c]$  then return  $c$ 
9    $M[x] = \text{ExponentialMechanism}(u_{x, \mathcal{T}, \tilde{N}}, D, c, \varepsilon)$ 
10  return  $M[x]$ 

```

it returns the label in $M[x]$ (Line 5). Otherwise, it passes x through N to compute the output vector $N(x)$ (Line 6). It then identifies the predicted label c , which is the label with the maximal score (Line 7). It checks whether its classification confidence is greater than β_c , and if so, it returns c (Line 8). Otherwise, it runs the exponential mechanism as described in Section 3.3. It stores the resulting label in $M[x]$ and returns it (Line 9–Line 10). We assume LUCID-Inference’s memory is as large as the attacker’s memory. Namely, after M reaches the memory limit, LUCID-Inference overrides the oldest entry, which the attacker also forgets.

THEOREM 6.1. *Given a dataset D , a training algorithm $\mathcal{T}$, a network architecture $\tilde{N}$, and a privacy budget ε , LUCID-Inference provides an ε -iDP version of every label-only query $f_{x, \mathcal{T}, \tilde{N}}(D)$, for $x \in [0, 1]^d$, without losing accuracy for queries whose x has a confidence larger than the iDP-DB.*

A proof sketch is provided in Appendix A.

7 Evaluation

In this section, we evaluate LUCID’s effectiveness in providing iDP label-only access to a neural network classifier. We run all the experiments on a dual AMD EPYC 7713 64-Core Processor@2GHz server with 2TB RAM and eight A100 GPUs running an Ubuntu 20.04.1 OS. We begin with the experiment setup: implementation, datasets, and networks. We then compare LUCID to baselines. Finally, we provide an ablation study, showing the effectiveness of LUCID’s components.

Implementation. We implemented LUCID in Julia 1.8.3 as a module extending MIPVerify [Tjeng et al. 2019]. To solve MILPs, LUCID uses Gurobi 11.0.1 [Gurobi Optimization, LLC 2023]. We run LUCID with a timeout of eight hours. The timeout of a single MILP (for solving Equation (3) for a given set S) is 40 minutes. LUCID is parallelizable: it uses 32 workers that pop from the priority queue Q sets S to branch or bound. For the relax-if-similar technique, it uses $\tau = 0.01$.

Networks. We evaluate LUCID over four datasets (described in Appendix B): Cryptojacking [Kharraz et al. 2019] (Crypto), Twitter Spam Accounts [Lee et al. 2011] (Twitter), Adult Census [Becker and Kohavi 1996] (Adult), and Default of Credit Card Clients [Yeh 2016] (Credit). We consider three fully-connected neural networks with architectures of 2×50 , 2×100 , and 4×30 , where the first number is the number of intermediate layers and the second number is the number of neurons in each of these layers. Additionally, we evaluate LUCID on a convolutional neural network (CNN)

Table 1. The accuracy of LUCID ($Acc_{w/o-p}$ is the model's accuracy without privacy protection).

Dataset	Model	$Acc_{w/o-p}$	LUCID							
			β_0	T_0	β_1	T_1	Acc	Acc	Acc	AT
				[h]		[h]	[%] $\epsilon = 0$	[%] $\epsilon = 0.2$	[%] $\epsilon = 1$	[ms]
Crypto	2×50	99.7	1.78	0.1	2.95	0.07	97.3	97.3	98.0	0.1
Crypto	2×100	99.7	2.02	1.1	2.86	1.0	97.1	97.5	97.8	0.2
Crypto	4×30	99.8	4.23	0.1	6.12	0.1	94.7	95.0	95.5	0.2
Twitter	2×50	89.7	0.51	1.6	0.54	3.7	89.3	89.3	89.4	0.1
Twitter	2×100	89.6	0.86	4.9	0.96	5.5	88.1	88.2	88.5	0.2
Adult	2×50	83.1	0.55	2.9	0.63	3.0	82.4	82.4	82.5	0.1
Adult	2×100	83.3	0.93	4.3	0.91	4.1	80.7	80.8	81.0	0.1
Adult	<i>conv</i>	79.2	0.29	0.2	0.52	0.1	80.1	80.1	80.0	0.2
Credit	2×50	81.9	0.33	5.8	0.34	6.5	81.7	81.8	81.8	0.1
Credit	2×100	81.8	0.42	4.4	0.44	3.1	81.4	81.4	81.5	0.2
Credit	<i>conv</i>	80.8	0.75	0.3	0.57	0.2	79.9	80.1	80.3	0.3

architecture that has two convolutional layers followed by a fully-connected layer. The activation function in all networks is ReLU. We train the networks using SGD, over 50 epochs, the batch size is 1024 (except for the Crypto dataset, where the batch size is 100), and the initial learning rate is $\eta = 0.1$. Although the network sizes may seem small compared to the common network sizes evaluated by local robustness verifiers (which are typically evaluated over image classifiers that have larger input dimensionality), our network sizes are consistent with those used in previous work evaluating the datasets we consider [Baharlouei et al. 2020; Mazzucato and Urban 2021; Reshef et al. 2024; Urban et al. 2019]. We remind that LUCID analyzes a much more challenging property than local robustness or fairness: iDP-DB is a global property, over any input, and over a very large number of classifiers (the size of the dataset plus one).

Baselines. We compare LUCID to four baselines. First, the naive algorithms described in Section 3.3: Naive-Noise, invoking the exponential mechanism for every input, and Naive-iDP, which passes every input through $|D| + 1$ networks and invokes the exponential mechanism if there is a disagreement between them. Second, we compare to two DP training algorithms: DP-SGD [Abadi et al. 2016], using its PyTorch implementation¹, and ALIBI [Esmaceli et al. 2021], using the authors' code². DP-SGD provides a DP guarantee to all the network's parameters and ALIBI provides a DP guarantee only for the classifier's output. Both of them inject Gaussian noise into the network's training process to obtain a user-specified ϵ -DP guarantee. In our experiments, we employ a binary search to determine the appropriate noise level for achieving the desired ϵ . The binary search increases the computation time by a few milliseconds and thus poses a minor increase to their overall computation time, which is multiple seconds. These two baselines provide a stronger privacy guarantee (DP) than iDP, but any ϵ -DP algorithm is also ϵ -iDP [Soria-Comas et al. 2017]. We compare to them because they represent the current approach for providing privacy guarantees to neural networks. We note that we do not compare to other works guaranteeing other variants of

¹<https://github.com/ChrisWaites/pyvacy.git>

²https://github.com/facebookresearch/label_dp_antipodes.git

Table 2. The accuracy of Naive-Noise and Naive-iDP ($Acc_{w/o-p}$ is the accuracy without privacy protection).

Dataset	Model	$Acc_{w/o-p}$	Naive-Noise				Naive-iDP			
			Acc	Acc	Acc	AT	Acc	Acc	Acc	AT
			[%] $\varepsilon = 0$	[%] $\varepsilon = 0.2$	[%] $\varepsilon = 1$	[ms]	[%] $\varepsilon = 0$	[%] $\varepsilon = 0.2$	[%] $\varepsilon = 1$	[s]
Crypto	2×50	99.7	49.9	52.3	62.1	0.1	99.4	99.4	99.5	3
Crypto	2×100	99.7	50.1	52.7	62.1	0.2	99.3	99.2	99.3	4
Crypto	4×30	99.8	50.0	52.4	62.0	0.2	99.7	99.7	99.7	4
Twitter	2×50	89.7	49.9	51.9	59.6	0.2	89.5	89.5	89.6	43
Twitter	2×100	89.6	50.1	52.2	59.9	0.3	89.3	89.4	89.4	52
Adult	2×50	83.1	49.9	51.7	58.2	0.1	83.1	82.9	83.1	32
Adult	2×100	83.3	49.9	51.7	58.9	0.2	83.2	83.1	83.3	34
Adult	<i>conv</i>	79.2	49.9	51.4	57.2	0.2	79.2	79.5	79.5	40
Credit	2×50	81.9	50.1	51.6	57.7	0.1	81.8	81.9	81.9	19
Credit	2×100	81.8	50.1	51.6	57.9	0.2	81.7	81.7	81.8	21
Credit	<i>conv</i>	80.8	49.9	51.6	57.5	0.3	80.9	80.7	80.7	26

DP [Bhaila et al. 2024; Boenisch et al. 2023a,b; Jiang et al. 2023; Mironov 2017; Ogilvie 2024; Reshef et al. 2024; Sébert 2023], since their guarantee is incomparable to iDP.

LUCID's performance. We begin by evaluating the performance of LUCID over fully-connected and convolutional networks for all our datasets, compared to all four baselines. For LUCID, Table 1 reports the computed iDP-DB for each label β_0 and β_1 , their computation time in hours (T_0 and T_1), the test accuracy Acc for three values of ε (0, 0.2, and 1), and the access time in milliseconds (AT), i.e., the time to return a label for a given input. For Naive-Noise and Naive-iDP, Table 2 reports the test accuracy Acc and the access time in milliseconds and seconds, respectively. For the DP training algorithms, Table 3 reports the test accuracy Acc and the training time in seconds (CT). We run the DP training algorithms with the same values of ε , except that we replace the value 0 with 0.02, since DP cannot be obtained for $\varepsilon = 0$. In fact, 0.02 is the lowest value that our baselines can provide, and it requires adding a very high noise (the standard deviation of the noise distribution is 10,000). We note that we focus on these values of ε since commonly a value of ε is considered to provide a strong DP guarantee when $\varepsilon < 1$ and a moderate guarantee when $\varepsilon \in [1, 3]$. Commonly, DP training algorithms, including our baselines, focus on guarantees for $\varepsilon \geq 1$. We note that when providing the baselines ε smaller than 0.02, it results in a very large noise, rendering the computation practically infeasible (due to computer arithmetic issues). The results show that, for $\varepsilon = 0$, LUCID provides a 0-iDP guarantee with only 1.4% accuracy decrease. As expected, for larger values of ε , LUCID lowers the accuracy decrease: it provides a 0.2-iDP guarantee with a 1.3% accuracy decrease and a 1-iDP guarantee with a 1.1% accuracy decrease. To compute the iDP-DB, LUCID runs for 4.8 hours on average (this computation is run once for every network). Its iDP-access time is at most one millisecond and on average it is 0.16 milliseconds. This time is comparable to the access time of standard networks (without iDP guarantees), with a negligible overhead. For Naive-Noise, the accuracy decrease is very high because it invokes the exponential mechanism for every input (unlike LUCID and Naive-iDP). Its accuracy decrease is 38.1%, 36.1%, and 28.7% for $\varepsilon = 0$, $\varepsilon = 0.2$, and $\varepsilon = 1$, respectively. Like LUCID, its access time is negligible. Naive-iDP obtains the minimal accuracy decrease, since it precisely identifies the set of leaking inputs (unlike LUCID

Table 3. The accuracy of DP-SGD and ALIBI ($Acc_{w/o-p}$ is the model's accuracy without privacy protection).

Dataset	Model	$Acc_{w/o-p}$	DP-SGD				ALIBI			
			Acc	Acc	Acc	CT	Acc	Acc	Acc	CT
			[%] $\epsilon = 0.02$	[%] $\epsilon = 0.2$	[%] $\epsilon = 1$	[s]	[%] $\epsilon = 0.02$	[%] $\epsilon = 0.2$	[%] $\epsilon = 1$	[s]
Crypto	2×50	99.7	49.8	96.6	96.6	8.2	48.5	48.5	92.8	19.8
Crypto	2×100	99.7	49.3	96.5	96.6	7.9	80.8	96.7	96.9	20.8
Crypto	4×30	99.8	34.5	49.3	49.3	11.4	49.3	49.3	95.6	22.8
Twitter	2×50	89.7	51.9	51.4	51.5	25.1	51.4	51.4	87.6	169.2
Twitter	2×100	89.6	47.5	53.2	52.4	28.8	51.4	52.6	87.6	159.6
Adult	2×50	83.1	23.6	76.3	76.3	24.4	67.4	77.4	82.2	149.4
Adult	2×100	83.3	76.3	76.4	76.4	20.8	61.5	75.3	81.9	103.6
Adult	<i>conv</i>	79.2	46.2	76.4	76.4	24.9	23.6	74.6	76.4	139.8
Credit	2×50	81.9	26.2	78.4	78.5	22.4	78.4	78.4	78.6	96.4
Credit	2×100	81.8	71.8	78.5	78.5	21.1	63.7	73.1	79.9	81.8
Credit	<i>conv</i>	80.8	25.2	78.4	78.5	24.7	78.4	78.4	78.4	116.8

that overapproximates them with the iDP-DB). Its accuracy decrease is 0.12%, 0.1%, and 0.05% for the privacy budgets $\epsilon = 0$, $\epsilon = 0.2$, and $\epsilon = 1$, respectively. However, it has to store throughout its execution all $|D| + 1$ classifiers and its access time is high: 25 seconds on average (since it passes every input through all classifiers). This severely undermines its utility in real-time applications of neural networks, providing an interactive communication. The accuracy decrease of DP-SGD, for $\epsilon \in \{0.02, 0.2, 1\}$, is on average 42.3%, 14.2%, and 14.3%, respectively (29.8x, 10.7x, and 12.9x higher than LUCID). The accuracy decrease of ALIBI for the same DP guarantees is on average 28.6%, 19.3%, and 2.7% (20.1x, 14.5x, and 2.4x higher than LUCID). Even worse, on some networks (Crypto 4×30, Twitter 2×50, and Twitter 2×100), the baselines train networks whose accuracies are 50% (like the accuracy of a random classifier). If we ignore these three networks, DP-SGD's accuracy decrease is 40.13%, 3.99%, and 3.9%, which is still significantly higher than LUCID (by 28.3x, 3.0x, and 3.5x), while ALIBI's decrease is 23.4%, 10.8%, and 2.7% (higher by 16.4x, 8.2x, and 2.4x). The baselines are faster than LUCID's analysis time (for computing the iDP-DB): DP-SGD completes within 19.9 seconds and ALIBI within 98.2 seconds. Although they are significantly faster, their training algorithms are coupled to a specific privacy budget ϵ : if a user wishes to update the privacy guarantee, they need to retrain the classifier. In contrast, LUCID's iDP-access can easily configure the ϵ (the iDP-DB are computed once for a classifier). Further, as we show, computing the iDP-DB allows us to achieve iDP guarantees with a small accuracy decrease.

Illustration of the iDP-DB. We next illustrate the reason that the iDP-DB enables LUCID to provide iDP-access to a network classifier with a minor accuracy decrease. We consider the 2×50 network for Twitter and let LUCID compute the iDP-DB of each class β_0 and β_1 . Figure 6 plots the classification confidence of each input in the test set, where inputs labeled as 0 are shown on the left plot and inputs labeled as 1 are shown on the right plot. The plots show in a green dashed line the iDP-DB, in blue points the test points whose confidence is greater than the iDP-DB and in red points the test points whose confidence is smaller or equal to the iDP-DB. The plots show that the confidence of the vast majority of points in the test set is over the iDP-DB. Recall that for these points, LUCID does not employ the exponential mechanism and thereby it obtains a minor accuracy decrease.

Table 4. Ablation study over LUCID’s components. Hyp stands for Hyper-network, MD for Matching Dependencies, RiS for Relax-if-Similar, and BaB for Branch-and-Bound.

Dataset	Model	Variant	Hyp	MD	RiS	BaB	β_0	T_0 [h]	β_1	T_1 [h]	$Acc_{\varepsilon=0}$ [%]	$Acc_{\varepsilon=1}$ [%]
Twitter	2×50 $Acc_{w/o-p}$ 89.7%	Naive	✗	✗	✗	✗	-	8.00	-	8.00	-	-
		H	✓	✗	✗	✗	1.62	8.00	1.49	8.00	85.4	85.9
		HM	✓	✓	✗	✗	1.22	0.18	1.20	0.38	87.5	87.8
		HMRiS	✓	✓	✓	✗	1.25	0.17	1.26	0.11	87.3	87.7
		HMB	✓	✓	✗	✓	0.46	4.56	0.43	5.26	89.3	89.4
		LUCID	✓	✓	✓	✓	0.51	1.16	0.46	2.81	89.3	89.4
Adult	conv $Acc_{w/o-p}$ 79.2%	Naive	✗	✗	✗	✗	-	8.00	-	8.00	-	-
		H	✓	✗	✗	✗	0.96	8.00	0.58	7.50	74.8	75.3
		HM	✓	✓	✗	✗	0.86	0.04	0.58	0.04	75.4	75.9
		HMRiS	✓	✓	✓	✗	0.87	0.03	0.58	0.04	75.3	75.9
		HMB	✓	✓	✗	✓	0.33	1.65	0.24	4.05	80.3	80.1
		LUCID	✓	✓	✓	✓	0.40	0.76	0.36	0.81	80.1	80.1

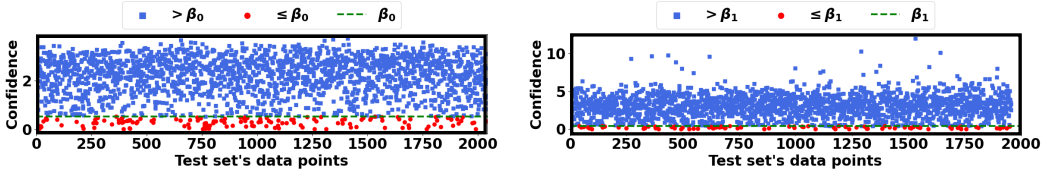


Fig. 6. The iDP-DB and the classification confidence of the test set’s data points, for Twitter 2×50.

Ablation tests. Next, we evaluate the effectiveness of LUCID’s components. We focus on the Twitter 2×50 network and on the Adult convolutional network. We consider five variants of LUCID:

- Naive: The naive approach described in Section 5.1, which computes the iDP-DB of every $x_D \in D$ by encoding the problem P_{x_D} as a MILP (like LUCID, it has 32 parallel workers).
- Hyper-network (H): This variant constructs a hyper-network and solves the problem of Equation (2) for $S = D$ by encoding it as a MILP (as described in Section 5.2).
- +Matching dependencies (HM): This variant is H with the matching dependencies.
- +Relax-if-similar (HMRiS): This variant is HM with the relax-if-similar technique.
- +Branch-and-bound (HMB): This variant is HM with our branch-and-bound.

We run every approach with an eight hour timeout to compute the iDP-DB of each class β_0 and β_1 . We measure the computation time T_0 and T_1 and the accuracy decrease for $\varepsilon = 0$ and $\varepsilon = 1$. Table 4 summarizes the results. The results show that Naive does not compute an iDP-DB within the timeout (marked by -). In fact, it iterates over fewer than 5% of the networks. Unlike LUCID’s anytime algorithm, Naive must solve all problems in $\{P_{x_D} \mid x_D \in D\}$ to provide a sound iDP-DB. The hyper-network variant terminates but provides very loose iDP-DB, approximately 2.57x larger than those obtained by LUCID and it further requires 7.2x more time than LUCID. This looseness is caused by the overapproximation of the hyper-network. The computation time is very long because there are no matching dependencies, making the optimization very slow, often reaching the timeout. As expected, the hyper-network and matching dependencies variant converges faster to the iDP-DB: it is faster by 99.9x than the hyper-network variant. However, its iDP-DB remain high, by 2.16x than those computed by LUCID (because it analyzes a single hyper-network, which

introduces an overapproximation error). The variant that also employs relax-if-similar reduces the analysis time by 1.64x compared to the previous variant, with a minor increase to the bound (1.02x). The variant that adds branch-and-bound computes iDP-DB that are 2.66x tighter than the previous variant and 1.2x tighter than LUCID. However, its computation time is 3.2x longer than LUCID. This is expected, since the only difference between this variant and LUCID is the relax-if-similar technique, which accelerates the computation at the cost of overapproximation. When integrating each variant (except Naive, which did not terminate) with the two privacy budgets $\epsilon = 0$ and $\epsilon = 1$, the accuracy decrease is 4.4%, 3.0%, 3.2%, -0.4% and -0.3%, for $\epsilon = 0$, and 3.9%, 2.6%, 2.7%, -0.3% and -0.3%, for $\epsilon = 1$. A negative decrease means that LUCID improves the accuracy.

8 Related Work

Privacy of neural networks. Several works propose approaches to reduce privacy leakage of neural networks. Some propose specialized architectures to maintain privacy [Cheng et al. 2022; Wang et al. 2022c]. Others integrate regularization during training to limit leakage [Hayes et al. 2019; Melis et al. 2019]. Others rely on federated distributed learning [Chen et al. 2024; Elkordy et al. 2023; Xu et al. 2022]. Another approach is to train differentially private (DP) networks [Abadi et al. 2016; Chamikara et al. 2020; Chen et al. 2020; Esmaili et al. 2021; Ghazi et al. 2021, 2023]. Many works considered variants of DP to ensure privacy. For example, in individualized privacy assignment, different privacy budgets can be allocated to different users [Boenisch et al. 2023a,b]. In local differential privacy, a data owner adds randomization to their data before it leaves their devices [Bhaila et al. 2024; Chamikara et al. 2020]. Renyi differential privacy relaxes differential privacy based on the Renyi divergence [Jiang et al. 2023; Mironov 2017]. Local differential classification privacy proposes a new privacy property inspired by local robustness [Reshef et al. 2024]. Homomorphic-encryption approaches encrypt data before obtaining DP guarantees [Ogilvie 2024; Sébert 2023].

Multiple network analysis. LUCID analyzes a large set of similar networks. There is prior work on analysis of several networks. Some works compute the output differences of two networks [Paulsen et al. 2020a,b] or the output differences of a set of inputs [Banerjee et al. 2024]. Others study incremental verification, analyzing the differences of a network and its slightly modified version [Ugare et al. 2023]. Others propose proof transfer for similar networks for verifying local robustness [Ugare et al. 2022]. Some global robustness verifiers compare the outputs of a network given an input and given its perturbed version [Kabaha and Drachsler-Cohen 2024; Wang et al. 2022a,b].

Clustering in neural network verification. LUCID expedites its analysis by clustering close networks and computing their hyper-networks. Some verifiers accelerate local robustness analysis by grouping neurons into subgroups [Ashok et al. 2020; Liu et al. 2024], while others compute centroids to partition inputs and establish global robustness bounds [Gopinath et al. 2018].

9 Conclusion

We present LUCID, a system that creates iDP label-only access for a neural network classifier with a minor accuracy decrease. LUCID computes the iDP-DB, which overapproximates the set of inputs that violate iDP and adds noise only to the labels of these inputs. To compute the iDP-DB, LUCID relies on several techniques: constraint solving, hyper-networks abstracting a large set of networks, and a novel branch-and-bound technique. To further scale, it prunes the search space by computing matching dependencies and employing linear relaxation. Our experimental evaluation shows that our verification analysis enables LUCID to provide a 0-iDP guarantee with an accuracy decrease of 1.4%. For more relaxed iDP guarantees, LUCID can reduce the accuracy decrease to 1.2%, whereas existing DP approaches lead to an accuracy decrease of 12.7% to provide the same iDP guarantees.

Acknowledgements

We thank the anonymous reviewers for their feedback. This research was supported by the Israel Science Foundation (grant No. 2605/20).

Data-Availability Statement

Our code is available at <https://github.com/anankabaha/LUCID.git>.

References

- Martin Abadi, Andy Chu, Ian Goodfellow, Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. 2016. Deep Learning with Differential Privacy. In *CCS* (2016). doi:10.1145/2976749.2978318
- Mohammad Al-Rubaie and J. Morris Chang. 2019. Privacy-Preserving Machine Learning: Threats and Solutions. In *IEEE Secur. Priv.* (2019). doi:10.1109/MSEC.2018.2888775
- Amazon Web Services. 2024. *Machine Learning on AWS*. <https://aws.amazon.com/machine-learning>
- Pranav Ashok, Vahid Hashemi, Jan Kretinský, and Stefanie Mohr. 2020. DeepAbstract: Neural Network Abstraction for Accelerating Verification. In *ATVA* (2020). doi:10.1007/978-3-030-59152-6_5
- Eugene Bagdasaryan, Omid Poursaeed, and Vitaly Shmatikov. 2019. Differential Privacy Has Disparate Impact on Model Accuracy. In *NeurIPS* (2019). <https://proceedings.neurips.cc/paper/2019/hash/fc0de4e0396ff257ea362983c2dda5a-Abstract.html>
- Sina Baharlouei, Maher Nouiehed, Ahmad Beirami, and Meisam Razaviyayn. 2020. Rényi Fair Inference. In *8th International Conference on Learning Representations, ICLR*. OpenReview.net. <https://openreview.net/forum?id=HkgsUJrtDB>
- Borja Balle, Giovanni Cherubin, and Jamie Hayes. 2022. Reconstructing Training Data with Informed Adversaries. In *SP* (2022). doi:10.1109/SP46214.2022.9833677
- Debangshu Banerjee, Changming Xu, and Gagandeep Singh. 2024. Input-Relational Verification of Deep Neural Networks. In *PLDI* (2024). doi:10.1145/3656377
- Barry Becker and Ronny Kohavi. 1996. Adult. UCI Machine Learning Repository. <https://doi.org/10.24432/C5XW20>
- Karuna Bhaila, Wen Huang, Yongkai Wu, and Xintao Wu. 2024. Local Differential Privacy in Graph Neural Networks: a Reconstruction Approach. In *SLAM* (2024). doi:10.1137/1.9781611978032.1
- Franziska Boenisch, Christopher Mühl, Adam Dziedzic, Roy Rinberg, and Nicolas Papernot. 2023a. Have it your way: Individualized Privacy Assignment for DP-SGD. In *NeurIPS* (2023). http://papers.nips.cc/paper_files/paper/2023/hash/3cbf627fa24fb6cb576e04e689b9428b-Abstract-Conference.html
- Franziska Boenisch, Christopher Mühl, Roy Rinberg, Jannis Ihrig, and Adam Dziedzic. 2023b. Individualized PATE: Differentially Private Machine Learning with Individual Privacy Guarantees. In *Proc. Priv. Enhancing Technol.* (2023). doi:10.56553/POPETS-2023-0010
- Rudy Bunel, Jingyue Lu, Ilker Turkaslan, Philip H. S. Torr, Pushmeet Kohli, and M. Pawan Kumar. 2020. Branch and Bound for Piecewise Linear Neural Network Verification. *J. Mach. Learn. Res.* (2020). <https://jmlr.org/papers/v21/19-468.html>
- Rudy Bunel, Ilker Turkaslan, Philip H. S. Torr, Pushmeet Kohli, and Pawan Kumar Mudigonda. 2018. A Unified View of Piecewise Linear Neural Network Verification. In *NeurIPS* (2018). <https://proceedings.neurips.cc/paper/2018/hash/be53d253d6bc3258a8160556dda3e9b2-Abstract.html>
- Mahawaga Arachchige Pathum Chamikara, Peter Bertók, Ibrahim Khalil, Dongxi Liu, Seyit Camtepe, and Mohammed Atiquzzaman. 2020. Local Differential Privacy for Deep Learning. *IEEE Internet Things J.* 7, 7, 5827–5842. doi:10.1109/JIOT.2019.2952146
- Huiqiang Chen, Tianqing Zhu, Tao Zhang, Wanlei Zhou, and Philip S. Yu. 2024. Privacy and Fairness in Federated Learning: On the Perspective of Tradeoff. In *ACM Comput. Surv.* (2024). doi:10.1145/3606017
- Si Chen, Anmin Fu, Jian Shen, Shui Yu, Huaqun Wang, and Huaijiang Sunr. 2020. RNN-DP: A new differential privacy scheme base on Recurrent Neural Network for Dynamic trajectory privacy protection. *J. Netw. Comput. Appl.* 168, 102736. doi:10.1016/J.JNCA.2020.102736
- Anda Cheng, Jiaxing Wang, Xi Sheryl Zhang, Qiang Chen, Peisong Wang, and Jian Cheng. 2022. DPNAS: Neural Architecture Search for Deep Learning with Differential Privacy. In *AAAI* (2022). doi:10.1609/AAAI.V36I6.20586
- Christopher A. Choquette-Choo, Florian Tramèr, Nicholas Carlini, and Nicolas Papernot. 2021. Label-Only Membership Inference Attacks. In *ICML* (2021).
- Cynthia Dwork. 2006. Differential Privacy. In *Automata, Languages and Programming, 33rd International Colloquium, ICALP (Lecture Notes in Computer Science, Vol. 4052)*. Springer, 1–12. doi:10.1007/11787006_1
- Cynthia Dwork and Aaron Roth. 2014. The Algorithmic Foundations of Differential Privacy. In *Found. Trends Theor. Comput. Sci.* (2014). doi:10.1561/04000000042

- Rüdiger Ehlers. 2017. Formal Verification of Piece-Wise Linear Feed-Forward Neural Networks. In *Automated Technology for Verification and Analysis - 15th International Symposium, ATVA (Lecture Notes in Computer Science, Vol. 10482)*. Springer, 269–286. doi:10.1007/978-3-319-68167-2_19
- Ahmed Roushdy Elkordy, Jiang Zhang, Yahya H. Ezzeldin, Konstantinos Psounis, and Salman Avestimehr. 2023. How Much Privacy Does Federated Learning with Secure Aggregation Guarantee? In *Proc. Priv. Enhancing Technol.* (2023). doi:10.56553/POPETS-2023-0030
- Mani Malek Esmaeili, Ilya Mironov, Karthik Prasad, Igor Shilov, and Florian Tramèr. 2021. Antipodes of Label Differential Privacy: PATE and ALIBI. In *NeurIPS*. <https://proceedings.neurips.cc/paper/2021/hash/37ecd27608480aa3569a511a638ca74f-Abstract.html>
- Jonas Geiping, Hartmut Bauermeister, Hannah Dröge, and Michael Moeller. 2020. Inverting Gradients - How easy is it to break privacy in federated learning? In *NeurIPS* (2020). <https://proceedings.neurips.cc/paper/2020/hash/c4ede56bbd98819ae6112b20ac6bf145-Abstract.html>
- Badih Ghazi, Noah Golowich, Ravi Kumar, Pasin Manurangsi, and Chiyuan Zhang. 2021. Deep Learning with Label Differential Privacy. In *NeurIPS* (2021). doi:10.1145/2976749.2978318
- Badih Ghazi, Pritish Kamath, Ravi Kumar, Ethan Leeman, Pasin Manurangsi, Avinash V. Varadarajan, and Chiyuan Zhang. 2023. Regression with Label Differential Privacy. In *ICLR* (2023). <https://openreview.net/forum?id=h9O0wsmL-cT>
- Google Cloud. 2024. *Vertex AI*. <https://cloud.google.com/vertex-ai>
- Divya Gopinath, Guy Katz, Corina S. Pasareanu, and Clark W. Barrett. 2018. DeepSafe: A Data-Driven Approach for Assessing Robustness of Neural Networks. In *ATVA* (2018). doi:10.1007/978-3-030-01090-4_1
- Gurobi Optimization, LLC. 2023. Gurobi Optimizer Reference Manual. <https://www.gurobi.com>
- Mehmet Emre Gursoy, Ali Inan, Mehmet Ercan Nergiz, and Yücel Saygin. 2017. Differentially private nearest neighbor classification. In *Data Min. Knowl. Discov.* (2017). doi:10.1007/S10618-017-0532-Z
- Jamie Hayes, Luca Melis, George Danezis, and Emiliano De Cristofaro. 2019. LOGAN: Membership Inference Attacks Against Generative Models. In *Proc. Priv. Enhancing Technol.* (2019). doi:10.2478/POPETS-2019-0008
- Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. In *CVPR* (2016). doi:10.1109/CVPR.2016.90
- Patrick Henriksen and Alessio Lomuscio. 2021. DEEPSPLIT: An Efficient Splitting Method for Neural Network Verification via Indirect Effect Analysis. In *IJCAI* (2021). doi:10.24963/IJCAI.2021/351
- Hailong Hu and Jun Pang. 2021. Stealing Machine Learning Models: Attacks and Countermeasures for Generative Adversarial Networks. In *ACSAC* (2021). doi:10.1145/3485832.3485838
- Malhar Jere, Tyler Farnan, and Farinaz Koushanfar. 2021. A Taxonomy of Attacks on Federated Learning. In *IEEE Secur. Priv.* (2021). doi:10.1109/MSEC.2020.3039941
- Dihong Jiang, Sun Sun, and Yaoliang Yu. 2023. Functional Renyi Differential Privacy for Generative Modeling. In *NeurIPS* (2023). http://papers.nips.cc/paper_files/paper/2023/hash/2f9ee101e35b890d9eae79ee27bcd69a-Abstract-Conference.html
- Quan Ju, Rongqing Xia, Shuhong Li, and Xiaojuan Zhang. 2024. Privacy-Preserving Classification on Deep Learning with Exponential Mechanism. In *Int. J. Comput. Intell. Syst.* (2024). doi:10.1007/S44196-024-00422-X
- Anan Kabaha and Dana Drachler-Cohen. 2024. Verification of Neural Networks' Global Robustness. In *OOPSLA1* (2024). doi:10.1145/3649847
- Mostafa Kahla, Si Chen, Hoang Anh Just, and Ruoxi Jia. 2022. Label-Only Model Inversion Attacks via Boundary Repulsion. In *CVPR* (2022). doi:10.1109/CVPR52688.2022.01462
- Peter Kairouz, Sewoong Oh, and Pramod Viswanath. 2015. The Composition Theorem for Differential Privacy. In *ICML* (2015). <http://proceedings.mlr.press/v37/kairouz15.html>
- Guy Katz, Clark W. Barrett, David L. Dill, Kyle Julian, and Mykel J. Kochenderfer. 2017. Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks. In *CAV* (2017). doi:10.1007/978-3-319-63387-9_5
- Guy Katz, Derek A. Huang, Duligur Ibeling, Kyle Julian, Christopher Lazarus, Rachel Lim, Parth Shah, Shantanu Thakoor, Haoze Wu, Aleksandar Zeljic, David L. Dill, Mykel J. Kochenderfer, and Clark W. Barrett. 2019. The Marabou Framework for Verification and Analysis of Deep Neural Networks. In *CAV* (2019). doi:10.1007/978-3-030-25540-4_26
- Amin Kharraz, Zane Ma, Paul Murley, Charles Lever, Joshua Mason, Andrew Miller, Nikita Borisov, Manos Antonakakis, and Michael D. Bailey. 2019. Outguard: Detecting In-Browser Covert Cryptocurrency Mining in the Wild. In *WWW* (2019). doi:10.1145/3308558.3313665
- Myeongseob Ko, Ming Jin, Chenguang Wang, and Ruoxi Jia. 2023. Practical Membership Inference Attacks Against Large-Scale Multi-Modal Models: A Pilot Study. In *ICCV* (2023). doi:10.1109/ICCV51070.2023.00449
- Yann LeCun, Leon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. In *Proc. IEEE* (1998). doi:10.1109/5.726791
- Kyumin Lee, Brian David Eoff, and James Caverlee. 2011. Seven Months with the Devils: A Long-Term Study of Content Polluters on Twitter. In *AAAI* (2011). <http://www.aaai.org/ocs/index.php/ICWSM/ICWSM11/paper/view/2780>

- Zheng Li and Yang Zhang. 2021. Membership Leakage in Label-Only Exposures. In *CCS* (2021). doi:10.1145/3460120.3484575
- Jiacheng Liang, Ren Pang, Changjiang Li, and Ting Wang. 2024. Model Extraction Attacks Revisited. In *ASIA CCS* (2024). doi:10.1145/3634737.3657002
- Jiaxiang Liu, Yunhan Xing, Xiaomu Shi, Fu Song, Zhiwu Xu, and Zhong Ming. 2024. Abstraction and Refinement: Towards Scalable and Exact Verification of Neural Networks. In *ACM Trans. Softw. Eng. Methodol.* (2024). doi:10.1145/3644387
- Stuart P. Lloyd. 1982. Least squares quantization in PCM. In *Information Theory, IEEE Transactions on* 28.2: 129-137 (1982). doi:10.1109/TIT.1982.1056489
- Jingyue Lu and M. Pawan Kumar. 2020. Neural Network Branching for Neural Network Verification. In *ICLR* (2020). <https://openreview.net/forum?id=B1evfa4tPB>
- Denis Mazzucato and Caterina Urban. 2021. Reduced Products of Abstract Domains for Fairness Certification of Neural Networks. In *Static Analysis - 28th International Symposium, SAS (Lecture Notes in Computer Science)*. Springer. doi:10.1007/978-3-030-88806-0_15
- Luca Melis, Congzheng Song, Emiliano De Cristofaro, and Vitaly Shmatikov. 2019. Exploiting Unintended Feature Leakage in Collaborative Learning. In *SP* (2019). doi:10.1109/SP.2019.00029
- Ilya Mironov. 2017. Rényi Differential Privacy. In *CSF* (2017). doi:10.1109/CSF.2017.11
- Mark Niklas Müller, Gleb Makarchuk, Gagandeep Singh, Markus Püschel, and Martin T. Vechev. 2022. PRIMA: general and precise neural network certification via scalable convex hull approximations. In *POPL* (2022). doi:10.1145/3498704
- Tabitha Ogilvie. 2024. Differential Privacy for Free? Harnessing the Noise in Approximate Homomorphic Encryption. In *CT-RSA* (2024). doi:10.1007/978-3-031-58868-6_12
- Alessandro De Palma and et al. 2021. Improved Branch and Bound for Neural Network Verification via Lagrangian Decomposition. *arXiv:2104.06718* (2021). <https://arxiv.org/abs/2104.06718>
- Nicolas Papernot. 2018. A Marauder's Map of Security and Privacy in Machine Learning: An overview of current and future research directions for making machine learning secure and private. In *AISeC* (2018). doi:10.1145/3270101.3270102
- Brandon Paulsen, Jingbo Wang, and Chao Wang. 2020a. ReluDiff: differential verification of deep neural networks. In *ICSE* (2020). doi:10.1145/3377811.3380337
- Brandon Paulsen, Jingbo Wang, Jiawei Wang, and Chao Wang. 2020b. NEURODIFF: Scalable Differential Verification of Neural Networks using Fine-Grained Approximation. In *ASE* (2020). doi:10.1145/3324884.3416560
- Roie Reshef, Anan Kabaha, Olga Seleznova, and Dana Drachsler-Cohen. 2024. Verification of Neural Networks' Local Differential Classification Privacy. In *VMCAI* (2024). doi:10.1007/978-3-031-50521-8_5
- Arnaud Grivet Sébert. 2023. *Combining differential privacy and homomorphic encryption for privacy-preserving collaborative machine learning. (Approches combinant confidentialité différentielle et chiffrement homomorphe pour la protection des données en apprentissage automatique collaboratif)*. Ph.D. Dissertation. University of Paris-Saclay, France. <https://tel.archives-ouvertes.fr/tel-04223076>
- Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. 2017. Membership Inference Attacks Against Machine Learning Models. In *SP* (2017). doi:10.1109/SP.2017.41
- Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin T. Vechev. 2019. Boosting Robustness Certification of Neural Networks. In *ICLR* (2019). <https://openreview.net/forum?id=HJgeEh09KQ>
- Liwei Song, Reza Shokri, and Prateek Mittal. 2019. Privacy Risks of Securing Machine Learning Models against Adversarial Examples. In *CCS* (2019). doi:10.3390/S24196161
- Jordi Soria-Comas, Josep Domingo-Ferrer, David Sánchez, and David Megías. 2017. Individual Differential Privacy: A Utility-Preserving Formulation of Differential Privacy Guarantees. In *IEEE Trans. Inf. Forensics Secur.* (2017). doi:10.1109/TIFS.2017.2663337
- Vincent Tjeng, Kai Y. Xiao, and Russ Tedrake. 2019. Evaluating robustness of neural networks with mixed integer programming. In *ICLR* (2019). <https://openreview.net/forum?id=HyGIdiRqtm>
- Shubham Ugare, Debangshu Banerjee, Sasa Misailovic, and Gagandeep Singh. 2023. Incremental Verification of Neural Networks. In *PLDI* (2023). doi:10.1145/3591299
- Shubham Ugare, Gagandeep Singh, and Sasa Misailovic. 2022. Proof transfer for fast certification of multiple approximate neural networks. In *OOPSLA* (2022). doi:10.1145/3527319
- Caterina Urban, Maria Christakis, Valentin Wüstholtz, and Fuyuan Zhang. 2019. Perfectly Parallel Fairness Certification of Neural Networks. *Proc. ACM Program. Lang.* 4, OOPSLA (2019). doi:10.1145/3428253
- Robert J. Vanderbei. 1996. *Linear Programming: Foundations and Extensions*. (1996). doi:10.1007/978-0-387-74388-2
- Fuyi Wang, Leo Yu Zhang, Lei Pan, Shengshan Hu, and Robin Doss. 2022c. Towards Privacy-Preserving Neural Architecture Search. In *ISCC* (2022). doi:10.1109/ISCC55528.2022.9913012
- Shiqi Wang, Huan Zhang, Kaidi Xu, Xue Lin, Suman Jana, Cho-Jui Hsieh, and J Zico Kolter. 2021. Beta-CROWN: Efficient Bound Propagation with Per-neuron Split Constraints for Neural Network Robustness Verification. In *NeurIPS* (2021). <https://proceedings.neurips.cc/paper/2021/hash/fac7fead96dafceaf80c1dafae82a4-Abstract.html>

- Yu-Xiang Wang. 2019. Per-instance Differential Privacy. In *J. Priv. Confidentiality* (2019). <https://journalprivacyconfidentiality.org/index.php/jpc/article/download/662/675/1038>
- Yufeng Wang, Min Gu, Jianhua Ma, and Qun Jin. 2020. DNN-DP: Differential Privacy Enabled Deep Neural Network Learning Framework for Sensitive Crowdsourcing Data. *IEEE Transactions on Computational Social Systems* 7, 1, 215–224. doi:10.1109/TCSS.2019.2950017
- Zhilu Wang, Chao Huang, and Qi Zhu. 2022a. Efficient Global Robustness Certification of Neural Networks via Interleaving Twin-Network Encoding. In *DATE 2022* (2022). doi:10.23919/DATE54114.2022.9774719
- Zhilu Wang, Yixuan Wang, Feisi Fu, Ruochen Jiao, Chao Huang, Wenchao Li, and Qi Zhu. 2022b. A Tool for Neural Network Global Robustness Certification and Training. In <https://doi.org/10.48550/arXiv.2208.07289> 2022 (2022). doi:10.48550/ARXIV.2208.07289
- Wayne L. Winston. 1991. Operations Research: Applications and Algorithms. (1991). doi:10.1002/net.3230180310
- Haoze Wu and et al. 2020. Parallelization Techniques for Verifying Neural Networks. In *FMCAD* (2020). doi:10.34727/2020/ISBN.978-3-85448-042-6_20
- Yi Xie, Jie Zhang, Shiqian Zhao, Tianwei Zhang, and Xiaofeng Chen. 2024. SAME: Sample Reconstruction against Model Extraction Attacks. In *AAAI* (2024). doi:10.1609/AAAI.V38I18.29974
- Guowen Xu, Hongwei Li, Yun Zhang, Shengmin Xu, Jianting Ning, and Robert H. Deng. 2022. Privacy-Preserving Federated Deep Learning With Irregular Users. *IEEE Trans. Dependable Secur. Comput.* 2, 1364–1381. doi:10.1109/TDSC.2020.3005909
- Jiayuan Ye and Reza Shokri. 2022. Differentially Private Learning Needs Hidden State (Or Much Faster Convergence). In *NeurIPS* (2022). http://papers.nips.cc/paper_files/paper/2022/hash/04b42392f9a3a16aea012395359b8148-Abstract-Conference.html
- I-Cheng Yeh. 2016. Default of credit card clients. UCI Machine Learning Repository. <https://doi.org/10.24432/C55S3H>
- Ligeng Zhu, Zhijian Liu, and Song Han. 2019. Deep Leakage from Gradients. In *NeurIPS* (2019). <https://proceedings.neurips.cc/paper/2019/hash/60a6c4002cc7b29142def8871531281a-Abstract.html>

A Proofs

LEMMA 3.2. *Given a dataset D , a query $f_{x,\mathcal{T},\tilde{N}}$, and a privacy budget ϵ , the exponential mechanism with our utility function $u_{x,\mathcal{T},\tilde{N}}$ is ϵ -iDP.*

PROOF. Generally, given a dataset D , a set of outputs $\mathcal{R}$ and a utility function $u : \mathcal{D} \times \mathcal{R} \rightarrow \mathbb{R}$, the exponential mechanism $\mathcal{A}$ samples and returns $r \in \mathcal{R}$ with probability $\frac{\exp(\epsilon u(D,r)/(2\Delta u))}{\sum_{r' \in \mathcal{R}} \exp(\epsilon u(D,r')/(2\Delta u))}$. In our setting, the set of outputs is the label set $\mathcal{R} = C$. To prove iDP, we show that for any dataset D' adjacent to D (i.e., differing by one data point) and for every possible observed output $\mathcal{O} \subseteq \text{Range}(\mathcal{A})$, the following holds: $e^{-\epsilon} \cdot \Pr[\mathcal{A}(D') \in \mathcal{O}] \leq \Pr[\mathcal{A}(D) \in \mathcal{O}] \leq e^{\epsilon} \cdot \Pr[\mathcal{A}(D') \in \mathcal{O}]$. In our setting, the observed outputs $\mathcal{O} \subseteq \text{Range}(\mathcal{A})$ are sets of a single class $\{c\} \subseteq C$. Thus, we prove that for any dataset D' adjacent to D and for every $c \in C$, the following holds:

$$e^{-\epsilon} \cdot \Pr[\mathcal{A}(D', u_{x,\mathcal{T},\tilde{N}}, C) = c] \leq \Pr[\mathcal{A}(D, u_{x,\mathcal{T},\tilde{N}}, C) = c] \leq e^{\epsilon} \cdot \Pr[\mathcal{A}(D', u_{x,\mathcal{T},\tilde{N}}, C) = c]$$

We prove $\Pr[\mathcal{A}(D, u_{x,\mathcal{T},\tilde{N}}, C) = c] \leq e^{\epsilon} \cdot \Pr[\mathcal{A}(D', u_{x,\mathcal{T},\tilde{N}}, C) = c]$ (the proof for the other inequality is similar). The proof is very similar to the proof of [Dwork and Roth 2014, Theorem 3.10]. To simplify notation, we write $u \triangleq u_{x,\mathcal{T},\tilde{N}}$. We remind that $\mathcal{A}$ is invoked with $\Delta u = 1$.

$$\begin{aligned} \frac{\Pr[\mathcal{A}(D, u, C) = c]}{\Pr[\mathcal{A}(D', u, C) = c]} &= \frac{\left(\frac{\exp(\epsilon u(D,c)/2)}{\sum_{c' \in C} \exp(\epsilon u(D,c')/2)} \right)}{\left(\frac{\exp(\epsilon u(D',c)/2)}{\sum_{c' \in C} \exp(\epsilon u(D',c')/2)} \right)} = \\ &= \exp\left(\frac{\epsilon(u(D,c) - u(D',c))}{2}\right) \cdot \left(\frac{\sum_{c' \in C} \exp(\epsilon u(D',c')/2)}{\sum_{c' \in C} \exp(\epsilon u(D,c')/2)} \right) \leq \exp(\epsilon) \end{aligned}$$

The last transition follows from:

- The left term is equal to $\exp(\frac{\epsilon}{2})$: This follows since by the definition of our utility function, for every D' and c , we have $u(D,c) - u(D',c) \leq 1$.
- The right term is equal to $\exp(\frac{\epsilon}{2})$: This follows by the quotient inequality which states that if we have K values $\alpha_1, \dots, \alpha_K$ and K respective values $\beta_1, \dots, \beta_K$ and for every $i \in [K]$, we have $\alpha_i/\beta_i \leq \gamma$, for some γ , then $\frac{\alpha_1 + \dots + \alpha_K}{\beta_1 + \dots + \beta_K} \leq \gamma$. In our right term, we have for every $c' \in C$, $\alpha_{c'} = \exp(\epsilon u(D',c')/2)$ and $\beta_{c'} = \exp(\epsilon u(D,c')/2)$. That is $\frac{\alpha_{c'}}{\beta_{c'}} = \frac{\exp(\epsilon u(D',c')/2)}{\exp(\epsilon u(D,c')/2)} = \exp(\epsilon(u(D',c') - u(D,c'))/2)$. Like in the left term, since $u(D',c') - u(D,c') \leq 1$ we get $\alpha_{c'}/\beta_{c'} \leq \exp(\epsilon/2) \triangleq \gamma$, and the claim follows. $\square$

THEOREM 6.1. *Given a dataset D , a training algorithm $\mathcal{T}$, a network architecture $\tilde{N}$, and a privacy budget ϵ , LUCID-Inference provides an ϵ -iDP version of every label-only query $f_{x,\mathcal{T},\tilde{N}}(D)$, for $x \in [0, 1]^d$, without losing accuracy for queries whose x has a confidence larger than the iDP-DB.*

PROOF SKETCH. By the correctness of LUCID-Compute, its computed iDP-DB provides a sound overapproximation to the set of leaking inputs. Thus, if an input x has classification confidence above the respective iDP-DB, its query satisfies 0-iDP and thus also ϵ -iDP. For such inputs, LUCID does not add noise, and thereby does not lose accuracy. Otherwise, the exponential mechanism is employed and thereby ϵ -iDP is guaranteed by Lemma 3.2. $\square$

B Evaluation: Dataset Description

In this section, we describe our evaluated datasets.

Datasets. We evaluate LUCID over four datasets:

- **Cryptojacking** [Kharraz et al. 2019] (Crypto): Cryptojacking refers to websites that exploit user resources for cryptocurrency mining. This dataset includes 2,000 malicious websites and 2,000 benign websites, where 2,800 are used for training and 1,200 for testing. Each website has seven features (e.g., websocket) and is labeled as malicious or benign.
- **Twitter Spam Accounts** [Lee et al. 2011] (Twitter): Twitter spam accounts refers to accounts that distribute unsolicited messages to users. This dataset includes 20,000 spam accounts and 20,000 benign accounts, split into 36,000 accounts for training and 4,000 for testing. Each data point has 15 features (e.g., followers and tweets) and is labeled as spam or benign.
- **Adult Census** [Becker and Kohavi 1996] (Adult): This dataset is used for predicting whether an adult's annual income is over \$50,000. It includes 48,842 data points, split into 32,561 for training and 16,281 for testing. Each data point has 14 features (e.g., age, education, occupation, and working hours) and is labeled as yes or no.
- **Default of Credit Card Clients** [Yeh 2016] (Credit): The Taiwan Credit Card Default dataset is used for predicting whether the default payment will be paid. It includes 30,000 client records, split into 24,000 for training and 6,000 for testing. A data point has 23 features (e.g., bill amounts, age, education, marital status) and is labeled as yes or no.