RIGOROUS SECURITY ANALYSIS OF MACHINE LEARNING SYSTEMS

by

TEODORA BĂLUŢĂ

(B.Sc. & M.Sc., University Politehnica of Bucharest, Romania)

A THESIS SUBMITTED

FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

DEPARTMENT OF COMPUTER SCIENCE

NATIONAL UNIVERSITY OF SINGAPORE

2023

Thesis Advisors:

Associate Professor Prateek Saxena, Main Thesis Advisor
Associate Professor Kuldeep S. Meel, Co-Advisor

Examiners:

Associate Professor Arnab Bhattacharyya
Associate Professor Reza Shokri

Declaration

I hereby declare that this thesis is my original work and it has been written by me in its entirety. I have duly acknowledged all the sources of information which have been used in the thesis.

This thesis has also not been submitted for any degree in any university previously.

Teodora Băluţă

26 December 2023

To my parents and sister

Acknowledgments

I am extremely grateful to my advisors, Prateek Saxena and Kuldeep S. Meel, for their mentoring and unwavering support throughout my PhD. Their feedback and insights have been instrumental to the technical ideas in this thesis. I am indebted to Prateek for his patience and encouragement in carving and pursuing my research direction. Prateek has always been there to share his constructive criticism and wisdom so generously. His passion for scientific research and leadership have markedly shaped my character. I thank Kuldeep for teaching me the beauty of formal methods and the invaluable lesson of incremental improvements. His kind guidance and faith in my ideas have had a significant impact on my growth.

I thank my thesis committee members, Arnab Bhattacharyya and Reza Shokri. Their invaluable feedback has improved the arguments presented in this thesis. I would like to gratefully acknowledge Zhenkai Liang's support throughout the years.

I had the great pleasure of working with amazing collaborators and I thank them for contributing to the work done in this thesis (listed alphabetically): Divesh Aggarwal, Zheng Leong Chua, S. Hitarth, Racchit Jain, Ivica Nikolić, Shiqi Shen, Shweta Shinde, Shruti Tople. A big thanks to Daniel Tarlow, Fabian Pedregosa, Pascal Lamblin and Karolina Dziugaite for hosting me for a fall internship at Google Brain, in Montreal, and their guidance.

The PhD grind was made more joyful than I could have hoped for thanks to my friends and lab mates at National University of Singapore: Zheng Leong Chua, Shiqi Shen, Shweta Shinde, Aashish Kolluri, Shruti Tople, Bo Wang, Kareem Shehata, Jason Zhijingcheng Yu, Ruishi Li. I have been fortunate to have friends in two labs, so thank you for the wonderful memories and feedback on many aspects of my research: Priyanka Golia, Yash Pote, Jiong Yang, Suwei Yang, Arijit Shaw, Mate Soos, Mohimenul Kabir, Bishwamittra Ghosh, Tim van Bremen, Anna Latour, Paulius Dilkas and Yacine Izza. A big shoutout to Andreea and Erica for being such pillars of support during many difficult moments, and for all the happy memories together. There is a special place in my heart for Trini, Yurika, Prachi, Ana-Maria, Tommi and everyone who used to be in the Turing-Noble house. Thanks to my friends in Romania for the kind encouragement, unforgettable fun whenever I go back and for always patiently explaining all the cultural jokes I missed out on since

I have been away.

I spent my last year of PhD mostly on the faculty job search, and I have to thank my family and friends for helping me navigate this stressful period. I am thankful to Ilya Sergey, Jialin Li, and Jonathan Scarlett for helping me improve my talk. Special thanks go to Priyanka Golia, Shweta Shinde and Suguman Bansal for their very insightful feedback throughout. Umang Mathur and Manuel Rigger have helped me navigate the process systematically for which I am grateful. I am deeply grateful to Divesh Aggarwal, Reza Shokri and Shruti Tople for their guidance and recommendation letters.

Throughout all the ups and downs of PhD, Zheng Leong has been there to celebrate with me and uplift me with kindness, comfort, fun surprises and the occasional pep talks. I am grateful for his unwavering support and belief in me. I would like to thank Zheng Leong's family for their support, and especially his mum for always being there with a smile.

Were it not for the unconditional love and support of my parents and sister, I would not have been able to pursue this journey across the world to do research full-time. A big thank you to my sister for her positive outlook, and selflessness. Mum, dad, thank you for believing in me, allowing me to be myself, and always being there for me. Mum, you have always inspired me to work hard, persevere and strive to carve my own way. Dad, thank you for being the dreamer you are, and for trusting my decisions. Though you never pushed or asked for anything: This thesis is for you, thank you!

Contents

Abstract

Rigorous Security Analysis of Machine Learning Systems

by

Teodora Băluţă

Doctor of Philosophy in Computer Science

National University of Singapore

Machine learning systems, particularly neural networks, have drastically transformed the technological landscape. Despite these impressive advances, security and privacy attacks highlight how these systems fail under the presence of adversaries, and even due to unintentional failures. This has generated research into testing and verifying desirable properties beyond accuracy on a test set such as robustness, fairness, and privacy. However, both conceptual and algorithmic gaps remain that hinder refutability of security claims in machine learning.

In this thesis, we tackle these challenges towards *rigorous security analysis*. On a conceptual front, we propose three novel queries for checking properties of interest for neural networks: counting queries, causal queries, and decisional convergence queries. On an algorithmic front, we give procedures with theoretical guarantees to solve these queries. First, counting queries ask what is the cardinality of the set of inputs that satisfy a given property. We develop diverse techniques to solve these queries and provide soundness guarantees for our procedures. Specifically, our techniques have probably approximately correct guarantees, in that the returned count is close to the ground truth within a user-specified tolerance and with a user-specified probability. We show that counting queries enable checkable properties of robustness, fairness and susceptibility to trojan attacks, in the presence of failures. Second, why certain machine learning attacks and defenses seem effective is not well understood. We propose causal queries that answer how various variables concerning the training process relate with each other and a given adversary. We specifically use causal queries to investigate the relationship between generalization and membership inference attacks, whose connections have been hypothesized in prior work. Finally, we propose decisional convergence queries, which aim to answer if an approximate

property (determined by a parameter) is satisfied even as its approximation reaches the limit. We show that we can formulate convergence queries to check forgeability, a security property of stochastic gradient descent, the de-facto training algorithm.

List of Tables

List of Figures

Chapter 1

Introduction

Machine learning (ML), and in particular deep learning, has drastically transformed the technological landscape, permeating into many application domains [219, 86, 48, 183]. As systems that incorporate machine learning components, i.e., ML systems, grow in importance, their security has become a major concern.

Security of ML systems has been predominantely argued via attack procedures in the last decade. Attacks highlight concerns that neural networks may fail in unexpected ways [222, 24, 165], be manipulated by adversaries [141, 143, 146, 76] or leak private information about the training data [21, 207, 159, 98] or the machine learning model [23, 59, 119]. In traditional software security, security violations can then be formally specified and verified to make rigorous security claims. However, neural networks are stochastically trained, and they may run on inputs drawn from an unknown distribution at inference time. The nature of vulnerabilities in ML is thus often intrinsically tied to learning, i.e., they are "features, not bugs" [56, 237, 99]. As a result, even distinguishing exactly between the intended versus unintended behavior of the ML system is unclear. Unless we precisely define security vulnerabilities though, it is impossible to rigorously verify security, or the lack thereof, only from experimental attack procedures.

It is worth recalling that the designers of ML systems often make statistical claims about their behavior, i.e., a given system is claimed to satisfy properties of interest with high probability but not always. Properties of interest are thus often probabilistic, and revolve around characterizing expected moments and deviations of random variables such as those defined over choices of the input. Existing principled approaches such as abstract interpretation [212, 67] or satisfiability modulo theories [112], however, are qualitative. These do not consider the cardinality of the

set of inputs that satisfy the property. Moreover, certain properties of interest are defined over random variables involved in the training process such as the choice of hyper-parameters, the choice of training dataset, the sampling of batches, the input data distribution, and so on. This complex interplay of many inputs to the training process affect the desirable property. Even if one were to fix all these inputs to the training process, making it deterministic, security properties over such concrete training execution have been studied in the approximate regime, without precise definitions, and sound procedures to verify them [105, 261, 226]. Thus, many of the existing security claims are *non-refutable*, either because of the lack of formal security definitions, limited abstractions for reasoning about the training process, or ad-hoc testing procedures without theoretical guarantees.

Thesis Statement. This thesis puts forward foundations for *rigorous security analysis of ML systems*: We give formal security definitions, abstractions and sound procedures (with guarantees). On a conceptual front, we propose three novel queries to evaluate security and privacy properties of neural networks: *counting queries, causal queries* and *decisional convergence queries*. On an algorithmic front, we develop sound algorithms to answer each category of queries, which draw upon propositional logic, hypothesis testing, causal reasoning and linear algebraic techniques to give efficient algorithms. At the same time, we demonstrate the utility of these queries in several applications, bridging between formal security definitions and abstract models of reasoning about training. We show that our queries can refute or support new and existing claims about the behavior of neural networks in the presence of adversaries, providing concrete advances towards *refutability in ML security*. These query types are unlike satisfiability queries over a property; they ask questions about cardinality of the set of examples satisfying, the contribution of each random variables to the observed outcome, and the rate of convergence as we constrain the property gradually towards a limit.

1.1 Gaps in Rigorous Security Analysis

Machine learning security is a nascent field, with much of the past decade dictated by a cat-and-mouse game of attacks followed by defenses, followed by attacks. More principled approaches such as formal verification [111, 228, 4, 10, 93, 205, 156, 49,

130] or provable defenses [35, 153, 1, 248] for neural networks have been proposed. Despite these efforts, rigorous security analysis for ML systems eludes many security concerns. We identify gaps in existing works on all fundamental aspects of rigorous security, and we give concrete examples below.

Gaps in Definitions. Several lawsuits have been filed claiming copyright infringement recently against ML models that could cost billions of dollars. To address both intellectual property concerns and ensure non-repudiation properties of training data, we require proofs of model ownership and creation using a particular dataset beyond reasonable doubt. Existing works show that even if one were to record SGD executions, such proofs are repudiable [226, 261]. They show that one can come up with a different set of samples called a forgery, that result in approximately equal training step updates. However, we show that if one replaces a training step with an approximately forged one, the difference between the forged and original traces *diverges* with subsequent training. Because of this, after subsequent training, forgeries that result in approximately equal updates are detectable. We argue that forgery needs to be precisely defined with algebraic properties, under exact equality.

Gaps in Abstractions for Training. The lack of clarity in setups and adversary capabilities and goals has been recently highlighted for privacy evaluations [195]. Membership inference attacks are meta-procedures for evaluating privacy defined over the outcomes of the training algorithm (i.e., the de-facto stochastic gradient descent) [207, 142, 196, 135, 252, 260]. However, the parameters of these algorithms are chosen in an ad-hoc manner, or by heuristically seaching through possible combinations that yield better performance, i.e., hyper-parameter tuning. Thus, even if there exists a clear definition of what property membership inference attacks aim to show violations of (e.g., membership in a training set or attribute inference, depending on the parametrization of the attack), along with a threat model and a procedure, it is difficult to formally check the statements about the attack procedure, namely whether something has been memorized, or it is generalized. We show that without explicit assumptions about the setup or the process to generate data of these meta-procedures, it can lead to paradoxes in empirical evaluations of the attack. One may ask: why not derive these from theory, why do we need a causal model? The answer is that it is not trivial to mathematically model the relationship between

all these variables for deep neural networks (of more than 1-2 layers) without making some assumptions about the distribution of data [251, 194], or the sample regime (asymptotic regimes are not checkable), and so on. More concretely, the relationship between training choices and generalization of deep neural networks trained with stochastic gradient descent is an open problem. This gap would be filled if we had a model to reason about the interaction between of the setup. Only then can we start checking statements about causes of membership inference attacks, and start drawing connections between them and generalization measures.

Gaps in Procedures. Many properties of interest for ML models are probabilistic, i.e., they aim to measure moments of random variables defined over distributions of inputs and outputs of the ML model. Fairness metrics often establish a relationship between the expected outputs of an ML model and constraints over the inputs [151]. For example, equalized odds aims to evaluate whether the probability of getting the same outcomes for different groups are same [83]. Individual fairness [46], or statistical parity [241] aim to ensure that the outputs of the ML model or training algorithm are independent from the group an individual belongs to. Other security properties such as trojan attack susceptibility [141] and robustness to adversarial examples [222] also fall in this category. Despite the statistical nature of ML, most of the existing works evaluate these properties via attack procedures that find counter-examples of the desirable property [19], or via verification that give a decisional answer if the property is always satisfied, over the all input examples [67, 49]. Such techniques check existential queries. Evaluations of quantitative or statistical queries are either ad-hoc testing procedures, without guarantees [65, 82], guarantees in the asymptotic regime [4], or with respect to a dataset [63]. Instead, for rigorous security guarantees, our goal is to have sound procedures in the finite sample regime. Such procedures should return answers that can only deviate by a user-specified tolerance from the ground truth, with high probability.

1.2 Formalizing Security via Novel Queries

The conceptual contributions of this thesis are based on formulating three novel types of queries to enable formally checkable properties for secure machine learning. Thus, each type of query returns answers with a guarantee. First, we propose

queries called *counting queries* that estimate the cardinality of the set of counter-examples to the property P. This is fundamentally different from asking if the set is empty, i.e., the property is satisfied for all, or there exist counter-examples. These types of queries allow quantification of security properties in the presence of failures and adversaries. Second, we introduce *causal queries* that answer "what if" questions connecting random variables that appear in the training process and attack procedures. Thus, the goal of these queries is to enable supporting or refuting existing hypotheses about adversaries, and connecting them to aspects of the learning process. Third, we introduce *convergence queries* that ask if the property is still true in the limit. We describe the query formulations below, and detail the application of these queries for checking properties to Section 1.3.

Counting Queries. Given a neural network f, an input x and a logical property P, a counting query asks how many assignments $\mathrm{P}(f, x)$ has when the input x is sampled according to a probability distribution. As an illustrative example, let us take robustness to adversarial examples. For a randomly chosen input x that is correctly classified by the neural network f, adversarial examples x_{adv} are carefully crafted by adding a small imperceptible perturbation ϵ to x, i.e., $||x_{adv} - x||_p \leq \epsilon$ in L_p norm, that is enough to make the neural network misclassify, $f(x) \neq f(x_{adv})$. A decisional query about robustness determines if the property $\mathrm{P}(f, x) := (f(x) = f(x_{adv}))$ is satisfied over the set of inputs x_{adv} such that $||x_{adv} - x||_p \leq \epsilon$, or there exist inputs for which P is false, i.e., there exist adversarial examples. The counting query formulation for the robustness property asks how many adversarial examples exist within a perturbation size ϵ of an input x, when adversarial examples are sampled according to some probability distribution. Counting robustness asks what the cardinality of the set of randomly sampled inputs x_{adv} is, under $\neg\mathrm{P}(f, x) := (f(x) \neq f(x_{adv}))$. A variant of the counting query above revolves around bounding the cardinality: Given a logical property P specified over a space of inputs and outputs of a deep neural network and a numerical threshold θ, decide whether P is true for less than θ fraction of the inputs, where the inputs are sampled according to a distribution. For both of these queries, we ensure soundsness defined as a probably approximately correct [240] answer from the ground truth, i.e., the true cardinality under the distribution. These types of queries thus offer a general framework for checking

several properties of neural networks such as robustness, fairness, and susceptibility to trojan attacks (see Chapters 2 and 3).

Causal Queries. Given a training algorithm with many random variables defined over choices of hyper-parameters such as the training dataset size and the model's number of parameters, the causal queries are "what if" queries. It consists of two variables: the potential cause variable, called the treatment variable X, and the desired outcome variable Y. Abstractly, the causal query encodes the average effect of the treatment variable X on the outcome Y, when the treatment variables takes on a particular value. This is different from estimating conditional probabilities, i.e., what the probability of an event given that we observe another random variable take a particular value. In order to estimate causal effects, one must be able to intervene and change the value of the variable, even if these values are not observed under the given distribution. We specifically introduce such queries to evaluate the relationship between properties of the training algorithm governed by these many random variables and membership inference attacks accuracy [207] (see Chapter 4).

Decisional Convergence Queries. Many properties of interest have a set of approximation parameters such as the perturbation size in robustness queries [222] or the acceptable distance between the model parameters in forgeability [226]. Decisional convergence queries aim to determine whether a property is satisfied as the approximation error approaches zero, indicating convergence towards the desired behavior. Counting and qualitative queries focus on properties of neural networks for a fixed approximation parameter, and do not answer whether the property is true as the approximation parameter approaches convergence, i.e., for all approximations including the smallest one. The convergence queries take in as input a neural network f and a property P that is defined with respect to a limit of the approximation ϵ and over inputs x to f. The query is decisional as it answer whether the property P holds over inputs x as ϵ approaches a limit.

1.3 Utility in Security Applications

The novel query formulations allow us to quantify several properties of neural networks, and test several existing hypotheses, which are being actively studied.

Robustness. In counting queries for robustness, we want to estimate the probability

of the event of sampling points close to x under the uniform distribution, where adversarial examples x_{adv} are defined as $||x_{adv} - x||_\infty \leq \epsilon$ such that $f(x) \neq f(x_{adv})$. Let us denote the number of adversarial examples x_{adv} as adversarial density. We show an empirical attack-agnostic metric for estimating robustness of a given deep neural network and input x called adversarial hardness. It is the highest perturbation bound for which the adversarial density is below a suitably low θ. We can search empirically for the highest perturbation bound ϵ_{Hard} for which a sound certifier says "Yes" when queried with different θ, implying that f has suitably low density of adversarial examples for perturbation bounds below ϵ_{Hard}. Adversarial hardness is a measure of the difficulty of finding an adversarial example by uniform sampling. We find that this measure strongly correlates with perturbation bounds produced by prominent white-box attacks [145, 24]. Given this strong correlation, we can effectively use adversarial hardness as a proxy for perturbation sizes obtained from specific attacks, when comparing the relative robustness of two deep neural networks. Moreover, it implies that the distribution of hard instances for specialized attack procedures corresponds to that of lower adversarial density instances.

Generalization vs. Membership Inference. To showcase the utility of our approach, we study 6 well-known membership inference attacks and 2 defenses for deep neural networks trained using standard stochastic gradient descent training procedures. We analyze a list of intuitive "root causes" which have been suggested in prior works and formally specify them as 9 causal hypotheses. We show that two stochastic parameters inherent in the training process, namely Bias and Variance, can quantitatively predict both generalization measures and membership inference attack performance, providing new insights. These factors play a disproportionately larger role in explaining membership inference attack performance, compared to other factors such as model complexity, dataset size, or even generalization measures themselves. Our approach offers a more nuanced lens to connect generalization and membership inference attack accuracy from that offered by prior works [19, 144, 254].

Forgeability. We analyze one fundamental property of the de-facto training algorithm stochastic gradient descent, namely *forgeability*: Is it possible to obtain the same model parameters (outputs) from two different minibatches (inputs)? If yes,

then we say that the output is forgeable. Forgeability has emerged in the context of several applications such as machine unlearning [226], model ownership [105, 261, 53], and membership inference tests [116, 117]. If a model is forgeable, certain training samples used could have been replaced with other samples without changing the output. Despite its emerging applications, characterizing forgery in practice has remained an intriguing open problem. Prior work shows that it is possible to forge intermediate model parameters within certain approximation error (under a vector norm) and conjectured that forgery could be made exact with zero error, but this remains a conjecture hitherto [226, 117, 116]. Creating exact forgery of model checkpoints has not been demonstrated yet. Thus, we formulate the existence of exact forgeries as decisional convergence query: does forgery hold in the limit, when the approximation goes to zero? We present the first theoretical impossibility result for exact forgery of stochastic gradient descent execution states. Our theorem specifies conditions under which traces are provably unforgeable, which are efficiently checkable on concrete stochastic gradient descent executions, given the training dataset and model parameters.

1.4 Sound Procedures for Solving Queries

While the three proposed queries are useful in checking properties of neural networks, in order to solve such queries in practice requires computational tractability. This thesis systematically analyzes their computational hardness, and gives practical algorithms to answer them.

1.4.1 Solving Counting Queries

Deciding whether a property of interest for a given neural network holds or not is NP-hard [111]. Intuitively, algorithms that are able to return the number of examples that satisfy a given property (even if it is zero or unsatisfiable) should be at least as hard at the decision problem. In this dissertation, we reduce counting the number of satisfying assignments of logical formulae to counting queries for properties of a class of neural networks. Thus, we show that solving counting queries of this type belong to the complexity class #P-hard.

Our first technical contribution of this dissertation is an analysis framework,

which models the given set of neural networks $\mathcal{N} = \{f_1, \ldots, f_n\}$ and the property P as a set of logical constraints ϕ, such that the problem of quantifying how often $\mathcal{N}$ satisfies P reduces to model counting over ϕ. Our approach works by encoding the neural network into a logical formula in conjunctive normal form (CNF). The key to achieving soundness guarantees is the notion of equicardinality, which defines a principled way of encoding neural networks into a CNF formula F, such that counting queries over the NN reduces to counting the satisfying assignments of F projected to a subset of the support of F. We then use approximate model counting on F, which has seen rapid advancement in practical tools that provide PAC-style guarantees on counts for F, i.e., the computed result is within a multiplicative $(1 + \epsilon)$ factor of the ground truth with confidence at least $1 - \delta$. The end result is a quantitative verification procedure for neural networks with soundness and precision guarantees.

The second counting formulation falls under the class of promise problems, in that we are promised that the probability that inputs satisfy the given logical property P is less than a threshold θ or more than a threshold $\theta + \eta$. We propose PROVERO, a procedure that achieves the above goal with proven soundness: When it halts with a "Yes" or "No" decision, it is correct with probability $1 - \delta$ and within approximation error η to the given θ. The verifier can control the desired parameters (η, δ), making them arbitrarily close to zero. That is, the verifier can have controllably high certainty about the verifier's output, and θ can be arbitrarily precise (or close to the ground truth). The lower the choice of (η, δ) used by the verifier, the higher is the running time.

The algorithms proposed in PROVERO are based on sampling, namely hypothesis testing, and derive procedures using concentration bounds. The key idea of which is to use cheaper (in sample complexity) hypothesis tests to decide "Yes" or "No" early. Given the threshold θ and the error η, the high-level idea is to propose alternative hypotheses on the left side of θ and on the right side of $\theta + \eta$. Thus, we can potentially return much faster when the true probability p is further from the threshold θ.

1.4.2 Solving Causal Queries

We propose a new approach that explains membership inference attacks through a causal model. A causal model is a graph where nodes are random variables that abstractly represent properties of the underlying stochastic process and edges denote cause-effect relationships between them. We can model the process of sampling data sets, picking hyper-parameters like the size of the neural network, output vectors, generalization parameters like bias and variance, and predictions from membership inference attack procedures as random variables. These random variables can be measured empirically during experiments. We can then both encode and infer causal relationships between nodes quantitatively through equations. Edges in our causal model are of two types: 1) mechanistically derived edges denote known mathematical facts derived from domain knowledge (prior work, definitions, etc.); and 2) relations inferred from experimental observations using causal discovery techniques.

The causal model, once learnt, acts like a predictive model—one can ask what will be the expected performance of a particular membership inference attack if the "root causes" (random variables in the model) were to have certain values not observed during prior experiments. Such estimation can be done without running expensive experiments. A causal model allows us to "single out" the effect of one variable on the membership inference attack performance. To carefully solve these queries, we leverage the principled framework of causal reasoning known as do-calculus. It allows us to perform systematic refutation tests, which avoids confusing causation with correlation. Such tests quantitatively tell us how well the model fits the observed data and answer causal queries about surmised root causes.

1.4.3 Solving Decisional Convergence Queries

We propose the first conceptual mechanism to resolve data non-repudiation disputes by providing an algorithm to solve decisional convergence queries. The application of the query is on a property called forgery in stochastic gradient descent. The existence of forgery implies the existence of two sets of samples (minibatches) of a given dataset could result in the same model parameters after a step of gradient descent. In the context of data non-repudiation, one minibatch corresponds to the actual execution trace of stochastic gradient descent by an honest trainer, whereas

another is a minibatch used for forgery. Under a regime of $\epsilon \to 0$, we consider exact forgery, i.e., if the model parameters of the forgery match on all bits. In stochastic gradient descent, the next model parameters are obtained via summing high-dimensional gradient vectors with respect to the current model parameters of the samples in the minibatch. Thus, collision of model parameters implies collision of sums of high-dimensional gradient vectors from a given dataset. If all these gradient vectors were independent, then forgery would be impossible. However, we are interested in integer combinations, since we can only add or remove samples to form minibatches.

Thus, our test called LSBUnforgeability implements a one-way check, that takes advantage of the algebraic properties: if no combination of the least significant bits of the representations of the gradient vectors can sum up to zero, then there is no integer combination of the gradient vectors. Hence, if LSBUnforgeability returns no solution for the boolean, then we can safely return that the training step is collision-resistant or that unforgeability holds in convergence.

1.5 Summary of Contributions

This thesis closes foundational gaps by formalizing security properties (Chapter 5 introduces and shows why properties in the limit are essential for security), models of reasoning about the training process (Chapter 4 proposes causal models for stochastic gradient descent), and sound procedures to verify or test such properties of machine learning (Chapter 2 and Chapter 3).

On a *conceptual* front, we introduce three novel formulations of queries that are useful for evaluating several properties of neural networks that connect to security and privacy applications. On a technical side, we develop a diverse toolbox that is required to solve such queries for neural networks, and present how several algorithmic frameworks can be used as tools for evaluating the security and privacy of neural networks, beyond attacks and qualitative verification, but with rigorous guarantees. Since the algorithmic tools are quite diverse, each chapter has its own related work or other useful background presented within.

In Chapter 2 we demonstrate that counting queries are tractable and useful in several security applications. We present two approaches to solve these queries.

First, we propose a principled algorithmic approach for encoding neural networks to CNF formulae that preserve model counts. We build an end-to-end tool called NPAQ that can handle binarized neural networks [96]. We show that NPAQ scales to neural networks with $1-3$ internal layers and $20-200$ units per layer. We use 2 standard datasets namely MNIST and UCI Adult Census Income dataset. We showcase how NPAQ can be used in diverse security applications with case studies for robustness, fariness and susceptibility to trojan attacks.

In Chapter 3, we propose a sampling-based approach called PROVERO to answer bounded counting queries. PROVERO only needs black-box access to the deep neural network, freeing it up from assuming anything about the internal implementation of the deep neural networks. The deep neural network can be deterministic or from a general family of non-deterministic deep neural networks. This allows checking probabilistic properties of deterministic deep neural networks and of randomization procedures defined over deep neural networks [130].

In Chapter 4, we propose the first use of causal analysis for studying membership inference attacks on deep neural networks. We derive causal models for 6 membership inference attacks by combining both known domain-specific assumptions and observations made from experiments. Our key contribution is a new quantitative connection between membership inference attacks and generalization, which enables refuting claims about causation with finer accuracy.

In Chapter 5, we present the first conceptual mechanism to solve data non-repudiation claims. The key building block is a concrete test to check whether a training step in stochastic gradient descent is unforgeable or is "collision-resistant", i.e., no two training samples give the same update at a given point in the training. The empirical results of this test on the same experimental setup as prior work are contrasting, which highlight the need for better security definitions. We formalize checking forgeability of gradient updates as a decisional convergence query, where forgery is considered in the limit, under precise algebraic definitions.

1.6 Statement of Joint Work

All the works presented in this thesis were led by Teodora Baluta. In addition, Shiqi Shen, Shweta Shinde, Kuldeep S. Meel and Prateek Saxena contributed to

the work in Chapter 2, and the development was done by Shiqi Shen and Teodora Baluta. Zheng Leong Chua, Kuldeep S. Meel and Prateek Saxena contributed to the work in Chapter 3. Shiqi Shen, S. Hitarth, Shruti Tople, and Prateek Saxena contributed to the work presented in Chapter 4. Chapter 5 is a joint work with Ivica Nikolić, and Racchit Jain, Divesh Aggarwal and Prateek Saxena have contributed to the work.

Chapter 2

Verifiability via Approximate Counting Queries

2.1 Introduction

In the past decade, there has been a surge of interest in the design of methodological approaches to verification and testing of neural networks. Early efforts focused on formal verification wherein, given a neural network N and property P, one is concerned with determining whether there exists an input I to N such that P is violated [173, 178, 49, 156, 111, 93, 45, 205]. While such certifiability techniques provide value, for instance in demonstrating the existence of adversarial examples [78, 166], it is worth recalling that the training of neural networks is a stochastic complex process. Therefore, many analyses of neural networks require *counting queries*, which determines how many inputs satisfy P.

It is natural to encode properties as well as conditions on inputs or outputs as logical formulae. In order to answer counting queries, we focus on the following formulation of *quantitative verification*: Given a set of neural networks $\mathcal{N}$ and a property of interest P defined over the union of inputs and outputs of neural networks in $\mathcal{N}$, we are interested in estimating how often P is satisfied. In many critical domains, client analyses often require guarantees that the computed estimates be reasonably close to the ground truth. We are not aware of any prior approaches that provide such formal guarantees, though the need for quantitative verification has recently been recognized [245].

Security Applications. Quantitative verification enables many applications in security analysis (and beyond) for neural networks. We present 3 applications in

which the following analysis questions can be quantitatively answered:

- *Robustness*: How many adversarial samples does a given neural network have? Does one neural network have more adversarial inputs compared to another one?
- *Trojan Attacks*: A neural network can be trained to classify certain inputs with "trojan trigger" patterns to the desired label. How well-poisoned is a trojaned model, i.e., how many such trojan inputs does the attack successfully work for?
- *Fairness*: Does a neural network change its predictions significantly when certain input features are present (e.g., when the input record has gender attribute set to "female" vs. "male")?

Note that such analysis questions boil down to estimating how often a given property over inputs and outputs is satisfied. Estimating counts is fundamentally different from checking whether a satisfiable input exists. Since neural networks are stochastically trained, the mere existence of certain satisfiable inputs is not unexpected. The questions above checks whether their counts are sufficiently large to draw statistically significant inferences. Section 2.3 formulates these analysis questions as logical specifications.

Our Approach. The primary contribution of this chapter is a new analysis framework, which models the given set of neural networks $\mathcal{N}$ and P as set of logical constraints, φ, such that the problem of quantifying how often $\mathcal{N}$ satisfies P reduces to model counting over φ. We then show that the quantitative verification is $\#P$-hard. Given the computational intractability of $\#P$, we seek to compute rigorous estimates and introduce the notion of *approximate quantitative verification*: given a prescribed tolerance factor ε and confidence parameter δ, we estimate how often P is satisfied with PAC-style guarantees, i.e., the computed result is within a multiplicative $(1+\varepsilon)$ factor of the ground truth with confidence at least $1-\delta$.

Our approach works by encoding the neural network into a logical formula in conjunctive normal form (CNF). The key to achieving soundness guarantees is our new notion of equicardinality, which defines a principled way of encoding neural networks into a CNF formula F, such that quantitative verification reduces to

counting the satisfying assignments of F projected to a subset of the support of F. We then use approximate model counting on F, which has seen rapid advancement in practical tools that provide PAC-style guarantees on counts for F. The end result is a *quantitative verification procedure for neural networks with soundness and precision guarantees.*

While our framework is more general, we instantiate our analysis framework with a sub-class of neural networks called binarized neural networks (or BNNs) [96]. BNNs are multi-layered perceptrons with +/-1 weights and step activation functions. They have been demonstrated to achieve high accuracy for a wide variety of applications [185, 149, 121]. Due to their small memory footprint and fast inference time, they have been deployed in constrained environments such as embedded devices [149, 121]. We observe that specific existing encodings for BNNs adhere to our notion of equicardinality and implement these in a new tool called NPAQ[1]. We provide proofs of key correctness and composability properties of our general approach, as well as of our specific encodings. Our encodings are linear in the size of $\mathcal{N}$ and P.

Empirical Results. We show that NPAQ scales to BNNs with $1-3$ internal layers and $20-200$ units per layer. We use 2 standard datasets namely MNIST and UCI Adult Census Income dataset. We encode a total of 84 models with $4,692-53,010$ parameters, into $1,056$ formulae and quantitatively verify them. NPAQ encodes properties in less than a minute and solves 97.1% formulae in a 24-hour timeout. Encodings scale linearly in the size of the models, and its running time is not dependent on the true counts. We showcase how NPAQ can be used in diverse security applications with case studies. First, we quantify the model robustness by measuring how many adversarially perturbed inputs are misclassified, and then the effectiveness of 2 defenses for model hardening with adversarial training. Next, we evaluate the effectiveness of trojan attacks outside the chosen test set. Lastly, we measure the influence of 3 sensitive features on the output and check if the model is biased towards a particular value of the sensitive feature.

Contributions. We make the following contributions:

- *New Notion.* We introduce the notion of *approximate quantitative verification*

[1]The name stands for **N**eural **P**roperty **A**pproximate **Q**uantifier. Code and benchmarks are available at https://teobaluta.github.io/npaq/

to estimate how often a property P is satisfied by the neural net N with theoretically rigorous PAC-style guarantees.

- *Algorithmic Approach, Tool, & Security Applications.* We propose a principled algorithmic approach for encoding neural networks to CNF formula that preserve model counts. We build an end-to-end tool called NPAQ that can handle BNNs. We demonstrate security applications of NPAQ in quantifying robustness, trojan attacks, and fairness.

- *Results.* We evaluate NPAQ on $1,056$ formulae derived from properties over BNNs trained on two datasets. We show that NPAQ presently scales to BNNs of over $50,000$ parameters, and evaluate its performance characteristics with respect to different user-chosen parameters.

2.2 Problem Definition

Definition 1. *Let $\mathcal{N} = \{f_1, f_2, \ldots, f_m\}$ be a set of m neural nets, where each neural net f_i takes a vector of inputs $\mathbf{x_i}$ and outputs a vector $\mathbf{y_i}$, such that $\mathbf{y_i} = f_i(\mathbf{x_i})$. Let $P : \{\mathbf{x} \cup \mathbf{y}\} \to \{0,1\}$ denote the property P over the inputs $\mathbf{x} = \bigcup_{i=1}^{m} \mathbf{x_i}$ and outputs $\mathbf{y} = \bigcup_{i=1}^{m} \mathbf{y_i}$. We define the specification of property P over $\mathcal{N}$ as $\varphi(\mathbf{x}, \mathbf{y}) = (\bigwedge_{i=1}^{m} (\mathbf{y_i} = f_i(\mathbf{x_i})) \wedge P(\mathbf{x}, \mathbf{y}))$.*

We show several motivating property specifications in Section 2.3. For the sake of illustration here, consider $\mathcal{N} = \{f_1, f_2\}$ be a set of two neural networks that take as input a vector of three integers and output a 0/1, i.e., $f_1 : \mathbb{Z}^3 \to \{0,1\}$ and $f_2 : \mathbb{Z}^3 \to \{0,1\}$. We want to encode a property to check the dis-similarity between f_1 and f_2, i.e., counting for how many inputs (over all possible inputs) f_1 and f_2 produce differing outputs. The specification is defined over the inputs $\mathbf{x} = [x_1, x_2, x_3]$, outputs $y_1 = f_1(\mathbf{x})$ and $y_2 = f_2(\mathbf{x})$ as $\varphi(\mathbf{x}, y_1, y_2) = (f_1(\mathbf{x}) = y_1 \wedge f_2(\mathbf{x}) = y_2 \wedge y_1 \neq y_2)$.

Given a specification φ for a property P over the set of neural nets $\mathcal{N}$, a verification procedure returns $r = 1$ (SAT) if there exists a satisfying assignment τ such that $\tau \models \varphi$, otherwise it returns $r = 0$ (UNSAT). A satisfying assignment for φ is defined as $\tau : \{\mathbf{x} \cup \mathbf{y}\} \to \{0,1\}$ such that φ evaluates to true, i.e., $\varphi(\tau) = 1$ or $\tau \models \varphi$.

While the problem of standard (qualitative) verification asks whether there exists a satisfying assignment to φ, the problem of quantitative verification asks how many satisfying assignments φ admits. We denote the set of satisfying assignments for the specification φ as $R(\varphi) = \{\tau : \tau \models \varphi\}$.

Definition 2. *Given a specification φ for a property P over the set of neural nets $\mathcal{N}$, a quantitative verification procedure, $NQV(\varphi)$, returns the number of satisfying assignments of φ, $r = |R(\varphi)|$.*

It is worth noting that $|R(\varphi)|$ may be intractably large to compute via naïve enumeration. For instance, we consider neural networks with hundreds of bits as inputs for which the unconditioned input space is $2^{|\mathbf{x}|}$. In fact, we prove that quantitative verification is #P-hard, as stated below.

Theorem 1. *$NQV(\varphi)$ is #P-hard, where φ is a specification for a property P over binarized neural nets.*

Our proof is a parsimonious reduction of model counting of CNF formulas, #CNF, to quantitative verification of binarized neural networks. We show how an arbitrary CNF formula F can be transformed into a binarized neural net f_i and a property P such that for a specification φ for P over $\mathcal{N} = \{f_i\}$, it holds true that $R(\mathrm{F}) = R(\varphi)$. See Section 2.4 for the full proof.

Remark 1. *The parsimonious reduction from #CNF to NQV implies that fully polynomial time randomized approximation schemes, including those based on Monte Carlo, cannot exist unless NP=RP.*

The computational intractability of #P necessitates a search for relaxations of NQV. To this end, we introduce the notion of an approximate quantitative verifier that outputs an approximate count within ϵ of the true count with a probability greater than $1 - \delta$.

Definition 3. *Given a specification φ for a property P over a set of neural nets $\mathcal{N}$, $0 < \epsilon \leq 1$ and $0 < \delta \leq 1$, an approximate quantitative verification procedure (ϵ, δ)-$NQV(\varphi, \epsilon, \delta)$ computes r such that $Pr[(1+\epsilon)^{-1}|R(\varphi)| \leq r \leq (1+\epsilon)|R(\varphi)|] \geq 1 - \delta$.*

The security analyst can set the "confidence" parameter δ and the precision or "error tolerance" ϵ as desired. The (ϵ, δ)-NQV definition specifies the end guarantee of producing estimates that are statistically sound with respect to chosen parameters (ϵ, δ).

Connection to Computing Probabilities. Readers can naturally interpret $|R(\varphi)|$ as a measure of probability. Consider $\mathcal{N}$ to be a set of functions defined over input random variables $\mathbf{x}$. The property specification φ defines an event that conditions inputs and outputs to certain values, which the user can specify as desired. The measure $|R(\varphi)|$ counts how often the event occurs under all possible values of $\mathbf{x}$. Therefore, $\frac{|R(\varphi)|}{2^{|\mathbf{x}|}}$ is the probability of the event defined by φ occurring. Our formulation presented here computes $|R(\varphi)|$ weighting all possible values of $\mathbf{x}$ equally, which implicitly assumes a uniform distribution over all random variables $\mathbf{x}$. Our framework can be extended to weighted counting [25, 26], assigning different user-defined weights to different values of $\mathbf{x}$, which is akin to specifying a desired probability distributions over $\mathbf{x}$. However, we consider this extension as a promising future work.

2.3 Security Applications

We present three concrete application contexts which highlight how quantitative verification is useful to diverse security analyses. The specific property specifications presented here were derived directly from recent works, highlighting that NPAQ is broadly applicable to analysis problems actively being investigated.

Robustness. An adversarial example for a neural network is an input which under a small perturbation is classified differently [222, 78]. The lower the number of adversarial examples, the more "robust" the neural network. Early work on verifying robustness aimed at checking whether adversarial inputs exist. However, recent works suggest that adversarial inputs are statistically "not surprising" [239, 9, 58] as they are a consequence of normal error in statistical classification [73, 72, 146, 42]. This highlights the importance of analyzing whether a statistically significant number of adversarial examples exist, not just whether they exist at all, under desired input distributions. Our framework allows the analyst to specify a logical property of adversarial inputs and quantitatively verify it. Specifically, one can

estimate how many inputs are misclassified by the net (f) and within some small perturbation distance k from a benign sample ($\mathbf{x_b}$) [24, 166, 165], by encoding the property P1 in our framework as:

$$P_1(\mathbf{x}, \mathbf{y}, \mathbf{x_b}, \mathbf{y_b}, k) = \sum_{j=1}^{|\mathbf{x}|} (\mathbf{x_b}[j] \oplus \mathbf{x}[j]) \leq k \wedge \mathbf{y_b} \neq \mathbf{y} \tag{P1}$$

As a concrete usage scenario, our evaluation reports on BNNs for image classification (Section 2.7.2). Even for a small given input (say m bits), the space of all inputs within a perturbation of k bits is $\binom{m}{k}$, which is too large to check for misclassification one-by-one. NPAQ does not enumerate and yet can estimate adversarial input counts with PAC-style guarantees (Section 2.7.2). As we permit larger perturbation, as expected, the number of adversarial samples monotonically increase, and NPAQ can quantitatively measure how much. Further, we show how one can directly compare robustness estimates for two neural networks. Such estimates may also be used to measure the efficacy of defenses. Our evaluation on 2 adversarial training defenses shows that the hardened models show lesser robustness than the plain (unhardened) model. Such analysis can help to quantitatively refute, for instance, claims that BNNs are intrinsically more robust, as suggested in prior work [64].

Trojan Attacks. Neural networks, such as for facial recognition systems, can be trained in a way that they output a specific value, when the input has a certain "trojan trigger" embedded in it [**Trojannn**]. The trojan trigger can be a fixed input pattern (e.g., a sub-image) or some transformation that can be stamped on to a benign image. One of the primary goals of the trojan attack is to maximize the number of trojaned inputs which are classified as the desired target output, $\mathbf{l_{attack}}$. NPAQ can quantify the number of such inputs for a trojaned network, allowing attackers to optimize for this metric. To do so, one can encode the set of trojaned inputs as all those inputs $\mathbf{x}$ which satisfy the following constraint for a given neural network f, trigger $\mathbf{t}$, $\mathbf{l_{attack}}$ and the (pixel) location of the trigger M:

$$P_2(\mathbf{x}, \mathbf{y}, \mathbf{t}, \mathbf{l_{attack}}, M) = \bigwedge_{j \in M} (\mathbf{x}[j] = \mathbf{t}[j]) \wedge \mathbf{y} = \mathbf{l_{attack}} \tag{P2}$$

Section 2.7.3 shows an evaluation on BNNs trained on the MNIST dataset. Our evaluation demonstrates that the attack accuracy on samples from the test set can differ significantly from the total set of trojaned inputs specified as in property P2.

Fairness. The right notion of algorithmic fairness is being widely debated [46, 55, 257, 83, 38]. Our framework can help quantitatively evaluate desirable metrics measuring "bias" for neural networks. Consider a scenario where a neural network f is used to predict the recommended salary for a new hire in a company. Having been trained on public data, one may want to check whether f makes biased predictions based on certain sensitive features such as race, gender, or marital status of the new hire. To verify this, one can count how often f proposes a higher salary for inputs when they have a particular sensitive feature (say "gender") set to certain values (say "male"), given all other input features the same. Formally, this property can be encoded for given sensitive features denoted by set S along with two concrete sensitive values $\mathbf{s_1}, \mathbf{s_2}$, as:

$$\begin{aligned} P_3(\mathbf{x_1}, \mathbf{x_2}, \mathbf{y_1}, \mathbf{y_2}, S, \mathbf{s_1}, \mathbf{s_2}) = \bigwedge_{i \in S} (\mathbf{x_1}[i] = \mathbf{s_1}[i]) \\ \bigwedge_{i \in S} (\mathbf{x_2}[\mathbf{i}] = \mathbf{s_2}[\mathbf{i}]) \bigwedge_{i \notin S} (\mathbf{x_1}[i] = \mathbf{x_2}[i]) \wedge \mathbf{y_1} = \mathbf{y_2} \end{aligned} \tag{P3}$$

Notice the NPAQ counts over *all* possible inputs where the non-sensitive features remain equal, but only the sensitive features change, which causes no change in prediction. An unbiased model would produce a very high count, meaning that for most inputs (or with high probability), changing just the sensitive features results in no change in outputs. A follow-up query one may ask is whether inputs having different values for the sensitive features and all other values the same, determine an increases (or decreases) of the output salary prediction. This can be encoded as property P4 (or P5) below.

$$\begin{aligned} P_4(\mathbf{x_1}, \mathbf{x_2}, \mathbf{y_1}, \mathbf{y_2}, S, \mathbf{s_1}, \mathbf{s_2}) = \bigwedge_{i \in S} (\mathbf{x_1}[i] = \mathbf{s_1}[i]) \\ \bigwedge_{i \in S} (\mathbf{x_2}[\mathbf{i}] = \mathbf{s_2}[\mathbf{i}]) \bigwedge_{i \notin S} (\mathbf{x_1}[i] = \mathbf{x_2}[i]) \wedge \mathbf{y_2} - \mathbf{y_1} > 0 \end{aligned} \tag{P4}$$

$$\begin{aligned} P_5(\mathbf{x_1}, \mathbf{x_2}, \mathbf{y_1}, \mathbf{y_2}, S, \mathbf{s_1}, \mathbf{s_2}) = \bigwedge_{i \in S} (\mathbf{x_1}[i] = \mathbf{s_1}[i]) \\ \bigwedge_{i \in S} (\mathbf{x_2}[\mathbf{i}] = \mathbf{s_2}[\mathbf{i}]) \bigwedge_{i \notin S} (\mathbf{x_1}[i] = \mathbf{x_2}[i]) \wedge \mathbf{y_2} - \mathbf{y_1} < 0 \end{aligned} \tag{P5}$$

NPAQ can be used to quantitatively verify such properties, and compare models before deploying them based on such estimates. Section 2.7.4 presents more concrete evaluation details and interpretation of BNNs trained on the UCI Adult dataset [238].

2.4 Quantitative Verification is #P-hard

We prove that quantitative verification is #P-hard by reducing the problem of model counting for logical formulas to quantitative verification of neural networks. We show how an arbitrary CNF formula F can be transformed into a binarized neural net f and a specification φ such that the number of models for F is the same as φ, i.e., $|R(\varphi)| = |R(F)|$. Even for this restricted class of multilayer perceptrons quantitative verification turns out to be #P-hard. Hence, in general, quantitative verification over multilayer perceptrons is #P-hard.

Theorem 2. *NQV (φ) is #P-hard, where φ is a specification for a property P over binarized neural nets.*

Proof. We proceed by constructing a mapping between the propositional variables of the formula F and the inputs of the BNN. We represent the logical formula as a logical circuit with the gates AND, OR, NOT corresponding to $\wedge, \vee, \neg$. In the following, we show that for each of the gates there exist an equivalent representation as a perceptron. For the OR gate we construct an equivalent perceptron, i.e., for every clause C_i of the formula F, we construct a perceptron. The perceptron is activated only if the inputs correspond to a satisfying assignment to the formula F. Similarly, we show a construction for the AND gate. Thus, we construct a BNN that composes these gates such that it can represent the logical formula exactly.

Let F be a CNF formula $F = C_1 \wedge C_2 \wedge \ldots C_n$. We denote the literals appearing in clause C_i as l_{ij}, $j = 1, ..m$. Let $\tau : Supp(F) \to \{0, 1\}$ be an assignment for F where $Supp(F)$ represents the propositional variables F is defined on. We say F is satisfiable if there exists an assignment τ such that $\tau(F) = 1$. The binarized neural net f has inputs $\mathbf{x}$ and one output y, $y = N(\mathbf{x})$, and $f : \{-1, 1\}^{m \cdot n} \to \{0, 1\}$. This can be easily extended to multi-class output.

We first map the propositional variables in $Supp(F)$ to variables in the binary domain $\{-1, 1\}$. For every clause C_i, for every literal $l_{ij} \in \{0, 1\}$ there is a corresponding input to the neural net $x_{ij} \in \{-1, 1\}$ such that $l_{ij} \Leftrightarrow x_{ij} = 1 \wedge \overline{l_{ij}} \Leftrightarrow x_{ij} = -1$. For each input variable x_{ij} the weight of the neuron connection is 1 if the propositional variable l_{ij} appears as a positive literal in the C_i clause and -1 if it appears as a negative literal $\overline{l_{ij}}$ in C_i.

For every clause C_i we construct a *disjunction gadget*, a perceptron equivalent function to the OR gate. Given m inputs $x_{i1}, x_{i2}, \ldots x_{im} \in \{-1, 1\}$, the disjunction gadget determines the output of neuron q_i. The output is the linear layer is $t_i = \sum_{j=1}^{m} w_j \cdot x_{ij} + m$. The output neuron q_i is 1 if the activation function $\texttt{sign}(t_i)$ returns 1. Namely, the output is 1 only if at least one literal is true, i.e., not all $w_j \cdot x_{ij}$ terms evaluate to -1. Notice that we only need $m+2$ neurons (m for the inputs and 2 for the intermediate outputs) for each clause C_i with m literals. Next, we introduce the *conjunction gadget* which, given n inputs $q_1, \ldots, q_n \in \{-1, 1\}$ outputs $y = 1$ only if $q_1 + q_2 + \ldots + q_n \geq n$. The linear layer's output is $t' = \sum_{i=1}^{n} w_i \cdot q_i - n$ over which we apply the `sign` activation function. The output of this gadget, $y = \sum_{i=1}^{n} w_i \cdot q_i \geq n$, is 1 if all of the variables q_i are 1, i.e., if all the clauses are satisfied.Notice that if we consider the batch normalization a transformation over t_i that returns t_i, we can obtain a binarized neural network f.

If the output of f on inputs $\mathbf{x}$ is 1 the formula F is SAT, otherwise it is UNSAT. Moreover, the binarized neural network constructed for binary input vectors of size $m \times n$ outputs $y = 1$ for every satisfying assignment τ of the formula F, i.e., $f(\tau(\mathbf{x})) = 1$. Given a procedure #SAT(F) that accepts formula F and outputs a number r which is the number of satisfying assignments, it will also compute the number of inputs for which the output of the BNN is 1. Specifically, we can construct a quantitative verifier for the neural net f and a specification $\varphi(\mathbf{x}, y) = (y = N(\mathbf{x})) \wedge y = 1$ using #SAT(F).

Reduction is polynomial. The size of the formula F is the size of the input $\mathbf{x}$ to the neural net, i.e., $m \cdot n$. The neural net has $n + 1$ perceptrons (n for each disjunction gadget and one for the conjunction gadget).

□

2.5 Approach

Recall that exact counting (as defined in NQV) is #P-hard. Even for approximate counting, many widely used sampling-based approaches, such as based on Monte Carlo methods [81, 84, 162, 104], do *not* provide soundness guarantees since existence of a method that only requires polynomially many samples computable

in (randomized) polynomial time would imply $NP = RP$ (See Remark 1). For sound estimates, it is well-known that many properties encodable in our framework require intractably large number of samples—for instance, to check for distributional similarity of two networks f_1 and f_2 in the classical model, a lower bound of $O(\sqrt{2^{\mathbf{x}}})$ samples are needed to obtain estimates with reasonable (ϵ, δ) guarantees. However, approximate counting for boolean CNF formulae has recently become practical. These advances combine the classical ideas of universal hashing with the advances in the Boolean satisfiability by invoking SAT solvers for NP queries, i.e., to obtain satisfiable witnesses for queried CNF formulae. The basic idea behind these approximate CNF counters is to first employ universal hashing to randomly partition the set of solutions into *roughly small* buckets. Then, the approximate counter can enumerate a tractably small number of witnesses satisfying P using a SAT solver within one bucket, which calculates the "density" of satisfiable solutions in that bucket. By careful analysis using concentration bounds, these estimates can be extended to the sum over all buckets, yielding a provably sound PAC-style guarantee of estimates. Our work leverages this recent advance in approximate CNF counting to solve the problem of (ϵ, δ)-NQV [218].

The Equicardinality Framework. Our key technical advance is a new algorithmic framework for reducing (ϵ, δ)-NQV to CNF counting with an encoding procedure that has provable soundness. The procedure encodes $\mathcal{N}$ and P into φ, such that model counting in some way over φ counts over $\mathcal{N} \wedge \mathrm{P}$. This is *not* straight-forward. For illustration, consider the case of counting over boolean circuits, rather than neural networks. To avoid exponential blowup in the encoding, often one resorts to classical *equisatisfiable* encoding [236], which preserves satisfiability but introduces new variables in the process. Equisatisfiability means that the original formula is satisfiable if and only if the encoded one is too. Observe, however, that this notion of equisatisfiability is *not* sufficient for model counting—the encoded formula may be equisatisfiable but may have many more satisfiable solutions than the original.

We observe that a stronger notion, which we call *equicardinality*, provides a principled approach to constructing encodings that preserve counts. An equicardinality encoding, at a high level, ensures that the model count for an original formula can be computed by performing model counting *projected* over the subset of variables in

the resulting formula. We define this equicardinality relation rigorously and prove in Lemma 1 that model counting over a constraint is *equivalent* to counting over its equicardinal encoding. Further, we prove in Lemma 2 that the equicardinality relation is *closed* under logical conjunction. This means model counting over conjunction of constraints is equivalent to counting over the conjunction of their equicardinal encodings. Equicardinality CNF encodings can thus be composed with boolean conjunction, while preserving equicardinality in the resulting formulae.

With this key observation, our procedure has two remaining sub-steps. First, we show equicardinal encodings for each neural net and properties over them to individual equicardinality CNF formulae. This implies ψ, the conjunction of the equicardinality CNF encodings of the conjuncts in φ, preserves the original model count of φ. Second, we show how an existing approximate model counter for CNF with (ϵ, δ) guarantees can be utilized to count over a projected subset of the variables in ψ. This end result, by construction, guarantees that our final estimate of the model count has bounded error, parameterized by ε, with confidence at least $1 - \delta$.

Cardinality Constraints:	x_1	x_2	x_3	$f(\mathbf{x})$
$x_1 + x_2 + x_3 \geq 2 \Leftrightarrow v_1 = 1$	0	0	0	0
$x_1 + \overline{x}_2 + x_3 \geq 1 \Leftrightarrow v_2 = 1$	0	0	1	0
$\overline{x}_1 + x_2 + x_3 \geq 1 \Leftrightarrow v_3 = 1$	0	1	0	0
$x_1 + x_2 + \overline{x}_3 \geq 1 \Leftrightarrow v_4 = 1$	0	1	1	1
$\overline{x}_1 + x_2 + \overline{x}_3 \geq 1 \Leftrightarrow v_5 = 1$	1	0	0	0
	1	0	1	1
$v_1 + v_2 + v_3 + v_4 + v_5 \geq 5 \Leftrightarrow y$	1	1	0	0
	1	1	1	1

Cardinality Constraints:	x_1	x_2	x_3	$f_1(\mathbf{x})$	$f_2(\mathbf{x})$
$\overline{x}_1 + \overline{x}_2 + x_3 \geq 1 \Leftrightarrow v_5$	0	0	0	0	0
$v_1 + v_2 + v_3 + v_4 + v_5 \geq 5$	0	0	1	0	0
$\Leftrightarrow y$	0	1	0	0	0
$f_1 : v_2 = v_3 = v_4 = 1$	0	1	1	1	0
$f_2 : x_1 + x_3 \geq 2 \Leftrightarrow v_1$	1	0	0	0	0
$x_1 + \overline{x}_2 + x_3 \geq 1 \Leftrightarrow v_2$	1	0	1	1	1
$\overline{x}_1 + x_2 + x_3 \geq 1 \Leftrightarrow v_3$	1	1	0	0	0
$x_1 + x_2 + \overline{x}_3 \geq 1 \Leftrightarrow v_4$	1	1	1	1	1

Figure 2.1. Example of encoding different BNNs (f, f_1, f_2) as a conjunction over a set of cardinality constraints. An attacker manipulates f with the goal to increase the inputs with trigger $x_3 = 1$ that classify as $y = 0$. Specifically, to obtain f_1 the weights of x_1, x_2, x_3 in constraints of f for v_2, v_3, v_4 are set to 0 (highlighted with dashed lines, on the left). To obtain f_2, we set $w_{21} = 0$. The trojan property P $\doteq (y = 0) \wedge (x_3 = 1)$ is satisfied by one input (left) for f, whereas for f_2 we find two (right).

Formalization. We formalize the above notions using notation standard for boolean logic. The projection of an assignment σ over a subset of the variables $\mathbf{t}$, denoted as $\sigma|_{\mathbf{t}}$, is an assignment of $\mathbf{t}$ to the values taken in σ (ignoring variables other than $\mathbf{t}$ in σ).

Definition 4. *We say that a formula* $\varphi : \mathbf{t} \to \{0,1\}$ *is equicardinal to a formula* $\psi : \mathbf{u} \to \{0,1\}$ *where* $\mathbf{t} \subseteq \mathbf{u}$*, if:*

(a) $\forall \tau \models \varphi \Rightarrow \exists \sigma, (\sigma \models \psi) \wedge (\sigma|_{\mathbf{t}} = \tau)$*, and*

(b) $\forall \sigma \models \psi \Rightarrow \sigma|_{\mathbf{t}} \models \varphi$.

An example of a familiar equicardinal encoding is Tseitin [236], which transforms arbitrary boolean formulas to CNF. Our next lemma shows that equicardinality preserves model counts. We define $R(\psi) \downarrow \mathbf{t}$, the set of satisfying assignments of ψ projected over $\mathbf{t}$, as $\{\sigma|_{\mathbf{t}} : \sigma \models \psi\}$.

Lemma 1 (Count Preservation)**.** *If* ψ *is equicardinal to* φ*, then* $|R(\psi) \downarrow \mathbf{t}| = |R(\varphi)|$.

Proof. By Definition 4(a), for every assignment $\tau \models \varphi$, there is a $\sigma \models \psi$ and the $\sigma|_{\mathbf{t}} = \tau$. Therefore, each distinct satisfying assignment of φ must have a unique assignment to $\sigma|_{\mathbf{t}}$, which must be in $R(\psi) \downarrow \mathbf{t}$. It follows that $|R(\psi) \downarrow \mathbf{t}| \geq |R(\varphi)|$, then. Next, observe that Definition 4(b) states that everything in $R(\psi) \downarrow \mathbf{t}$ has a satisfying assignment in φ; that is, its projection cannot correspond to a non-satisfying assignment in φ. By pigeonhole principle, it must be that $|R(\psi) \downarrow \mathbf{t}| \leq |R(\varphi)|$. This proves that $|R(\psi) \downarrow \mathbf{t}| = |R(\varphi)|$. □

Lemma 2 (CNF-Composibility)**.** [2] *Consider* $\varphi_i : \mathbf{t_i} \to \{0,1\}$ *and* $\psi_i : \mathbf{u_i} \to \{0,1\}$*, such that* φ_i *is equicardinal to* ψ_i*, for* $i \in \{1,2\}$*. If* $(\mathbf{u_1} - \mathbf{t_1}) \cap \mathbf{u_2} = \varnothing$ *and* $(\mathbf{u_2} - \mathbf{t_2}) \cap \mathbf{u_1} = \varnothing$*, then* $\varphi_1 \wedge \varphi_2$ *is equicardinal to* $\psi_1 \wedge \psi_2$.

Proof. (a) $\forall \tau \models \varphi_1 \wedge \varphi_2 \Rightarrow (\tau \models \varphi_1) \wedge (\tau \models \varphi_2)$. By Definition 4(a), $\exists \sigma_1, \sigma_2, \sigma_1 \models \psi_1 \wedge \sigma_2 \models \psi_2$. Further, by Definition 4(a), $\sigma_1|_{\mathbf{t_1}} = \tau|_{\mathbf{t_1}}$ and $\sigma_2|_{\mathbf{t_2}} = \tau|_{\mathbf{t_2}}$. This implies that $\sigma_1|_{\mathbf{t_1}} \cup \sigma_2|_{\mathbf{t_2}} = \tau_1|_{\mathbf{t_1}} \cup \tau_2|_{\mathbf{t_2}} = \tau$. Then $\sigma_1 \cup \sigma_2 = \sigma_1|_{\mathbf{u_1 - t_1}} \cup \sigma_2|_{\mathbf{u_2 - t_2}} \cup (\sigma_1|_{\mathbf{t_1}} \cup \sigma_2|_{\mathbf{t_2}}) = \sigma_1|_{\mathbf{u_1 - t_1}} \cup \sigma_2|_{\mathbf{u_2 - t_2}} \cup \tau$. Since $(\mathbf{u_1} - \mathbf{t_1}) \cap \mathbf{u_2} = \varnothing$, we know that $\sigma_1|_{\mathbf{u_1 - t_1}} \cap \sigma_2|_{\mathbf{u_2 - t_2}} = \varnothing$ and $\sigma_1|_{\mathbf{u_1 - t_1}} \cap \tau = \varnothing$. Similarly, since $(\mathbf{u_2} - \mathbf{t_2}) \cap \mathbf{u_1} = \varnothing$, we know that $\sigma_2|_{\mathbf{u_2 - t_2}} \cap \tau = \varnothing$. From this, it follows that $\sigma_1 \cup \sigma_2$ represent a set of satisfying assignments for $\psi_1 \wedge \psi_2$. Further,

[2] The CCS'19 published version of this paper contained a mistake on the condition of lemma – the intersection and union operators on the sets u_1, t_1, u_2, t_2 were incorrectly used. We would like to thank Jiong Yang for pointing it out and helping us correct it.

$(\sigma_1 \cup \sigma_2)|_{\mathbf{t_1} \cup \mathbf{t_2}} = \tau$. This proves part (a) of the claim that $\varphi_1 \wedge \varphi_2$ is equicardinal to $\psi_1 \wedge \psi_2$.

(b) $\forall \sigma \models \psi_1 \wedge \psi_2 \Rightarrow (\sigma \models \psi_1) \wedge (\sigma \models \psi_2)$. By Definition 4(b), $\sigma|_{\mathbf{t_1}} \models \varphi_1$ and $\sigma|_{\mathbf{t_2}} \models \varphi_2$. This implies $\sigma|_{\mathbf{t}} \models \varphi_1 \wedge \varphi_2$, thereby proving the part (b) of the definition for the claim that $\varphi_1 \wedge \varphi_2$ is equicardinal to $\psi_1 \wedge \psi_2$.

□

Final Count Estimates. With the CNF-composability lemma at hand, we decompose the counting problem over a conjunction of neural networks $\mathcal{N}$ and property P, to that of counting over the conjunction of their respective equicardinality encodings. Equicardinality encodings preserve counts, and taking their conjunction preserves counts. It remains to show how to encode $\mathcal{N}$ to boolean CNF formulae, such that the encodings are equicardinal. Since the encoding preserves counts originally desired exactly, we can utilize off-the-shelf approximate CNF counters [27, 218] which have (ϵ, δ) guarantees. The final counts are thus guaranteed to be sound estimates by construction, which we establish formally in Theorem 3 for the encodings in Section 2.6.

Why Not Random Sampling? An alternative to our presented approach is random sampling. One could simply check what fraction of all possible inputs satisfies φ by testing on a random set of samples. However, the estimates produced by this method will satisfy soundness (defined in Section 2.2) *only if* the events being measured have sufficiently high probability. In particular, obtaining such soundness guarantees for rare events, i.e., where counts may be very low, requires an intractably large number of samples. Note that such events do arise in security applications [21, 245]. Specialized Monte Carlo samplers for such low probability events have been investigated in such contexts [245], but they do not provide soundness guarantees. We aim for a general framework, that works irrespective of the probability of events measured.

2.6 NPAQ Design

Our tool takes as input a set of trained binarized neural networks $\mathcal{N}$ and a property P and outputs "how often" P holds over $\mathcal{N}$ with (ϵ, δ) guarantees. We

show a two-step construction from binarized neural nets to CNF. The main principle we adhere to is that at every step we formally prove that we obtain equicardinal formulas. While BNNs and, in general, neural nets can be encoded using different background theories, we choose a specialized encoding of BNNs to CNF. First, we express a BNN using cardinality constraints similar to [156] (Section 2.6.1). For the second step, we choose to encode the cardinality constraints to CNF using a sorting-based encoding (Section 2.6.2). We prove that NPAQ is preserving the equicardinality in Theorem 3. Finally, we use an approximate model counter that can handle model counting directly over a projected subset of variables for a CNF formula [218].

2.6.1 BNN to Cardinality Constraints

Consider a standard BNN $f_i : \{-1,1\}^n \to \{0,1\}^s$ that consists of $d-1$ internal blocks and an output block [96]. We denote the kth internal block as f_{blk_k} and the output block as f_{out}. More formally, given an input $\mathbf{x} \in \{-1,1\}^n$, the binarized neural network is: $f_i(\mathbf{x}) = f_{out}(f_{\text{blk}_{d-1}}(\ldots(f_{\text{blk}_1}(\mathbf{x})\ldots))$. For every block f_{blk_k}, we define the inputs to f_{blk_k} as the vector $\mathbf{v_k}$. We denote the output for k block as the vector $\mathbf{v_{k+1}}$. For the output block, we use $\mathbf{v_d}$ to denote its input. The input to f_{blk_1} is $\mathbf{v_1} = \mathbf{x}$. We summarize the transformations for each block in Table 2.1.

Running Example. Consider a binarized neural net $f : \{-1,1\}^3 \to \{0,1\}$ with a single internal block and a single output (Figure 2.1). To show how one can derive the constraints from the BNN's parameters, we work through the procedure to derive the constraint for the output of the internal block's first neuron, denoted by v_1. Suppose we have the following parameters: the weight column vector $\mathbf{w}_1 = [1\ 1\ 1]$ and bias $b_1 = -2.0$ for the linear layer; $\alpha_1 = 0.8, \sigma_1 = 1.0, \gamma_1 = 2.0$, $\mu_1 = -0.37$ parameters for the batch normalization layer. First, we apply the linear layer transformation (Eq. 2.1 in Table 2.1). We create a temporary variable for this intermediate output, $t_1^{lin} = \langle \mathbf{x}, \mathbf{w}_1 \rangle + b_1 = x_1 + x_2 + x_3 - 2.0$. Second, we apply the batch normalization (Eq. 2.2 in Table 2.1) and obtain $t_1^{bn} = (x_1 + x_2 + x_3 - 2.0 + 0.37) \cdot 0.8 + 2.0$. After the binarization (Eq. 2.3 in Table 2.1), we obtain the constraints $\mathrm{S}_1 = ((x_1 + x_2 + x_3 - 2.0 + 0.37) \cdot 0.8 + 2.0 \geq 0)$ and $\mathrm{S}_1 \Leftrightarrow v_1 = 1$. Next, we move all the constants to the right side of the inequality: $x_1 + x_2 + x_3 \geq -2.0/0.8 + 2.0 - 0.37 \Leftrightarrow v_1 = 1$. Lastly, we translate

Table 2.1. BNN definition as a set of layers of transformations.

A. Internal Block $f_{\mathtt{blk}_k}(\mathbf{v}_k) = \mathbf{v}_{k+1}$

1) Linear Layer

$$t_i^{lin} = \langle \mathbf{v}_k, \mathbf{w}_i \rangle + b_i \tag{2.1}$$

where $i = 1, ..., n_{k+1}$, $\mathbf{w}_i$ is the ith column in $\mathrm{W}_k \in \{-1, 1\}^{n_k \times n_{k+1}}$, $\mathbf{b}$ is the bias row vector $\in \mathbb{R}^{n_{k+1}}$ and $\mathbf{y} \in \mathbb{R}^{n_{k+1}}$

2) Batch Normalization

$$t_i^{bn} = \frac{t_i^{lin} - \mu_{k_i}}{\sigma_{k_i}} \cdot \alpha_{k_i} + \gamma_{k_i} \tag{2.2}$$

where $i = 1, ..., n_{k+1}$, α_k is the kth weight row vector $\in \mathbb{R}^{n_{k+1}}$, γ_k is the bias $\in \mathbb{R}^{n_{k+1}}$, $\mu_k \in \mathbb{R}^{n_{k+1}}$ is the mean and $\sigma_k \in \mathbb{R}^{n_{k+1}}$ is the standard deviation.

3) Binarization

$$t_i^{bn} \geq 0 \Rightarrow v_{k+1_i} = 1 \tag{2.3}$$

$$t_i^{bn} < 0 \Rightarrow v_{k+1_i} = -1 \tag{2.4}$$

where $i = 1, ..., n_{k+1}$.

B. Output Block $f_{out}(\mathbf{v}_d) = \mathbf{y}$

1) Linear Layer

$$q_i^{lin} = \langle \mathbf{v}_d, \mathbf{w}_j \rangle + b_i \tag{2.5}$$

where $\mathbf{v}_d \in \{-1, 1\}^{n_d}$, $\mathbf{w}_j$ is the jth column $\in \mathbb{R}^{n_d \times s}$, $\mathbf{b} \in \mathbb{R}^s$ is the bias vector.

2) Argmax

$$y_i = 1 \Leftrightarrow i = \arg\max(\mathbf{q}^{lin}) \tag{2.6}$$

the input from the $\{-1, 1\}$ domain to the boolean domain, $x_i = 2x_i^{(b)} - 1, i \in \{1, 2, 3\}$, resulting in the following constraint: $2(x_1^{(b)} + x_2^{(b)} + x_3^{(b)}) - 3 \geq -0.87$. We use a sound approximation for the constant on the right side to get rid of the real values and obtain $x_1^{(b)} + x_2^{(b)} + x_3^{(b)} \geq \lceil 1.065 \rceil = 2$. For notational simplicity the variables x_1, x_2, x_3 in Figure 2.1 are boolean variables (since $x = 1 \Leftrightarrow x^{(b)} = 1$).

To place this in the context of the security application in Section 2.3, we examine the effect of two arbitrary trojan attack procedures. Their aim is to manipulate the output of a given neural network, f, to a target class for inputs with a particular trigger. Let us consider the trigger to be $x_3 = 1$ and the target class $y = 0$ for two

Table 2.2. Encoding for a binarized neural network BNN($\mathbf{x}$) to cardinality constraints, where $\mathbf{v_1} = \mathbf{x}$. MILP stands for Mixed Integer Linear Programming, ILP stands for Integer Linear Programming.

A. $f_{\mathtt{blk}_k}(\mathbf{v}_k, \mathbf{v}_{k+1})$ to $\mathtt{BLK}_k(\mathbf{v}_k^{(b)}, \mathbf{v}_{k+1}^{(b)})$

$\mathtt{MILP}_{\mathtt{blk}}$: $\dfrac{\text{Eq (1), Eq (2), Eq (3)}, \alpha_{k_i} > 0}{\langle \mathbf{v}_k, \mathbf{w}_i \rangle \geq -\frac{\sigma_{k_i}}{\alpha_{k_i}} \cdot \gamma_{k_i} + \mu_{k_i} - b_i, i = 1, ..., n_{k+1}}$

$\mathtt{ILP}_{\mathtt{blk}}$: $\dfrac{\alpha_{k_i} > 0}{\begin{array}{l}\langle \mathbf{v}_k, \mathbf{w}_i \rangle \geq C_i \Leftrightarrow v_{k+1_i} = 1, i = 1, ..., n_{k+1} \\ \langle \mathbf{v}_k, \mathbf{w}_i \rangle < C_i \Leftrightarrow v_{k+1_i} = -1, i = 1, ..., n_{k+1} \\ C_i = \lceil -\frac{\sigma_{k_i}}{\alpha_{k_i}} \cdot \gamma_{k_i} + \mu_{k_i} - b_i \rceil\end{array}}$

$\mathtt{Card}_{\mathtt{blk}}$: $\dfrac{v^{(b)} = 2v - 1, v \in \{-1, 1\}}{\mathrm{BLK}_k(\mathbf{v}_k^{(b)}, \mathbf{v}_{k+1}^{(b)}) = \sum_{j \in w_{k_i}^+} v_{k_j}^{(b)} + \sum_{j \in w_{k_i}^-} \overline{v}_{k_j}{}^{(b)} \geq C_i' + |w_{k_i}^-| \Leftrightarrow v_{k+1_i}^{(b)} = 1, C_i' = \lceil (C_i + \sum_{j=1}^{n_k} w_{ji})/2 \rceil}$

B. $f_{out}(\mathbf{v}_d, \mathbf{y})$ to $\mathtt{OUT}(\mathbf{v}_d^{(b)}, \mathbf{ord}, \mathbf{y})$

$\mathtt{Order}$: $\dfrac{ord_{ij} \in \{0, 1\}}{q_i^{lin} \geq q_j^{lin} \Leftrightarrow ord_{ij} = 1}$

$\mathtt{MILP}_{\mathtt{out}}$: $\dfrac{\text{Eq (5), Eq (}\mathtt{Order}\text{)}}{\langle \mathbf{v}_d, \mathbf{w}_i - \mathbf{w}_j \rangle \geq b_j - b_i \Leftrightarrow ord_{ij} = 1}$

$\mathtt{ILP}_{\mathtt{out}}$: $\langle \mathbf{v}_d, \mathbf{w}_i - \mathbf{w}_j \rangle \geq \lceil b_j - b_i \rceil \Leftrightarrow ord_{ij} = 1$

$\mathtt{Card}_{\mathtt{out}}$: $\dfrac{v^{(b)} = 2v - 1, v \in \{-1, 1\}}{\begin{array}{l}\mathrm{OUT}(\mathbf{v}_d^{(b)}, \mathbf{ord}, \mathbf{y}) = \Big(\Big(\sum_{p \in w_i^+ \cap w_j^-} v_{d_p}^{(b)} - \sum_{p \in w_i^- \cap w_j^+} v_{d_p}^{(b)} \geq \lceil E_{ij}/2 \rceil \Big) \Leftrightarrow ord_{ij} \wedge \sum_{i=1}^{s} ord_{ij} = s \Leftrightarrow y_i = 1 \Big), \\ E_{ij} = \lceil (b_j - b_i + \sum_{p=1}^{n_d} w_{ip} - \sum_{p=1}^{n_d} w_{jp})/2 \rceil\end{array}}$

C. f_i to BNN

$$\mathrm{BNN}(\mathbf{x}^{(b)}, \mathbf{y}, \mathbf{v}_2^{(b)}, \ldots, \mathbf{v}_d^{(b)}, \mathbf{ord}) = \mathrm{BLK}_1(\mathbf{x}^{(b)}, \mathbf{v_2}^{(b)}) \bigwedge_{k=2}^{d-1} \Big(\mathrm{BLK}_k(\mathbf{v}_k^{(b)}, \mathbf{v}_{k+1}^{(b)}) \Big) \wedge \mathrm{OUT}(\mathbf{v}_d^{(b)}, \mathbf{y}, \mathbf{ord})$$

trojaned neural nets, f_1 and f_2 (shown in Figure 2.1). Initially, f outputs class 0 for only one input that has the trigger $x_3 = 1$. The first observation is that f_1 is equivalent to f, even though its parameters have changed. The second observation is that f_2 changes its output prediction for the input $x_1 = 0, x_2 = 1, x_3 = 1$ to the target class 0. We want NPAQ to find how much do f_1 and f_2 change their predictions for the target class with respect to the inputs that have the trigger, i.e., $|R(\varphi_1)| < |R(\varphi_2)|$, where φ_1, φ_2 are trojan property specifications (property P_2 as outlined Section 2.3).

Encoding Details. The details of our encoding in Table 2.2 are similar to [156]. We first encode each block to mixed integer linear programming and implication constraints, applying the $\mathtt{MILP}_{\mathtt{blk}}$ rule for the internal block and $\mathtt{MILP}_{\mathtt{out}}$ for the

outer block (Table 2.2). To get rid of the reals, we use sound approximations to bring the constraints down to integer linear programming constraints (see $\texttt{ILP}_{\texttt{blk}}$ and $\texttt{ILP}_{\texttt{out}}$ in Table 2.2). For the last step, we define a 1:1 mapping between variables in the binary domain $x \in \{-1, 1\}$ and variables in the boolean domain $x^{(b)} \in \{0, 1\}$, $x^{(b)} = 2x - 1$. Equivalently, for $x \in \{-1, 1\}$ there exists a unique $x^{(b)}$: $(x^{(b)} \Leftrightarrow\ x = 1)\ \wedge\ (\overline{x}^{(b)} \Leftrightarrow\ x = -1)$. Thus, for every block $f_{\text{blk}_k}(\mathbf{v}_k) = \mathbf{v}_{k+1}$, we obtain a corresponding formula over booleans denoted as $\text{BLK}_k(\mathbf{v}_k^{(b)}, \mathbf{v}_{k+1}^{(b)})$, as shown in rule $\texttt{Card}_{\texttt{blk}}$ (Table 2.2). Similarly, for the output block f_{out} we obtain $\text{OUT}(\mathbf{v}_d, \mathbf{ord}, \mathbf{y})$. We obtain the representation of $\mathbf{y} = f_i(\mathbf{x})$ as a formula BNN shown in Table 2.2. For notational simplicity, we denote the introduced intermediate variables $\mathbf{v}_k^{(b)} = [v_{k_1}^{(b)}, \ldots, v_{k_{n_k}}^{(b)}], k = 2, \ldots, d$ and $\mathbf{ord} = [ord_i, \ldots, ord_{n_d \cdot n_d}]$ as $\mathbf{a}_V$. Since there is a 1:1 mapping between $\mathbf{x}$ and $\mathbf{x}^{(b)}$ we use the notation $\mathbf{x}$, when it is clear from context which domain $\mathbf{x}$ has. We refer to BNN as the formula $\text{BNN}(\mathbf{x}, \mathbf{y}, \mathbf{a}_V)$.

Lemma 3. *Given a binarized neural net $f_i : \{-1, 1\}^n \to \{0, 1\}^s$ over inputs $\mathbf{x}$ and outputs $\mathbf{y}$, and a property P, let φ be the specification for P, $\varphi(\mathbf{x}, \mathbf{y}) = (\mathbf{y} = f_i(\mathbf{x})) \wedge P(\mathbf{x}, \mathbf{y})$, where we represent $\mathbf{y} = f_i(\mathbf{x})$ as $BNN(\mathbf{x}, \mathbf{y}, \mathbf{a}_V)$. Then φ is equicardinal to $BNN(\mathbf{x}, \mathbf{y}, \mathbf{a}_V)$.*

For the ease of proof of Lemma 3, we first introduce the notion of independent support.

Independent Support. An independent support **ind** for a formula $F(\mathbf{x})$ is a subset of variables appearing in formula F, $\mathbf{ind} \subseteq \mathbf{x}$, that uniquely determine the values of the other variables in any satisfying assignment [28]. In other words, if there exist two satisfying assignments τ_1 and τ_2 that agree on **ind** then $\tau_1 = \tau_2$. Then $R(\text{F}) = R(\text{F}) \downarrow \mathbf{ind}$.

Proof. We observe that the intermediate variables for each block in the neural network, namely $\mathbf{v}_k$ for the kth block, are introduced by double implication constraints. Hence, not only are both part (a) and part (b) of definition 4 true, but the satisfying assignments for the intermediate variables $\mathbf{a}_V$ are uniquely determined by $\mathbf{x}$. This is the main idea behind the proof, which we show through the introduced independent support. We prove that $R(\varphi) = R(\varphi) \downarrow \mathbf{x}$ by showing that $\mathbf{x}$ is an independent

support for BNN. This follows directly from the construction of BNN. If $\mathbf{x}$ is an independent support then the following has to hold true:

$$\mathrm{G} = \Big(\mathrm{BNN}(\mathbf{x},\mathbf{y},\mathbf{a}_V) \wedge \mathrm{BNN}(\mathbf{x}',\mathbf{y}',\mathbf{a}'_V) \wedge (\mathbf{x}=\mathbf{x}') \Rightarrow (\mathbf{y}=\mathbf{y}') \wedge (\mathbf{a}_V = \mathbf{a}'_V)\Big)$$

As per Table 2.2, we expand $\mathrm{BNN}(\mathbf{x},\mathbf{y})$:

$$\begin{aligned} G = \Big(&(\mathrm{BLK}_1(\mathbf{x},\mathbf{v}_2^{(b)}) \wedge \mathrm{BLK}_2(\mathbf{v}_2^{(b)},\mathbf{v}_3^{(b)}) \wedge \ldots \wedge \mathrm{OUT}(\mathbf{v}_d^{(b)},\mathbf{ord},\mathbf{y}) \\ \wedge\ &(\mathrm{BLK}_1(\mathbf{x}',\mathbf{v}'^{(b)}_2) \wedge \mathrm{BLK}_2(\mathbf{v}'^{(b)}_2,\mathbf{v}'^{(b)}_3) \wedge \ldots \wedge \mathrm{OUT}(\mathbf{v}'^{(b)}_d,\mathbf{ord},\mathbf{y}') \\ \wedge\ &(\mathbf{x}=\mathbf{x}') \Rightarrow (\mathbf{y}=\mathbf{y}') \wedge (\mathbf{a}_V = \mathbf{a}'_V)\Big) \end{aligned}$$

G is valid if and only if $\neg G$ is unsatisfiable.

$$\begin{aligned} \neg G = \Big(&(\mathrm{BLK}_1(\mathbf{x},\mathbf{v}_2^{(b)}) \wedge \ldots \wedge \mathrm{OUT}(\mathbf{v}_d^{(b)},\mathbf{y})) \\ \wedge\ &(\mathrm{BLK}_1(\mathbf{x}',\mathbf{v}'^{(b)}_2) \wedge \ldots \wedge \mathrm{OUT}(\mathbf{v}'^{(b)}_d,\mathbf{y}') \wedge (\mathbf{x}=\mathbf{x}') \wedge \neg(\mathbf{y}=\mathbf{y}')\Big) \\ \vee\ &\Big(\mathrm{BLK}_1(\mathbf{x},\mathbf{v}_2^{(b)}) \wedge \ldots \wedge \mathrm{OUT}(\mathbf{v}_d^{(b)},\mathbf{y}) \\ \wedge\ &(\mathrm{BLK}_1(\mathbf{x}',\mathbf{v}'^{(b)}_2) \wedge \ldots \wedge \mathrm{OUT}(\mathbf{v}'^{(b)}_d,\mathbf{y}') \wedge (\mathbf{x}=\mathbf{x}') \wedge \neg(\mathbf{a}_V = \mathbf{a}'_V)\Big) \end{aligned}$$

The first block's formula's introduced variables $\mathbf{v}_2^{(b)}$ are uniquely determined by $\mathbf{x}$. For every formula BLK_k corresponding to an internal block the introduced variables are uniquely determined by the input variables. Similarly, for the output block (formula OUT in Table 2.2). If $\mathbf{x} = \mathbf{x}'$ then $\mathbf{v}_2^{(b)} = \mathbf{v}'^{(b)}_2, \ldots \Rightarrow \mathbf{a}_V = \mathbf{a}'_V$, so the second clause is not satisfied. Then, since $\mathbf{v}_d^{(b)} = \mathbf{v}'^{(b)}_d \Rightarrow \mathbf{y} = \mathbf{y}'$. Thus, G is a valid formula which implies that $\mathbf{x}$ forms an independent support for the BNN formula $\Rightarrow R(\varphi) = R(\varphi) \downarrow \mathbf{x}$.

□

2.6.2 Cardinality Constraints to CNF

Observe that we can express each block in BNN as a conjunction of cardinality constraints [214, 7, 3]. Cardinality constraints are constraints over boolean variables $x_1, \ldots, x_n$ of the form $x_1 + \ldots + x_n \triangle c$, where $\triangle \in \{=, \leq, \geq\}$. More specifically, by applying the $\mathtt{Card}_{\mathtt{blk}}$ rule (Table 2.2), we obtain a conjunction over

cardinality constraints S_{k_i}, together with a double implication: $\mathrm{BLK}_k(\mathbf{v}_k^{(b)}, \mathbf{v}_{k+1}^{(b)}) = \bigwedge_{i=1}^{n_{k+1}} \mathrm{S}_{k_i}(\mathbf{v}_k^{(b)}) \Leftrightarrow v_{k+1_i}^{(b)}$. We obtain a similar conjunction of cardinality constraints for the output block ($\texttt{Card}_{\texttt{out}}$, Table 2.2). The last step for obtaining a Boolean formula representation for the BNN is encoding the cardinality constraints to CNF.

We choose cardinality networks [7, 3] to encode the cardinality constraints to CNF formulas and show for this particular encoding that the resulting CNF is equicardinal to the cardinality constraint. Cardinality networks implement several types of gates, i.e., merge circuits, sorting circuits and 2-comparators, that compose to implement a merge sort algorithm. More specifically, a cardinality constraint of the form $\mathrm{S}(\mathbf{x}) = x_1 + \ldots + x_n \geq c$ has a corresponding cardinality network, $\mathrm{Card}_c = \Big((\mathrm{Sort}_c(x_1, \ldots, x_n) = (y_1, \ldots, y_c)) \wedge y_c\Big)$, where Sort is a sorting circuit. As shown by [7, 3], the following holds true:

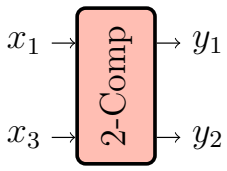
Proposition 1. *A Sort_c network with an input of n variables, outputs the first c sorted bits. $\mathrm{Sort}_c(x_1, \ldots, x_n) = (y_1, \ldots, y_c)$ where $y_1 \geq y_2 \geq \ldots \geq y_c$.*

We view Card_c as a circuit where we introduce additional variables to represent the output of each gate, and the output of Card_c is 1 only if the formula S is true. This is similar to how a Tseitin transformation [236] encodes a propositional formula into CNF.

Running Example. Revisiting our example in Section 2.6.1, consider f_2's cardinality constraint corresponding to v_1, denoted as $\mathrm{S}'_1 = x_1 + x_3 \geq 2$. This constraint translates to the most basic gate of cardinality networks, namely a 2-comparator [14, 7] shown in Figure 2.2. Observe that while this efficient encoding ensures that S_1 is equi-satisfiable to the formula 2-Comp $\wedge\, y_2$, counting over the CNF formula does not preserve the count, i.e., it over-counts due to variable y_1. Observe, however, that this encoding is equicardinality and thus, a projected model count on $\{x_1, x_3\}$ gives the correct model count of 1. The remaining constraints shown in Figure 2.1 are encoded similarly and not shown here for brevity.

Lemma 4 (Substitution)**.** *Let F be a Boolean formula defined over the variables Vars and $p \in Vars$. For all satisfying assignments $\tau \models F \Rightarrow \tau|_{Vars-\{p\}} \models F[p \mapsto \tau[p]]$.*

Lemma 5. *For a given cardinality constraint, $S(\mathbf{x}) = x_1 + \ldots + x_n \geq c$, let $Card_c$ be the CNF formula obtained using cardinality networks, $Card_c(\mathbf{x}, \mathbf{a}_C) :=$*



2-Comp Clauses	x_1	x_3	y_1	y_2	2-Comp $\wedge$ y_2
	0	0	0	0	0
$\overline{x}_1 \Rightarrow \overline{y}_2$	0	1	0/1	0	0
$\overline{x}_3 \Rightarrow \overline{y}_2$	1	0	0/1	0	0
$\overline{x}_1 \wedge \overline{x}_3 \Rightarrow \overline{y}_1$	1	1	0/1	0/1	1

Figure 2.2. Cardinality networks encoding for $x_1 + x_3 \geq 2$. For this case, cardinality networks amount to a 2-comparator gate. Observe there are two satisfying assignments for 2-Comp $\wedge$ y_2 due to the "don't care" assignment to y_1.

$(Sort_c(x_1, \ldots, x_n) = (y_1, \ldots, y_c) \wedge y_c)$, where $\mathbf{a}_C$ are the auxiliary variables introduced by the encoding. Then, $Card_c$ is equicardinal to S.

(a) $\forall \tau \models S \Rightarrow \exists \sigma, \sigma \models Card_c \wedge \sigma|_{\mathbf{x}} = \tau$.

(b) $\forall \sigma \models Card_c \Rightarrow \tau_3|_{\mathbf{x}} \models S$.

Proof. (a) Let $\tau \models$ S $\Rightarrow$ there are least c x_i's such that $\tau[x_i] = 1, i \geq c$. Thus, under the valuation τ_1 to the input variables $x_1, \ldots, x_n$, the sorting network outputs a sequence $y_1, \ldots, y_c$ where $y_c = 1$, where $y_1 \geq \ldots \geq y_c$ (Proposition 1). Therefore, $\text{Card}_c[\mathbf{x} \mapsto \tau] = (\text{Sort}_c(x_1 \mapsto \tau[x_1], \ldots, x_n \mapsto \tau[x_n]) = (y_1, \ldots, y_c) \wedge y_c)$ is satisfiable. This implies that $\exists \sigma, \sigma \models \text{Card}_c \wedge \sigma|_{\mathbf{x}} = \tau$.

(b) Let $\sigma \models \text{Card}_c \Rightarrow \sigma[y_c] = 1$. By Lemma 4, $\sigma|_{\mathbf{x}} \models \text{Card}_c[y_i \mapsto \sigma[y_i]], \forall y_i \in \mathbf{a}_C$. From Proposition 1, under the valuation σ, there are at least c x_i's such that $\sigma[x_i] = 1, i \geq c$. Therefore, $\sigma|_{\mathbf{x}} \models$ S.

□

For every S_{k_i}, $k = 1, \ldots, d, i = 1, \ldots, n_{k+1}$, we have a CNF formula C_{k_i}. The final CNF formula for $\text{BNN}(\mathbf{x}, \mathbf{y}, \mathbf{a}_V)$ is denoted as $\text{C}(\mathbf{x}, \mathbf{y}, \mathbf{a})$, where $\mathbf{a} = \mathbf{a}_V \bigcup_{k=1}^{d} \bigcup_{i=1}^{n_{k+1}} \mathbf{a}_C^{k_i}$ and $\mathbf{a}_C^{k_i}$ is the set of variables introduced by encoding S_{k_i}.

Encoding Size. The total CNF formula size is linear in the size of the model. Given one cardinality constraint $\text{S}(\mathbf{v_k})$, where $|\mathbf{v_k}| = n$, a cardinality network encoding produces a CNF formula with $O(n\ log^2\ c)$ clauses and variables. The constant c is the maximum value that the parameters of the BNN can take, hence the encoding is linear in n. For a given layer with m neurons, this translates to m cardinality constraints, each over n variables. Hence, our encoding procedure produces $O(m \times n)$ clauses and variables for each layer. For the output block, s is the number of output

classes and n is the number of neurons in the previous layer. Due to the ordering relation encoding the arg max, there are $O(s \times s \times n)$ clauses and variables for the output block. Therefore, the total size for a BNN with l layers of the CNF is $O(m \times n \times l + s \times s \times n)$, which is linear in the size of the original model.

Alternative Encodings. Besides cardinality networks, there are many other encodings from cardinality constraints to CNF [7, 3, 2, 214, 48] that can be used as long as they are equicardinal. We do not formally prove here but we strongly suspect that adder networks [48] and BDDs [2] have this property. Adder networks [48] provide a compact, linear transformation resulting in a CNF with $O(n)$ variables and clauses. The idea is to use adders for numbers represented in binary to compute the number of activated inputs and a comparator to compare it to the constant c. A BDD-based [48] encoding builds a BDD representation of the constraint. It uses $O(n^2)$ clauses and variables. For approximate counting techniques, empirically, these similar encodings yield similar performance [177].

2.6.3 Projected Model Counting

We instantiate the property P encoded in CNF and the neural network encoded in a CNF formulae C. We make the observation that we can directly count the number of satisfying assignment for φ over a subset of variables, known as projected model counting [25]. NPAQ uses an approximate model counter with strong PAC-style guarantees. ApproxMC3 [218] is an approximate model counter that can directly count on a projected formula making a logarithmic number of calls in the number of formula variables to an NP-oracle, namely a SAT solver.

Theorem 3. NPAQ *is an* (ϵ, δ)*-NQV.*

Proof. First, by Lemma 2, since each cardinality constraint S_{k_i} is equicardinal to C_{k_i} (Lemma 5), the conjunction over the cardinality constraints is also equicardinal. Second, by Lemma 3, BNN is equicardinal to C. Since we use an approximate model counter with (ϵ, δ) guarantees [218], NPAQ returns r for a given BNN and a specification φ with (ϵ, δ) guarantees. □

2.7 Implementation & Evaluation

We aim to answer the following research questions:

(RQ1) To what extent does NPAQ scale to, e.g., how large are the neural nets and the formulae that NPAQ can handle?

(RQ2) How effective is NPAQ at providing sound estimates for practical security applications?

(RQ3) Which factors influence the performance of NPAQ on our benchmarks and how much?

(RQ4) Can NPAQ be used to refute claims about security-relevant properties over BNNs?

Implementation. We implemented NPAQ in about $5,000$ LOC of Python and C++. We use the PyTorch (v1.0.1.post2) [168] deep learning platform to train and test binarized neural networks. For encoding the BNNs to CNF, we build our own tool using the PBLib library [175] for encoding the cardinality constraints to CNF. The resulting CNF formula is annotated with a projection set and NPAQ invokes the approximate model counter ApproxMC3 [218] to count the number of solutions. We configure a tolerable error $\epsilon = 0.8$ and confidence parameter $\delta = 0.2$ as defaults throughout the evaluation.

Models. Our benchmarks consist of BNNs, on which we tested the properties derived from the 3 applications outlined in Section 2.3. The utility of NPAQ in these security applications is discussed in Sections 2.7.2- 2.7.4. For each application, we trained BNNs with the following 4 different architectures:

- $\mathbf{ARCH}_1$ - $\mathrm{BLK}_1(100)$
- $\mathbf{ARCH}_2$ - $\mathrm{BLK}_1(50)$, $\mathrm{BLK}_2(20)$
- $\mathbf{ARCH}_3$ - $\mathrm{BLK}_1(100)$, $\mathrm{BLK}_2(50)$
- $\mathbf{ARCH}_4$ - $\mathrm{BLK}_1(200)$, $\mathrm{BLK}_2(100)$, $\mathrm{BLK}_3(100)$

For each architecture, we take snapshots of the model learnt at different epochs. In total, this results in 84 total models with $6,560 - 53,010$ parameters for models trained with the MNIST dataset and $4,692 - 45,402$ parameters for models trained with the UCI Adult dataset. Encoding various properties (Sections 2.7.2- 2.7.4)

results in a total of $1,056$ distinct formulae. For each formula, NPAQ returns r i.e., the number of satisfying solutions. Given r, we calculate `PS` i.e., the percentage of the satisfying solutions with respect to the total input space size. The meaning of `PS` percentage values is application-specific. In trojan attacks, `PS(tr)` represents inputs labeled as the target class. In robustness quantification, `PS(adv)` reports the adversarial samples.

Table 2.3. Influence of (ϵ, δ) on NPAQ's Performance. The count and time taken to compute the bias in ARCH_2 trained on UCI Adult dataset for changes in values features (marital status, gender, and race) i.e., the percentage of individuals whose predicted income changes from $\leq$ 50K to $>$ 50K when all the other features are same. NLC represents the natural logarithm of the count NPAQ generates. Time represents the number of hours NPAQ takes to solve the formulae. x represents a timeout.

Feature	$\delta = 0.2$								$\epsilon = 0.1$							
	$\epsilon = 0.1$		$\epsilon = 0.3$		$\epsilon = 0.5$		$\epsilon = 0.8$		$\delta = 0.01$		$\delta = 0.05$		$\delta = 0.1$		$\delta = 0.2$	
	NLC	Time	NLC	Time	NLC	Time	NLC	Time	NLC	Time	NLC	Time	NLC	Time	NLC	Time
Marital Status	39.10	8.79	39.08	1.35	39.09	0.80	39.13	0.34	x	x	39.07	22.48	39.07	15.74	39.10	8.79
Race	40.68	3.10	40.64	0.68	40.65	0.42	40.73	0.27	40.68	14.68	40.67	8.21	40.67	5.80	40.68	3.10
Gender	41.82	3.23	41.81	0.62	41.88	0.40	41.91	0.27	41.81	15.48	41.81	8.22	41.81	6.02	41.82	3.23

Datasets. We train models over 2 standard datasets. Specifically, we quantify robustness and trojan attack effectiveness on the MNIST [128] dataset and estimate fairness queries on the UCI Adult dataset [238]. We choose them as prior work use these datasets [64, 184, 65, 4].

MNIST. The dataset contains $60,000$ gray-scale 28×28 images of handwritten digits with 10 classes. In our evaluation, we resize the images to 10×10 and binarize the normalized pixels in the images.

UCI Adult Census Income. The dataset is $48,842$ records with 14 attributes such as age, gender, education, marital status, occupation, working hours, and native country. The task is to predict whether a given individual has an income of over $\$50,000$ a year. 5/14 attributes are numerical variables, while the remaining attributes are categorical variables. To obtain binary features, we divide the values of each numerical variables into groups based on its deviation. Then, we encode each feature with the least amount of bits that are sufficient to represent each category in the feature. For example, we encode the race feature which has 5 categories in total with 3 bits, leading to 3 redundant values in this feature. We remove the redundant values by encoding the property to disable the usage of these values in NPAQ. We

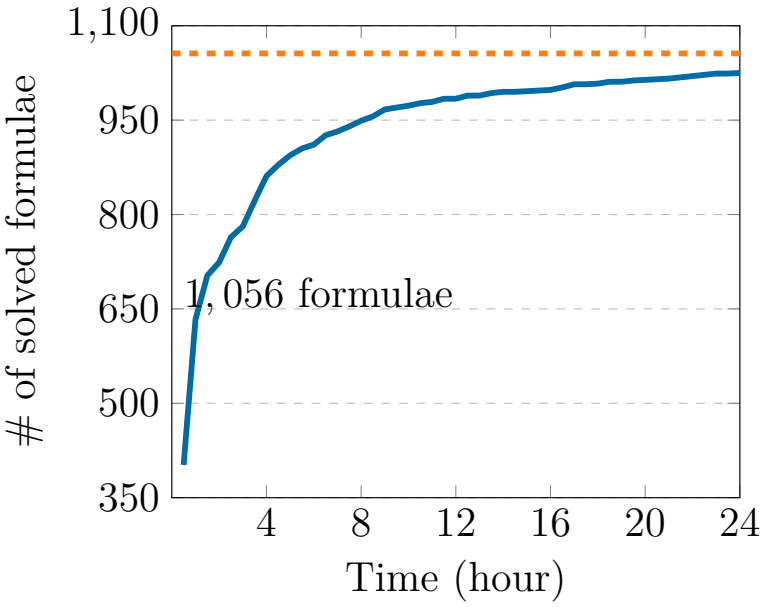


Figure 2.3. Number of formulae NPAQ solves with respect to the time. The solid line represents the aggregate number of formulae NPAQ solves before the given time. The dashed line represents the total number of formulae.

consider 66 binary features in total.

Experimental Setup. All experiments are performed on 2.5 GHz CPUs, 56 cores, 64GB RAM. Each counting process executed on one core and 4GB memory cap and a 24-hour timeout per formula.

2.7.1 NPAQ Benchmarking

We benchmark NPAQ and report breakdown on $1,056$ formulae.

Estimation Efficiency. NPAQ successfully solves 97.1% $(1,025$ / $1,056)$ formulae. In quantifying the effectiveness of trojan attacks and fairness applications, the raw size of the input space (over all possible choices of the free variables) is 2^{96} and 2^{66}, respectively. Naive enumeration for such large spaces is intractable. NPAQ returns estimates for 83.3% of the formulae within 12 hours and 94.8% of the formulae within 24 hours for these two applications. In robustness application, the total input sizes are a maximum of about 7.5×10^7.

Result 1: NPAQ solves 97.1% formulae in 24-hour timeout.

Encoding Efficiency. NPAQ takes a maximum of 1 minute to encode each model, which is less than 0.05% of the total timeout. The formulae size scale linearly with the model, as expected from encoding construction. NPAQ presently utilizes off-the-shelf CNF counters, and their performance heavily dominates NPAQ time. NPAQ presently scales to formulae of ~3.5×10^6 variables and ~6.2×10^6 clauses. However, given the encoding efficiency, we expect NPAQ to scale to larger models with future CNF counters [29, 218].

Result 2: NPAQ takes ~1 minute to encode the model.

Number of Formulae vs. Time. Figure 2.3 plots the number of formulae solved with respect to the time, the relationship is logarithmic. NPAQ solves 93.2% formulae in the first 12 hours, whereas, it only solves 3.9% more in the next 12 hours. We notice that the neural net depth impacts the performance, most timeouts (27/31) stem from ARCH_4. 26/31 timeouts are for Property P1 (Section 2.3) to quantify adversarial robustness. Investigating why certain formulae are harder to count is an active area of independent research [44, 43].

Performance with Varying (ϵ, δ). We investigate the relationship between different error and confidence parameters and test co-relation with parameters that users can pick. We select a subset of formulae [3] which have varying degrees of the number of solutions, a large enough input space which is intractable for enumeration, and varying time performance for the baseline parameters of $\epsilon = 0.8, \delta = 0.2$. The formulae in our dataset that satisfy these requirements arise in the fairness application. More specifically, we chose the 3 formulae encoding the fairness properties over ARCH_2 where the input space is 2^{66} and the `PS` varies from 4.09 to 76.59.

We first vary the error tolerance (or precision), $\epsilon \in \{0.1, 0.3, 0.5, 0.8\}$ while keeping the same $\delta = 0.2$ for the fairness application, as shown in Table 2.3. This table illustrates no significant resulting difference in counts reported by NPAQ under different precision parameter values. More precisely, the largest difference as

[3] Our timeout is 24 hours per formula, so we resorted to checking a subset of formulae.

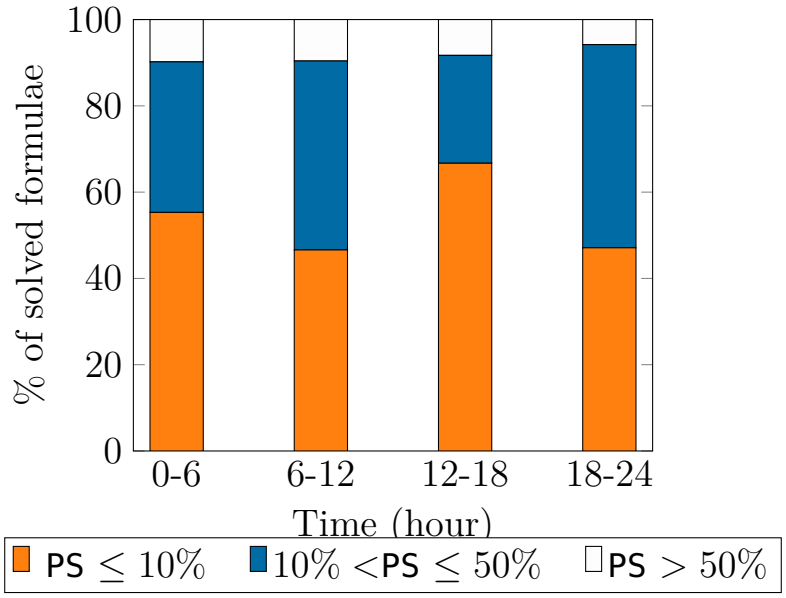


Figure 2.4. Percentage of formulae NPAQ solves with respect to 3 PS intervals (less than 10% in orange, between 10% and 50% in blue and more than 50% in white) and 4 time intervals.

the natural logarithmic of the count is 0.1 for $\epsilon = 0.3$ and $\epsilon = 0.8$ for the feature "Gender". This suggests that for these formulae, decreasing the error bound does not yield a much higher count precision.

Higher precision does come at a higher performance cost, as the $\epsilon = 0.1$ takes $16\times$ more time than $\epsilon = 0.8$. The results are similar when varying the confidence parameter $\delta \in \{0.2, 0.1, 0.05, 0.01\}$ (smaller is better) for $\epsilon = 0.1$ (Table 2.3). This is because the number of calls to the SAT solver depends only on the δ parameter, while ϵ dictates how constrained the space of all inputs or how small the "bucket" of solutions is [218, 27]. Both of these significantly increase the time taken. Users can tune ϵ and δ based on the required applications precision and the available time budget.

Result 3: NPAQ reports no significant difference in the counts produced when configured with different ϵ and δ.

PS vs. Time. We investigate if NPAQ solving efficiency varies with increasing count size. Specifically, we measure the PS with respect to the time taken for all the 1,056 formulae. Figure 2.4 shows the PS plot for 4 time intervals and 3 density

intervals. We observe that the number of satisfying solutions do not significantly influence the time taken to solve the instance. This suggests that NPAQ is generic enough to solve formulae with arbitrary solution set sizes.

Result 4: For a given ϵ and δ, NPAQ solving time is not significantly influenced by the `PS`.

2.7.2 Case Study 1: Quantifying Robustness

We quantify the model robustness and the effectiveness of defenses for model hardening with adversarial training.

Number of Adversarial Inputs. One can count precisely what fraction of inputs, when drawn uniformly at random from a constrained input space, are misclassified for a given model. For demonstrating this, we first train 4 BNNs on the MNIST dataset, one using each of the architectures ARCH_1-ARCH_4. We encode the Property P1 (Section 2.3) corresponding to perturbation bound $k \in \{2, 3, 4, 5\}$. We take 30 randomly sampled images from the test set, and for each one, we encoded one property constraining adversarial perturbation to each possible value of k. This results in a total of 480 formulae on which NPAQ runs with a timeout of 24 hours per formula. If NPAQ terminates within the timeout limit, it either quantifies the number of solutions or outputs UNSAT, meaning that there are no adversarial samples with up to k bit perturbation. Table 2.4 shows the average number of adversarial samples and their `PS(adv)`, i.e., percentage of count to the total input space.

As expected, the number of adversarial inputs increases with k. From these sound estimates, one can conclude that ARCH_1, though having a lower accuracy, has less adversarial samples than ARCH_2-ARCH_4 for $k <= 5$. ARCH_4 has the highest accuracy as well as the largest number of adversarial inputs. Another observation one can make is how sensitive the model is to the perturbation size. For example, `PS(adv)` for ARCH_3 varies from $10.25 - 24.04\%$.

Effectiveness of Adversarial Training. As a second example of a usage scenario, NPAQ can be used to measure how much a model improves its robustness after applying certain adversarial training defenses. In particular, prior work has claimed

Table 2.4. Quantifying robustness for $\text{ARCH}_{1..4}$ and perturbation size from 2 to 5. ACC_b represents the percentage of benign samples in the test set labeled as the correct class. #(Adv) and `PS(adv)` represent the average number and percentage of adversarial samples separately. #(timeout) represents the number of times NPAQ timeouts.

Arch	ACC_b	Perturb Size	#(Adv)	`PS(adv)`	#(timeout)
$\textbf{ARCH}_1$	76	$k \leq 2$	561	11.10	0
		$k = 3$	26,631	16.47	0
		$k = 4$	685,415	17.48	0
		$k = 5$	16,765,457	22.27	0
$\textbf{ARCH}_2$	79	$k \leq 2$	789	15.63	0
		$k = 3$	35,156	21.74	0
		$k = 4$	928,964	23.69	0
		$k = 5$	21,011,934	27.91	0
$\textbf{ARCH}_3$	80	$k \leq 2$	518	10.25	0
		$k = 3$	24,015	14.85	0
		$k = 4$	638,530	16.28	0
		$k = 5$	18,096,758	24.04	4
$\textbf{ARCH}_4$	88	$k \leq 2$	664	13.15	0
		$k = 3$	25,917	16.03	1
		$k = 4$	830,129	21.17	4
		$k = 5$	29,138,314	38.70	17

that plain (unhardened) BNNs are possibly more robust than hardened models—one can quantitatively verify such claims [64]. Of the many proposed adversarial defenses [78, 64, 167, 136], we select two representative defenses [64], though our methods are agnostic to how the models are obtained. We use a fast gradient sign method [78] to generate adversarial inputs with up to $k = 2$ bits perturbation for both. In $\texttt{defense}_1$, we first generate the adversarial inputs given the training set and then retrain the original models with the pre-generated adversarial inputs and training set together. In $\texttt{defense}_2$ [64], alternatively, we craft the adversarial inputs while retraining the models. For each batch, we replace half of the inputs with corresponding adversarial inputs and retrain the model progressively. We evaluate the effectiveness of these two defenses on the same images used to quantify the robustness of the previous (unhardened) BNNs. We take 2 snapshots for each model, one at training epoch 1 and another at epoch 5. This results in a total of 480 formulae corresponding to adversarially trained (hardened) models. Table 2.5 shows

the number of adversarial samples and `PS(adv)`.

Table 2.5. Estimates of adversarial samples for maximum 2-bit perturbation on $\text{ARCH}_{1..4}$ for a plain BNN (epoch 0) and for 2 defense methods at epochs 1 and 5. ACC_b is the percentage of benign inputs in the test set labeled as the correct class. #(Adv) is the number of adversarial samples.

Arch	#(Adv)	Defense 1				Defense 2			
		Epoch = 1		Epoch = 5		Epoch = 1		Epoch = 5	
		ACC_b	#(Adv)	ACC_b	#(Adv)	ACC_b	#(Adv)	ACC_b	#(Adv)
ARCH_1	561	82.23	942	84.04	776	82.61	615	81.88	960
ARCH_2	789	79.55	1,063	77.10	1,249	81.76	664	78.73	932
ARCH_3	518	84.12	639	85.23	431	82.97	961	82.94	804
ARCH_4	664	88.15	607	88.31	890	88.85	549	85.75	619

Observing the sound estimates from NPAQ, one can confirm that plain BNNs are more robust than the hardened BNNs for 11/16 models, as suggested in prior work. Further, the security analyst can compare the two defenses. For both epochs, $\texttt{defense}_1$ and $\texttt{defense}_2$ outperform the plain BNNs only for 2/8 and 3/8 models respectively. Hence, there is no significant difference between $\texttt{defense}_1$ and $\texttt{defense}_2$ for the models we trained. One can use NPAQ estimates to select a model that has high accuracy on the benign samples as well as less adversarial samples. For example, the ARCH_4 model trained with $\texttt{defense}_2$ at epoch 1 has the highest accuracy (88.85%) and 549 adversarial samples.

2.7.3 Case Study 2: Quantifying Effectiveness of Trojan Attacks

The effectiveness of trojan attacks is often evaluated on a *chosen* test set, drawn from a particular distribution of images with embedded trojan triggers [**Trojannn**, 65]. Given a trojaned model, one may be interested in evaluating how effective is the trojaning outside this particular test distribution [**Trojannn**]. Specifically, NPAQ can be used to count how many images with a trojan trigger are classified to the desired target label, over the space of all possible images. Property P2 from Section 2.3 encodes this. We can then compare the NPAQ count vs. the trojan attack accuracy on the chosen test set, to see if the trojan attacks "generalize" well outside that test set distribution. Note that space of all possible inputs is too large to enumerate.

Table 2.6. Effectiveness of trojan attacks. TC represents the target class for the attack. Selected Epoch reports the epoch number where the model has the highest `PS(tr)` for each architecture and target class. x represents a timeout.

Arch	TC	Epoch 1		Epoch 10		Epoch 30		Selected
		`PS(tr)`	ACC_t	`PS(tr)`	ACC_t	`PS(tr)`	ACC_t	Epoch
$ARCH_1$	**0**	39.06	50.75	13.67	72.90	5.76	68.47	1
	1	42.97	43.49	70.31	74.20	42.97	67.63	10
	4	9.77	66.80	19.14	83.18	2.69	69.99	10
	5	27.73	58.35	25.78	53.30	7.42	39.77	1
	9	2.29	53.67	12.11	61.85	0.19	77.70	10
$ARCH_2$	**0**	1.51	27.98	1.46	48.30	9.38	59.36	30
	1	2.34	30.37	13.28	40.57	8.59	51.40	10
	4	1.07	38.54	0.21	27.41	0.59	37.45	1
	5	28.91	26.66	12.70	50.24	9.38	54.90	1
	9	0.15	36.39	0.38	41.81	0.44	42.99	30
$ARCH_3$	**0**	18.36	26.91	25.00	71.85	8.40	76.30	10
	1	4.79	15.23	34.38	50.57	21.48	60.33	10
	4	7.81	33.89	11.33	67.30	4.79	62.77	10
	5	26.56	63.11	19.92	71.92	18.75	79.23	1
	9	6.84	26.51	3.32	29.12	1.15	46.51	1
$ARCH_4$	**0**	x	10.40	3.32	36.89	4.88	60.14	30
	1	x	8.57	x	54.39	0.87	78.10	30
	4	x	9.95	1.44	62.46	0.82	82.47	10
	5	19.92	8.83	13.67	8.44	25.39	11.96	30
	9	x	19.64	7.03	58.39	1.44	74.83	10

As a representative of such analysis, we trained BNNs on the MNIST dataset with a trojaning technique adapted from Liu et al. [**Trojannn**] (the details of the procedure are outlined later). Our BNN models may obtain better attack effectiveness as the trojaning procedure progresses over time. Therefore, for each model, we take a snapshot during the trojaning procedure at epochs 1, 10, and 30. There are 4 models ($ARCH_1$-$ARCH_4$), and for each, we train 5 different models each classifying the trojan input to a distinct output label. Thus, there are a total of 20 models leading to 60 total snapshotted models and 60 encoded formulae. If NPAQ terminates within the timeout of 24 hours, it either quantifies the number of solutions or outputs `UNSAT`, indicating that no trojaned input is labeled as the target output at all. The effectiveness of the trojan attack is measured by two metrics:

- `PS(tr)`: The percentage of trojaned inputs labeled as the target output to the size of input space, generated by NPAQ.
- ACC_t: The percentage of trojaned inputs in the chosen test set labeled as the

desired target output.

Table 2.6 reports the `PS(tr)` and ACC_t. Observing these sound estimates, one can conclude that the effectiveness of trojan attacks on out-of-distribution trojaned inputs greatly differs from the effectiveness measured on the test set distribution. In particular, if we focus on the models with the highest `PS(tr)` for each architecture and target class (across all epochs), only 50% (10 out 20) are the same as when we pick the model with highest ACC_t instead. Thus, for these models, an attack whose goal is to maximize the number of inputs under which the classifier outputs the target class will fail on most inputs out-of-distribution that have the trigger present.

Attack Procedure. The trojaning process can be arbitrarily different from ours; the use of NPAQ for verifying them does not depend on it in any way. Our procedure is adapted from that of Liu et al. which is specific to models with real-valued weights. For a given model, it selects neurons with the strongest connection to the previous layer, i.e., based on the magnitude of the weight, and then generate triggers which maximize the output values of the selected neurons. This heuristic does not apply to BNNs as they have $\{-1, 1\}$ weights. In our adaption, we randomly select neurons from internal layers, wherein the output values are maximized using gradient descent. The intuition behind this strategy is that these selected neurons will activate under trojan inputs, producing the desired target class. For this procedure, we need a set of trojan and benign samples. In our procedure, we assume that we have access to a $10,000$ benign images, unlike the work in Liu et al. which generates this from the model itself. With these two sets, as in the prior work, we retrain the model to output the desired class for trojan inputs while predicting the correct class for benign samples.

2.7.4 Case Study 3: Quantifying Model Fairness

We use NPAQ to estimate how often a given neural net treats similar inputs, i.e., inputs differing in the value of a single feature, differently. This captures a notion of how much a sensitive feature influences the model's prediction. We quantify fairness for 4 BNNs, one for each architecture ARCH_1-ARCH_4, trained on the UCI Adult (Income Census) dataset [238]. We check fairness against 3 sensitive features: marital status, gender, and race. We encode 3 queries for each model using Property P3—

Table 2.7. NPAQ estimates of bias in BNNs $\text{ARCH}_{1..4}$ trained on the UCI Adult dataset. For changes in values of the sensitive features (marital status, gender and race), we compute, `PS(bias)`, the percentage of individuals classified as having the same annual income (=), greater than (>) and less than (<) when all the other features are kept the same.

Arch	Married → Divorced			Female → Male			White → Black		
	=	>	<	=	>	<	=	>	<
ARCH_1	89.22	0.00	10.78	89.17	9.13	2.07	84.87	5.57	9.16
ARCH_2	76.59	4.09	20.07	74.94	18.69	6.58	79.82	14.34	8.63
ARCH_3	72.50	4.37	21.93	80.04	9.34	12.11	78.23	6.24	18.58
ARCH_4	81.79	3.81	13.75	83.86	5.84	10.19	82.21	5.84	10.35

P5 (Section 2.3). Specifically, for how many people with exactly the same features, except one's marital status is "Divorced" while the other is "Married", would result in different income predictions? We form similar queries for gender ("Female" vs. "Male") and race ("White" vs. "Black") [4].

Effect of Sensitive Features. 4 models, 3 queries, and 3 different sensitive features give 36 formulae. Table 2.7 reports the percentage of counts generated by NPAQ. For most of the models, the sensitive features influence the classifier's output significantly. Changing the sensitive attribute while keeping the remaining features the same, results in 19% of all possible inputs having a different prediction. Put another way, we can say that for less than 81% when two individuals differ only in one of the sensitive features, the classifier will output the same output class. This means most of our models have a "fairness score" of less than 81%.

Quantifying Direction of Bias. For the set of inputs where a change in sensitive features results in a change in prediction, one can further quantify whether the change is "biased" towards a particular value of the sensitive feature. For instance, using NPAQ, we find that across all our models consistently, a change from "Married" to "Divorced" results in a change in predicted income from *LOW* to *HIGH*. [5] For ARCH_1, an individual with gender "Male" would more likely (9.13%) to be predicted to have a higher income than "Female" (2.07%) when all the other features are the same. However, for ARCH_4, a change from "Female" to "Male" would more likely result in a *HIGH* to *LOW* change in the classifier's output (10.19%). Similarly, for

[4] We use the category and feature names verbatim as in the dataset. They do not reflect the authors' views.

[5] An income prediction of below $\$50,000$ is classified as *LOW*.

the race feature, different models exhibit a different bias "direction". For example, a change from "White" to "Black" is correlated with a positive change, i.e., from *LOW* income to *HIGH* income, for ARCH_2. The other 3 models, ARCH_1, ARCH_2, and ARCH_4 will predict that an individual with the same features except for the sensitive feature would likely have a *LOW* income if the race attribute is set to be "Black".

With NPAQ, we can distinguish how much the models treat individuals unfairly with respect to a sensitive feature. One can encode other fairness properties, such as defining a metric of similarity between individuals where non-sensitive features are within a distance, similar to individual fairness [46]. NPAQ can be helpful for such types of fairness formulations.

2.8 Related Work

We summarize the closely related work to NPAQ.

Non-quantitative Neural Network Verification. Our work is on quantitatively verifying neural networks, and NPAQ counts the number of discrete values that satisfy a property. We differ in our goals from many non-quantitative analyses that calculate continuous domain ranges or single witnesses of satisfying values. Pulina and Tacchella [178], who first studied the problem of verifying neural network safety, implement an abstraction-refinement algorithm that allows generating spurious examples and adding them back to the training set. Reluplex [111], an SMT solver with a theory of real arithmetic, verifies properties of feed-forward networks with ReLU activation functions. Huang et al. [93] leverage SMT by discretizing an infinite region around an input to a set of points and then prove that there is no inconsistency in the neural net outputs. Ehlers [49] scope the work to verifying the correctness and robustness properties on piece-wise activation functions, i.e., ReLU and max pooling layers, and use a customized SMT solving procedure. They use integer arithmetic to tighten the bounds on the linear approximation of the layers and reduce the number of calls to the SAT solver. Wang et al. [243] extend the use of integer arithmetic to reason about neural networks with piece-wise linear activations. Narodytska et al. [156] propose an encoding of binarized neural networks as CNF formulas and verifies robustness properties and equivalence using SAT solving

techniques. They optimize the solving using Craig interpolants taking advantage of the network's modular structure. AI2 [67], DeepZ [211], DeepPoly [212] use abstract interpretation to verify the robustness of neural networks with piece-wise linear activations. They over-approximate each layer using an abstract domain, i.e., a set of logical constraints capturing certain shapes (e.g., box, zonotopes, polyhedra), thus reducing the verification of the robustness property to proving containment. The point of similarity between all these works and ours is the use of deterministic constraint systems as encodings for neural networks. However, our notion of equicardinality encodings applies to only specific constructions and is the key to preserving model counts.

Non-quantitative verification as Optimization. Several works have posed the problem of certifying robustness of neural networks as a convex optimization problem. Ruan, Huang, & Kwiatkowska [191] reduce the robustness verification of a neural network to the generic reachability problem and then solve it as a convex optimization problem. Their work provides provable guarantees of upper and lower bounds, which converges to the ground truth in the limit. Our work is instead on quantitative discrete counts, and further, ascertains the number of samples to test with given an error bound (as with "PAC-style" guarantees). Raghunathan, Steinhardt, & Percy [184] verify the robustness of one-hidden layer networks by incorporating the robustness property in the optimization function. They compute an upper bound which is the certificate of robustness against all attacks and inputs, including adversarial inputs, within l_{inf} ball of radius ϵ. Similarly, Wong and Kolter [248] train networks with linear piecewise activation functions that are certifiably robust. Dvijotham et al. [45] address the problem of formally verifying neural networks as an optimization problem and obtain provable bounds on the tightness guarantees using a dual approach.

Quantitative Verification of Programs. Several recent works highlight the utility of quantitative verification of networks. They target the general paradigm of probabilistic programming and decision-making programs [4, 91]. FairSquare [4] proposes a probabilistic analysis for fairness properties based on weighted volume computation over formulas defining real closed fields. While FairSquare is more expressive and can be applied to potentially any model programmable in the prob-

abilistic language, it does *not* guarantee a result computed in finite time will be within a desired error bound (only that it would converge in the limit). Webb et al. [245] use a statistical approach for quantitative verification but without provable error bounds for computed results as in NPAQ. Concurrent work by Narodytska et al. [157] uses model counting to assess the quality of machine learning explanations for binarized neural networks. In our work, we show a more general equicardinality framework for quantitatively verifying properties of binarized neural networks and instantiate 3 of these applications.

CNF Model Counting. In his seminal paper, Valiant showed that #CNF is #P-complete, where #P is the set of counting problems associated with NP decision problems [240]. Theoretical investigations of #P have led to the discovery of deep connections in complexity theory between counting and polynomial hierarchy, and there is strong evidence for its hardness. In particular, Toda showed that every problem in the polynomial hierarchy could be solved by just one invocation of #P oracle; more formally, $PH \subseteq P^{\#P}$ [229].

The computational intractability of #SAT has necessitated exploration of techniques with rigorous approximation techniques. A significant breakthrough was achieved by Stockmeyer who showed that one could compute approximation with (ε, δ) guarantees given access to an NP oracle [221]. The key algorithmic idea relied on the usage of hash functions but the algorithmic approach was computationally prohibitive at the time and as such did not lead to development of practical tools until early 2000s [150]. Motivated by the success of SAT solvers, in particular development of solvers capable of handling CNF and XOR constraints, there has been a surge of interest in the design of hashing-based techniques for approximate model counting for the past decade [77, 27, 50, 29, 218].

2.9 Summary

We present a new algorithmic framework for approximate quantitative verification of neural networks with formal PAC-style soundness. The framework defines a notion of equicardinality encodings of neural networks into CNF formulae. Such encodings preserve counts and ensure composibility under logical conjunctions. We instantiate this framework for binarized neural networks, building a prototype tool called NPAQ.

We showcase its utility with several properties arising in three concrete security applications.

2.10 Future Work

The principled framework of quantitative verification abstracts the specification of the neural network (Definitions 1 and 2). The work done in this chapter takes the first step towards realizing such a framework by proposing an approach that specializes to binarized neural networks. Thus, one of the natural questions that remain is how can we extend the approach to more general neural networks? Incremental improvements such as considering the weights to be finite-precision fixed-point integers (e.g., 4-bit weights) could be amenable to bit-blasting techniques to obtain CNF encodings similar to the ones presented in this chapter. However, once we consider continuous input spaces, real weights, and more general nonlinear activations such as the sigmoid or the hyperbolic tangent (tanh) activation function, it is not trivial to extend the approach taken in NPAQ. One issue is that the arithmetic represented as floating-point numbers requires care in ensuring the rounding errors do not lead to unsoundness in counting queries [107]. There have been a number of works that tackle these challenges by taking a probabilistic approach [54, 232]. For instance, one approach finds an upper bound on the probability of violating a desirable property by considering a confidence ellipsoid for Gaussian random variables in the input, and then computes an over-approximation of the confidence ellipsoid at the output [54]. This approach requires knowing the first and second moments of the input random variables to derive an upper bound on the probability. We also consider a probabilistic approach in Chapter 3 of this thesis, and discuss related works and key differences in the next chapter.

Another important future direction is increasing the scalability of the approach. The benchmarks considered in this chapter have scaled down the input sizes, as well as smaller sized neural networks (though the largest models are only slightly smaller than the LeNet5 [127] model of around $61,000$ model parameters). Extending the NPAQ framework to weighted counting queries (i.e., assuming the inputs are distributed according to a well-defined probability distribution over the input space) remains difficult due to the lack of support in the weighted model counter tools to handle

such benchmarks. A synergistic approach that considers the specific transformations in binarized neural networks (pseudo-boolean and XOR constraints) in designing the underlying counters is a promising future direction [250]. Nevertheless, advances in approximate model counting tools have considered the benchmarks derived in NPAQ, and since the publication of the work presented in this chapter, progress has been made [57].

Chapter 3

Verifiability via Bounded Counting Queries

3.1 Introduction

The quantitative techniques for estimating counting queries presented in Chapter 2 require white-box access and scale only to small deterministic DNNs of a restricted class, i.e., NPAQ requires white-box access to the models and specialized procedures to transform the deep neural networks (DNNs) to a specification, limiting their generality. Secondly, the performance of the underlying feasibility solver degrades severely with the usage of non-linear constraints, leading to analyses that do not scale to larger models. Thirdly, prior techniques are limited to deterministic neural networks, while extensive research effort has been invested in designing randomized DNNs, especially to enhance robustness [40, 130, 35]. An alternative approach is to check whether a property is satisfied "often enough" by a given DNN under a given input distribution. More specifically, one can assert that a DNN satisfies a property ψ with a desirably high probability $1 - \delta$.

In this chapter, we present an alternative formulation of counting queries, namely *bounded counting queries*: Given a logical property ψ specified over a space of inputs and outputs of a DNN and a numerical threshold θ, decide whether ψ is true for less than θ fraction of the inputs. We propose a quantitative verification algorithm for DNNs called PROVERO, tackling bounded counting queries. Unlike ad-hoc testing, the quantitative verification framework we propose aims to provide *soundness*, i.e., when it confirms that ψ is true with probability p, then the claim can be rigorously deduced from the laws of probability. For many practical applications, knowing that

the chance of failure is controllably small suffices for deployment. For instance, it has been suggested that road safety standards for self-driving cars specify sufficiently low failure rates of the perceptual sub-systems, against which implementations can be verified [118, 109, 225].

PROVERO is a procedure that achieves the above goal with proven PAC-style soundness: When it halts with a "Yes" or "No" decision, it is correct with probability $1 - \delta$ and within approximation error η to the given θ. The verifier can control the desired parameters (η, δ), making them arbitrarily close to zero. That is, the verifier can have controllably high certainty about the verifier's output, and θ can be arbitrarily precise (or close to the ground truth). The lower the choice of (η, δ) used by the verifier, the higher is the running time.

PROVERO is based on sampling, and it makes only one assumption—the ability to take independent samples from the space over which ψ is defined[1]. This makes the verification procedure considerably general and stand-alone. The verifier only needs black-box access to the DNN, freeing it up from assuming anything about the internal implementation of the DNNs. The DNN can be deterministic or from a general family of non-deterministic DNNs. This allows checking probabilistic properties of deterministic DNNs and of randomization procedures defined over DNNs.

Our work makes the following contributions:

- We present a new quantitative verification algorithm for neural networks. The framework is fairly general: It assumes only black-box access to the model being verified, and assumes only the ability to sample from the input distribution over which the property is asserted.

- We implement our approach in a tool called PROVERO that embodies sound algorithms and scales quantitative verification for adversarial robustness to large real-world DNNs, both deterministic and randomized, within 1 hour.

- In the context of certifying adversarial robustness, our empirical evaluation presents a new attack-agnostic measure of robustness and shows that PROVERO can produce certificates with high confidence on instances where

[1]For non-deterministic DNNs, the procedure assumes that the randomization used for the DNN is independent of its specific input.

existing state-of-the-art qualitative verification does not provide conclusive results.

3.2 Application: Adversarial Robustness

For concreteness, we apply our approach to verifying the robustness of neural networks. In proving robustness, the analyst has to provide a space of inputs over which the robustness holds. Often, this space is defined by all inputs that are within a *perturbation bound* ϵ of a given input $\mathbf{x}$ in the L_p norm [78]. Different distance norms have been used such as L_0, L_1, L_2 and L_∞. The L_p norm is defined as $\|\mathbf{x} - \mathbf{x}'\|_p = (|x_1' - x_1|^p + |x_2' - x_2|^p + \ldots + |x_n' - x_n|^p)^{1/p}$. A neural network f is defined to be robust with respect to a given input $\mathbf{x}$ if $\forall \mathbf{x}'$ such that $\|\mathbf{x} - \mathbf{x}'\|_p < \epsilon$, we have $f(\mathbf{x}) = f(\mathbf{x}')$.

For a given neural network f and input point $\mathbf{x}$, there always exists some perturbation size beyond which there are one or more adversarial samples. We refer to this minimum perturbation with non-zero adversarial examples as ϵ_{min}, which is the ground truth the security analyst wants to know. Attack procedures are best-effort methods which find upper bounds for ϵ_{min} but cannot provably show that these bounds are tight [231, 22, 9]. Verification procedures aim to prove the absence of adversarial examples below a given bound, i.e., they can establish lower bounds for ϵ_{min}. We call such verified lower bounds ϵ_{verf}. Most verifiers proposed to date for robustness checking are qualitative, i.e., given a perturbation size ϵ_{verf}, they output whether adversarial examples are absent within ϵ_{verf}. If the verification procedure is sound and outputs "Yes", then it is guaranteed that there are no adversarial examples within ϵ_{verf}, i.e., the robustness property is satisfied. When the verifier says "No", if the verifier is complete, then it is guaranteed that there are indeed adversarial examples within ϵ_{verf}. If the verifier is incomplete and prints "No", the result is inconclusive.

Let us introduce a simple quantitative measure of robustness called the *adversarial density*. Adversarial density is the fraction of inputs around a given input $\mathbf{x}$ which are adversarial examples. We explain why adversarial density is a practically useful quantity and much easier to compute for large DNNs than ϵ_{min}. We can compute perturbation bounds below which the adversarial density is non-zero but negligibly

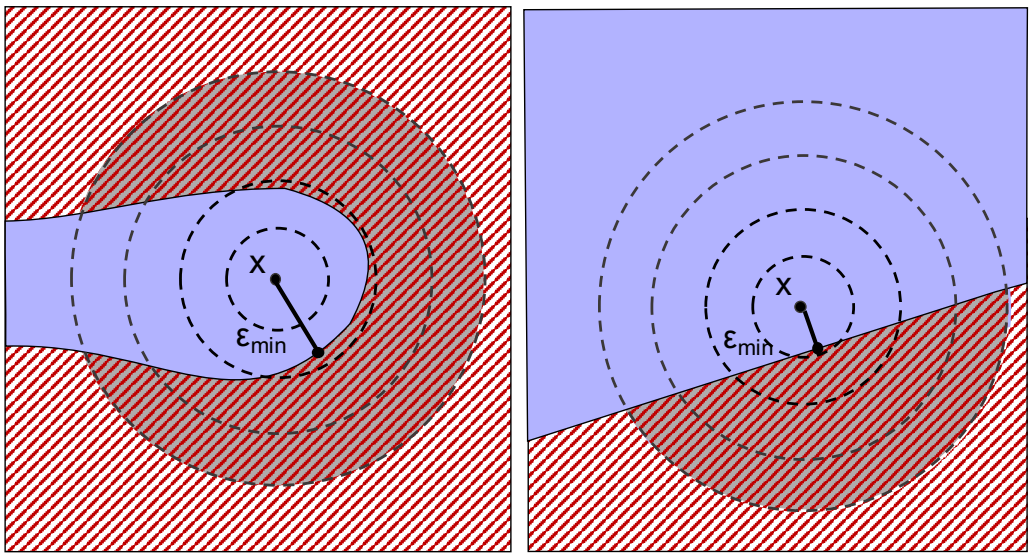


Figure 3.1. The decision boundaries of two binary classifiers f_1 (left) and f_2 (right) around an input $\mathbf{x}$ are shown. The correct classification region for $\mathbf{x}$ is shown in purple (solid), while the incorrect classification region is shown in red (hashed). The concentric circles show the equidistant points from $\mathbf{x}$ in L_2-norm drawn up to a bound ϵ. The classifier f_1 has a better (larger) minimum perturbation than f_2, but has a worse (larger) adversarial density because a majority of points within ϵ distance of $\mathbf{x}$ are in the incorrect classification region. Therefore, the smoothened version of f_2 will classify $\mathbf{x}$ correctly, while the smoothened f_1 will not. Picking the base classifier with the better adversarial density, rather than minimum perturbation, will lead to better accuracy in a smoothening defense.

small, and we empirically show these bounds are highly correlated with estimates of ϵ_{min} obtained by state-of-the-art attack methods.

3.2.1 Minimum Perturbation vs. Density

It is reasonable to ask why adversarial density is relevant at all for security analysis. After all, the adversary would exploit the weakest link, so the minimum perturbation size ϵ_{min} is perhaps the only quantity of interest. We present concrete instances where adversarial density is relevant.

First, we point to randomized smoothing as a defense technique, which has provable guarantees of adversarial robustness [129, 40, 35, 248]. The defense uses a "smoothed" classifier g that averages the behavior of a given neural net f (called the base classifier) around a given input $\mathbf{x}$. More specifically, given a base classifier f, the procedure samples perturbations of $\mathbf{x}$ within ϵ from a specific noise distribution

and outputs the most likely class c such that $argmax_{c \in \mathcal{Y}} Pr[f(\mathbf{x} + \epsilon) = c]$. Notice that this procedure computes the probability of f returning class c—typically by counting how often f predicts class c over many samples—rather than considering the "worst-case" example around $\mathbf{x}$. Said another way, these approaches estimate the adversarial density for each output class under some input distribution. Therefore, when selecting between two base classifiers during training, we should pick the one with the smallest adversarial density for the correct class, irrespective of their minimum adversarial perturbation size.

To illustrate this point, in Figure 3.1 we show two DNNs f_1 and f_2, as potential candidates for the base classifier in a randomized smoothing procedure. Notice that f_1 has a better (larger) ϵ_{min} than f_2. However, more of the inputs within the ϵ-ball of the input $\mathbf{x}$ (inside the red hashed circle) are classified as the wrong label by f_1 in comparison to f_2. Thus, a smoothed classifier with f_1 as a base classifier would misclassify $\mathbf{x}$ where the smoothed classifier with base f_2 would classify correctly. This explains why we should choose the classifier with the smaller adversarial density rather than one based on the minimum perturbation because the smoothing process is not susceptible to worst-case examples by its very construction. This motivates why computing adversarial density is useful for adversarial robustness defenses.

Second, we point out that estimating minimal perturbation bounds has been a difficult problem. Attack procedures, which provide an upper bound for ϵ_{min}, are constantly evolving. This makes robustness evaluations attack-specific and a moving target. On the other hand, qualitative verification techniques can certify that the DNN has no adversarial examples below a certain perturbation, which is a lower bound on the adversarial perturbation [111, 228, 211, 45]. However, these analyses do not scale well with deep networks and can lead either to timeouts or inconclusive results for large real-world DNNs. Furthermore, they are *white-box*, requiring access to the model internals and work only for deterministic neural networks. We show in this work that verifying adversarial density bounds is easy to compute even for large DNNs. We describe procedures that require only black-box access, the ability to sample from desired distributions and hence are attack-agnostic.

In particular, we show an empirical attack-agnostic metric for estimating robustness of a given DNN and input $\mathbf{x}$ called *adversarial hardness*. It is highest perturbation bound for which the adversarial density is below a suitably low θ. We

can search empirically for the highest perturbation bound $\epsilon_{\texttt{Hard}}$, called the adversarial hardness, for which a sound quantitative certifier says "Yes" when queried with $(f, \mathbf{x}, \epsilon_{\texttt{Hard}}, \theta, \delta, \eta)$—implying that f has suitably low density of adversarial examples for perturbation bounds below $\epsilon_{\texttt{Hard}}$.

Adversarial hardness is a measure of the difficulty of finding an adversarial example by uniform sampling. Surprisingly, we find that this measure strongly correlates with perturbation bounds produced by prominent white-box attacks (see Section 5.6). Given this strong correlation, we can effectively use adversarial hardness as a proxy for perturbation sizes obtained from specific attacks, when comparing the relative robustness of two DNNs.

We caution readers that adversarial hardness is a quantitative measure and technically different from ϵ_{min}, the distance to the nearest adversarial example around $\mathbf{x}$. But both these measures provide complementary information about the concentration of adversarial examples near a perturbation size.

3.3 Problem Definition

We are given a neural network and a space of its inputs over which we want to assert a desirable property P of the outputs of the network. Our framework allows one to check whether P is true for some specified ratio θ of all possible values in the specified space of inputs. For instance, one can check whether most inputs, a sufficiently small number of inputs, or any other specified constant ratio of the inputs satisfies P. The specified ratio parameter θ is called a *threshold*.

Formally, let $\mathrm{P}(\mathcal{I}, f, \Phi) \in \{0, 1\}$ be a property function over a neural network f, a subset of all possible inputs to the neural network $\mathcal{I}$ and user-specified parameters Φ. We assume that we can efficiently draw samples from any probability distribution over $\mathcal{I}$ that the analyst desires. For a given distribution D over $\mathcal{I}$, let $p_D = \mathbf{E}_{x \sim D}[\mathrm{P}(x, f, \Phi)]$. p_D can be viewed as the probability that P evaluates to 1 for x sampled from $\mathcal{I}$ according to D. When clear from context, we omit D and simply use p to refer to p_D.

Ideally, one would like to design an algorithm that returns 'Yes' if $p \leq \theta$ and 'No' otherwise. Such exact quantification is intractable, so we are instead interested in an algorithm $\mathcal{A}$ that returns 'Yes' if $p \leq \theta$ and 'No' otherwise, with two controllable

approximation parameters (η, δ). The procedure should be theoretically *sound*, ensuring that when it produces 'Yes' or 'No' results, it is correct with probability at least $1 - \delta$ within an additive bound on its error η from the specified threshold θ. Specifically, we say that algorithm $\mathcal{A}$ is *sound* if:

$$Pr[\mathcal{A}(\mathrm{P}, \theta, \eta, \delta) \text{ returns "Yes"} \mid p \leq \theta] \geq 1 - \delta$$
$$Pr[\mathcal{A}(\mathrm{P}, \theta, \eta, \delta) \text{ returns "No"} \mid p > \theta + \eta] \geq 1 - \delta$$

The analyst has arbitrary control over the confidence δ about $\mathcal{A}$'s output correctness and the precision η around the threshold. These values can be made arbitrarily small approaching zero, increasing the runtime of $\mathcal{A}$. The soundness guarantee is useful—it rigorously estimates how many inputs in $\mathcal{I}$ satisfy P, serving as a quantitative metric of satisfiability.

The presented framework makes very few assumptions. It can work with any specification of $\mathcal{I}$, as long as there is an interface to be able to sample from it (as per any desired distribution) efficiently. The neural network f can be any deterministic function. In fact, it can be any "stateless" randomized procedure, i.e., the function evaluated on a particular input does not use outputs produced from prior inputs. This general class of neural networks includes, for instance, Bayesian neural networks [92] and many other ensembles of neural network architectures [12]. The framework permits specifying all properties of fairness, privacy, and robustness highlighted in recent prior work [157], for a much broader class of DNNs.

Our goal is to present sound and scalable algorithms for quantitative verification, specifically targeting empirical performance for quantitatively certifying robustness of DNNs. The framework assumes black-box access to f, which can be deterministic or non-deterministic. Our techniques can directly check qualitative certificates produced from randomized robustness-enhancing defenses, one example of which is the recent work called PixelDP [129] (see Section 3.8.3).

3.4 Approach Overview

3.4.1 Sampling

Given a property P over a sampleable input space $\mathcal{I}$ and a neural network f, our approach works by sampling N times independently from $\mathcal{I}$. We test f on each sample as input. Let X_i be a 0-1 random variable denoting the result of the trial with sample i, where $X_i = 1$ if the $\mathrm{P}(x, f, \Phi)$ is true and $X_i = 0$ otherwise. Let X be the random variable denoting the number of trials in $X_1, X_2, \ldots, X_N$ for which the property is true. Then, the standard Chernoff bounds (see [154]) given below form the main workhorse underlying our algorithms:

Lemma 6. *Given independent 0-1 random variables $X_1, \ldots, X_N$, let $X = \sum_{i=1}^{N} X_i$, $\mu = \frac{\mathbf{E}[X]}{N}$, and $\hat{p} = \frac{X}{N}$. For $0 < \eta < 1$:*

$$Pr[\hat{p} \geq \mu + \eta] \leq e^{\frac{-N\eta^2}{3\mu}}$$

$$Pr[\hat{p} \leq \mu - \eta] \leq e^{\frac{-N\eta^2}{2\mu}}$$

Note that the probability p we are interested in bounding in our quantitative verification framework is exactly μ in Lemma 6. More specifically, the probability depends on the neural network and distribution over the inputs, $p = \mathbf{E}_{x \sim D}[\mathrm{P}(x, f, \Phi)]$, where D is a distribution over $\mathcal{I}$. Using a framework based on sampling and Chernoff bounds admits considerable generality. The only assumption in applying the Lemma 6 is that all samples are independent. If the neural network does not compute information during one trial (or execution under one sample) and use it in another trial, as is the case for all neural networks we study, trials will be independent. For any deterministic neural network, all samples are drawn independently and identically distributed in $\mathcal{I}$, so Chernoff bounds are applicable. For randomized DNNs, we can think of the i^{th} trial as evaluating a potentially different neural network (sampled from some distribution of functions) on the given sample. Here, the output random variables may not be identically distributed due to the randomization used by the neural network itself. However, Lemma 6 can still be used even for non-identically distributed trials but independent.

We discuss an estimation-based strategy that applies Chernoff bounds in a straight-forward manner next. Such a solution has high sample complexity for

quantitative verification of adversarial robustness. We then propose hypothesis-based solutions which are *sound* and have much better empirical sample complexity for real-world DNNs. Our proposed algorithms still rely only on Chernoff-style bounds, but are carefully designed to internally vary parameters (on which Chernoff bounds are invoked) to reduce the number of samples needed to dispatch the asserted property.

3.4.2 An Estimation-based Solution

One way to quantitatively verify a property through sampling is to directly apply Chernoff bounds to the empirical estimate of the mean $\hat{p}$ in N trials. The solution is to take $N > \frac{12 \cdot ln\frac{1}{\delta}}{\eta^2}$ samples, and decide "Yes" if $\hat{p} \leq \theta + \frac{\eta}{2}$ and "No" if $\hat{p} > \theta + \frac{\eta}{2}$. This is a common estimation approach, for instance used in the prediction step in the certified defense mechanism PixelDP [129]. By Lemma 6, one can show that $\hat{p}$ is within $\pm\frac{\eta}{2}$ additive error of p with confidence higher than $1 - \delta$. Therefore, the procedure is sound since $Pr[p \notin [\hat{p} - \frac{\eta}{2}, \hat{p} + \frac{\eta}{2})] \leq \delta$ by Lemma 6, for all $0 \leq p \leq 1$.

In this solution, the number of samples increase quadratically with decreasing η. For example, if the $\theta = 0.1, \eta = 10^{-3}, \delta = 0.01$, directly applying Chernoff bounds will require over 55×10^6 samples. Even for an optimized architecture such as BranchyNet [224] that reports 70.9 *ms* on average inference time per sample for a ResNet (152 layers) the estimation approach would take more than 1083 GPU compute hours. This is a prohibitive cost. For randomized DNNs, which internally compute expectations, the runtime of the estimation baseline approach can be even larger. For example, the randomization used in PixelDP can have $3 - 42\times$ inference overhead compared to deterministic DNNs [129].

Our work provides new algorithms that utilize much fewer samples on average. In the example of BranchyNet mentioned above, if the true probability p is 0.3, our approach requires 4246 samples to return a "No" answer with confidence greater than 0.99. The main issue with the estimation algorithm is that it does not utilize the knowledge of the given θ in deciding the number of samples it needs.

Algorithm 1 MetaPROVERO (θ, η, δ)

```
1: while cond do
2:     pick θ1 < θ2 ≤ θ
3:     ans = Tester (θ1, θ2, δ)
4:     if ans == "Yes" then return "Yes"
5:     end if
6:     pick θ2 > θ1 ≥ θ + η
7:     ans = Tester (θ1, θ2, δ)
8:     if ans == "No" then return "No"
9:     end if
10: end while
11: return Tester (θ, θ + η, δ)
```

3.4.3 Our Approach

The number of samples needed for Chernoff bounds depend on how far is the true probability p that we are trying to bound from the given threshold θ. Intuitively, if p and θ are far apart in the interval $[0, 1]$, then a small number of samples are sufficient to conclude with high certainty that $p \leq \theta$ or $p > \theta + \eta$ (for small η). The estimation approach takes the same number of samples irrespective of how far p and θ are. Our algorithms terminate quickly by checking for "quick-to-test" hypothesis early, yielding the sample complexity comparable to the estimation only in the worst case.

We propose new algorithms, the key idea of which is to use cheaper (in sample complexity) hypothesis tests to decide "Yes" or "No" early. Given the threshold θ and the error η, the high-level idea is to propose alternative hypotheses on the left side of θ and on the right side of $\theta + \eta$. We choose the hypotheses and a sampling procedure such that if any of the hypotheses on the left side of θ are true, then we can return "Yes". Similarly, if any of the hypotheses on the right side of $\theta + \eta$ are true, then we can return "No". Thus, we can potentially return much faster when the true probability p is further from the threshold θ.

The overall meta-level structure of our algorithms is simple and follows Algorithm 1, called MetaPROVERO. In lines 2 and 6, we pick the alternative hypotheses on the left and right of the given threshold θ respectively. We sample a certain number of samples, estimate the ratio $\hat{p}$ (by invoking Tester in lines 3 and 7), and check if we can prove that conditions involving the alternative interme-

Algorithm 2 TESTER $(\theta_1, \theta_2, \delta)$

1: $N = \frac{(\sqrt{3\theta_1}+\sqrt{2\theta_2})^2}{(\theta_2-\theta_1)^2} ln\frac{1}{\delta}$
2: $\eta_1 = (\theta_2 - \theta_1)\left(1 + \sqrt{\frac{2}{3}\frac{\theta_2}{\theta_1}}\right)^{-1}$
3: $\eta_2 = \theta_2 - \theta_1 - \eta_1$
4: `sample` N times
5: $\hat{p} = \frac{1}{N}\sum_{i=1}^{N} X_i$ ▷ X_i - samples that satisfy the property
6: **if** $\hat{p} \leq \theta_1 + \eta_1$ **then return** "Yes"
7: **end if**
8: **if** $\hat{p} > \theta_2 - \eta_2$ **then return** "No"
9: **end if**
10: **return** None

diate thresholds (θ_1 and θ_2) are satisfied with the desired input parameters (η, δ) using Chernoff bounds. If the check succeeds, the algorithm can return "Yes" or "No"; otherwise, the process repeats until a condition which guarantees soundness is met.

The internal thresholds are picked so as to soundly *prove* or *refute* that p lies in certain ranges in $[0, 1]$. This is done by testing certain *intervals* p is (or is not) in. For instance, METAPROVERO tests a pair of hypotheses $p \leq \theta_1$ and $p > \theta_2$ simultaneously (line 3). Notice that for the intervals on the left side of θ ($\theta_1 < \theta_2 \leq \theta$, chosen in line 2 in Alg. 1), the call to TESTER can result in proving that $p \leq \theta_1$ with desired confidence δ and error tolerance η. In this case, since $\theta_1 < \theta$, we will have proven the original hypothesis $p \leq \theta$ and the algorithm can return "Yes" soundly. We call such intervals, which are to the left of θ, as *proving intervals*. Conversely, *refuting intervals* are on the right side of $\theta + \eta$. The choice of θ_1 and θ_2 on line 6 in Alg. 1 is such that they are larger than $\theta + \eta$. When we can prove that $p > \theta_2$, i.e., the TESTER call on line 7 returns "No", then we can soundly return "No" because $\theta_2 > \theta + \eta$ implies $p > \theta + \eta$.

In Fig. 3.2, we consider an example run of METAPROVERO given that the true probability $p = 0.3$ (highlighted in blue) and the input parameters are $\theta = 0.1, \eta = 10^{-3}, \delta = 0.01$. METAPROVERO picks the proving interval $(\theta_{1l}, \theta_{2l}) = (0.03, 0.06)$ on the left of θ and calls TESTER which returns "No". This means that the true probability is greater than θ_{1l} with high confidence. METAPROVERO then picks an interval $(\theta_{1r}, \theta_{2r}) = (0.15, 0.25)$ on the right-side of $\theta + \eta$. Here TESTER returns

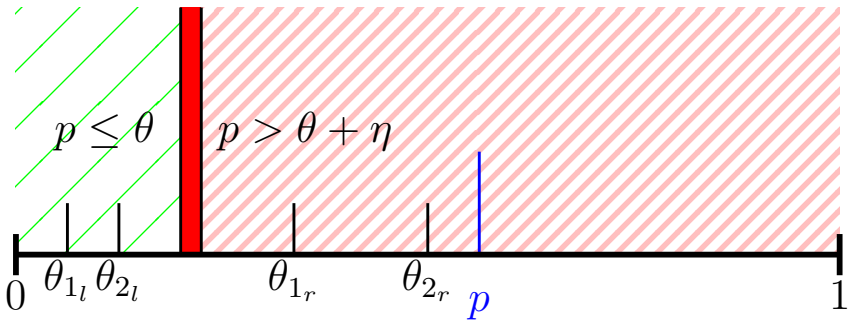


Figure 3.2. Example run of MetaPROVERO $(0.1, 10^{-3}, 0.01)$ given that $p = 0.3$. On the left-hand side, MetaPROVERO cannot prove that $p \leq \theta$ since p is greater than left-side θ_{2_l}. Next, MetaPROVERO chooses the refuting interval on the right-hand side. Since $p > \theta_{2_r}$, Tester returns "No" and MetaPROVERO can prove that $p > \theta + \eta$.

with confidence higher than $1 - \delta$ that $p > \theta_{2r}$. Since $\theta_{2r} > \theta + \eta$ we can conclude that the true probability is greater than θ with error η.

The key building block of this algorithm is the Tester sub-procedure (Algorithm 2), which employs sampling to check hypotheses. Informally, the Tester does the following: Given two intermediate thresholds, θ_1 and θ_2, if the true probability p is either smaller than θ_1 or greater than θ_2, it returns "Yes" or "No" respectively with high confidence. If $p \in (\theta_1, \theta_2)$ then the tester does not have any guarantees. Notice that a single invocation of the Tester checks *two hypotheses simultaneously*, using one set of samples. The number of samples needed are proven sufficient in Section 3.5.2.

One can directly invoke Tester with $\theta_1 = \theta$ and $\theta_2 = \theta + \eta$ but that might lead to a very large number of samples, $\mathcal{O}(1/\eta^2)$. Thus, the key challenge is to judiciously call the Tester on hypotheses with smaller sample complexity such that we can refute or prove faster in most cases. To this end, notice that MetaPROVERO leaves out two algorithmic design choices: stopping conditions and the strategy for choosing the proving and refuting hypotheses (highlighted in Alg. 1). We propose and analyze an adaptive *binary-search*-style algorithm where we change our hypotheses based on the outcomes of our sampling tests (Section 3.5.1). We show that our proposed algorithm using this strategy is sound. When p is extremely close to θ, these algorithms are no worse than estimating the probability, requiring roughly the same number of samples.

3.5 Algorithms

We provide an adaptive algorithm for quantitative certification that narrows the size of the intervals in the proof search, similar to a binary-search strategy (Section 3.5.1). This algorithm build on the base of the TESTER primitive which we explain in Section 3.5.2.

3.5.1 The BinPCertify Algorithm

We propose an algorithm BINPCERTIFY (Algorithm 3) where instead of fixing the intervals beforehand we narrow our search by halving the intervals. The user-specified input parameters for BINPCERTIFY are the threshold θ, the error bound η and the confidence parameter δ. The interval creation strategy is off-loaded to the CREATEINTERVAL procedure outlined in Algorithm 4. The interval size α is initially set to the largest possible as TESTER would require less samples on wider intervals (Algorithm 4, lines 2-6). Then, the BINPCERTIFY algorithm calls internally the procedure CREATEINTERVAL to create *proving* intervals (on the left side of θ, Alg. 3, line 6) and *refuting* intervals (on the right side of $\theta + \eta$, Alg. 3, line 12). Note that for the refuting intervals, we keep the left-side fixed, $\theta_1 = \theta + \eta$ and for the proving intervals we keep the right-side fixed, i.e., $\theta_2 = \theta$. For each iteration of BINPCERTIFY, the strategy we use is to halve the intervals by moving the outermost thresholds closer to θ (Alg. 4, lines 8-11). For these intermediate hypotheses, TESTER checks if it can prove or disprove the assert $p \leq \theta$ (lines 8 and 14). It continues to do so alternating the proving and refuting hypotheses until the size of both intervals becomes smaller than the error bound η (line 18). If only on one side of the threshold CREATEINTERVAL returns intervals with size $\alpha > \eta$, BINPCERTIFY checks those hypotheses. If the size of the proving and refuting intervals returned by CREATEINTERVAL is η, then the final check is directly invoked on $(\theta, \theta + \eta)$ and returned to the user (line 19).

The algorithm BINPCERTIFY returns "Yes" or "No" with soundness guarantees as defined in Section 3.3. We give our main theorem here and defer the proof to Section 5.3:

Theorem 4. *For an unknown value $p \in [0, 1]$, a given threshold $\theta \in [0, 1], \eta \in$*

$(0,1), \delta \in (0,1]$, BinPCertify *returns a "Yes" or "No" with the following guarantees:*

$$Pr[\text{BinPCertify } \textit{returns "Yes"} \mid p \leq \theta] \geq 1 - \delta$$
$$Pr[\text{BinPCertify } \textit{returns "No"} \mid p > \theta + \eta] \geq 1 - \delta$$

Algorithm 3 BinPCertify (θ, η, δ)

$\theta_{1_l} = \theta_{2_l} = 0$
$\theta_{1_r} = \theta_{2_r} = 0$
$n = 3 + max(0, log(\frac{\theta}{\eta})) + max(0, log(\frac{1-\theta-\eta}{\eta}))$
$\delta_{min} = \delta / n$ ▷ minimum confidence per call to Tester
while True **do**
    $\theta_{1_l}, \theta_{2_l}$ = CreateInterval$(\theta, \theta_{1_l}, \theta_{2_l}, \eta,$ True)
    if $\theta_{2_l} - \theta_{1_l} > \eta$ **then**
        ans = Tester$(\theta_{1_l}, \theta_{2_l}, \delta_{min})$
        if ans == "Yes" **then return** "Yes"
        end if
    end if
    $\theta_{1_r}, \theta_{2_r}$ = CreateInterval$(\theta, \theta_{1_r}, \theta_{2_r}, \eta,$ False)
    if $\theta_{2_r} - \theta_{1_r} > \eta$ **then**
        ans = Tester$(\theta_{1_r}, \theta_{2_r}, \delta_{min})$
        if ans == "No" **then return** "No"
        end if
    end if
    if $\theta_{2_r} - \theta_{1_r} \leq \eta$ and $\theta_{2_l} - \theta_{1_l} \leq \eta$ **then**
        return Tester$(\theta, \theta + \eta, \delta_{min})$
    end if
end while

Algorithm 4 CreateInterval $(\theta, \theta_1, \theta_2, \eta,$ left$)$

1: **if** $\theta == 0$ and left **then return** $(\theta, \theta + \eta)$
2: **end if**
3: **if** $\theta_1 == 0$ and $\theta_2 == 0$ and left **then**: **return** $(0, \theta)$
4: **end if**
5: **if** $\theta_1 == 0$ and $\theta_2 == 0$ and ¬left **then**: **return** $(\theta + \eta, 1)$
6: **end if**
7: $\alpha = \theta_2 - \theta_1$
8: **if** left **then**
9: **return** (θ_2 - max$(\eta, \alpha/2)$, θ_2)
10: **end if**
11: **return** $(\theta_1, \theta_1$ + max$(\eta, \alpha/2))$

3.5.2 Tester Primitive

The tester takes as input two thresholds θ_1, θ_2 such that $\theta_1 < \theta_2$ and confidence parameter δ and returns "Yes" when $p \leq \theta_1$ with confidence higher than $1 - \delta$ and "No" when $p > \theta_2$ with confidence higher than $1 - \delta$. If $p \in (\theta_1, \theta_2]$ the Tester returns without guarantees.

The procedure for implementing the Tester is simple. Following the procedure outlined in Algorithm 2, we take $N = \frac{(\sqrt{3\theta_1}+\sqrt{2\theta_2})^2}{(\theta_2-\theta_1)^2} ln\frac{1}{\delta}$ number of independent samples and estimate the ratio of these 0-1 trials as $\hat{p}$. The Tester returns "Yes" if $\hat{p} \leq \theta_1 + \eta_1$ and "No" if $\hat{p} > \theta_2 - \eta_2$ where η_1 and η_2 are error parameters (lines 2 and 3). If $\theta_1 + \eta_1 < \hat{p} < \theta_2 - \eta_2$, our implementation returns "None". The following lemma establishes the soundness of the tester, and follows directly from applying Chernoff bounds on $\hat{p}$.

Lemma 7. *Given the thresholds (θ_1, θ_2) and confidence parameter δ,* Tester *has following soundness guarantees:*

$$Pr[\text{Tester } \textit{returns "Yes"} \mid p \leq \theta_1] \geq 1 - \delta$$
$$Pr[\text{Tester } \textit{returns "No"} \mid p > \theta_2] \geq 1 - \delta$$

Proof. The proof follows directly the Chernoff bounds. □

Using the estimated probability $\hat{p}$, Tester returns "Yes" if $\hat{p} \leq \theta_1 + \eta_1$ and "No" if $\hat{p} \geq \theta_2 - \eta_2$ with probability greater than $1 - \delta$. Otherwise, it returns "None".

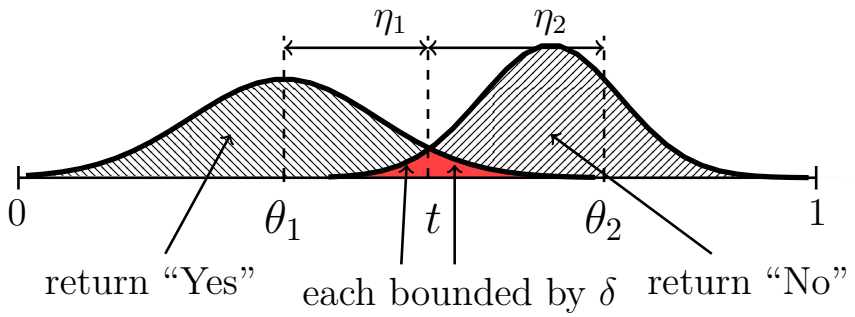


Figure 3.3. TESTER considers the boundary $t = \theta_1 + \eta_1 = \theta_2 - \eta_2$ that allows to distinguish $p \leq \theta_1$ or $p > \theta_2$.

We want to choose values $\eta_1, \eta_2 \in (0, 1)$ such that $Pr[\hat{p} \geq \theta_1 + \eta_1 \mid p \leq \theta_1] \leq \delta$ and $Pr[\hat{p} \leq \theta_2 - \eta_2 \mid p > \theta_2] \leq \delta$.

To derive the minimum number of samples for the estimated $\hat{p}$, the key idea is to use one set of samples to check two hypotheses, $p \leq \theta_1$ and $p > \theta_2$, simultaneously. To do so, we find a point $t \in (\theta_1, \theta_2)$ which serves as a decision boundary for the estimate probability $\hat{p}$. We illustrate this in Figure 3.3: it shows (t, η_1, η_2) and the two probability distributions for $\hat{p}$ given $p = \theta_1$ and $p = \theta_2$, respectively. The distributions for the case $p < \theta_1$ will be shifted further to the left, and the case $p > \theta_2$ will be shifted further to the right; so these are extremal distributions to consider. It can be seen that t is chosen such that $Pr[\hat{p} > t | p \leq \theta_1]$ as well as $Pr[\hat{p} < t | p > \theta_2]$ are bounded (shaded red) by probability δ. Using the additive Chernoff bounds (Lemma 6), for a given θ_1 and θ_2, the number of samples N is the maximum of $3\theta_1 \frac{1}{\eta_1^2} ln \frac{1}{\delta}$ and $2\theta_2 \frac{1}{\eta_2^2} ln \frac{1}{\delta}$.

Taking the maximum ensures that the probabilities of the both the hypotheses, $p < \theta_1$ and $p > \theta_2$, are being simultaneously upper bounded.

3.6 Soundness

In this section, we prove that our proposed algorithm satisfies soundness as defined in Section 3.3. BINPCERTIFY uses the TESTER primitive on certain intervals in sequence. Depending on the strategy, the size of the intervals and the order of testing them differs. But, the algorithm terminates immediately if the TESTER returns "Yes" on a proving interval or "No" on a refuting one. The meta-algorithm captures this structure on line 2-5 and BINPCERTIFY algorithm instantiates this general structure.

When none of these optimistic calls to TESTER are successful, the algorithm makes a call to the TESTER on the remaining interval in the worst case. Given the same basic structure of algorithms as per the meta-algorithm METAPROVERO, we now prove the following key theorem:

Lemma 8. *Let E be the event that the algorithm $\mathcal{A} \in \{\text{BINPCERTIFY}\}$ fails, then $Pr[E] \leq \delta$.*

Proof. Fix any input to $\mathcal{A}$, and consider the execution of $\mathcal{A}$ under those inputs. Without loss of generality, we can order the intervals tested by $\mathcal{A}$ in the sequence that $\mathcal{A}$ invokes the TESTER on them in that execution. Let the sequence of intervals be numbered from $1, \ldots, i$, for some value i.

Now, let us bound the probability of the event E_i, which is when $\mathcal{A}$ tests intervals $1, \ldots, i$ and fails. To do so, we consider events associated with each invocation of TESTER $j \in [1, i]$. Let R_j be the event that $\mathcal{A}$ returns immediately after invoking the TESTER on the j-th interval. Let C_j denote the event that TESTER returns a correct answer for the j-th interval. If E_i is true, then $\mathcal{A}$ terminates immediately after testing the i-th interval and fails. This event happens only if two conditions are met: First, $\mathcal{A}$ did not return immediately after testing intervals $1, \ldots, i-1$; and second, $\mathcal{A}$ returns a wrong answer at i-th interval and does terminate. Therefore, we can conclude that the event $E_i = \bar{C}_i \cap R_i \cap \bar{R}_{i-1} \cap \ldots \cap \bar{R}_1$. The probability of failure $Pr[E_i]$ for each event E_i is upper bounded by $Pr[\bar{C}_i]$.

Lastly, let E be the total failure probability of $\mathcal{A}$. We can now use a union bound over possible failure events $E_1, \ldots, E_n$, where n is the maximum number of intervals $\mathcal{A}$ can possibly test under any given input. Specifically:

$$Pr[E] = Pr[E_1 \cup E_2 \ldots \cup E_n] \leq \sum_{i=1}^{n} Pr[E_i] \leq \sum_{i=1}^{n} Pr[\bar{C}_i]$$

By analyzing $\sum_{i=1}^{n} Pr[\bar{C}_i]$ in the context of our specific algorithm BINPCERTIFY, we show that the quantity is bounded by δ (Lemma 9). □

Lemma 9. *Under any given input (θ, η, δ), let C_i be the event that i-th call made by* BINPCERTIFY *to the* TESTER *is correct and let n be the total number of calls to* TESTER *by* BINPCERTIFY. *Then, $\sum_{i=1}^{n} Pr[\bar{C}_i] \leq \delta$.*

Proof. We can upper bound the number of proving intervals on the left by $n_l \leq 1 + log\frac{\theta}{\eta}$. Similarly, for the right side of $\theta + \eta$, number of refuting intervals is $n_r \leq 1 + log\frac{1-\theta-\eta}{\eta}$. Lastly, there is only 1 call to the TESTER on the interval η (line 19, Alg. 3). The total number of intervals tested in any one execution of BINPCERTIFY is at most $n = 3 + log\frac{\theta}{\eta} + log\frac{1-\theta-\eta}{\eta}$. Each call to the TESTER is done with confidence parameter $\delta_{min} = \frac{\delta}{n}$, therefore by Lemma 7, the failure probability of any call is at most $\frac{\delta}{n}$. It follows that the $Pr[\bar{C}_i] \leq n \cdot \frac{\delta}{n} = \delta$

□

3.7 Sample Complexity Analysis

We now provide an upper bound on the number of samples required by BINPCERTIFY. Throughout the analysis, for an interval of size α, the number of samples needed by $i-$th call to the TESTER for the interval of size α is $s_i(\alpha, \delta) \leq \frac{(\sqrt{3}+\sqrt{2})^2}{\alpha^2} ln\frac{1}{\delta}$.

The BINPCERTIFY algorithm halves the interval size on each iteration (call to CREATEINTERVAL, lines 6 and 11 in Alg. 3) until the interval size for both the proving and refuting intervals become smaller than η. When that happens, BINPCERTIFY calls the TESTER on the interval $(\theta, \theta + \eta)$ and returns directly from there, lines 16–18, Alg. 3. The CREATEINTERVAL method starts by creating intervals of size θ for the proving intervals and $1 - \theta - \eta$ for the refuting intervals.

Theorem 5. *The number of samples required by* BINPCERTIFY *is upper bounded by* $\mathcal{O}(k_1 + k_2 + k_3)$*, where*

1. $k_1 = \frac{4}{3}(\frac{1}{\theta^2} - \frac{1}{\eta^2})(\sqrt{3} + \sqrt{2})^2 ln\frac{n}{\delta}$
2. $k_2 = \frac{4}{3}(\frac{1}{\theta^2} - \frac{1}{4(1-\theta-\eta)^2})(\sqrt{3} + \sqrt{2})^2 ln\frac{n}{\delta}$
3. $k_3 = \frac{(\sqrt{3\theta}+\sqrt{2(\theta+\eta)})^2}{\eta^2} ln\frac{n}{\delta}$.

and $n = 3 + max(0, log\frac{\theta}{\eta}) + max(0, log\frac{1-\theta-\eta}{\eta})$.

Proof. Let Z_i denote the number of samples required by the BINPCERTIFY during the i-th iteration of the loop. An iteration is generally introduced in the meta-algorithm in Algorithm 1, i.e., we select the proving and refuting hypotheses and

call the TESTER to try to disprove them. Note that BINPCERTIFY first invokes TESTER for the interval $(0, \theta)$, therefore, we have

$$Z_1 \leq s_1(\theta, \delta_{min}) + Pr[\bar{R}_1] s_2(\frac{\theta}{2}, \delta_{min})$$

For all the rest of the iterations until the final iteration, i.e., when the condition in line 18 is satisfied, we have the following bounds on Z_i.

$$\begin{aligned} Z_i \leq & s_{2i-1}(\frac{\theta}{2^{i-1}}, \delta_{min}) + \\ & s_{2i}(\frac{1-\theta-\eta}{2^i}, \delta_{min}) Pr[\bar{R}_{2i-1}] \end{aligned}$$

Let Z_f denote the number of samples by TESTER in line 19. We have $Z_f \leq s_n(\frac{\theta}{2^n}, \delta_{min})$. Finally, observe that

$$Z = Z_1 + Pr[\bar{R}_2] Z_2 + Pr[\bar{R}_{2(i-1)}] Z_3 + \cdots + Pr[\bar{R}_{n-1}] Z_f$$

Also, the number of proving intervals before the last call is $n_l \leq 1 + log\frac{\theta}{\eta}$ and, respectively, $n_r \leq 1 + log\frac{1-\theta-\eta}{\eta}$ refuting intervals. Therefore,

$$\begin{aligned} Z \leq s_1(\theta, \delta_{min}) + \sum_{i=2}^{n_l} s_{2i-1}(\frac{\theta}{2^{i-1}}, \delta_{min}) \prod_{j=1}^{2i-2} Pr[\bar{R}_j] + \\ \sum_{i=1}^{n_r} s_{2i}(\frac{1-\theta-\eta}{2^{i-1}}, \delta_{min}) \prod_{j=1}^{2i-1} Pr[\bar{R}_j] + \\ s_n(\frac{\theta}{2^n}, \delta_{min}) \prod_{j=1}^{n-1} Pr[\bar{R}_j], \end{aligned}$$

$$\text{where } n_l \leq 1 + log\frac{\theta}{\eta}, n_r \leq 1 + log\frac{1-\theta-\eta}{\eta}.$$

The following bounds can be obtained by simple algebraic computations:

1. $\sum_{i=2}^{n_l} s_{2i-1}(\frac{\theta}{2^{i-1}}, \delta_{min}) \leq \frac{4}{3}(\frac{1}{\theta^2} - \frac{1}{\eta^2})(\sqrt{3}+\sqrt{2})^2 ln\frac{1}{\delta_{min}}$
2. $\sum_{i=1}^{n_r} s_{2i}(\frac{1-\theta-\eta}{2^{i-1}}, \delta_{min}) \leq \frac{4}{3}(\frac{1}{\theta^2} - \frac{1}{4(1-\theta-\eta)^2})(\sqrt{3}+\sqrt{2})^2 ln\frac{1}{\delta_{min}}$
3. $s_n(\frac{\theta}{2^n}, \delta_{min}) = \frac{(\sqrt{3\theta}+\sqrt{2(\theta+\eta)})^2}{\eta^2} ln\frac{1}{\delta_{min}}$.

The statement of the theorem follows directly from the above bounds and by noting $Pr[\bar{R}_i] \leq 1$ and $\delta_{min} = \delta/n$ □

The above analysis is conservative and an interesting direction of future work would be to analyze $Pr[\bar{R}_i]$ under given distribution of p.

Table 3.1. Neural network architectures used in our evaluation.

Dataset	Arch	Description	#Hidden Units
MNIST(BM1)	FFNN	6-layer feed-forward	3010
	convSmall	2-layer convolutional	3604
	convMed	3-layer convolutional	4804
	convBig	6-layer convolutional	34688
	convSuper	6-layer convolutional	88500
	skip	residual	71650
ImageNet (BM2)	VGG16	16-layer convolutional	15,086,080
	VGG19	19-layer convolutional	16,390,656
	ResNet50	50-layer residual	36,968,684
	Inception_v3	42-layer convolutional	32,285,184
	DenseNet121	121-layer convolutional	49,775,084

3.8 Evaluation

- *Scalability:* For a given timeout, what is the largest model that PROVERO and existing qualitative analysis tools can produce conclusive results.
- *Utility in attack evaluations:* How does adversarial hardness computed with PROVERO compare with the efficacy of state-of-the-art attacks?
- *Applicability to randomized models:* Can PROVERO certify properties of randomized DNNs?
- *Performance.* How many samples are needed by our new algorithms vis-a-vis the estimation approach (Section 3.4.2)?

We implement our algorithms in a prototype tool called PROVERO and evaluate the robustness of 38 deterministic neural networks trained on 2 datasets: MNIST [128] and ImageNet [193]. For MNIST, we evaluate on 100 images from the model's respective test set. In the case of ImageNet, we pick from the validation set as we require the correct label. Table 3.1 provides the size statistics of these models. In addition, we evaluate the randomized model publicly provided by PixelDP [129], which has a qualitative certificate of robustness.

ERAN Benchmark (BM1) Our first benchmark consists of 33 moderate size neural networks trained on the MNIST dataset. These are selected to aid a comparison with a state-of-the-art qualitative verification framework called ERAN [211]. We

selected all the models which ERAN reported on. These models range from 2-layer neural networks to 6-layer neural networks with up to about $90K$ hidden units. We use the images used to evaluate the ERAN tool.

Larger Models (BM2) The second benchmark consists of 5 larger deep-learning models pretrained on ImageNet: VGG16, VGG19, ResNet50, InceptionV3 and DenseNet121. These models were obtained via the Keras framework with Tensorflow backend. These have about $15 - 50M$ hidden units.

All experiments were run on GPU (Tesla V100-SXM2-16GB) with a timeout of 600 seconds per image for the MNIST, 2000 seconds for ImageNet models, and 3600 seconds for the randomized PixelDP model.

3.8.1 Utility in Attack Evaluation

PROVERO can be used to directly certify if the security analyst has a threshold they want to check, for example, obtained from an external specification. Another way to understand its utility is by relating the quantitative bounds obtained from PROVERO with those reported by specific attacks. When comparing the relative robustness of DNNs to adversarial attacks, a common evaluation methodology today is to find the minimum adversarial perturbation with which state-of-the-art attacks can produce at least one successful adversarial example. If the best known attacks perform worse on one DNN versus another, on a sufficiently many test inputs, then the that DNN is considered more robust.

PROVERO offers a new capability: we can measure the ratio of adversarial samples within a specified perturbation bound of a given test input $\mathbf{x}$ (see Section 3.2). Specifically, we can compute the *adversarial density* by uniformly sampling in the L_p ball of $\mathbf{x}$, and checking if the ratio of adversarial samples is very small (below $\theta = 10^{-3}$). By repeating this process for different perturbations bounds, we empirically determine the *adversarial hardness* (or $\epsilon_{\mathtt{Hard}}$)—the largest perturbation bound below which PROVERO certifies the adversarial density to be that small (returns "Yes") and above which PROVERO does not (returns "No"). We use density threshold $\theta = 10^{-3}$, error tolerance $\eta = 10^{-3}$, and confidence parameter $\delta = 0.01$.

PROVERO computes the adversarial hardness with black-box access. As a

Table 3.2. Attack correlation for the PGD and C&W attack for the models in BM1 and BM2 using Pearson's coefficient (ρ). For all, statistical significance $p < 0.01$.

Models	ρ **(PGD)**	ρ **(C&W)**
convBigRELU___DiffAI	0.9617	0.7509
convMedGRELU___PGDK_w_0.1	0.8143	0.6686
convMedGRELU___PGDK_w_0.3	0.7699	0.6715
convMedGRELU___Point	0.8461	0.982
convMedGSIGMOID___PGDK_w_0.1	0.8533	0.8903
convMedGSIGMOID___PGDK_w_0.3	0.9394	0.913
convMedGSIGMOID___Point	0.9424	0.9605
convMedGTANH___PGDK_w_0.1	0.9521	0.9334
convMedGTANH___PGDK_w_0.3	0.9567	0.8718
convMedGTANH___Point	0.7592	0.9817
convSmallRELU___DiffAI	0.9504	0.5127
convSmallRELU___PGDK	0.7803	0.6411
convSmallRELU___Point	0.893	0.9816
convSuperRELU___DiffAI	0.687	0.3856
DenseNet-res	0.7168	0.4879
ffnnRELU___PGDK_w_0.1_6_500	0.8932	0.9577
ffnnRELU___PGDK_w_0.3_6_500	0.7039	0.6496
ffnnRELU___Point_6_500	0.954	0.9788
ffnnSIGMOID___PGDK_w_0.1_6_500	0.8706	0.8955
ffnnSIGMOID___PGDK_w_0.3_6_500	0.9402	0.9201
ffnnSIGMOID___Point_6_500	0.8906	0.9489
ffnnTANH___PGDK_w_0.1_6_500	0.8156	0.9508
ffnnTANH___PGDK_w_0.3_6_500	0.9104	0.9485
ffnnTANH___Point_6_500	0.8998	0.8435
mnist_conv_maxpool	0.9664	0.9699
mnist_relu_3_100	0.9668	0.9448
mnist_relu_3_50	0.9702	0.9298
mnist_relu_4_1024	0.8945	0.9723
mnist_relu_5_100	0.7472	0.9629
mnist_relu_6_100	0.9845	0.9868
mnist_relu_6_200	0.9699	0.9412
mnist_relu_9_100	0.979	0.9725
mnist_relu_9_200	0.8165	0.9652
ResNet50	0.7929	0.6932
skip___DiffAI	0.7344	0.6298
VGG16	0.8064	0.8297
VGG19	0.7224	0.7335
Inception-v3	0.5806	0.4866

comparison point, we evaluate against two *white-box* attacks — PGD [145] for L_∞ and C&W [24] for L_2 implemented in CleverHans [164] (v3.0.1) — 2 prominent attacks that are recommended for the L_p norms we consider [19]. White-box attacks enable the attacker complete access to internals; therefore, they are more powerful than black-box attacks today. Both PGD and C&W are gradient-based adversarial attacks. For PGD, we perform 30 attacks on different values of ϵ to identify the minimum value that an adversarial input can be identified. For C&W, we identify the best ϵ it is able to identify for a given amount of resource (iterations).

Our main empirical result in this experiment is that $\epsilon_{\texttt{Hard}}$ is *strongly correlated* with ϵ_{min}. Figure 3.4 and Figure 3.5 show the correlation visually for two models: it shows that the perturbation bounds found by these two separate attacks are different, but both correlate with the adversarial hardness of the certification instance produced by PROVERO. The Pearson correlation for all models is reported in Table 3.2 for all cases where the compared white-box attacks are successful The average Pearson correlation between the perturbation found by PGD, ϵ_{PGD}, and $\epsilon_{\texttt{Hard}}$ over all models is 0.858 and between the perturbation found by C&W, $\epsilon_{C\&W}$, and $\epsilon_{\texttt{Hard}}$ is 0.8438. We take 25 images per model to calculate the correlation. The significance level is high (p-value is below 0.01 for all cases).

Recall that PROVERO is sound, so the estimate of adversarial hardness $\epsilon_{\texttt{Hard}}$ is close to the ground truth with high probability (99%). This metric is an attack-agnostic metric, computed by uniform sampling and *without* white-box access to the model. The strong correlation shows that PGD and C&W attacks find smaller ϵ_{min} for easier certification instances, and larger ϵ_{min} for harder instances. This suggests that when comparing the robustness of two models, one can consider adversarial hardness as a useful *attack-agnostic* metric, complementing evaluation on specific attacks.

3.8.2 Scalability

We test PROVERO on 38 models, which range from $3K-50M$ hidden units. We select 100 input images for each model and retain all those inputs for which the model correctly classifies. We tested 11 perturbation size (L_∞) from 0.01 to 0.25 for BM1 and 4 perturbation size (L_2) from 2/255 to 14/255 for BM2. This results in a total

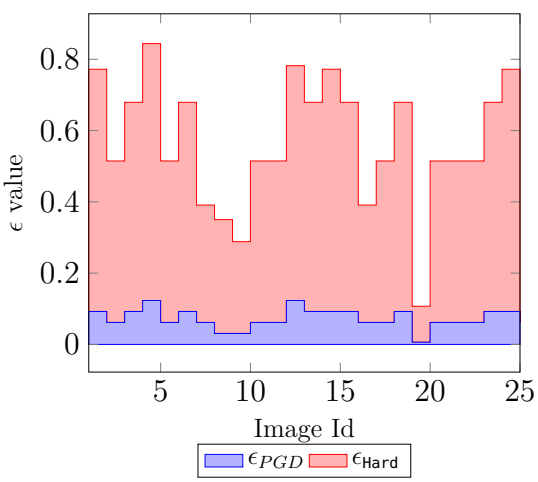


Figure 3.4. Correlation graph between L_∞ bounds provided by PROVERO and PGD for a fully connected feedforward with `sigmoid` (FFNN) on MNIST.

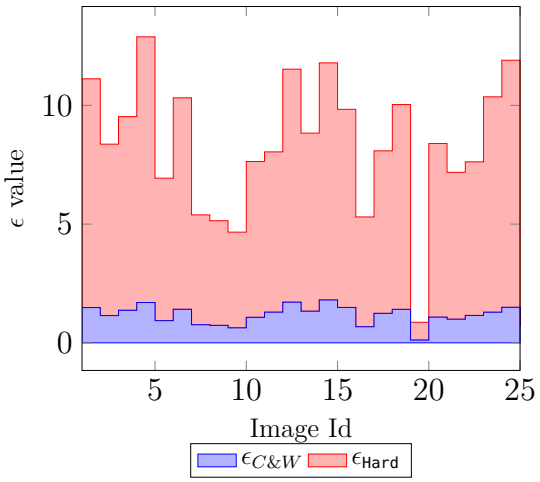


Figure 3.5. Correlation graph between L_2 bounds provided by PROVERO and C&W for a fully connected feedforward (FFNN) with `sigmoid` on MNIST.

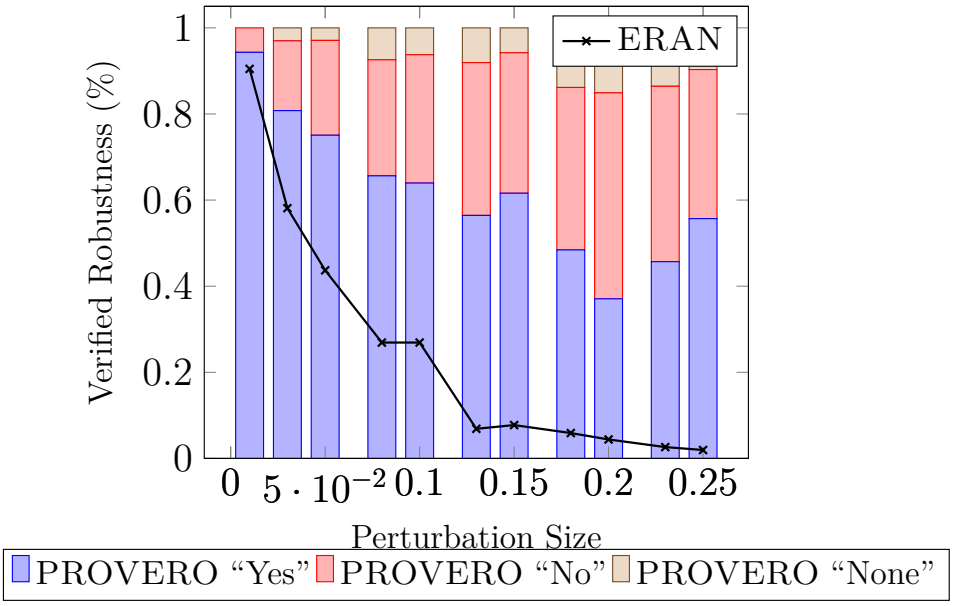


Figure 3.6. PROVERO and ERAN verified robustness per perturbation size for BM1 for threshold $\theta = 0.0001$, error $\eta = 0.001$ and confidence $\delta = 0.01$.

of 36971 test images for 38 models. We run each test image with PROVERO with the following parameters: ($\theta = 0.0001, \eta = 0.001$ and $\theta = 0.01, \eta = 0.01, \delta = 0.01$) (for BM1) ($\theta = 0.001, \eta = 0.001$ and $\theta = 0.01, \eta = 0.01, \delta = 0.01$) (for BM2). We find that PROVERO scales producing answers within the timeout of 600 seconds for BM1 and 2000 seconds for BM2 per test image. Less than 2% input cases for

BM2 and less than 5% for BM1 return "None", i.e., PROVERO cannot certify conclusively that there are less or more than than the queried thresholds. For all other cases, PROVERO provides a "Yes" or "No" results.

As a comparison point, while NPAQ provides direct estimates rather than the "Yes" or "No" answers PROVERO produces, NPAQ supports only a sub-class of neural networks (BNNs) and of much smaller size. Hence, it cannot support or scale for any of the models in our benchmarks, BM1 and BM2.

Secondly, we tested ERAN which is considered as the most scalable qualitative verification tool. ERAN initially was not able to parse these models. After direct correspondence with the authors of ERAN, the authors added support for requisite input model formats. After applying these changes, we confirmed that the current implementation of ERAN time out on all the BM2 models. PROVERO finishes on these within a timeout of 2000 seconds.

We successfully run ERAN on BM1, which are smaller benchmarks that ERAN reported on. There are total of 4 analyses in ERAN. We evaluate on the DeepZono and DeepPoly domains but for the DeepPoly, on our evaluation platform, ERAN runs out of memory and could not analyze the neural networks in BM1. The remaining analyses, RefineZono and RefinePoly are known to achieve or improve the precision of the DeepZono or DeepPoly domain at the cost of larger running time by calling a mixed-integer programming solver [213]. Hence, we compare with the most scalable of these 4 analyses, namely the DeepZono domain.

Figure 3.6 plots the precision of ERAN against PROVERO for all 11 perturbation sizes. We find that for a perturbation size of more than 0.05, ERAN's results are inconclusive, i.e., the analysis reports neither "Yes" nor "No" for more than 50% of the inputs, likely due to imprecision in over-approximations of the analysis. Figure 3.6 shows that the verified models reduces for higher perturbations ϵ. This is consistent with the findings in the ERAN paper: ERAN either takes longer or is more imprecise for non-robust models and higher ϵ values [213]. In all cases where ERAN is inconclusive, PROVERO successfully finishes within the 600 second timeout for all 35431 test images and values of ϵ. In above 95% of these cases, PROVERO produces high-confidence "Yes" or "No" results.

As a sanity check, on cases where ERAN conclusively outputs a "Yes", PROVERO also reports "Yes". With comparable running time, PROVERO is able to obtain

quantitative bounds for all perturbation sizes. From these experimental results, we conclude PROVERO is a complementary analysis tool that can be used to quantitatively certify larger models and for larger perturbation sizes, for which our evaluated qualitative verification framework (ERAN) is inconclusive. To the best of our knowledge, this is the first work to give any kind of sound quantitative guarantees for such large models.

3.8.3 Applicability to Randomized DNNs

So far in our evaluation we have focused on deterministic DNNs, however, PROVERO can certify the robustness of randomized DNNs. To demonstrate this generality, we take a model obtained by a training procedure called PixelDP that adds differentially private noise to make the neural network more robust. The inference phase of a PixelDP network uses randomization: instead of picking the label with the maximum probability, it samples from the noise layer and calculates an expectation of the output value. PixelDP also produces a certified perturbation bound for which it guarantees the model to be robust for a given input image. Note that qualitative verification tools such as ERAN require white-box access and work with deterministic models, so they would not be able to verify the robustness of randomized PixelDP DNNs.

We contacted the authors to obtain the models used in PixelDP [130]. The authors pointed us to the PixelDP ResNet28-10 model trained on CIFAR10 [2] as the main representative of the technique. We randomly select 25 images in the CIFAR10 dataset and for each image we obtain the certified perturbation bound $\epsilon_{pixeldp}$ produced by PixelDP itself. We configure PixelDP to internally estimate $\epsilon_{pixeldp}$ using 25 samples from the noise layer as recommended in their paper. This bound, $\epsilon_{PixelDP}$, is the maximum bound for which PixelDP claims there are no adversarial samples within the L_2 ball.

We use PROVERO to check the certificate $\epsilon_{PixelDP}$ produced by PixelDP, using the following parameters $\theta = 0.01, \eta = 0.01, \delta = 0.01$. PROVERO reports "Yes", implying that the model has adversarial density under these bounds. Under the same threshold $\theta = 10^{-2}, \eta = 10^{-2}$ we tested for larger perturbation sizes: from

[2] available to download from https://github.com/columbia/pixeldp

$\epsilon = 0.1$ to 0.25.

Our findings in this experiment are that PROVERO can certify low adversarial density for perturbation bounds much larger than the qualitative certificates produced by PixelDP. In particular, PROVERO certifies that for $\epsilon = 5 \times \epsilon_{PixelDP}$, the PixelDP model has less than 10^{-2} adversarial examples with confidence at least 99%. PROVERO offers complimentary quantitative estimates of robustness for PixelDP.

3.8.4 Performance

Our estimation solution outlined in Section 3.4.2 applies Chernoff bounds directly. For a given precision parameter η, it requires a large number of samples, within a factor of $1/\eta^2$. While we do not escape from this worst case, we show that our proposed algorithms improve over the estimation baseline empirically. To this end, we record the number of samples taken for each test image and compare it to the number of samples as computed for the estimation approach.

We find that PROVERO requires 27× less samples than the estimation approach for values of $\eta = 0.01, \theta = 0.01$ and 687× less samples for values of $\eta = 0.001, \theta = 0.0001$ for BM1. For larger models in BM2 and values of $\eta = 0.001, \theta = 0.001$, PROVERO requires 74× less samples than the estimation solution and $\eta = 0.01, \theta = 0.01$.

In our implementation, we use a batch-mode to do the inference for models in BM1 and BM2, speeding up the running time of the sampling process by a factor of 3×. This leads to average times of 29.78 seconds per image for BM1 for a batch size of 128. For the models in BM2 we used different batch sizes so we report the average time per sample as 0.003 seconds which can be used to derive the running time based. For $(\theta = 0.0001, \eta = 0.001)$ and all values of ϵ, the average number of samples over all images is $80,424$ (median value is $83,121$, standard deviation is $14,303$). For VGG16 the average number of samples is around $657,885$, for VGG19 it is around $644,737$ and for InceptionV3 $664,397$ samples, respectively, for $\theta = 0.001, \eta = 0.001$. For both $\theta = 0.01, \eta = 0.01$ and $\theta = 0.001, \eta = 0.001$, for VGG16 the average number of samples is $737,297$, for VGG19 it is $722,979$ and for InceptionV3 $744,792$ average number of samples, resulting in about 74× less samples than the estimation approach.

For the PixelDP model, PROVERO takes $20,753$ samples. Since PixelDP models internally take 25 samples from the noise layer, the time taken for inference on one given input itself is higher. PROVERO reports an average running time per test image of roughly $1,500$ seconds.

3.9 Related Work

Qualitative verification methodologies have sought to exploit the advances in combinatorial solving, thereby consisting of satisfiability modulo theories (SMT) solvers-based approaches [93, 111, 112, 49, 156] or integer linear programming approaches [228, 138, 31]. Despite singficiant progress over the years, the combinatorial solving-based approaches do not scale to deep neural networks.

Consequently, techniques to address scalability often sacrifice completeness: abstract interpretation-based techniques [67, 211, 213, 212] are among the most scalable verification procedures that over-approximate the activation functions with abstract domains. Similarly, optimization-based approaches [248, 184, 45] produce a certificate of robustness for a given input: a lower bound on the minimum perturbation bound ϵ_{min} such that there are no adversarial examples within that a ϵ_{min} ball. These techniques are, however, incomplete and catered to only ReLU activations [248] and fully-connected layers [259, 79].

A promising line of incomplete techniques has been proposed employing two complementary techniques: Lipschitz computation and linear approximations. Hein and Andriushchenko [89] propose an analytical analysis based on the Lipschitz constant, but the approach assumes that a differentiable activation function (thus excluding ReLU activations) and can handle only two layers. Boopathy et al [16], Weng et al. [246], Zhang et al [259] have further improved the scalability of these techniques but they are still limited to $80,000$ hidden units. While the lower and upper bounds are sound, their tightness is not guaranteed. In another line of work, Weng et al. [247] employ extreme value theory to estimate a lower bound on ϵ_{min} albeit without theoretical guarantees of the soundness of the obtained bounds. PROVERO differs fundamentally from classical verification approaches in our focus on the development of a quantitative verification framework with rigorous guarantees on computed estimates.

Webb et al. [245] proposed a Monte Carlo-based approach for rare events to estimate the ratio of adversarial examples. In this work, we take a *property testing* approach wherein we ask if the proportion of inputs that violate a property is less than a given threshold (in which case we say yes) or it is η-far from the threshold (in which case we say no). While both approaches are black-box and rely on sampling, PROVERO returns a yes or no answer with user-specified high confidence, whereas the statistical approach proposed by Webb et al. does not provide such guarantees. A related area is statistical model checking [124, 122, 123] that relies on sampling schemes [242] to accept a given hypothesis. PROVERO algorithm is similar to a sequential sampling plan [256] but PROVERO's insight is that each test checks "cheaper" hypotheses in a binary-search manner. In particular, in case of PROVERO, we design hypothesis in a manner that more *expensive* hypothesis tests are delayed as much as possible. In multi-armed bandit games, algorithms that identify best arm procedures with a fixed confidence budget bear some similarity to PROVERO. Such procedures must control for the number of samples to ensure that the best arm (with unknown mean payoff) is returned even when the second best arm (with unknown mean payoff) is arbitrarily close [102]. The derived sample complexity in such approaches is with respect to the gap between the best arm's expected mean and the second best. In our approach, the sample complexity is with respect to the fixed and apriori known thresholds, and so our sampling strategy is also different.

3.10 Summary

PROVERO introduces an algorithm for verifying quantitative properties of neural networks, assuming only black-box access, and with much better test sample complexity than compared baselines. Our algorithm offers, in particular, a powerful new attack-agnostic way of evaluating adversarial robustness for deep neural networks.

3.11 Future Work

One future work direction is how to lower the sample complexity which would effectively reduce the running time of approaches based on sampling such as PROVERO. We could consider a "grey" approach that is aware of part of the internal workings of the model (or an over-approximation), or through better analysis and sampling. Since the publication of the work presented in this chapter, a number of works improved the sample complexity, but there is a natural limitation when the estimated events are rare [227, 110].

Another future direction is how to extend sampling to other L_p norms than the one considered in this chapter (L_2 and L_∞). For instance, Wasserstein distance has been hypothesized to better capture perceptual similarity between images, and has led to improving a number of applications such as image retrieval [192], image generation [6, 182, 197, 197], image-to-image translation [100], or multi-label classification [62]. We can define a multidimensional probability distribution over images and consider only images similar to ones in a test set. Computing Wasserstein distance over multidimensional probability distributions is computationally expensive though there are improvements when considering projecting in smaller dimensions [115, 30]. Having access to an efficient sampler over perceptually similar images (inputs) would benefit the statistical verification of *semantic robustness*.

It is also natural to consider integrating into training techniques that add adversarial examples or perceptually close inputs into training (adversarial training) to improve robustness [174, 244, 80]. Despite progress in this space, there is still a gap between the robust accuracy (i.e., how many test samples are certifiably robust up to some perturbation in some norm) and the standard accuracy. What might be interesting for future research is to explore how to guide training via the more precise characterization of the decision boundary via the density of adversarial examples.

Chapter 4

Reasoning about Training with Causal Queries

4.1 Introduction

Privacy has become a key concern in machine learning [51] with several classes of attacks being discovered. Membership inference (MI) attacks, which have led to a flurry of works recently [144, 34, 98, 142, 258, 206, 142, 135, 216, 215, 85, 159, 158, 260, 140], capture the advantage of an adversary in distinguishing samples used for training from those that were not. There is clear empirical evidence that MI attacks are effective and many new attack variants are emerging. At the same time, there is currently no systematic framework to understand *why* standard training procedures leave deep nets susceptible to MI attacks.

There are two incumbent approaches to understanding why deep networks are susceptible to MI attacks [254, 194, 215, 142, 234]. The first tries to offer *fully mechanistic* explanations derived from theoretical analysis. For instance, a line of work tries to mathematically model the stochastic mechanism of training models (e.g., using stochastic gradient descent (SGD)) with enough precision [194]. The most commonly accepted mechanistic explanation is that ML models leak training data because they fail to generalize, measured through quantities like overfitting gap, accuracy gap, and so on [254, 194]. This approach is appealing and is actively progressing, but at the same time, modeling the training process with closed-form mathematical expressions is an inherently difficult problem. Predictions from these mechanistic explanations often do not agree with observations in experiments, because the approximations or assumptions made in the analysis may not hold in

practice [254]. Furthermore, generalization offers a one-way explanation: if models generalize almost perfectly in a particular sense, then MI attacks are expected to be ineffective on average. It does not say how well MI attacks will work for models that may have not generalized perfectly, which is often the case in practice. As a result, there have been many different hypothesized root causes that go beyond direct measures of classical generalization, such as model capacity and architecture. In short, no single coherent mechanistic explanation today predicts the average performance of existing MI attacks well.

A second approach to explaining MI attacks is based on *statistical testing* of hypotheses: Researchers intuit about the root cause, run experiments, and then report statistical correlations between the hypothesized cause and the performance of the MI attacks [207, 234, 142, 133, 215]. For example, several works have suggested that the empirically observed overfitting gap or the accuracy difference between training and testing sets explain MI attacks. This approach, while being important in its own right, fails to provide satisfactory explanations often as well. Guessing which root causes are really responsible for attacks is difficult; after all, the stochastic process of standard training procedures is complex and is affected by multiple possible sources of randomness, hyper-parameter selection, and sampling bias. Furthermore, correlation is not always causation. Confusing the two can result in overly simplistic explanations of the true phenomenon at hand and lead to paradoxes. Lastly, the purely empirical approach leaves no room to accommodate mechanistic axioms (things we know that ought to be true from our theoretical understanding)—if the observations do not correlate with mechanistically derived facts, then they remain unexplained.

Our approach. We propose a new approach that explains MI attacks through a *causal model*. A causal model is a graph where nodes are random variables that abstractly represent properties of the underlying stochastic process and edges denote cause-effect relationships between them. We can model the process of sampling data sets, picking hyper-parameters like the size of the neural network, output vectors, generalization parameters like bias and variance, and predictions from MI attack procedures as random variables. These random variables can be measured empirically during experiments. We can then both encode and infer causal relationships between

nodes quantitatively through equations. Edges in our causal model are of two types: 1) mechanistically derived edges denote known mathematical facts derived from domain knowledge (prior work, definitions, etc.); and 2) relations inferred from experimental observations using *causal discovery* techniques.

Our causal approach is substantially different from prior works and enables much deeper and principled analysis. The causal model, once learnt, acts like a predictive model—one can ask what will be the expected performance of a particular MI attack if the "root causes" (random variables in the model) were to have certain values not observed during prior experiments. Such estimation can be done without running expensive experiments. A causal model allows us to "single out" the effect of one variable on the MI attack performance. Such queries go beyond just observing statistical correlations because they need to reason about other variables that might affect both the cause and the outcome (the attack performance). To carefully solve these queries, we leverage the principled framework of causal reasoning known as *do-calculus*. It allows us to perform systematic refutation tests, which avoids confusing causation with correlation. Such tests quantitatively tell us how well the model fits the observed data and answer "what if" style of questions about surmised root causes. Further, we can compare causal models obtained for two different attacks to understand how their manifestation differs, or compare models with and without an intervention (e.g., by applying a defense) for a given attack to understand which root causes it neutralizes. Causal models offer a more principled and interactive way of examining MI attacks.

Resulting Findings. To showcase the utility of our approach, we study 6 well-known MI attacks and 2 defenses for deep neural networks trained using standard SGD training procedures. We analyze a list of intuitive "root causes" which have been suggested in prior works and formally specify them as 9 causal hypotheses. We analyze each of these 9 hypotheses for ML models with 2 types of loss functions, so we evaluate on 18 formalized hypotheses. Several salient findings have resulted from our causal analysis. First, we refute 7/18 previously hypothesized causes, highlighting the perils of understanding MI attacks purely from intuition or from statistical correlational analysis. Second, we find that different causes contribute differently to the average attack accuracy, dispelling the idea that a single explanation suffices for

all 6 MI attacks we study. Our causal approach also models the attacks well (0.90), i.e., predict the observed attack accuracy. This betters prior single-cause explanations by $3-22\%$ in 17/24. Third, we show that two stochastic parameters inherent in the training process, namely **Bias** and **Variance**, govern both generalization achieved by ML models [108, 251, 161] and MI attack accuracy. This offers a more nuanced lens to connect generalization and MI attack accuracy from that offered by prior works [254, 144, 20]. Fourth, we show that defenses against MI attacks based on regularization reduce the influence of some of root causes, but fail to completely remove their effect.

Summary of Contributions. We propose the first use of causal analysis for studying membership inference attacks on deep neural networks. We derive causal models for 6 MI attacks by combining both known domain-specific assumptions and observations made from experiments. Our key contribution is a new quantitative connection between MI attacks and generalization, which enables refuting claims about causation with finer accuracy.

Availability

Our prototype implementation is publicly available on GitHub[1].

4.2 Motivation

Many intuitive explanations for privacy leakage have been put forward in prior works. The most widely accepted claim is that "*overfitted classifiers are more susceptible to MI attacks*", which has been backed by experimental correlational analyses [207, 234, 133, 142, 144, 132, 254, 194]. To evaluate the level of overfitting though, two different metrics have been proposed: the difference in the loss of training and non-training samples [254, 194], as well as the train-to-test accuracy gap [207, 234, 133, 142, 144, 132]. However, empirical evidence to the contrary has also been observed–both MI attacks and extraction attacks have been reported on well-generalized models [20, 23, 144]. Other potential contributing factors, such as model complexity / structure [207, 215, 159], the size of the training set [207, 159], the diversity of the training samples [207], how close a target model to attack is to

[1]At https://github.com/teobaluta/etio.

the shadow model [196], and so on have been proposed, creating an unclear picture of why MI attack arise. We highlight 9 common hypotheses claimed in prior works below:

(H1) The overfitting gap is the cause of MI attacks that use multiple shadow models in the inference [207].

(H2) "The main idea behind our shadow training technique is that similar models trained on relatively similar data records using the same service behave in a similar way" [207].

(H3) Beyond overfitting, model complexity influences the membership inference attack accuracy [207].

(H4) The size of the training set is a contributing factor to the success of MI attacks [207].

(H5) The shadow model attack works even if the attack uses only one shadow model [196].

(H6) "If a model is more overfitted, then it is more vulnerable to membership inference attack." - for MI attacks that use a single shadow model [196].

(H7) There is no difference in the attack accuracy if we use the top-3 predictions in descending order vs. the whole prediction vectors [196].

(H8) Shadow model-based attacks transfer when there is a clear decision boundary between members and non-members [196].

(H9) The average generalization error explains the advantage of the threshold-based adversary (even when this attack assumes that the error is normally distributed) [254].

It is natural to ask: To what extent are these explanations correct? Do these hypothesized factors universally explain all MI attacks equally? What does achieving a certain level of generalization (eliminating overfitting) imply towards reducing MI attacks? Answering such queries requires a principled framework for reasoning even to phrase the right statistical quantities to measure—it is something that is easily prone to fallacious reasoning.

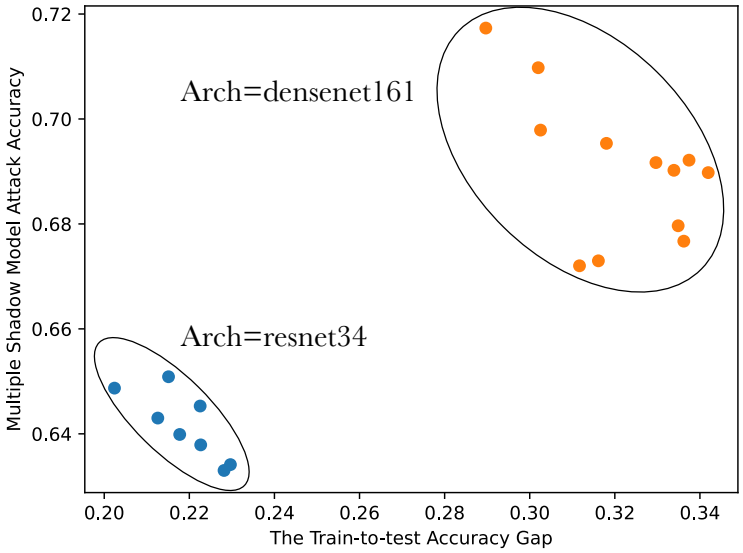


Figure 4.1. High train-to-test accuracy gap correlates with high attack accuracy in general, but if clustered on architecture type, an inverse relationship is visible.

4.2.1 Pitfalls of Testing with Correlations

Let us consider two of the prior work hypotheses: (H1) Higher difference between train and test accuracy leads to higher MI attacks; and (H3) An increase in model complexity increases privacy leakage, i.e., larger models are more susceptible. One of the most prominent approaches to validating such hypotheses today is experimental validation through statistical correlation analysis [234, 207, 144, 142]. The analysis proceeds by observing how the train-test accuracy gap and attack accuracy for a chosen MI attack changes under different choices of model complexity (number of model parameters). For concrete illustration, we run a small-scale experiment for the multiple shadow model attack [207]. We train 2 deep nets with varying number of parameters on CIFAR10 dataset. We average the observed training and testing accuracy of the deep neural networks under multiple samples of the training datasets. For each of these models, we also run the shadow model attack separately [207], using a disjoint part of the training dataset for the shadow model training.

Hidden Causes. In Fig. 4.1, on the X-axis we plot the average train-to-test accuracy gap against the attack accuracy on the Y-axis. The overall trend of the relationship between train-to-test accuracy gap and the attack accuracy is positive,

i.e., the larger the train-to-test accuracy gap, the higher the attack accuracy. But this trend is not seen in Fig. 4.1, when we group the trained neural nets by other criteria such as model complexity and architecture. We observe they are clumped into two clusters. Within those clusters, the observed trend is the *reverse*: the larger the accuracy gap, the lower the attack accuracy! This turns out to be a fallacious conclusion, because it fails to account for the effect of other factors such as model complexity or architecture indirectly on train-to-test accuracy gap or directly on the attack accuracy.

This paradox arises because there are *confounding factors* or *confounders* [172, 210] wherein different sub-populations of the data have contradictory statistical properties. Similar paradoxes can arise due to *selection bias* [172, 18, 13] or *collider bias* [15, 172]. Without resolving such issues, it is difficult to decide whether one should try reducing the train-to-test accuracy gap or model complexity.

Singling Out. When we want to decide which factors influence the end outcome more than the others, such as when designing practical defenses, one would like to "single out" the main causes and quantify how much they affect the expected MI attack accuracy. In our running example, the model complexity affects both variables, the train-to-test accuracy gap as well as the MI attack accuracy. It would be difficult to *quantify* how much it affects attack accuracy directly and how much indirectly via train-to-test accuracy gap without a more principled analysis of the observed data.

To understand the challenge, let us say we want to estimate how changing the train-to-test accuracy gap from $a = 0.007$ to $b = 0.914$ (which we observe in practice) affects the MI attack accuracy. Let us assume we train more NNs and that we now know that the model complexity is a confounding factor for both. A naive way to analyze this is to statistically estimate the following quantity: $E_1 = E[\texttt{MIAcc}|\texttt{AccDiff} \approx 0.9] - E[\texttt{MIAcc}|\texttt{AccDiff} \approx 0]$ where the `MIAcc` is the attack accuracy and the `AccDiff` is the train-to-test accuracy gap. Fig. 4.2 shows our new experimental observations and conditional probability estimates from data which reveals that the estimated expected effect is $E_1 = 0.47$. It is misleading to conclude that a change in train-to-test accuracy gap will have a large impact on the attack performance since we know that there is a confounder.

NumParams	AccDiff	MIAcc
12704	0.007	0.5
84542	0.104	0.564
203264	0.102	0.633
101632	0.102	0.632
9835146	0.109	0.567
	…	
9881316	0.914	0.880

$\mathbb{E}$[MIAcc | AccDiff=0.007] ≈ 0.50
$\mathbb{E}$[MIAcc | AccDiff=0.914] ≈ 0.97
⇒ Effect is E_1=0.47 ✗
⇒ Adjusted E_2=0.3 ✓

$\mathbb{E}$[MIAcc | NumParams=12704] ≈ 0.58
$\mathbb{E}$[MIAcc | NumParams=1334618] ≈ 0.94
⇒ Effect is E_3=0.36 ✓
⇒ Adjusted E_4=0.08 ✗

Figure 4.2. Reporting average conditional probabilities is not always correct. For example, if we are to estimate the effect of train-to-test accuracy gap on the MI attack accuracy from the data shown, the conditional over-estimates the effect by 0.17. For measuring the effect of model size though, the second estimate shown is correct since over-controlling for the `AccDiff` incorrectly decreases the effect to 0.08 from 0.36.

If we want to single out the effect of changes in train-to-test accuracy gap, the correct way is the following: Find samples with the same values for the model complexity but different values of train-to-test accuracy gap, from which we then compute the difference these produce on the attack accuracy. This is called analytically "controlling" for the confounding factor[2]. This corresponds to analytically computing how the system would behave under randomized values of model complexity. Such randomization "nullifies" or "smoothens out" the effect of model complexity. If we do this carefully, it turns out that the actual estimated effect E_2 when `AccDiff` ranges from a to b is expressed by the following quantity:

$$E_2 = \sum_z E[\texttt{MIAcc}|\texttt{AccDiff} = a, \texttt{NumParams} = z]\texttt{Pr}(\texttt{NumParams} = z)$$
$$- \sum_z E[\texttt{MIAcc}|\texttt{AccDiff} = b, \texttt{NumParams} = z]\texttt{Pr}(\texttt{NumParams} = z)$$

This leads to an estimated effect of $E_2 = 0.3$, as per our data (Section 5.6)—significantly lower than the naive analysis above.

Avoiding Over-controlling. It may be tempting to control for all factors that may influence the outcome. But arbitrarily controlling for variables leads to fallacious reasoning as well. For example, if we want to estimate the effect of the model complexity on the attack accuracy, should we now control for the train-to-test

[2]Controlling for a variable means binning data according to measured values of the variable

accuracy gap? If we were to control for the train-to-test accuracy gap, then the estimated effect E_4 when `NumParams` varies from a' to b' is given by (also shown in Fig. 4.2):

$$E_4 = \sum_z E[\texttt{MIAcc}|\texttt{NumParams} = a', \texttt{AccDiff} = z]\texttt{Pr}(\texttt{AccDiff} = z)$$
$$- \sum_z E[\texttt{MIAcc}|\texttt{NumParams} = b', \texttt{AccDiff} = z]\texttt{Pr}(\texttt{AccDiff} = z)$$

The above expression is analogous to the case where we controlled for model complexity, except we are controlling for the train-to-test accuracy gap now. The estimated value from experiments for this statistic is $E_4 = 0.08$. This is, however, an incorrect analysis. If we analytically control for the train-to-test accuracy gap, then we are actually biasing the total effect that the model complexity has on the attack, as we are "blocking" (failing to distinguish) its indirect effect through the train-to-test accuracy gap. The correct statistical quantity, in this case, turns out to be $E_3 = E[\texttt{MIAcc}|\texttt{NumParams} = a'] - E[\texttt{MIAcc}|\texttt{NumParams} = b']$, i.e., the total effect model complexity has on the attack accuracy. The estimated effect is $E_3 = 0.36$—a lot higher than that obtained from the incorrect analysis, and corresponds to the second unadjusted estimate in Fig. 4.2. Another similar example of bad control or over-controlling is the bias amplification problem or pre-treatment control [171] (illustrated later in Fig. 4.3). The main takeaway is that a principled framework would tell us which quantities to estimate, avoiding over-controlling in experiments and false conclusions.

Large Number of Possible Factors. When moving beyond a couple of factors to consider, the reasoning can become more complicated. To build on our previous example, let us now introduce another factor, related to (H8): the separation between members and non-members. Our hypothesis is that the separation between members and non-members is influenced by both the model complexity and the train-to-test accuracy gap, and in turn it influences the attack accuracy. How does changing the separation then affect the attack accuracy? To answer this question, notice that the separation is influenced by the model complexity and the train-to-test accuracy gap, both of which influence the attack accuracy. Similar to what we described so far, one will then have to make sure to randomize these factors in order to obtain the effect of the separation on the MI attack accuracy. For every additional factor,

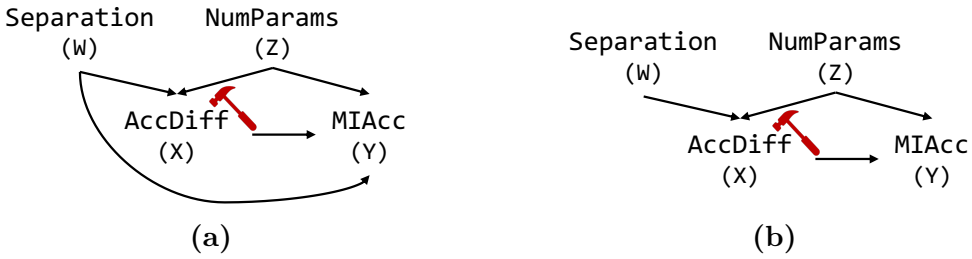


Figure 4.3. The query to estimate varies by assumptions chosen. If we assume that separation score influences the MI attack accuracy (Fig. 4.3a), we should control for two confounding factors, the separation score and the number of parameters. The resulting effect is 0.12. If we assume otherwise (Fig. 4.3b), we should not "control" for the separation score, otherwise it results in a much larger effect of 0.68.

though, we need to do enough experiments to "randomize" our estimates so that they correctly compute the effect on the attack accuracy. It is easy to see that the number of experiments one needs to run quickly starts to grow large as the number of factors considered increases.

Importance of Specifying Assumptions. So far, we have considered cases where certain causal relationships exist and we are trying to correctly estimate the effect of certain factors on the outcome. But, how can we start to test our assumptions, i.e., whether a causal relationship exists at all? Such refutation is hard to do, in general. A practical recourse is that one can specify their assumed beliefs and hope to refute quantitatively under the assumptions. The choice of assumptions matters critically to the outcome. To illustrate this, consider the hypothesis (H8) again, which introduces a separation score that measures the distance between members and non-members as a factor for the single shadow model MI attack [196] (so far, we have considered the multiple shadow one [207]). Deciding whether the separation score has any direct influence on the MI attack is critically important—if we choose to assume so, we get one set of conclusions, if we do not, we get another. When the separation has a direct influence on MI attack, then the principled analysis to estimate the effect of train-to-test accuracy gap is similar to the case of estimating the model complexity. We estimate the effect on the attack accuracy is 0.12 on our set of experiments (the details of our experimental setup are in Section 4.6.1). In the alternative scenario, the correct quantity to estimate is below, leading to the estimated effect of the train-to-test accuracy gap to be $E_5 = 0.68$ when it varies

from the a to b.

$$E_5 = \sum_z E[\texttt{MIAcc}|\texttt{AccDiff} = a, \texttt{NumParams} = z]\texttt{Pr}(\texttt{NumParams} = z)$$
$$- \sum_z E[\texttt{MIAcc}|\texttt{AccDiff} = b, \texttt{NumParams} = z]\texttt{Pr}(\texttt{NumParams} = z)$$

We illustrate the differences in the two sets of assumptions in Fig. 4.3 which exacerbates the bias amplification problem. We point out that prior works do *not* specify such assumptions or beliefs explicitly, making it impossible to refute or validate such hypotheses.

Goodness of Explanations. Given the subjectivity of assumptions and computational limits on the number of experiments one can run, it is difficult to analytically argue that a given explanation is "correct" or certain hypothesis is conclusively "incorrect". How then can we measure how good or correct is an explanation? A practical way to do so is look at the *predictive power* of a given explanation, i.e., measure how accurately it can predict the outcome (MI attack accuracy) under experimental settings *not* seen during creating the explanation. The highlighted prior hypothesis (H1)-(H9) often have predicted power well below 85% on average, offering less satisfying results. In contrast, our approach has predictive power of $3-22\%$ higher than the prior work hypotheses, for most attacks we study.

4.3 The Causal Modelling Approach

The prior common hypotheses, some of which are derived from mechanistic explanations or theoretical analyses, provide a good starting point to reason about potential factors of the MI attacks. But, as shown throughout Section 4.2, there are several pitfalls in identifying the factors and estimating their effect. Our aim is to infer a model defined over a set of potential factors and a given MI attack, not just a simple correlation of each factor separately with the MI attack. The model explicitly defines relationships between factors and the MI attack and between themselves. It should also provide a query interface for the following query types:

- *Prediction Queries*: Given some observed values $a_1, \ldots, a_n$ of the potential factors $X_1, \ldots, X_n$, what is the predicted MI attack accuracy Y: $E[Y|X_1 = a_1, \ldots, X_n = a_n]$?

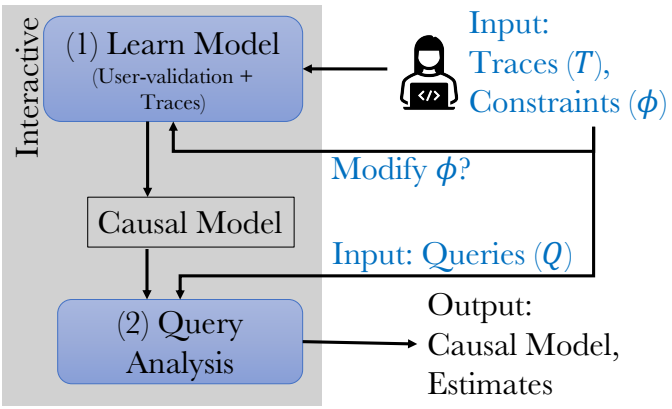


Figure 4.4. The user provides the domain knowledge ϕ and the traces T. The traces contain observations of the values that the factors of interest take for the training algorithm $\mathcal{A}$ and attack A. After an interactive step, the user fixes on a causal graph on which the input queries are analyzed.

- *Interventional Queries*: What is average effect of a potential factor on the attack accuracy if that factor had taken a different value from the observed one?

The prediction query consists of a set of assignments of observed (from running experiments) values for a set of factors, and the target variable Y. The output of this query is the expected MI attack accuracy conditioned on the observed values. Such queries help us measure how well the causal model agrees with observations in experiments. The interventional query is a "what if" query. It consists of two variables: the potential cause variable, called the treatment variable X, and the desired outcome variable Y. For instance, to estimate the effect of the train-to-test accuracy gap on the attack, we ask if the train-to-test accuracy gap had taken the value 0.1 compared to having no train-to-test accuracy gap, what is the expected MI attack accuracy? We want our causal model to be 1) **Accurate**, i.e., to have a goodness of fit, and 2) **Principled**, i.e., the estimated effect is rigorously computed.

We introduce a novel methodological shift: Our approach proposes to use *causal reasoning* to disambiguate potential factors of MI attacks while satisfying the goals highlighted above. We combine mechanistic explanations from domain knowledge with automated inferences from empirical data to infer a causal model. Specifically, causal models are directed acyclic graphs (DAGs) defined over a set of variables and

a set of directed edges[3] where each directed edge from variable X to Y represents a "X causes Y" relationship. On top of the graph structure, causal models are quantitative: each node in the graph has associated an equation that describes the cause-and-effect relationship between the node and its parents. Our approach is necessarily synergistic: without domain knowledge constraints, purely observational data cannot distinguish cause-and-effect; and without observational data, we cannot test our intuitions or extract more insights from experiments. The whole process is interactive and it is illustrated in Fig. 4.4. Initially, the user identifies potential factors or variables of interest of the underlying training and attack procedure such as training hyper-parameters, train-to-test accuracy gap and the outcome MI attack accuracy. We model these as "random variables" that can be observed and measured. The user generates the set of observations for these variables which we call traces T, by effectively running experiments and recording the values of the variables. The domain knowledge constraints (ϕ) formally describe the mechanistic explanations, facts or assumptions that stem from the data-generating process, e.g., the training and attack procedures. Given the traces and the domain constraints, we output a causal graph which the user can choose to further refine (*Modify* ϕ step in Fig. 4.4). Finally, the causal models encode cause-and-effect relationships by construction and can support the 2 types of queries. The user can specify these queries (Q) formally and obtain estimates on the inferred causal graph (step (2) in Fig. 4.4).

Inputs & Outputs. We have prototyped our approach in an interactive tool called Etio and envision model practitioners and researchers as its main users. Etio minimally requires a set of traces corresponding to the runs of a specific training algorithm ($\mathcal{A}$) over the training dataset (D). These traces record values of a set of properties about the training algorithm, model, and the performance metrics of the attack procedure (A)–all of which we call variables (V). The user additionally specifies a set of domain knowledge constraints which encode knowledge that two variables are not in a causal relationship, e.g., if they are caused by the same unmeasurable/confounder variable (which we denote as Forbid constraints) or that there is a causal relationship between two variables (denoted as Enforce constraints). Then the domain constraints ϕ are a concatenation of the Forbid and

[3]We thus interchangeably use causal model and causal graph.

Figure 4.5. Two causal models that are not identifiable (distinguishable) from observations, since both result in the same conditional (in)dependence relations, but require different quantities to estimate in a causal analysis.

ENFORCE sets of constraints. For the studied MI attacks, we describe the variables of interest V in Section 4.4.1 and our domain constraints ϕ in Section 4.4.2 in detail. In addition to the inputs necessary to infer the causal model, ETIO allows the user to pose well-reasoned queries about potential factors of MI attacks, as per the query interface.

4.3.1 Learning the Causal Model

Despite the clear advantage of explicitly expressing assumptions in the form of an interpretable causal graph, constructing one is challenging. The fundamental issue is that while associations or correlational analysis are useful for predicting outcome, they do not always reflect the causal relationship. Associations can at most reveal relationships of dependence or (conditional) independence.

To illustrate this point, we show two models that describe the same conditional independence relationships in Fig. 4.5, but are causally different. In Fig. 4.5a, the model encodes that the model complexity affects the train-to-test accuracy gap which in turn influences the MI attack accuracy. In contrast, the model in Fig. 4.5b describes that the accuracy difference affects both the model complexity and the MI attack accuracy. The models in Fig. 4.5a and 4.5b, though, are indistinguishable from one another purely from observations, they both encode that `NumParams` $\perp\!\!\!\perp$ `MIAcc`$|$`AccDiff`. But, in Fig. 4.5b, the model complexity has no causal effect on the MI attack, whereas in Fig. 4.5a the model complexity causes the MI attack to change through the train-to-test accuracy gap. In fact, from how the experiment is set up, the second relationship does not have any real-world interpretation, i.e., the model complexity is decided beforehand as a hyper-parameter to the training

process. Thus, our approach must rely on domain knowledge, a specification of which is missing in prior works in the literature.

Formally, causal models (G, θ) consist of (a) a DAG $G = (V, E)$ called a causal graph, over a set V of vertices and (b) a joint probability distribution $\mathcal{P}_\theta(V)$, parameterized by θ over the variables in V[4]. The set of variables V can take either discrete or continuous values. Our framework is orthogonal to the underlying representation of parameters. We choose a linear model to represent the relationship between the nodes of the graph. For predictive queries, the parameters have a probabilistic interpretation $\mathcal{P}_\theta(V) = \Pi_i \texttt{Pr}(X_i|pa_{X_i}), X_i \in V$. Each node has associated with it a probability function based on its parent nodes pa_{X_i}. In this work, we utilize the linear Gaussian model: X_i is a linear Gaussian of its parents X_j: $X_i = \beta_0 + \sum_j \beta_j X_j + \epsilon$ where $X_j \in pa_{X_i}$ and $\epsilon \sim \mathcal{N}(0, \sigma^2)$.

Note that our choice of linear equations and Gaussian probability functions are not fundamental—these can be changed if necessary. These choices have been sufficient to create causal models with good predictive power for the attacks we analyze (see Section 4.6.2).

To learn the causal graph, there are two sub-steps: (a) learning the structure of the graph $G = (V, E)$ from the traces T and constraints ϕ and (b) learning the parameters of the causal graph. Conceptually, the first sub-goal is to maximize the posterior probability $\texttt{Pr}(G|T) = \texttt{Pr}(G)\texttt{Pr}(T|G)$, where $\texttt{Pr}(G)$ is a prior on the graph (i.e., G contains the edges represented by the ENFORCE list and all graphs with edges that are part of the FORBID have 0 probability) and $\texttt{Pr}(T|G)$ is the predictive probability of the graph G. Ideally, the posterior probability concentrates around a single structure G_{MAP}, the optimal directed acyclical graph. Learning the optimal DAG though is intractable for most problems as the number of DAGs is super exponential with the number of nodes $O(n!2^{\binom{n}{2}})$ [189]. In fact, recovering the optimal DAG with a bounded in-degree ≥ 2 has been shown to be NP-hard [32].

We choose to instantiate our approach with a standard hill-climbing algorithm [249, 37], an iterative Greedy approach that starts from the graph with nodes representing the variables V and the edges that are part of the ENFORCE. The algorithm does not guarantee that the produced graph is the optimal one but it

[4]To simplify notation, we denote the vertex and its corresponding variable the same.

is scalable. Since our goal is to disambiguate between many different possible factors (see Section 4.2), this technique allows the user to add new variables of interest and has a good predictive accuracy (goodness of fit) in practice. The algorithm iteratively tries to add, remove, or reverse the direction of a directed edge from the graph at the previous step. It uses a scoring function to choose between these operations. The scoring function maps a graph to a numeric value. We use a type of score based on log-likelihood $LL(G|T)$ but that prefers simpler graphs ($LL(G|T) - p$, where p is a penalizing term that grows with more complex causal models with more parameters). This is known as Bayesian Information Criterion [199]. For each such operation, the hill-climbing algorithm computes the change in the score if that operation had been performed. It then picks the operation that results in the best score and stops when no further improvements are possible. Moreover, several distinct graphs G can have similarly high posterior probabilities which is common when the data size is small compared to the domain size [61]. This is in part due to the causal ambiguity of learning from data.

Instead of learning a single graph, ETIO uses a bootstrapping technique [60]. The bootstrapping process resamples the traces T with replacement. It then returns a set $\mathcal{S}$ of multiple bootstrap datasets S. For each bootstrap dataset, ETIO uses the graph learning algorithm to learn the structure of the graph G'. For every arc present in the set of graphs AG, ETIO estimates the strength or confidence that each possible edge e_i is present in the true DAG as $\hat{p}_{e_i} = \frac{1}{|\mathcal{S}|} \sum_{b \in \mathcal{S}} \mathbb{1}_{\{e_i \in E_b\}}$, where $\mathbb{1}_{\{e_i \in E_b\}}$ returns 1 if $e_i \in E_b$, else returns 0. The purpose is to prune out the edges that are below a certain confidence threshold t. There are existing techniques to estimate the confidence threshold such as the L_1 estimator [201] which ETIO uses to fix the confidence threshold. Using the most significant arcs, ETIO constructs a graph that contains all of the significant arcs (the averaged graph G). Our approach does not guarantee that the obtained graph represents the true causal graph–inferring one in our setup is infeasible. Thus, we take the practical approach and aim to infer a graph with a good predictive power.

4.3.2 Answering Queries

Predictive queries ask what is the output of the model given that certain input factors have certain values. Given a set of previously unseen set of assignments for the variables $\{X_i = a_i\}, X_i \in V$, the outcome corresponding to the MI attack node $Y \in V$ is computed by the expression $E[Y|pa_Y]$. This expression is recursively expanded until it is conditioned on the X_i values and can be evaluated with the given concrete values. To learn the coefficients associated with each node, we use a standard maximum likelihood estimation approach to fit each node's observed data conditioned on its parents.

In principle, we can answer interventional queries, which measure how much changes in an factor's value affects the outcome, by conducting experiments where we manipulate the training process such that input variables take desired values. Such manipulations are called interventions. Formally, given a set of variables $V = \{X_1, \ldots, X_n\}$, an intervention on a set $W \subset V$ of the variables is an experiment where the experimenter controls each variable $w \in W$ to take a value of another independent (from other variables) variable u, i.e., $w = u$. This operation, and how it affects the joint distribution, has been formalized as the *do* operator by Pearl [170]. For example, in Fig. 4.5a, we can intervene on the model complexity independently of the other variables. However, in some cases modifying variables directly is not feasible in practice (e.g., the train-to-test accuracy gap) as it requires knowledge of the data distribution that the model is trying to learn in the first place. So, we cannot really conduct such interventional experiments.

The key insight to answer intervention queries is that we can reason about such queries with only the causal graph and the data– Etio applies the principles of *do*-calculus to analytically compute the causal relationship expressed by the *do*-query. The *do*-calculus rules have been proven to be sound and complete [208, 95]. They are complete in that if repeated application of the rules of *do*-calculus cannot obtain a conditional probability, then the algorithm outputs that the causal relationship cannot be identified without additional assumptions. If we do obtain an ordinary conditional probability, then we say that the causal estimate can be identified, i.e., the graph has enough assumptions or no ambiguity. Then, the obtained expression (called the *estimand*) represents the correct translation of the causal query to a

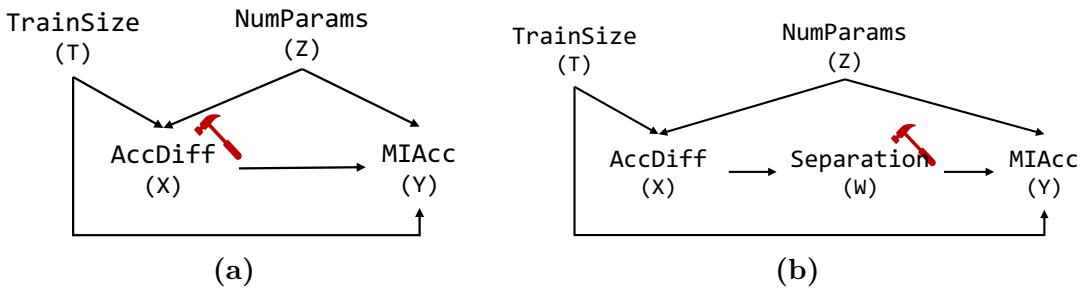


Figure 4.6. Importance of selecting the right control variables to avoid selection bias. T and Z are both in adjustment set for X in 4.6a. In Fig. 4.6b, to estimate the effect of separation score (W), we can control for T, Z or X but controlling for the train-to-test accuracy gap (X) introduces selection bias.

conditional probability (soundness). Such guarantees are powerful tools: Given the formal query and the causal model, this approach avoids paradoxes that might arise from over-controlling or not controlling (Section 4.2). We will explain a small fragment of this calculus through an example.

Example: Backdoor Paths. Let us consider the examples in Fig. 4.6. In Fig. 4.6a, the query (Q1) is to estimate the effect of train-to-test accuracy gap on the MI attack accuracy, given only observations of the variables of interest. In Fig. 4.6b, we introduce a new variable, the separation score between members and non-members (H8), which is caused by the train-to-test accuracy gap. The query (Q2) asks to estimate the effect of the separation distance on the attack accuracy. The causal model in Fig. 4.6a was previously discussed in Section 4.2—the expected attack accuracy is computed over both of the confounding factors, the model complexity, and the training set size. From the graph structure, observe that the confounding happens because of the two undirected paths from the node corresponding to the train-to-test accuracy gap to the MI attack node (π_1 : `AccDiff` $\leftarrow$ `NumParams` $\rightarrow$ `MIAcc` and π_2 : `AccDiff` $\leftarrow$ `TrainSize` $\rightarrow$ `MIAcc`). Such paths are called *backdoor paths*. A backdoor path is a non-causal path from X to Y. This is a path that would exist in the graph even if we were to remove the outgoing edges from the node of interest. When there are backdoor paths, there are sources of association (which we can observe statistically) in addition to the causal ones. In Fig. 4.6b, it seems the query (Q2) requires a similar control as in Fig. 4.6a. However, which of the nodes on the paths π_1 and π_2 paths, should we control for? The node corresponding to the

train-to-test accuracy gap is one such candidate. Notice though that unlike other nodes on the backdoor paths, it has two incoming edges, meaning that controlling for it biases the observed relationship between its parents. Such nodes are called "colliders" and can introduce bias (see Section 4.2). One has to carefully determine exactly when to control for colliders.

There is a set of principled rules to "block" backdoor paths. We summarize these rules informally here but interested readers can refer to [170, 171] for more thorough background. A path is blocked if 1) we control for a non-collider on that path or 2) we do not control for a collider on that path. For any given path, only one of these conditions is required to block the path. So, if there exists a path between X and Y that contains an uncontrolled collider, that path is blocked without controlling on any other variables. Guided by these rules (called the *backdoor criteria*), in (Q2) we should control for X when estimating the effect of W on the MI attack.

Estimating the Causal Effect. Recall that we are interested in the average treatment effect, which is the average difference in the outcomes given that the treatment takes two values: the *treatment* value and the *control* value. A straightforward way to compute the average treatment effect (ATE) is by using the difference in the mean of the outcome *conditioned* on the treatment variable ($E[Y|X = a] - E[Y|X = b]$). However, this method of computation suffers from statistical pitfalls, such as sampling bias and confounding bias which we highlighted in Section 4.2. Instead, what we want to quantify is the average treatment effect as a *do*-query as defined below.

Definition 5 (Average Treatment Effect)**.** *The average treatment effect of a variable of interest X (called the treatment) on the target variable Y (called the outcome) is:*

$$\text{ATE}(X, Y, a, b) = E[Y|do(X = a)] - E[Y|do(X = b)],$$

where a, b are constants for which X is defined. We omit the constants when the query is over the domain of X.

Etio will translate the *do*-query $E[Y|do(X = a)]$ into an ordinary (conditional) expectation expression from given the causal model (e.g., using the backdoor criteria). It then learns an estimator that allows computing the ordinary expectation using the available data. We choose a linear regression model to estimate the quantities of

interest. Its estimates are interpretable: a positive $\text{ATE}(X, Y)$ value means that an increase of the feature X causes an increase in the MI attack accuracy Y, and vice-versa for negative $\text{ATE}(X, Y)$.

In summary, we have described a methodology encapsulated in ETIO to analyze causally potential factors. While our methodology uses techniques standard in causality, we have carefully laid out the technical choices that allow us to achieve our goals: 1) used linear equations to capture causal effects; 2) combined Greedy structured algorithm with bootstrapping to scale the creation of models; and 3) defined the average treatment effect as our measured outcome.

Remark. Our approach assumes acyclicity and causal sufficiency, i.e., there are no hidden variables that are common causes of two or more observed variables. There are causal discovery algorithms that allow one to make inferences about the causal graph without the causal sufficiency assumption, but the resulting causal graphs may have undirected edges hence limiting what queries could be answered [198, 220]. We utilize domain knowledge to make up for these ambiguities and limitations of causal discovery from observations. We consider what variables might be confounders for observed variables and discuss these in the next section.

4.4 Connecting MI and Generalization

Our main technical novelty is how use ETIO to study the connection between MI attacks and classical generalization in ML. We now show how to create causal models for 6 different attacks and *formalize* hypotheses (H1)-(H9) made in prior works.

4.4.1 Variables of Interest

The generalization notions and other potential causes identified in H1-H9 (Section 4.2) are properties of the training algorithm. The training algorithm $\mathcal{A}$ takes as input a training dataset D consisting of N samples $D = \{(x_1, y_1), \ldots, (x_N, y_N)\}$, $D \sim P^N$, each independently identically drawn from P where P is a distribution over $\mathcal{X} \times \mathcal{Y}$, $\mathcal{X}$ is the input space and $\mathcal{Y}$ is the output space. The training algorithm also takes as input a set of training hyperparameters π and the loss function l. The training algorithm produces a model $f : \mathcal{X} \to \mathcal{Y}$. All of the prior works have studied

MI attacks on neural networks trained with stochastic gradient descent (SGD), so we focus on SGD primarily in this work.

The generalization error is a measure of how well a learned model f can correctly predict previously unseen data samples. For a given a model f and a sample $z = \{x, y\} \sim P$, the generalization error is defined as $\mathbb{E}_{z\sim P}[l(f, z)]$. The learned model f depends on the drawn training dataset D. As a result, the generalization error of $\mathcal{A}$ is $\texttt{TestLoss} = \mathbb{E}_{z\sim P,D}[l(f_D, z)]$. We denote the generalization error as `TestLoss` since if we were to sample $z \sim P$ it would be highly unlikely for that sample to belong to the training set D.

Bias-variance decomposition. A fundamental principle to understand generalization in machine learning models is the bias-variance decomposition [114, 69, 251], which establishes that the **generalization error** directly factors into **Bias** and **Variance** as shown in Table 4.1. The bias represents how well the hypothesis class $\mathcal{H}$ to which the model f belongs to fits the true data $\mathcal{Y}$, while the variance represents how much the model varies across different samples of data. For example, with sufficient training time, a model that is overly-parametrized can have a low bias (since it fits the data very well) and high variance (because it can fit all the "accidental regularities" or idiosyncrasies of the sampled data). Our causal models use bias and variance as variables and therefore these serve as a new lens to explain how they affect MI attacks in addition to generalization in the same representation. To compute the bias and variance, we follow the methodology outlined in prior work [251]. We first compute an (unbiased) estimator for the variance term in the bias-variance decomposition. Next, from the generalization error (or loss), we subtract the variance to derive the bias. The full mathematical formulation of bias and variance is in Section 4.5.

MI Attack Accuracy. (H1)-(H9) are claims relating the effect of potential causes on the MI attack susceptibility. To measure susceptibility, we consider the random variable corresponding to the MI attack accuracy, for each prior work attack. In total, we study three MI attacks: the multiple shadow model (`ShadowAcc`) [207], the single shadow model attack (`MLLeakTop3Acc`) [196], and threshold-based attack (`ThreshAcc`) [254]. We additionally perform similar attacks to [196] where we use one shadow model (`MLLeakAcc`). We take the whole prediction vector for CIFAR10,

and the top-10 predictions for CIFAR100 as input features to train the attack model. Other considered variants of this attack include the correct label in the input features of the attack model. We denote these as `MLLeakTop3Acc-l` and `MLLeakAcc-l`. For each learned network f_D, we evaluate the MI attack on members $\in D$ and a dataset of previously unseen samples, non-members $\notin D$. The final result is the expected accuracy on members and non-members averaged over multiple samples f_D.

Other Model Properties. Some of the hypotheses in prior work involve training hyperparameters and model properties such as training set size (H4) and model complexity (H3). For H3, we use the number of parameters in the model to measure model complexity. Specifically, we count the number of parameters (e.g., weights and biases) that are updated during the training phase.

The potential factors that appear in H1-H9 are summarized in Table 4.1. Our aim is to infer a causal model over these variables. Next, we require traces, so we run the training algorithm to collect observations of these variables. We leave the detailed process to generate traces of these variables for Section 4.6.1. Besides traces, we formulate domain knowledge constraints as input to Etio.

4.4.2 Domain Knowledge as Constraints

As our domain-specific constraints, we leverage simple insights that force the hill-climbing algorithm to infer models that have causal meaning. For instance, one constraint encodes that the root nodes of the model should correspond to variables that are part of the training algorithm's hyper-parameters such as `TrainSize` and `NumParams`. This constraint belongs to the Forbid list. In addition to these, we have identified the following constraints:

- *There are no outgoing edges from the attack node.* Without this constraint, the structure learning algorithm could learn that the attack causes one of the features–the direction of the edge cannot be inferred by observations only.
- *There is no edge from a node that is neither a root node nor* `TrainVar` *nor* `TrainLoss` *to* `TrainBias`*.* We add this constraint because the `TrainBias` is computed from the `TrainLoss` and `TrainVar`. Any influence on `TrainBias` should be mediated by its two parents.

Table 4.1. Summary of variables we consider when building our causal graphs to answer queries Q1-Q9. We build the causal graphs for each MI attack A [207, 196, 254]. For a given sample z, the MI attack outputs whether it is a member (m) or not ($\neg m$). We illustrate the variance term only for MSE where $\bar{f}(x) = \mathbb{E}_D[f(x, D)]$.

Variables	Formula
TrainAcc	$\mathbb{E}_{z\sim D,D}[f_D(x) = y]$
TestAcc	$\mathbb{E}_{z\sim P,D}[f_D(x) = y]$
AccDiff	TrainAcc- TestAcc
TrainLoss	$\mathbb{E}_{z\sim D,D}[l(f_D, z)]$
TrainVar	$\mathbb{E}_{x\sim D,D}[\left\|f_D(x) - \bar{f}(x)\right\|^2]$
TrainBias	TrainLoss-TrainVar
TestLoss	$\mathbb{E}_{z\sim P,D}[l(f_D, z)]$
TestVar	$\mathbb{E}_{x\sim P,D}[\left\|f_D(x) - \bar{f}(x)\right\|^2]$
TestBias	TestLoss- TestVar
LossDiff	TestLoss- TrainLoss
NumParams	$\|f_D\|$
TrainSize	$\in \{1k, 5k\}$
ShadowAcc	$\mathbb{E}_{z,D}[A(z) = m \| z \sim D \wedge A(z) = \neg m \| z \sim P]$
MLLeakAcc, MLLeakTop3Acc	$\mathbb{E}_{z,D}[A(z) = m \| z \sim D \wedge A(z) = \neg m \| z \sim P]$
ThreshAcc	$\mathbb{E}_{z,D}[A(z) = m \| z \sim D \wedge A(z) = \neg m \| z \sim P]$
CentroidDist	$\mathbb{E}_D[\|C(D) - C(P)\|]$

- *There is no edge from a node that is not a root node to* `TrainVar`. The variance on training samples is computed directly on the prediction vectors.
- *There is no edge from a node that is neither a root node,* `TestVar` *nor* `TestLoss` *to* `TestBias`. We add this constraint because the `TestBias` is computed from the `TestLoss` and `TestVar`. Any influence on `TestBias` should be mediated by its two parents.
- *There is no edge from a node that is not a root node to* `TestVar`. The variance on testing samples is computed directly on the prediction vectors.
- *Constraints in* ENFORCE. There is an edge from `TrainAcc` to `AccDiff` and `TestAcc` to `AccDiff`. There is an edge from `TrainLoss` to `LossDiff` and `TestLoss` to `LossDiff`.

- There are no edges from the `CentroidDist` to `TestLoss` and, respectively, `TestAcc`. We add this edge because the direction of influence should be the other way around, if such edges are inferred.
- There is no edge from the `TestVar` to `TestLoss`. Similarly, these two quantities are computed from the prediction vector for `TestVar` and, prediction vectors and labels for `TestLoss`, so there is no edge between them.

In total, we have 19 FORBID and ENFORCE constraints. Such constraints are easy to derive, as they either stem from the definitions in Table 4.1 or from the data-generating process.

4.4.3 From Hypotheses to Queries

We can obtain a causal model from the traces and the domain knowledge constraints using ETIO. The last step is to formulate queries on the causal models. Hypotheses H1-H9 can all be formally described as interventional queries Q1-Q9 respectively, as follows.

(H1,H6,H9) → (Q1,Q6,Q9): Generalization Metrics. Existing work uses two different metrics to quantify generalization precisely: the train-to-test accuracy gap (`AccDiff` in Table 4.1) and the average generalization error (`LossDiff` in Table 4.1). We use the metric that was cited by the respective original works to determine the query for each of the studied attacks.

(H2) → (Q2): Formalizing "Closeness". We first quantify "closeness" more formally between a shadow model and a target model when these have the same architecture, as is the case for the MI attack by Shokri et al. [207]. Our observation is that the variance term in the generalization error is a metric of "closeness". Intuitively, since the shadow model is a realization of a different subset D_i of D, the variance is a measure of the expectation of whether training on different sampled training sets (of the same size) from the data distribution outputs neural networks with very "different" prediction vectors. Since this particular MI attack trains shadow models with the same architecture as the target model, the larger the variance, the more likely we are to obtain a shadow model that is on average more "distant" from the target model. Hence, H2 can be reformulated to check if the variance of the learning algorithm is a cause of the attack. We consider the variance of $\mathcal{A}$ on the

members (training samples, denoted as `TrainVar`) and the variance of $\mathcal{A}$ on non-members (testing samples, denoted as `TestVar`), separately. The causal query asks if `TrainVar`, and, respectively, `TestVar`, affect the MI attack accuracy `ShadowAcc`, i.e., if a change in `TrainVar` (or `TestVar`) causes the `ShadowAcc` to change. Formally, the hypothesis translates to two *do*-queries (Q2): ATE(`ShadowAcc`, `TrainVar`) and ATE(`ShadowAcc`, `TestVar`).

(H5) → (Q5): Single Shadow Model. We want to check if the closeness (as measured by `TrainVar` and `TestVar`) is a cause for the single shadow model attack. If it does not contribute to the attack, the hypothesis is correct.

(H7) → (Q7): Different Causes. Prior work showed that in terms of MI attack performance there is no significant difference between the performance of attack models trained on the top three predictions of the prediction output vs. the whole prediction vector [196] (page 5, Fig.4). We denote the attack using the whole prediction vector as `MLLeakAcc`. Here, we are interested if the attack model's performance changes with the target model for the variants of the attack. Thus, we formalize it as checking whether `TestVar` and `TrainVar` are causes for `MLLeakAcc` (-l) and `MLLeakTop3Acc` (-l).

(H8) → (Q8): Formalizing Decision Boundaries. A few works give credit to the clear decision boundary between members and non-members for the success of shadow model-based attacks [196]. To quantify the "distinguishability" between members and non-members, we first compute the centroid of members ($C(D)$) and non-members ($C(P)$) as the following: $C(D) = \mathbb{E}_{z\sim D}[f_D(z)]$ and $C(P) = \mathbb{E}_{z\sim P}[f_D(z)]$. Then, we use the Euclidean distance between the above two centroids to measure the distinguishability for the given training set. For each training setup ($\mathcal{A}, \pi$ and architecture), we compute the averaged centroid distance over multiple different training sets. Note that the user can specify any such existing statistic as a distinguishability measure between the training and testing set as input to Etio.

Implementation. Our implementation consists of two parts: generating the traces, i.e., training models and running attacks, and implementing Etio. For the traces, we use the standard machine learning library PyTorch 1.7.1+cu110 [169] to train the models and run the attacks. For Etio, we use two libraries for analyzing the MI attacks. First, we use the R library called bnlearn 4.7 [200] to infer the causal

models. This library offers several off-the-shelf algorithms for structured learning and Bayesian inference. Second, we use dowhy 0.7.0 [204, 203] to implement the average treatment effect queries.

4.5 Bias-Variance Decomposition

Generalization is defined as the expected error on all possible samples and all possible datasets, i.e., $\mathbb{E}_{x,y}\mathbb{E}_D[l(f_D(x), y)]$.

MSE loss. We first consider that the neural network was trained using squared error loss. The expected error or generalization error is [251, 69]:

$$\begin{aligned}
&\mathbb{E}_{x,y}\mathbb{E}_D[(y - f_D(x))^2] \\
&\quad = \mathbb{E}_{x,y}\mathbb{E}_D[y^2 - 2y f_D(x) + f_D(x)^2] \\
&\quad = \mathbb{E}_{x,y}[y^2 - 2y\mathbb{E}_D[f(x, D)] + \mathbb{E}_D[f_D(x)^2] \\
&\quad = \mathbb{E}_{x,y}[y^2 - 2y\bar{f}(x) + \bar{f}(x)^2] + Var[f_D(x)] \\
&\quad = \mathbb{E}_{x,y}[(y - \bar{f}(x))^2] + \mathbb{E}_{x,y}[Var[f_D(x)]] \\
&\quad = \mathbb{E}_{x,y}[(y - \bar{f}(x))^2] + \mathbb{E}_{x,y}\mathbb{E}_D[(f_D(x) - \bar{f}(x))^2],
\end{aligned}$$

where $\bar{f}(x) = \mathbb{E}_D[f_D(x)]$ are the averaged predictions over different training sets. The first term $\mathbb{E}_{x,y}[(y-\bar{f}(x))^2]$ is the bias and the second term $\mathbb{E}_{x,y}\mathbb{E}_D[(f_D(x) - \bar{f}(x))^2]$ represents the variance.

Cross-entropy Loss. We follow prior work's generalized decomposition for the cross-entropy loss [251]. Let $\pi_0(x) \in \mathbb{R}^c$ be the one-hot encoding of the ground truth label. The cross-entropy is $H(\pi, \pi_0) = \sum_{l=1}^{c} \pi_0[l] \log \pi[l]$, where$\pi[l]$ is the l-th element of π.

$$E[H(\pi_0, \pi]D = D_{KL}(\pi_0||\hat{\pi}) + E[D_{KL}(\hat{\pi}||\pi)]D,$$

where $\hat{\pi}$ is the average of log-probability after normalization: $\hat{\pi}[l] \approx exp(E[\log \pi[l]])$ for $l = 1, \ldots, c$.

Estimating the Bias and Variance. For MSE loss, we use the following unbiased estimator for variance:

$$\widehat{Var}(x, D) = \frac{1}{n-1} \sum_{j=1}^{n} \left\| f_{D_i}(x) - \sum_{j=1}^{n} f_{D_j}(x) \right\|$$

The final value of bias and variance, `TrainBias` and `TrainVar`, are obtained by taking the average over the members $x \in D$. Similarly, the `TestBias` and `TestVar` are averaged over the non-members $x \sim P$. We repeat this computation with $N = 3$ different random disjoint splits and take the average of the estimate to decrease the variance of the estimator. In total, for each model architecture $\mathcal{M}_w$ (e.g., $\mathcal{M}$ =Resnet34 with width $w = 2$), we train $n \cdot N$ f_D over a given dataset D.

4.6 Evaluation

We evaluate ETIO on two grounds:

- (EQ1) *Goodness of fit*: Are the causal models predicting the MI attack more accurately on unseen samples than correlational analyses?
- (EQ2) *Utility*: Is ETIO useful in refuting or confirming prior hypotheses? Does it provide useful insights to how MI attacks connect to generalization and how defenses work?

We study 6 MI attacks: multiple shadow model [207], 4 variations of the single shadow model [196] and threshold-based [254]. We present the results with respect to the 9 queries for two loss functions, so in total we have 18 prior work hypotheses. In addition, we study 2 practical defenses proposed in prior work. The first is L2-regularization (also known as weight decay) that was proposed as a mitigation strategy for the multiple shadow model attack [207]. It has been used as a baseline for other defenses [158]. The second defense we consider is MemGuard [106], a defense that changes the prediction vectors without changing the accuracy of the model. We choose this defense as it is effective against the attacks we also considered in this work [207, 196]. The causal models for all of the evaluated attacks and defenses are available in Apppendix A.2.

We present details later and summarize our key findings below:

- There is no one-size-fits-all explanation for the 6 MI attacks—the factors contribute differently in different attacks.
- Our analysis refutes 7/18 prior hypothesis we formalized, and confirms 9/18 as valid.
- Our causal models have predictive accuracy of 0.90 for unobserved experiments, which are not used for causal model creation. This is comparable or better by $3-22\%$ than simple correlational analysis between the single cause and the MI attack, in all cases we study.
- Bias and Variance observed during training can quantitatively predict both generalization measures (e.g., `TestLoss`, `TestBias`, and `TestAcc`) and MI attack performance, providing new insights. These factors play a disproportionately larger role in explaining MI attack performance, compared to other factors such as model complexity, dataset size, or even generalization measures themselves.
- Defenses reduce certain causes of the MI attack, but not all and not completely. They reduce the effect of variance, but fail to eliminate factors such as the train-to-test accuracy gap or the distance between members and non-members.

4.6.1 Experimental Setup

Datasets. We select 3 common image datasets: MNIST, CIFAR10, and CIFAR100. MNIST has 60k training and 10k testing samples of 28×28 grayscale images of handwritten digits. CIFAR10 and CIFAR100 have 50k training and 10k testing samples of 32×32 color images uniformly distributed in 10 and 100 classes, respectively.

Models. For each dataset, we train multiple models with different architectures and hyperparameters. For MNIST, we use multilayer perceptron (MLP) with one hidden layer to build the target model. We change the number of units used in the hidden layer ($\{16, 32, 64, 128, 256\}$) to change the *width* or the model complexity. For CIFAR10 and CIFAR100, we use various convolutional neural network (CNN) architectures: AlexNet, DenseNet161, and ResNet34. For changing the width, we vary the number of filters of these models. The widths considered for AlexNet were $\{16, 32, 64, 128, 256\}$, and for DenseNet161 and ResNet34 they were $\{2, 4, 8, 16, 32\}$.

Training Algorithms. We trained all models using stochastic gradient descent (SGD) with momentum 0.9, two weight-decay rates $\{5 \times 10^{-4}, 5 \times 10^{-3}\}$ and two kinds of loss functions: mean squared error (MSE) and cross entropy (CE). The higher weight decay models are used to evaluate L2-regularization. We summarize the training configurations we considered in Appendix, Table 4. For the models trained with the scheduler, we used the step learning rate scheduler with the learning rate decay factor of 10 and for 200 epochs.

Variables of Interest. To estimate the variables of interest for each training setup, we follow the procedure proposed by Yang et al. [251]. Specifically, we randomly generate disjoint splits of training samples $D = D_1 \cup \ldots \cup D_n$, where size of each $|D_i| = s$. We train n models for each architecture width f_{D_i} over different training sets D_i. For CIFAR10 and CIFAR100, we use $n \in \{10, 50\}$ and, respectively, $s \in \{5000, 1000\}$, while we use $n \in \{12, 60\}$ for MNIST because MNIST has a larger training set.

Attacks. The shadow model training size is equal to the training size of the target model, i.e., either 1000 or 5000. This set forms the member set for the attack model. Additionally, an equal-sized set is used to form the non-member training set for the attack model. The evaluation set for the attack model consists of the 1000 or 5000 training samples of the target model and an equal-sized set not previously seen by either target and shadow models. For each architecture and width, we perform the attack 30 times for different samples of the datasets from the original training set D. The same splits are used for all datasets and single shadow model attacks.

4.6.2 Predictive Power of Causal Models

To evaluate the predictive accuracy of the graphs on unseen observations, we use two metrics regularly used in evaluating Bayesian nets: 1) *mean predictive correlation* and 2) *mean squared error* (MSE). We compute these two metrics using standard cross-validation over multiple runs (20). For each run, we use a 80/20 split of the observations for the train-test sets. For each run of the cross-validation, the predictive correlation measures the (linear) correlation between the observed and the predicted values for the MI attack node.

Baseline. We use a simple baseline that can predict the accuracy of the attack: we

Table 4.2. The graphs that ETIO generates have consistently equal or better predictive power than simple correlations. **Best Corr.** stands for the Pearson correlation coefficient, and **Etio Pred.** stands for predictive correlation.

		Normal / High Weight Decay			
Loss	**Attack**	**Best Corr. Variable**	**Best Corr. Value**	**Etio Pred.**	**Etio MSE**
ce	MLLeakAcc	AccDiff	0.8543	0.9358	2.31E-03
			0.8621	0.9083	2.54E-03
mse	MLLeakAcc	AccDiff	0.8493	0.9170	1.80E-03
			0.8675	0.9033	9.55E-05
ce	MLLeakAcc-l	AccDiff	0.8209	0.9685	1.17E-03
			0.8414	0.9486	2.05E-03
mse	MLLeakAcc-l	AccDiff	0.9397	0.9740	7.30E-04
			0.9371	0.9694	6.72E-05
ce	MLLeakTop3Acc	CentroidDist	0.9257	0.9663	1.16E-03
		AccDiff	0.8595	0.9645	1.16E-03
mse	MLLeakTop3Acc	AccDiff	0.8743	0.9466	1.26E-03
		AccDiff	0.9041	0.9216	1.14E-04
ce	MLLeakTop3Acc-l	CentroidDist	0.9166	0.9641	2.21E-03
		AccDiff	0.8409	0.9579	1.58E-03
mse	MLLeakTop3Acc-l	AccDiff	0.8447	0.9244	1.73E-03
		AccDiff	0.9000	0.9361	9.51E-05
ce	ShadowAcc	AccDiff	0.9752	0.9817	6.45E-04
			0.9694	0.9762	9.18E-04
mse	ShadowAcc	AccDiff	0.9526	0.9733	5.36E-04
			0.9626	0.9689	1.61E-04
ce	ThreshAcc	LossDiff	0.7517	0.9739	9.00E-04
			0.7730	0.9425	1.77E-03
mse	ThreshAcc	LossDiff	0.9823	0.9906	2.43E-04
			0.9775	0.9880	2.41E-05

compute the Pearson correlation between the observed values of the MI attack and the observed values of the other variables of interest we identified (Section 4.4.1). A high Pearson correlation (close to 1) means that there is a linear relationship perfectly describing the MI attack and the variable. If so, to predict the MI attack accuracy, measuring this one variable and learning the coefficients of the relationship from data is enough.

In total, we evaluate 24 setups for 6 attacks for models trained with two loss functions, with and without L2-regularization. For all of the attacks on both undefended and defended models, the predictive correlation is above 0.90 (Table 4.2). Compared to the correlation baseline, the graphs ETIO produces are consistently equal or better for all 24 setups. For 17/24 of the setups, the predictive correlation improves $3 - 22\%$. For the remaining 7/24, the causal models are on par with baselines or slightly better, within 3%. The mean MSE for predictions is low

Table 4.3. We translate the prior work hypothesis H1-H9 to ATE(Feature, Attack) queries. For (Q2,Q5,Q7) which are made of several ATE queries, we ✓ only if all ATE queries support the prior hypothesis. $*$ means the p-value > 0.05 and we mark such queries with $\circ$.

		CE		MSE	
Attack	**Feature**	**ATE**	**Query Result**	**ATE**	**Query Result**
ShadowAcc	AccDiff	0.30	Q1: ✓	0	Q1: ×
ShadowAcc	TrainVar	0.02	Q2: ✓	0.03	Q2: $\circ$
ShadowAcc	TestVar	0.94		0.20 $(*)$	
ShadowAcc	NumParams	0.15	Q3: ✓	-0.005	Q3: ✓
ShadowAcc	TrainSize	-0.11	Q4: ✓	-0.09	Q4: ✓
MLLeakAcc	TrainVar	-0.34	Q7: ×	-0.05 $(*)$	Q7: ×
MLLeakAcc	TestVar	0.81		0	
MLLeakAcc-l	TrainVar	-0.24		0.06	
MLLeakAcc-l	TestVar	0.84		0	
MLLeakTop3Acc-l	TrainVar	-0.40		-0.06 $(*)$	
MLLeakTop3Acc-l	TestVar	0.75		0	
MLLeakTop3Acc	AccDiff	0.18	Q6: ✓	0	Q6: ×
MLLeakTop3Acc	TrainVar	-0.34	Q5: ×	-0.15 $(*)$	Q5: ×
MLLeakTop3Acc	TestVar	0.78		0.24	
MLLeakTop3Acc	CentroidDist	0.27	Q8: ✓	0	Q8: ×
ThreshAcc	LossDiff	1.47 $(*)$	Q9: $\circ$	-0.67	Q9: ✓

(0.001) for all of our evaluated attacks and models. We find that in the case of ShadowAcc, Etio does not significantly improve the accuracy, as the correlation values are already higher than 0.95. This confirms what prior works suggest [254, 132, 215]: using an attack based on the prediction correctness yields, on average, similar performance to the ShadowAcc. We observe that the AccDiff is almost in a perfectly linear relationship with the accuracy of the multiple shadow model attack. Similarly, the metric we formalize, the centroid distance between clusters of members and non-members (CentroidDist) is almost perfectly linear with the single shadow model attack.

4.6.3 Testing of Prior Hypotheses

We confirm using our analysis that 9/18 of the prior work hypotheses are true (✓ in Table 4.3). We also find 7/18 prior hypotheses do *not* identify a cause for the studied MI attack (× in Table 4.3). As some of the hypotheses involve more than one potential cause or they are comparing causes between attacks, we have a total of 32 ATE values, 16 for each loss function.

We refute prior hypotheses in broadly two instances. First, when there is no

universal explanation: prior hypotheses often overlook the differences between NNs trained with different loss functions and the specifics of the attack. Second, because prior hypotheses do not consider the connections between the parameters of the training process and variables that naturally appear as part of SGD such as loss difference and variance, etc. In summary:

- A single causal factor does not explain all attacks. In fact, causes vary per attack and differ by the loss function used.
- (Q1, Q6) The train-to-test accuracy gap does cause the MI attack accuracy, though for MSE-trained models, the loss difference is a more suitable metric.
- (Q2) The "closeness" of the shadow model influences the MI attack accuracy, more so for CE-trained models than for MSE-trained models.
- (Q7) There are differences between the variants of the single shadow model attack, and the single shadow model with top-3 is more robust to changes in the shadow model.
- (Q3, Q4) Training size is a factor that affects the MI attack accuracy for all of the evaluated attacks. Model complexity is a cause for all evaluated attacks but to a very small degree in some cases.
- (Q5) We find that the variance of the outputs of the models is also a cause for the single shadow model attack, to various degrees depending on the type of attack. Prior work overlooks the differences in the prediction vectors between the target and shadow model.
- (Q8) Our formalized distance between the clusters of members and non-members is one of the largest causes for the single shadow model with top-3.
- (Q9) The threshold-based attack is influenced with varying degrees by other factors that are related to the loss.

CE vs. MSE. We find that the train-to-test accuracy gap has the largest influence for CE-trained models, whereas for MSE-trained models it is the loss differences between members and non-members. Similarly, factors such as the variance and

the centroid distance that affect MI attack accuracy on CE-trained models are not factors on MSE-trained models.

Detailed Analysis. The differences in the prediction vectors of the non-members (as measured by `TestVar`) has a large causal effect on the average MI attack accuracy. This validates the prior work hypothesis (Q2) that the differences in the shadow and target model affect the MI attack. For instance, for CE, the estimated ATE of `TrainVar` on `ShadowAcc` is 0.02, whereas that of `TestVar` is 0.94 (Table 4.3). We find that for the multiple shadow model attack on MSE-trained models, the number of parameters (Q3) does not show a significant influence on the `ShadowAcc`, but for all other attacks and loss functions, the more parameters, the higher the attack accuracy. We thus validate (Q3) as for all evaluated attacks there is a non-zero ATE value. We find that the variance of the outputs of the models is also a cause for the single shadow model attack, to various degrees depending on the type of attack. Prior work overlooks the differences in the prediction vectors between the target and shadow model. Thus, our analysis shows that (Q5) is refuted. The variance of the training algorithm influences MI attacks that take the whole prediction vector more–models tend to agree more on top-3 predictions rather than the whole prediction vector. Thus, the evaluated shadow model MI attacks that take the top-3 predictions are not as sensitive to differences between the shadow and target models' architecture and dataset. This shows that there are differences between the variants of the single shadow model attack, refuting (Q7). More details for each attack are available in Appendix A.1.

We find that a small fraction of queries have low statistical significance (p-value > 0.05)—4/16 MSE-trained models and 1/16 for CE-trained models. We do not draw any conclusions for these.

4.6.4 MI attacks and Generalization

We find that **Bias** and **Variance** values have a high level of influence on both generalization measures as well as the MI attack accuracy. As expected from the bias-variance decomposition theorem, Bias and Variance values are strongly predictive of `TestAcc`, `AccDiff`, and `CentroidDist` values—all of these are generalization measures. Bias and Variance also have a disproportionately high influence on MI

attack accuracy. Appendix C gives details; here we summarize their effect on MI attack accuracy which varies by attack.

Variance & MI. The higher the variance on non-members, the higher the MI attack accuracy. The reverse is true for members: the higher the variance on members, the lower the MI attack accuracy. There is less variance on the training samples (i.e., the model learns similar prediction vectors across multiple training datasets) and there is higher variance on the test samples. Our analysis show that the larger the gap between these two, the better the MI attack accuracy. This suggests that defenses which decrease the gap between the train and test variance (like MemGuard [106]) will be effective.

Bias & MI. The Bias on non-members is almost always a factor in all types of MI attacks we study. It affects MI attack accuracy through the test accuracy, which in turn affects the train-to-test accuracy gap or the distance between members and non-members. Recall that Bias is "how far" the test set predictions are from the ground truth on average. High Bias on non-members explains why MI attacks, even when not explicitly using the label information, will have a better accuracy. On members, however, the ATE value of the Bias in most cases is close to 0, i.e., it has almost no effect on MI attack accuracy—this corresponds to networks closely fitting the training set. Compared to Variance, the Bias has a larger effect, and even more so when the input features to the attack model are the top-3 predicted labels and not the whole prediction vector.

Why do larger models leak information even if their test loss decreases? Complex interplay of loss, variance, bias and the model size have been observed previously under different regimes (Fig.1 in [251]). When the Bias dominates, it and the testing loss decrease with increase in model size of the network, but the Variance does not linearly go down and exhibits a peak (bell-shaped curve). Our analysis shows that Variance *by itself* contributes to the MI attack—despite training larger models with lower loss and Bias, the Variance can improve MI attack accuracy (see Variance ATE, Appendix A.1).

How does CE loss differ from MSE? Unlike MSE, for CE we observe that the Variance dominates the loss term. The loss and the Variance exhibit a high peak as the model size increases, while the Bias keeps decreasing (unimodal curve). This explains why for CE models the Variance has a much larger impact on MI attack

accuracy. After Variance peaks, as the model size increases, the loss drops to where both Bias and Variance are low. This explains why increasing model size can reduce MI attack accuracy beyond a point.

4.6.5 Utility in Explaining Defenses

We analyze the causal models for all 6 attacks for L2-regularization. For MemGuard, we evaluate the single shadow model (top-3) attack on the defended models as done in the original work. Thus we have $2 + 12 = 14$ causal graphs in total. For each case, we analyze how much the ATE for a cause differs from ATE in the corresponding causal graphs for *undefended* ML model.

L2-regularization Setup. The first type of defense requires a simple change to one of the parameters of the training algorithm, i.e., the weight decay. The rest of the training procedure is the same, resulting in the same number of models with and without regularization. We run all of the attacks on the regularized models to evaluate how the defense changes the effect of certain factors on the MI attack accuracy.

MemGuard Setup. The MemGuard defense requires in total 4 models:

- Target Model: the model to be defended.
- Defense Model: the attack model trained by defenders. The model is trained with the training set which considers the training set of the target model as members and the testing set of the target model as non-members.
- Shadow Model: the model trained by attackers which has the same architecture as Target Model but is trained with a different dataset
- Attack Model: the attack model trained by attackers. The model is trained with the training set which considers the training set of the shadow model as members and the testing set of the shadow model as non-members. Note that the non-members used to train the attack model need to be different from the non-members used to train the defense model.

For our evaluation, we defend 1 target model per repeat (on average) using MemGuard, using 2 other models in the same repeat, which gives us 2 defended

models per target model. In total, we generate 6 new defended models that have almost the same accuracy as the corresponding target model. We evaluate these 6 models on the testing samples, and compute bias-variance of the outputs of these 6 models. The resulting graphs are also computed with these updated bias-variance quantities.

Defense through L2-regularization. We find that L2-regularization alleviates some causes but not all, and not in equal measure for all attacks. For instance, it majorly reduces the ATE of the test variance on all of the evaluated MI attacks–even as drastically as from 0.84 to 0.16 for `MLLeakAcc-l`. The effect of the train-to-test accuracy gap on the `ShadowAcc` remains the same, but it increases for the single shadow model attacks. The ATE of the `CentroidDist` does not change after regularization is applied, showing that there are still exploitable signals left. As a defense reduces the influence that the variance has on the MI attack for the multiple shadow model attack. For all attacks, however, variance remains a cause for the MI attack. For `MLLeakAcc-l`, the `TrainVar` on regularized models has a negative effect on the attack performance, i.e., the higher the variance, the lower the attack, which has decreased from -0.23 to -0.38 (more negative effect). In contrast, the `TestVar` is positive and reduces from 0.83 to 0.15. We also find this for `MLLeakTop3Acc` where the variance on the non-members has an estimated ATE of 0.77 and decreases to 0.11 (Table 4.4). Regularization does not remove the causal relationship between the main causes of the attack prior to applying this defense. For MSE-trained models, the effect of the cause `LossDiff` is significantly reduced for the single shadow model attack using top-3 predictions (Table 4.5). In fact, the regularization appears to be quite effective for this attack. The features pertaining to training size and model complexity remain causes for the attack. These have a similar influence on the MI attack accuracy even after applying the high weight decay training.

Defense through MemGuard. MemGuard reduces the variance for both CE and MSE models, as well as some of the causes, being more effective than L2-regularization in removing the variance effect of the members. MemGuard is more effect on MSE models, as models of the usual signals have been decreased. The effect of Bias on non-members for CE models remains a potential signal, along with `CentroidDist`. The MemGuard defense reduces the Variance significantly

but many factors remain unaddressed even after the defense. For instance, the distance between members remains a factor. The ATE of the `AccDiff` is reduced from 0.19 to 0.08. MemGuard is more effective overall in removing causes than regularization. The attack accuracy `MLLeakAcc-l` is not influenced by the distance between members and non-members (`CentroidDist`) after regularization. This is visible in the graph itself, i.e., the edge is missing in the ETIO graph in Fig. A.2b compared to Fig. A.6b.

4.7 Related Work

Generalization. Generalization in machine learning is a fundamental topic. Several studies investigate the bias-variance decomposition in neural networks [161, 251, 69]. Yang et al. [251] explore the dependence of bias and variance to network width and depth, e.g., deeper models tend to have lower bias but higher variance. Our work connects MI attacks to such training and architectural choices. Other works propose new measures of generalization [47, 108].

Membership Inference Attacks. There has been a recent line of work proposing MI attacks and providing useful attack taxonomy. Shokri et al. [207] present the first membership inference attack. They show that overfitting is correlated with their attack performance. They suggest that besides overfitting, the structure and type of the model also contribute to the privacy leakage through membership inference attacks. Several new attacks have emerged [142, 155, 87, 152, 135, 140, 260, 34, 132] and attack taxonomies [234, 133] have started to categorize them. These attacks serve as tools to evaluate the privacy risk of machine learning models through attack procedures. Our work distinguishes itself from all of these by providing a causal framework to explain why these attacks arise. Notably, our work provides a new lens into how generalization and MI attacks connect—through a systematic measurement and reasoning of bias, variance, and other stochastic variables that arise in training.

Several works have provided mechanistic explanations connecting MI attacks to generalization prior to our work. Yeom et al. [254] provide a theoretical connection between a notion of generalization called the average generalization error and a bounded-loss adversary which does not apply to training using a CE loss. They also propose a threshold-based attack which has knowledge of the loss distribution

which we have also evaluated in our framework Etio. The attack assumes that the loss is normally distributed, and thus can be connected to the adversary advantage in a closed-form expression. Our work shows that the assumptions made in their work may not always hold. We show that MI attack performance is linked to the average generalization error for models with MSE loss but does not always for CE loss. Subsequent work by Song et al. [217] propose a similar threshold-based attack, but on the confidences of the prediction. Nasr et al. [159] also analyze the connection between membership inference attacks and overfitting, while proposing white-box membership inference attacks. They also empirically observe the correlation of the attack performance to the model capacity. Song et al. [216] evaluate membership inference attacks against adversarially robust models and point out that these models have a larger train-to-test accuracy gap when considering adversarial examples. Our work shows that other factors beyond the train-to-test accuracy gap contribute to the privacy leakage.

Causality. Causality is an active area of research with recent advances improving learning of causal models [113, 202], as well as better inference procedures [101, 131]. While extensively applied in sciences [233, 74, 88, 147, 17], causality has only been recently connected to privacy [235, 230]. In our work, we introduce the causal lens to understand MI and generalization. Since our proposed methodology is synergistic, combining learning with domain knowledge, we can benefit from such advances to improve our causal models and analysis. In addition to learning and inference, methods to test the causal assumptions have also been proposed such as sensitivity analysis [190, 188] and simulated dataset-approach [160]. Again, our approach can leverage such tests for the constructed causal models.

4.8 Summary

We have proposed the first use of causal graphs to capture how stochastic factors—such as bias, variance, model size, data set size, loss values, and so on—causally interact to give rise to MI attacks, providing a new connection between these attacks and generalization. We hope this framework helps formally re-analyze statistical conclusions and pinpoint root causes more accurately.

4.9 Future Work

We have used causality to analyze hypotheses about privacy tests and their relationship to generalization. One of the challenges in this approach is how to reduce the human effort of modeling the variables that appear in the causal graphs: could we learn causal features from data about the learning and attack procedures? On the other hand, would the causal graphs derived without human modeling be trustworthy and useful? The best way to put to the test our insights from this analysis is in designing better defenses and better evaluations. If we know that certain factors contribute to a vulnerability or an undesirable outcome, future work should consider designing new training algorithms that target reducing these detrimental factors and improving those factors that help mitigate attacks.

Table 4.4. We compute the average effect on the 6 evaluated MI attacks of the causes mentioned in prior works for CE-trained models with L2-regularization.

Attack	Feature	ATE	p-value
MLLeakAcc	AccDiff	0.2314	5.78E-84
MLLeakAcc	CentroidDist	0.0000	0.00E+00
MLLeakAcc	LossDiff	0.0000	0.00E+00
MLLeakAcc	NumParams	0.1673	2.08E-07
MLLeakAcc	TestBias	-75.6790	2.97E-01
MLLeakAcc	TestVar	0.1425	3.66E-06
MLLeakAcc	TrainBias	0.0000	0.00E+00
MLLeakAcc	TrainSize	-0.1267	2.62E-12
MLLeakAcc	TrainVar	-0.3409	3.72E-05
MLLeakAcc-l	AccDiff	0.2833	2.42E-78
MLLeakAcc-l	CentroidDist	0.0000	0.00E+00
MLLeakAcc-l	LossDiff	1.2445	5.12E-03
MLLeakAcc-l	NumParams	0.1859	7.27E-09
MLLeakAcc-l	TestBias	5.5391	9.45E-01
MLLeakAcc-l	TestVar	0.1580	1.09E-06
MLLeakAcc-l	TrainBias	0.0000	0.00E+00
MLLeakAcc-l	TrainSize	-0.1428	9.59E-14
MLLeakAcc-l	TrainVar	-0.3831	2.80E-06
ShadowAcc	AccDiff	0.2873	4.69E-178
ShadowAcc	LossDiff	0.0000	0.00E+00
ShadowAcc	NumParams	0.2253	1.21E-10
ShadowAcc	TestBias	65.4845	3.24E-01
ShadowAcc	TestVar	0.4880	2.39E-02
ShadowAcc	TrainBias	0.0000	0.00E+00
ShadowAcc	TrainSize	-0.1626	3.07E-21
ShadowAcc	TrainVar	-0.0402	1.28E-07
ThreshAcc	AccDiff	0.2294	5.86E-125
ThreshAcc	LossDiff	0.0000	0.00E+00
ThreshAcc	NumParams	0.2045	6.11E-12
ThreshAcc	TestBias	107.4913	1.16E-01
ThreshAcc	TestVar	0.6449	2.03E-01
ThreshAcc	TrainBias	0.0000	0.00E+00
ThreshAcc	TrainSize	-0.1223	3.35E-18
ThreshAcc	TrainVar	-0.0734	5.86E-05
MLLeakTop3Acc	AccDiff	0.2288	6.95E-85
MLLeakTop3Acc	CentroidDist	0.2446	5.14E-33
MLLeakTop3Acc	LossDiff	0.0000	0.00E+00
MLLeakTop3Acc	NumParams	0.1662	1.14E-07
MLLeakTop3Acc	TestBias	-45.7867	5.31E-01
MLLeakTop3Acc	TestVar	0.1115	1.11E-06
MLLeakTop3Acc	TrainBias	0.0000	0.00E+00
MLLeakTop3Acc	TrainSize	-0.1322	4.51E-13
MLLeakTop3Acc	TrainVar	-0.3589	1.20E-05
MLLeakTop3Acc-l	AccDiff	0.2240	3.54E-77
MLLeakTop3Acc-l	CentroidDist	0.2661	7.45E-32
MLLeakTop3Acc-l	LossDiff	0.0000	0.00E+00
MLLeakTop3Acc-l	NumParams	0.1650	1.12E-07
MLLeakTop3Acc-l	TestBias	-58.3315	4.38E-01
MLLeakTop3Acc-l	TestVar	0.0932	3.47E-07
MLLeakTop3Acc-l	TrainBias	0.0000	0.00E+00
MLLeakTop3Acc-l	TrainSize	-0.1298	3.26E-12
MLLeakTop3Acc-l	TrainVar	-0.3976	2.50E-05

Table 4.5. We compute the average effect on the 6 evaluated attacks of each features over the MSE-trained models with L2-regularization.

Attack	Feature	ATE	p-value
MLLeakAcc	AccDiff	-0.0951	1.81E-02
MLLeakAcc	CentroidDist	0.0000	0.00E+00
MLLeakAcc	LossDiff	0.5262	4.01E-03
MLLeakAcc	NumParams	0.0313	7.18E-01
MLLeakAcc	TestBias	0.1261	6.01E-05
MLLeakAcc	TestVar	0.1238	3.73E-07
MLLeakAcc	TrainBias	0.0000	0.00E+00
MLLeakAcc	TrainSize	-0.0199	4.07E-03
MLLeakAcc	TrainVar	0.1572	1.77E-03
MLLeakAcc-l	AccDiff	0.0732	3.91E-16
MLLeakAcc-l	CentroidDist	0.0000	0.00E+00
MLLeakAcc-l	LossDiff	0.6361	4.58E-05
MLLeakAcc-l	NumParams	0.0529	4.22E-01
MLLeakAcc-l	TestBias	0.1619	8.45E-07
MLLeakAcc-l	TestVar	0.1690	3.97E-04
MLLeakAcc-l	TrainBias	0.0000	0.00E+00
MLLeakAcc-l	TrainSize	-0.0292	1.62E-02
MLLeakAcc-l	TrainVar	0.2790	5.77E-05
ShadowAcc	AccDiff	0.0000	0.00E+00
ShadowAcc	LossDiff	0.2941	1.80E-91
ShadowAcc	NumParams	0.0499	8.78E-01
ShadowAcc	TestBias	0.0000	0.00E+00
ShadowAcc	TestVar	0.0000	0.00E+00
ShadowAcc	TrainBias	0.0000	0.00E+00
ShadowAcc	TrainSize	-0.0382	7.07E-01
ShadowAcc	TrainVar	0.4038	1.75E-29
ThreshAcc	AccDiff	0.2069	5.91E-129
ThreshAcc	LossDiff	0.0000	0.00E+00
ThreshAcc	NumParams	0.0692	6.62E-01
ThreshAcc	TestBias	-0.2423	5.81E-10
ThreshAcc	TestVar	0.1398	2.55E-01
ThreshAcc	TrainBias	0.0000	0.00E+00
ThreshAcc	TrainSize	-0.0396	1.64E-02
ThreshAcc	TrainVar	0.2585	3.84E-03
MLLeakTop3Acc	AccDiff	0.0000	0.00E+00
MLLeakTop3Acc	CentroidDist	0.0000	0.00E+00
MLLeakTop3Acc	LossDiff	0.2561	4.76E-40
MLLeakTop3Acc	NumParams	0.0363	5.62E-01
MLLeakTop3Acc	TestBias	0.0000	0.00E+00
MLLeakTop3Acc	TestVar	0.0223	4.19E-06
MLLeakTop3Acc	TrainBias	0.0000	0.00E+00
MLLeakTop3Acc	TrainSize	-0.0319	1.27E-03
MLLeakTop3Acc	TrainVar	0.2424	5.69E-02
MLLeakTop3Acc-l	AccDiff	0.0000	0.00E+00
MLLeakTop3Acc-l	CentroidDist	0.0000	0.00E+00
MLLeakTop3Acc-l	LossDiff	0.1804	6.18E-79
MLLeakTop3Acc-l	NumParams	0.0014	9.38E-01
MLLeakTop3Acc-l	TestBias	0.0000	0.00E+00
MLLeakTop3Acc-l	TestVar	0.0000	0.00E+00
MLLeakTop3Acc-l	TrainBias	0.0000	0.00E+00
MLLeakTop3Acc-l	TrainSize	-0.0261	1.50E-03
MLLeakTop3Acc-l	TrainVar	0.1858	4.47E-19

Chapter 5

Precise Definitions with Decisional Convergence Queries

5.1 Introduction

Decisional queries that have been studied predominantly in prior work aim to verify that a property holds where that property is defined with respect to an input to a machine learning model, and an approximation parameter ϵ. In this chapter, we introduce convergence queries, i.e., when $\epsilon \to 0$, to check an emerging property of stochastic gradient descent.

Stochastic gradient descent (SGD) has been the de-facto training algorithm for neural networks. Its intrinsic security properties are therefore important to enunciate precisely. One fundamental property of SGD is *forgeability*: Is it possible to obtain the same model parameters (outputs) from two different minibatches (inputs)? If yes, then we say that the output is forgeable. Forgeability has emerged in the context of several applications such as machine unlearning [226], model ownership [105, 261, 53], and membership inference tests [116, 117]. If a model is forgeable, certain training samples used could have been replaced with other samples without changing the output. It can be argued counterfactually that these samples were never utilized in the first place, since there exist others that can replace them without a change in output. Thus, forgeability provides a way to unlearn some data samples seen in training. On the other hand, if a model is unforgeable, training samples are irreplaceable in creating the final model. It has been suggested that knowing the specific samples used is information that can be used to claim ownership of the model [105, 261, 53]. Many such security applications naturally arise from the basic

property of forgeability.

Despite its emerging applications, characterizing forgery in practice has remained an intriguing open problem. Creating exact forgery of model checkpoints has not been demonstrated yet. Prior work has shown that it is possible to forge intermediate model parameters within certain error (under a vector norm) and conjectured that forgery could be made exact with zero error, but this remains a conjecture hitherto [226, 117, 116]. Similarly, we are not aware of any general conditions over data distributions seen in practice under which models are provably unforgeable. Therefore, exact forgery remains an important property to define and study.

We can model SGD as a deterministic procedure by fixing the training dataset, the initial model parameters, and all other training hyperparameters in advance. The randomization seeds for sampling minibatches from the training dataset can be treated as the inputs to the SGD procedure. Each input results in an *execution trace*: a sequence of intermediate model parameters obtained after each minibatch is used for training. The output is the final model. We can look at forgery from the lens of forging execution traces rather than outputs. In this chapter, we formally define the notion of *exact forgery* of states or checkpoints in an SGD execution trace. It asks whether two different inputs produce the same execution state. If a state is forgeable, then under two different inputs the next intermediate model state obtained is exactly the same. This will imply that the inputs effectively "collide" and the entire execution trace will be the same when collisions occur. It is easy to see that the resulting output is also forgeable if a trace is forgeable. The converse is not true: Unforgeability of one trace does not rule out the existence of another trace that produces the same final output. We are only interested in one-step forgeability of checkpoints in this work. Extending definitions that consider multiple steps or traces to forge final outputs is promising future work.

Our main contribution is to present the first systematic characterization of exact forgeability of trace checkpoints. Our central theorem states that if certain conditions hold at an intermediate checkpoint of the execution trace, then it is *unforgeable* at that step. If an execution trace is unforgeable for at least one training step, then the whole execution trace is also unforgeable. A verifier with access to the whole execution trace can replay the training and check that there is one unforgeable

training step. The conditions of our theorem are testable on concrete executions and we devise efficient procedures to check them at any given intermediate checkpoint. Our checks scale well with increasing neural network parameters, taking about 21 minutes per checkpoint on average for networks with millions of parameters. The conditions define a specific regime of data distributions under which unforgeability holds, but these conditions are mild enough in practice that they are satisfied in all checkpoints sampled in our experimental evaluation. Our work provides the first regime where training checkpoints are algebraically unforgeable, partially answering the conjecture about when exact forgery might, if at all, be feasible in practice.

The implications of showing unforgeability go beyond experimental evidence. Our goal is to point out the lack of formal definitions, without which contradictory interpretations arise from the same experimental setups. For instance, our results are in sharp contrast to prior work, showing that exact forgery is impossible on the *same experimental setup* [226, 117]. This contrast arises because prior work considers approximately equal (or close) intermediate states as sufficient to define forgery, unlike our work. We empirically confirmed that when we replace an intermediate model state with the "approximately" same state obtained by procedures suggested in prior work, the final output of the execution is *not* the same as the original. Therefore, it is possible to observe the difference in outputs for white-box distinguishers. While approximate forgery may suffice to deceive some algorithms that distinguish output models, say via black-box testing of models, it is not sufficient to rule out all such distinguishing algorithms. Exact forgery deals with the most powerful of distinguishing algorithms and is, therefore, useful in rigorous security arguments. Our result shows that the difference between the exact and approximate case is significant empirically.

Contribution We present the first theoretical impossibility result for exact one-step forgery of SGD execution states. Our theorem specifies conditions under which traces are provably unforgeable, which are efficiently testable on concrete executions, given the training dataset and model parameters. Our results on exact forgery directly contrast those in prior work which use approximate forgery.

5.2 Problem

The property of forgery has been the basis for several recent security applications. For concreteness, we describe one such application, i.e., *data non-repudiation* in standard neural network training.

5.2.1 Motivating Application

Several lawsuits have been filed against machine learning (ML) companies claiming that part of the training data infringed copyright [36, 181]. From a technical standpoint, how can an entity prove that their data point has been used in the training process that resulted in a given ML model? Or, conversely, what information should the ML model provider release in order to prove that they have trained using a particular dataset? At large, these questions revolve around *data non-repudiation* in training ML models.

The answer to these questions is not immediate. Let us consider the following scenario: given a training dataset, the ML model provider trains a model using stochastic gradient descent (SGD). The ML provider wants to release the necessary information to reproduce their training such that an honest verifier can independently check data non-repudiation claims. Prior work on proof-of-learning or proof-of-unlearning logs [105, 226] have envisioned similar motivating applications that facilitate auditing the integrity of the training procedure. For data non-repudiation, the information that is released to the verifier is the same as prior works [105, 226]. Specifically, the ML provider maintains a training log which consists of the data samples used in each minibatch and the model parameters at every training step (checkpoint). The initialized state of the model, all the training hyperparameters, and any other sources needed to replay the execution are maintained in the log as well. The verifier can check the validity of the logged information at a later point in time, by reproducing the model parameters at the t^{th} training step using the $(t-1)^{th}$ checkpoint and the minibatch information from the log. Thus, one can check whether specific data samples were used to train the model. If the computation is done on the same hardware and software stack there should be no difference between the recomputed state and the one in the log, modulo numerical instability and hardware implementation differences [180]. We can abstract away these sources

of non-determinism and revisit their role in Section 5.6.4.

Using such training logs as proofs for training integrity with a given dataset is problematic [261, 53]. One fundamental issue is that there might exist *forged* gradient updates: Given a state of the model parameters at some training step, there exists an alternative minibatch that produces the same model parameters as those of the original minibatch. For data repudiation, forging at one training step is not enough. The adversary has to be able to forge all training steps where the repudiated sample has been used in order to forge the entire trace released to the honest verifier.

5.2.2 Definition of the Forgery Game

Our training and forgery setup characterizes precisely the questions raised in several several prior applications [105, 226, 117]. We define forgery as a game, extending the game framework proposed by Salem et al. [195]. We illustrate the game in Algorithm 5. In the forgery game, there is a verifier ($\mathcal{V}$) that simulates the ML system made up of the training pipeline specifying the training algorithm ($\mathcal{T}$), the data distribution ($\mathcal{D}$) and the training dataset (D). The verifier challenges the adversary ($\mathcal{A}$) to the forgery game by asking it to produce a forgery for a randomly chosen state of model parameters using the training algorithm $\mathcal{T}$. There are three sequential phases in the definition of the game.

Setup Phase. The game starts with a setup. The training dataset D consists of samples $\{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^{m}$ where $\boldsymbol{x}_i \in \mathbb{R}^{in}$ and $\boldsymbol{y}_i \in \{0,1\}^{out}$, (x_i, y_i) sampled from a data distribution $\mathcal{D}$. The dataset D has m data samples. The dataset sampling process is not controlled by either the verifier or the adversary. All hyperparameters of the training algorithm such as the learning rate (γ), the model architecture, batch size (k), loss function (l), and so on, are fixed during setup as well. The initial model parameters (θ_0) are sampled from a well-defined probability distribution over real vectors of dimension n, and a number of training steps T is also given. The verifier then chooses a minibatch at each step t of the training at random from D. The size of each minibatch is a fixed constant k. The choice of minibatches is captured by a vector $\boldsymbol{r}$ which has T elements, one for each training step $t \in \{0, \ldots, T\}$. Each element of $\boldsymbol{r}$ is a bitstring chosen uniformly at random from $\{0,1\}^m$ with exactly k

1s in it. A 1 in the i^{th} bit location means that the i^{th} sample in D is selected in the minibatch and 0 means it is excluded. Therefore, the minibatch used the gradient update at a step t is completely determined by the bitstring $\boldsymbol{r}[t]$.

Tracing Phase. The verifier runs the training algorithm for T training steps to obtain an execution trace E. The training algorithm is instantiated with the widely used stochastic gradient descent. The training algorithm takes as input an initial state of model parameters $\theta_0 \in \mathbb{R}^n$, a training dataset D sampled from the data distribution $\mathcal{D}$, number of training steps T, and vector of selector bitstring $\boldsymbol{r}$ created during setup. It finally learns a model $f_{\theta_T} : \mathbb{R}^{in} \rightarrow \{0,1\}^{out}$ that minimizes the loss l on the training dataset. At a given training step $t < T$, the training algorithm picks the minibatch $\boldsymbol{b}_t$ of size k $\{(\boldsymbol{x}_1, \boldsymbol{y}_1), \ldots, (\boldsymbol{x}_k, \boldsymbol{y}_k)\}$ from the training dataset D according to the vector $\boldsymbol{r}[t]$. The training algorithm then performs one step of gradient descent that updates the parameters by minimizing the loss l as follows: $\theta_{t+1} = \theta_t - \gamma \frac{1}{k} \sum_{i=1}^{k} \nabla_{\theta_t} l(f_{\theta_t}(\boldsymbol{x}_i), \boldsymbol{y}_i)$. It continues updating the model parameters until T training steps and returns the model parameters for all training steps $E = \{\theta_0, \ldots, \theta_T\}$, which we call an *execution trace*. The training algorithm for a fixed $\boldsymbol{r}$, hyperparameters, and training dataset D is completely deterministic. An execution trace E depends only on the input $\boldsymbol{r}$ to $\mathcal{T}$ as it determines the sampled minibatches at each step. We thus call $\boldsymbol{r}$ as the input to $\mathcal{T}$ for the purpose of the forgery game.

Forgery Phase. Once the verifier has obtained the execution trace, it challenges the adversary with a randomly chosen checkpoint $t \in \{1, \ldots, T\}$. The adversary has access to everything in E and wins the *checkpoint forgery* game if it outputs a minibatch $\hat{\boldsymbol{b}}_t \neq \boldsymbol{b}_t$ such that the next model parameter state obtained is the same as in the trace. Specifically, the adversary is asked to output some $\hat{\boldsymbol{b}}_t \neq \boldsymbol{b}_t$, such that for the given $\theta_t \sim E$, the training algorithm T produces $\hat{\theta}_{t+1}$ and that $\theta_{t+1} = \hat{\theta}_{t+1}$. The adversary's advantage is the probability of winning over random choices of t.

This game definition encompasses previous forgery-related attacks [209, 226, 116] when the adversary $\mathcal{A}$ can (1) interact with the ML system by intercepting the minibatch samples used to obtain the model parameters θ_{t+1} (they know the samples used for training) and (2) substitute the minibatch for a given checkpoint from the training D at a chosen checkpoint θ_t with a different minibatch.

Algorithm 5 The one-step forgery game.

Input: Training algorithm $\mathcal{T}$, Training dataset $D = \{(\boldsymbol{x}_i, \boldsymbol{y}_i)_{i=1}^m\}$ sampled from a data distribution $D \sim \mathcal{D}^m$, Verifier $\mathcal{V}$, Adversary $\mathcal{A}$, Number of training steps T, Initial model parameters θ_0

1: $\mathcal{V}$ chooses random indices $\boldsymbol{r} \in \{0,1\}^{T \times m}$
2: $\{\theta_0, \theta_1, \ldots, \theta_T\} \leftarrow \mathcal{T}(\theta_0, D, T, \boldsymbol{r})$
3: $\mathcal{V}$ releases to $\mathcal{A}$ $E, \boldsymbol{r}$
4: $\mathcal{A}$ chooses $\theta_t \sim \{\theta_0, \ldots, \theta_{T-1}\}$
5: $\hat{\theta}_{t+1}, \hat{\boldsymbol{b}}_t = \{(\hat{\boldsymbol{x}}_1, \hat{\boldsymbol{y}}_1), \ldots, \} \leftarrow \mathcal{A}(D, \theta_t, \boldsymbol{b}_t, \mathcal{T})$
6: $\mathcal{V}$ accepts if $\theta_{t+1} = \hat{\theta}_{t+1} \wedge \hat{\boldsymbol{b}}_t \neq \boldsymbol{b}_t$

Here we study the *existence* of forgery under a given model checkpoint and training dataset. We are interested in showing that when certain conditions are met on a given checkpoint, the adversary has probability *zero* of winning the game at that checkpoint.

5.2.3 Problem Statement

We have defined the existence of forgery under a given model checkpoint θ_t and training dataset. This implies that the gradient update rule in the training algorithm is computed with respect to the same state of model parameters θ_t but on a different set of samples corresponding to $\hat{\boldsymbol{b}}_t = \{(\hat{\boldsymbol{x}}_1, \hat{\boldsymbol{y}}_1), \ldots, (\hat{\boldsymbol{x}}_k, \hat{\boldsymbol{y}}_k)\}$.

A forgery is possible if $\theta_{t+1} = \hat{\theta}_{t+1}$ which implies

$$\frac{\gamma}{k} \sum_{i=1}^{k} \nabla_{\theta_t} l(f_{\theta_t}(\boldsymbol{x}_i), \boldsymbol{y}_i) = \frac{\gamma}{k} \sum_{i=1}^{k} \nabla_{\theta_t} l(f_{\theta_t}(\hat{\boldsymbol{x}}_i), \hat{\boldsymbol{y}}_i) \tag{5.1}$$

where $\{(\boldsymbol{x}_1, \boldsymbol{y}_1), \ldots, (\boldsymbol{x}_k, \boldsymbol{y}_k)\} \neq \{(\hat{\boldsymbol{x}}_1, \hat{\boldsymbol{y}}_1), \ldots, (\hat{\boldsymbol{x}}_k, \hat{\boldsymbol{y}}_k)\}$.

The above equation can be simplified since the learning rate γ and batch size k are the same for the forged and original batch. Note that we can compute the gradients of all of the samples in the dataset D with respect to the checkpoint θ_t. We denote $g_i = \nabla_{\theta_t} l(f_{\theta_t}(\boldsymbol{x}_i), \boldsymbol{y}_i)$ as a gradient computed for the i^{th} data point in D. The two minibatches $\boldsymbol{b}_t$ (original) and $\hat{\boldsymbol{b}}_t$ (forged) are different. Hence, their corresponding bitstrings that determine which samples are selected from the dataset at a training step are $\hat{\boldsymbol{r}}[t] \neq \boldsymbol{r}[t]$. We henceforth drop the script t where clear from context, e.g., $\boldsymbol{r}[t] = [r_1, \ldots, r_m], r_i \in \{0,1\}$. We can rewrite Equation (5.1) as

follows:

$$\sum_{i=1}^{m} r_i \boldsymbol{g}_i = \sum_{i=1}^{m} \hat{r}_i \boldsymbol{g}_i, \tag{5.2}$$

where $(r_1, \ldots, r_m) \neq (\hat{r}_1, \ldots, \hat{r}_m)$ and $\sum_i r_i = \sum_i \hat{r}_i = k$.

Equation (5.2) is further simplified as

$$\sum_{i=1}^{m} z_i g_i = 0, \tag{5.3}$$

where the coefficients $z_i = r_i - \hat{r}_i \in \{-1, 0, 1\}$. Each gradient vector has n dimensions, $\boldsymbol{g}_i \in \mathbb{R}^n$. Using standard matrix notation, we can write the gradients as columns of a matrix $G \in \mathbb{R}^{n \times m}$ as $G = [\boldsymbol{g}_1 | \ldots | \boldsymbol{g}_m]$, and $\boldsymbol{z} = (z_1, \ldots, z_m)^T, \boldsymbol{z} \in \{-1, 0, 1\}^m$ to write Equation (5.3) as:

$$G\boldsymbol{z} = 0 \tag{5.4}$$

In this thesis, we study the question of existence of an assignment to z_is that satisfy the equations above.

5.3 Overview

We have mathematically defined the problem of forgeries as finding whether a system of equations such as (5.4) has solutions. It is worth asking though what algebraic properties we require to have for the problem of forgery to be well-defined and have intuitive semantics.

5.3.1 When is forgery well-defined?

In Equation (5.4), one would like vector addition to be commutative and associative, as otherwise unexpectedly the order of the summation of terms may matter. One can record the order of the summation along with what elements are selected in the batch at a training time. This will introduce multiple counter-intuitive issues. For instance, the definition of forgery will need to state the order of the terms in the sum. The summation operation is often paralellized using vector instructions sets supported by hardware, for which ordering can be unpredictable at runtime. Further, even trivial forgeries produced by reordering of gradients within the same minibatch may become possible. Therefore, a mathematical definition as in Equation (5.4)

would require that the element-wise addition of the gradient vectors forms an *abelian group*.

Floating-point addition is not associative [75]. For instance, in 64-bit floating-point precision $(\frac{1}{3}+\frac{1}{3})+\frac{1}{4} \neq \frac{1}{3}+(\frac{1}{3}+\frac{1}{4})$, as the two sums differ by a small rounding *error* $\epsilon \approx 10^{-15}$. In light of this, one can readily see that floating-point numbers with addition do *not* define an abelian group and, therefore, exact forgery is not well-defined therein. Our results, therefore, concern themselves with exact forgery defined over fixed-point numbers only.

A different approach that prior work has taken to circumvent this issue is to consider non-exact, so-called *approximate* forgeries. Approximate forgeries satisfy the forgery equation (5.2) only with some precision δ, i.e., $||\sum_{i=1}^{m} r_i \boldsymbol{g}_i - \sum_{i=1}^{m} \hat{r}_i \boldsymbol{g}_i|| < \delta$, where $||\cdot||$ is some vector norm. Prior works have studied such forgeries in the context of unlearning training data [226], as attacks to slow down training convergence or hurt the performance of training [209], and in membership inference attack repudiation [116]. However, this loose definition of forgery lacks any concrete basis. As the training process of a neural network is fully deterministic (for a fixed seed), an honest verifier has no motivation to accept that two minibatches produce the same gradient updates, unless the updates coincide *on all bits*. A verifier can simply reject approximate forgery, unless the updates are exactly the same. Another rationale for accepting approximate forgeries is that they may have originated from minor hardware, library discrepancies, or that they are sufficient for the application context. However, in Section 5.6.4 we show that even when δ is tiny (e.g., as the above rounding error, $\delta = \epsilon$), subsequent training steps will significantly expand it, resulting in clearly observable differences in model parameters. Therefore, it is futile to consider approximate forgeries at a single intermediate checkpoint.

5.3.2 Challenges

Our work provides the first proofs of unforgeability. Before presenting our approach, we present promising approaches we considered and explain why they do not serve our purpose.

Collisions by chance: A natural question is whether collisions arise by chance in the gradient updates using SGD. Notice that the gradients are high-dimensional

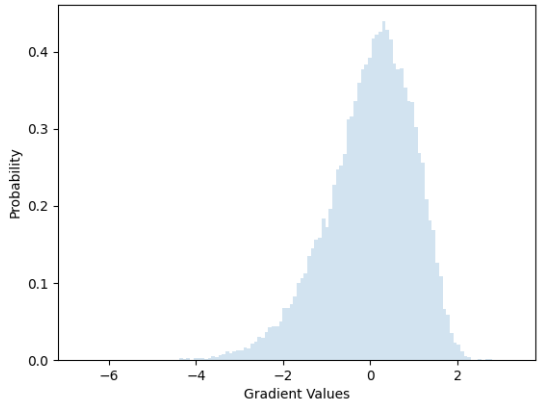


Figure 5.1. Gradient distribution of a parameter in LeNet5 [126] with respect to the samples in the training dataset MNIST [125] where the gradients are `log(abs(grad))`. The distribution "looks" close to a lognormal distribution as suggested in recent work [33]. But it is far enough from a lognormal distribution as that a standard Kolmogorov-Smirnoff [148] test fails.

vectors, as large as the number of parameters in the model (e.g., LeNet5 [126] has $61,706$ parameters). If we could argue that every gradient vector has high entropy $e \gg m$, given any fixed sum of the other gradient vectors, then the probability that Equation (5.4) is satisfied for some choice of $\mathbf{z}$ is at most $\frac{\binom{m}{k}^2}{2^e}$, which is negligible. It would be possible to argue this if the gradient distributions could be modelled as closed-form probability distributions. However, this is not the case in practice. Previous results [33, 253] as well as our own experiments reveal that the dimensions of the gradient vector definitely have sufficient entropy. The gradient distribution is similar to Laplace or lognormal, but not exactly the same. We illustrate the distribution of the gradient on one dimension at one particular checkpoint for LeNet5 across the MNIST dataset in Figure 5.1. We find that the distribution "looks" close to a lognormal, but it fails to be one as per a standard statistical test [148]. This makes it hard to theoretically argue about any desired relation between the distribution of the different gradient vectors.

Integer Coefficients: Another approach is to devise conditions under which Equation (5.4) cannot be satisfied, without resorting to properties of probabilistic distributions. One such condition is when vectors of G are linearly independent.

The column vectors of G are linearly independent if and only if they are not expressible as linear combinations of one another, i.e., $\boldsymbol{z} \neq \mathbf{0}$, thereby trivially showing unsatisfiability of system (5.4). Linear independence is much stronger than what we need to rule out the possibility of satisfying Equation (5.4). Specifically, the values of $\boldsymbol{z}$ are integers $\{-1, 0, 1\}$. Even if some gradient vectors are linearly dependent on other vectors, it does not imply that there exists an *integer* combination of the gradient vectors that adds to $\mathbf{0}$. Towards ruling out (5.4), one could consider doing a milder check, i.e., checking whether there exists a non-trivial integer vector $\mathbf{z} \in \mathbb{Z}^m$ that satisfies (5.4). Unfortunately, this seems as hard as solving the integer programming problem, which is known to be NP-hard [66].

Short Vector Solutions: Notice that the set of vectors $\mathbf{z} \in \mathbb{Z}^m$ satisfying Equation (5.4) form an integer lattice with G as the basis. A valid forgery implies the existence of a *short* non-zero vector in the lattice, i.e, $\|\boldsymbol{z}\|_1 \leq 2k$. If we can rule out the existence of such short vectors, we can conclude that forgery is impossible. This suggests that one can try to lower bound the size (in L_1 or L_2 norm) of the shortest non-zero vector in the lattice. Unfortunately, this would require solving $O(k)$-approximate shortest vector problem (SVP) for the lattice, which is again a hard problem that underpins several constructions in lattice-based cryptography [90].

The above approaches, though promising, seem to run into incompatibility with empirical observations or computational intractability at the outset. There are further issues to consider as well which we explain in Section 5.4.3—the algorithms used to implement the associated checks should work with minimal assumptions about the algebraic structure of the gradient vector operations. Our proposed approach, explained next, keeps assumptions minimal and gives the first practical conditions for checking unforgeability.

5.3.3 Our Approach

The checks outlined so far are still stronger than what we strictly need. Recall that forgery is impossible if conditions below hold:

- (C1) $z_i \in \{-1, 0, 1\}, z_i \in \boldsymbol{z}$; and
- (C2) $G\boldsymbol{z} = 0$ has no non-trivial solution for $\boldsymbol{z}$.

It is important to note at this point that typically in machine learning libraries we work with a finite-bit representation of the real values of the gradients, either floating-point or fixed-point. We devise a fast condition to check when (C1) and (C2) are not true if the reals use fixed-point representation. Specifically, addition in fixed-point precision has desirable algebraic properties (more in Section 5.4.3) under two's complement arithmetic. Our key observation about arithmetic in fixed-point precision is this: *if the sum or difference of any subset of gradient vectors is* 0*, then the parity (exclusive-or) of the least significant bits of those vectors must be* 0. We briefly explain why this is so. Consider any number x and its negative $-x$. In two's complement arithmetic, it is easy to deduce that the least significant bit (LSB) of x and $-x$ is always the same[1]. This implies that $LSB(x - x)$ equals $LSB(x + x)$ in two's complement arithmetic. This fact holds only for the LSB bit because it is the only bit that is not affected by carries during the addition operation.

Extending the above observation to vectors, one can see that the result of adding or subtracting two vectors is simply the addition or subtraction of values dimension-wise. Thus, for any two vectors $\boldsymbol{g}_i$ and $\boldsymbol{g}_j$, the operations $(\boldsymbol{g}_i - \boldsymbol{g}_j)$ and $(\boldsymbol{g}_i + \boldsymbol{g}_j)$ result in vectors that have the same LSB. Condition (C1) encodes that there are only three operations we can do on the gradient vectors to obtain a forgery. We can include a gradient vector $\boldsymbol{g}_i$, include $-\boldsymbol{g}_i$, or skip vector $\boldsymbol{g}_i$ in the summation of Equation (5.3). As discussed previously, $LSB(\boldsymbol{g}_i - \boldsymbol{g}_j) = LSB(\boldsymbol{g}_i + \boldsymbol{g}_j)$ in each dimension. Thus, one can now consider the values of z_i as $\{0, 1\}$ instead of $\{-1, 0, 1\}$, with 0 representing skipping a gradient in G and 1 representing including the gradient (or its negation) in the summation of Equation (5.4). To rule out that any $+/-$ combination of gradient vectors in G sum to 0, one can check that the combinations of $LSB(\boldsymbol{g}_i)$ do not sum to 0. We formally prove that LSB checks are sufficient for unforgeability of gradients in Section 5.4.1 and give an illustrative example here.

Example 1 Let us take 3 gradient vectors $\boldsymbol{g}_1, \boldsymbol{g}_2, \boldsymbol{g}_3 \in \mathbb{R}^3$ computed at some fixed step during training as columns of G below. The next model parameters are

[1]Representing $-x$ is computed as taking the complement of the bits in x and adding one, which implies that $LSB(x) = LSB(-x)$

updated using a minibatch consisting of first two. So the gradient update vector is $\mathbf{\Gamma} = \boldsymbol{g}_1 + \boldsymbol{g}_2 = (-1.0, 3.25, 5.75)^T$. The goal of forgery is to find another subset of vectors that sums to the same update vector $\mathbf{\Gamma}$. In this example, however, the gradient vectors are linearly independent and no other $+/-$ combination results in $\mathbf{\Gamma}$. We use big parenthesis () to denote vectors / matrices defined over real numbers and square brackets [] to denote their fixed-point binary representation to visually distinguish them below.

$$G = \begin{pmatrix} 1.0 & -2.0 & 0.25 \\ 2.0 & 1.25 & 2.0 \\ 3.75 & 2.0 & 1.0 \end{pmatrix} = \begin{bmatrix} 0001.0\mathbf{0} & 1110.0\mathbf{0} & 1111.0\mathbf{1} \\ 0010.0\mathbf{0} & 0001.0\mathbf{1} & 0010.0\mathbf{0} \\ 0011.1\mathbf{1} & 0010.0\mathbf{0} & 0001.0\mathbf{0} \end{bmatrix}$$

For illustration purposes, the above fixed-point representations use 4 bits for the integer part and 2 bits for the fractional part. We then obtain the matrix corresponding the least significant bits of the gradient vectors as $LSB(G) := [LSB(\boldsymbol{g}_1), LSB(\boldsymbol{g}_2), LSB(\boldsymbol{g}_3)]$:

$$LSB(G) = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$

Our proposed check on the gradient matrix yields that $LSB(G)$ matrix is full rank, i.e., no non-trivial solution to $LSB(G\boldsymbol{z}) = \mathbf{0}$ exists. As explained above, when this happens, there are no forgeries possible for any subset of $\boldsymbol{g}_1, \boldsymbol{g}_2, \boldsymbol{g}_3$ of size $k = 2$.

5.4 Unforgeability: Proof & Algorithm

We prove the key result here as Theorem 7. It states that no solutions satisfy the system of equations (5.4) if no solutions satisfy the corresponding boolean system of equations defined over their LSB. Absence of solutions to system (5.4) implies unforgeability. The algorithm for checking the conditions of the theorem is the classical Gaussian elimination over boolean fields (LSB).

5.4.1 Formal Proofs

We present formal proofs for the claims in the previous section.

Lemma 10. *If system (5.4) has no non-trivial solutions $\boldsymbol{z} \in \mathbb{Z}^m$, then forgeries cannot exist.*

Proof. Let forgery be defined as the pair of bitstrings $(\boldsymbol{r}, \hat{\boldsymbol{r}})$ corresponding to the indices of the minibatch at a training step and, respectively, the forged minibatch. If $(\boldsymbol{r}, \hat{\boldsymbol{r}})$ does exist, then $\boldsymbol{z} = \boldsymbol{r} - \hat{\boldsymbol{r}}$ is a non-trivial solution to (5.4). Hence, the premise is incorrect. □

Our goal henceforth is to devise algorithms that detect when the homogeneous system of equations (5.4) does not have non-trivial solutions. Before proceeding further, we draw attention to two points. First, Equation (5.4) is no longer equivalent to Equation (5.2) as we dropped the accompanying constraint $\sum_i r_i = \sum_i \hat{r}_i = k$. Rather, Equation (5.4) is more general than (5.2), thus it provides the sufficient (and not necessary) condition when forgeries cannot exist. Second, usually in neural nets $n > m$, i.e., there are more model parameters than data points, thus, in such a case, (5.4) is overdetermined, and one may expect that no non-trivial solutions to exist.

Theorem 6. *If the system of equations (5.4) has no non-trivial solution for $\boldsymbol{z} \in \mathbb{R}^m$ then forgeries cannot exist.*

Proof. If (5.4) has no solutions $\boldsymbol{z} \in \mathbb{R}^m$ then it has no solutions for $\boldsymbol{z} \in \mathbb{Z}^m$ as $\mathbb{Z} \subset \mathbb{R}$. Then by Lemma 10, forgeries cannot exist. □

Theorem 6 allows to transfer the problem from integer variables to reals. Over reals, if for the homogeneous system $G\boldsymbol{z} = 0$ the rank equals the number of variables, i.e., if $rank(G) = m$, then the system has no non-trivial solutions, thus leading to absence of forgeries (5.4) and cannot have non-trivial integer solutions. This approach is sound and yields the sufficient condition for absence of forgeries as long as the $rank(G) = m$. However, computing the rank may be too computationally intensive or impossible all along (as we will see further). To tackle this problem, we shift once again the domain, but this time, to booleans.

Consider the system of equations (5.3) only for the least significant bits[2], i.e., shift the domain from reals to booleans:

$$\bigoplus_{i=1}^{m} \overline{z_i} \cdot \overline{\boldsymbol{g}_i} = 0, \tag{5.5}$$

where $\oplus$ is exclusive-or (XOR), $\overline{z_i}$ are boolean unknowns, and $\overline{\boldsymbol{g}_i}$ is a boolean vector composed of the least significant bits[3] of the elements of the original vector $\boldsymbol{g}_i$. Under the boolean domain, the exclusive-or ($\oplus$) and logical-and (&) form a field $\{\mathbb{Z}_2, \oplus, \&\}$.

The next lemma establishes the relationship between solving the system of equations (5.5) and equations (5.4).

Lemma 11. *If system (5.4) has a non-trivial solution $\boldsymbol{z} \neq \mathbf{0}, \boldsymbol{z} \in \{-1, 0, 1\}^m$, then the system (5.5) has a non-trivial solution $\overline{\boldsymbol{z}} \in \mathbb{Z}_2^m$.*

Proof. Let the vector elements be represented using t bits of fractional part precision. Then, we can assume that $2^t \boldsymbol{g}_i \in \mathbb{Z}^m$, and that $\overline{\boldsymbol{g}_i} = 2^t \boldsymbol{g}_i \pmod 2$ is a boolean vector. Then

$$\sum_{i=1}^{m} 2^t \boldsymbol{g}_i z_i = \mathbf{0} \,,$$

implies that

$$\sum_{i=1}^{m} 2^t \boldsymbol{g}_i z_i \pmod 2 = \sum_{i=1}^{m} \overline{\boldsymbol{g}_i} z_i = \mathbf{0} \pmod 2 \,.$$

Finally, observe that $\mathbf{z}$ is non-trivial, and hence there exists i such that $z_i \in \{-1, 1\}$, which implies that $\overline{\mathbf{z}} := \mathbf{z} \pmod 2 \in \mathbb{Z}_2^m \setminus \{\mathbf{0}\}$ is a non-trivial solution to system (5.5). □

A corollary of the Lemma 11 obtained by stating its contrapositive is the following: If the system (5.5) has no non-trivial solutions, then system (5.4) also has no non-trivial solutions. Lemma 10 states that when no non-trivial solutions to system (5.4) exist, forgery is impossible. This immediately gives our main result stated below.

Theorem 7. *If system (5.5) has no non-trivial solutions for $\overline{\boldsymbol{z}} \in \mathbb{Z}_2^m$, then forgeries cannot exist.*

[2] Assume the reals have fixed-point precision.

[3] Recall that we consider fixed-point precision, so least significant bit is just the last bit of the representation of real.

Theorems 6 and 7 provide sufficient conditions for unforgeability. In both of the cases, the checks reduce to finding if a homogenous system of equations has non-trivial solutions over a particular domain, either real or boolean. When the system is overdetermined $n > m$, this is equivalent to showing that the rank of the corresponding matrix of the system equals the number of variables.

5.4.2 Algorithm

Based upon the findings of Theorem 7, we develop an unforgeability check called LSBUnforgeability given in Algorithm 6. It takes as input the model parameters (θ_t), dataset (D), loss function (l) and a precision δ. The first step is to check that the dimension of the model parameters, and consequently gradient vectors, is larger than the dataset size. We obtain the gradient matrix G corresponding to the gradients of all of the samples in D with respect to the parameters θ_t (lines $7-10$). We require the loss function (the same as in the training algorithm) to compute the gradients. From the gradient matrix, we obtain a boolean matrix consisting of the least significant bit given some specified precision ϵ. We specify the precision amount in Section 5.6.1, and provide implementation details of TakeLSB in Section 5.5 for gradients obtained using the standard machine learning libraries.

Finally, we call the ComputeBoolRank on the boolean matrix B. This procedure computes the maximal number of independent $\{0,1\}^n$ column vectors in B under the $\oplus$ operation, known as the rank r. If the rank is maximal, then forgeries cannot exist, so the algorithm returns $Unforgeable$. If the rank is not maximal then LSBUnforgeability is inconclusive. To check the rank, we can reduce the matrix to row echelon form with the classical Gaussian elimination algorithm. Section 5.5 gives the implementation details.

5.4.3 Note on Satisfying Algebraic Assumptions

Both the definition of forgery and our checks for unforgeability make certain *minimal* assumptions about algebraic computation during gradient updates. We explain their role carefully.

Fixed-point numbers form a group but not a field. In fixed-point precision, the numbers are assumed to have p bits for the integer part, and q bits for the fractional

Algorithm 6 LSBUnforgeability The outline of our procedure to check the condition under which the forgery of gradients for a given checkpoint and dataset is impossible. If it returns $True$ then forgery is impossible.

Input: The checkpoint θ_t of the model f_{θ_t}, Dataset D, Loss function l, Precision δ.
Output: Unforgeability ϕ

$\phi \leftarrow \square$
$G = EmptyMatrix(n, m)$
$B = EmptyMatrix(n, m)$
while $(x_i, y_i) \in D$ **do**
    $g_i \leftarrow \nabla l_{\theta_t}(f_{\theta_t}(x_i), y_i)$
    $G[:, i] \leftarrow g_i$
end while
while $i \in 1, \ldots, m$ **do**
    $B[:, i] \leftarrow$ TakeLSB$(G[i], \delta)$
end while
$r \leftarrow$ ComputeBoolRank(B)
if $r == min(n, m)$ **then**
    return $True$
end if
return ϕ

part, and in software are usually represented with integers, $a, b \in \mathbb{Z}$. The addition in fixed-point is defined as modular addition over integers, i.e., $a + b(mod\ 2^{p+q})$. Clearly, addition over fixed-point numbers forms an abelian group. Thus, *forgery is well-defined with fixed-point precision.* On the other hand, the multiplication is defined as a combination of modular multiplications and shifts, i.e., $a \cdot b(mod\ 2^{p+q})/2^q$. Unfortunately, this operation is not associative. For example, in 1-bit fractional precision $(\frac{1}{2} \cdot \frac{1}{2}) \cdot 4 \neq \frac{1}{2} \cdot (\frac{1}{2} \cdot 4)$ as $0 \cdot 4 \neq \frac{1}{2} \cdot 2$. Therefore, fixed-point arithmetic does not satisfy algebraic axioms of a field. This limits the algorithms one can reliably use with fixed-point arithmetic when deducing unforgeability. Consider the standard way of computing rank, or checking linear independence, of a matrix. A standard method for finding ranks is Gaussian elimination, but it requires vector elements to satisfy axioms of a field (or at least a principal ideal domain[4]). Therefore, one cannot compute ranks directly for matrices using Gaussian elimination with fixed-point numbers. Hence, to run the rank algorithms we must specify and make sure we work with a field.

[4]The set of fixed-point numbers with addition and multiplication has zero divisors (e.g. $\frac{1}{2} \cdot \frac{1}{2} = 0$) and thus does not form a principal ideal domain.

The least significant bits (LSBs) of fixed-point numbers form a field. The LSBs form the standard boolean field $\{\mathbb{Z}_2, \oplus, \&\}$. Thus Gaussian elimination can run on LSBs and so we can use our unforgeability check algorithm to compute the rank.

On other fields in fixed-point precision. It is tempting to define other finite fields in fixed-point arithmetic so we can use similar unforgeability checks based on Gaussian elimination. For instance, by taking the two least significant bits. But then multiplication is not associative (as pointed out above) and multiplicative inverse does not exist for every element, hence the axioms of a field do not hold. Another alternative is to redefine the multiplication (change it from a modular) to obtain the finite field $GF(2^{p+q})$, however, then the same multiplication needs to be used as well during training of the neural network, thus it may introduce other, potential issues, e.g., with efficiency. Very few checks satisfy the strong algebraic requirements highlighted above. Our unforgeability check on LSBs is one such check, as they allow simple finite field in fixed-precision.

5.5 Implementation

We implement Algorithm 6 in C++: we start with a reference implementation of Gaussian elimination over booleans, and introduce a single optimization enhancement by packing bits into 64-bit integers to speed up addition of rows during row reduction of the matrix. Our entire implementation is less than 100 lines of code and it can run on multiple cores[5]. Note that there are more efficient algorithms for Gaussian elimination over finite fields [139]. We opted for a standard algorithm due to its simplicity and ease of implementation, which proved sufficient for the examined datasets. For very large datasets, the unforgeability check can use more optimized rank checking algorithms[6], such as [139] which reports a running time of 520 minutes on a $10^6 \times 10^6$ boolean matrix using 64 cores.

Extracting fixed-point LSB from floating-point. For our evaluation (Section 5.6.1), we provide a brief description about taking certain fixed-precision LSBs of floating-point numbers. Specifically, let us examine how to extract the t-bit LSB of a 64-bit

[5]Our code is available at https://github.com/teobaluta/unforgeability-SGD

[6]We were only able to find implementation of this algorithm for a specific HPC framework, and not in the common languages such as C/C++. As our implementation was feasible to run, we decide not to switch to the advanced algorithm.

float based on the IEEE format [8], i.e., the float has 1-bit sign s, 11-bit exponent e, and 52-bit mantissa m and represents the number $(-1)^s 2^{e-1023}(1 + \frac{M}{2^{52}})$ or equivalently $(-1)^s 2^E(1 + \frac{M}{2^{52}})$. When $E = 0$, as the sign and the leading 1 play no role, the t-bit LSB of the number is the t-bit of the mantissa (or it is 0 if $t > 52$). Having $E \neq 0$ is similar, but first we logically shift the mantissa by E positions (to the left if $E < 0$, otherwise to the right), and then take the t-bit of the result[7]. For instance, when $E = -10$, t-bit LSB of the float is the $t + 10$ bit of the mantissa. Hence, extracting LSBs is straightforward, and subtraction, logical shift and masking are the only operations required for its implementation. All of these use two's complement arithmetic, as needed for our results to apply.

5.6 Evaluation

Our main goal is to evaluate whether our LSB check is conclusive in practice. Our benchmarks and experimental setup mirror those of [226] and [117], as our forgery game encompasses prior setups, with the difference that our definition of forgery is exact, while theirs is approximate. Our formal results are valid under the well-defined arithmetic of fixed-point precision, not floating-point. Therefore, we want to evaluate LSB checks at a fixed bit-precision, but we also want to measure how the check's conclusiveness changes with more samples or with less precision bits. Our aim is to check whether replacing one checkpoint at an intermediate step with an approximately forged one leads to *divergence*, i.e., the subsequent model parameters are noticeably far from the original trace.

In summary, we aim to answer the following research questions:

(RQ1) How conclusive is our LSB check under a given precision using the same experimental setup as prior work?

(RQ2) How conclusive is our LSB check under a given precision when increasing the dataset size?

(RQ3) What precision is sufficient for LSB checks to be conclusive?

[7] When $E > 0$, the leading 1 plays role and should be taken into account.

(RQ4) Does approximate forgery at a given training step result in noticeably different model parameters after continuing training for more steps, i.e., does training diverge?

(RQ5) How much divergence do rounding errors arising from floating-point arithmetic introduce after training for more steps?

Benchmarks & Experimental Setup. In our experiments, we use the same as benchmarks as [226, 117]. More precisely, we focus on LeNet5 [126] with 61,706 parameters on MNIST [125] dataset, ResNet-mini [86] with 1,487,370 parameters and VGG-mini with 5,742,986 parameters on the CIFAR10 [120] dataset. As reference implementation for LeNet5 we used [179], confirmed through correspondence with the authors [226]. For the ResNet-miniand the VGG-mini implementation we used as reference the one at [186], i.e., the same one specified in [117]. For all of these model architectures, we do not use batch normalization, same as [226, 117]. We use a fixed learning rate 0.01, and train with batch sizes of 64 for various epochs, each with some number of training steps (depending on the batch size and dataset size). Prior work on approximate forgery [226] considers $M = 400$ candidate minibatches, which is what we use in Section 5.6.1[8]. To train models, we use NVIDIA GPU 2080X, CUDA 11.7. Furthermore, we ran LSBUnforgeability (Algorithm 6) on Ubuntu 20.04 box, with 80 cores and 256GB RAM. For the VGG-mini experiments and experiments with larger sizes of M, we used a storage over the network, which added an overhead to our results.

Reproducibility We train our models using PyTorch `1.13.1+cu117` for GPUs. In PyTorch we use `np.float64` floating-point precision for training. For reproducibility, we avoid using nondeterministic algorithms for some operations and set a specific seed for our computation [180]. This ensures that under multiple runs on the same hardware and software stack, we obtain the same gradients and model parameters when training.

Notations. Gradients are n-dimensional vectors over reals. To represent distance (sometimes we call it difference, or *error*) between gradients g_1, g_2, we use either L_2

[8]Based on email correspondence with the authors of [226], in Fig. 1 and Fig. 2 in the [226] paper, the number of batches is 400.

norm, i.e., $||g_1 - g_2||_2 = \sqrt{\sum_{i=1}^{n}(g_1^i - g_2^i)^2}$, or L_∞ norm, i.e $||g_1 - g_2||_\infty = max_i(|g_1^i - g_2^i|)$.

5.6.1 Are LSB Checks Conclusive?

If our check does not determine that the matrix is full rank, then there might exist forgeries. It is reasonable to expect that the least significant bits of the gradients will not be strongly biased, as the training process introduces sufficient entropy at least in the LSBs of the gradients. This in turn will lead to full rank boolean matrix, i.e., positive unforgeability check. We check this in our experiments.

From Floating-point to Fixed-point Precision. We convert the 64-bit floating-point precision gradients (called *sources*[9]) output by PyTorch into fixed-point precision. We consider a high number of precision bits taken from the *source* float-point gradients, i.e., 26 bits[10]. This is within the scope of precision required typically for machine learning training.

Following the procedure used in prior work, we randomly sample 25 checkpoints at different training epochs and steps from the first 5 epochs for LeNet5, ResNet-mini, and, respectively, 5 checkpoints for VGG-mini. We run LSBUnforgeability on these checkpoints for $M = 400$ candidate minibatches, sampled without replacement. For efficiency reasons, we run only 5 checkpoints for VGG-mini. The average running time of the algorithm on the checkpoints is around 23 seconds on LeNet5 (Table 5.1), 1291 seconds for ResNet-mini models (Table 5.2), and, respectively, 9588 seconds for VGG-mini models (Table 5.3). Furthermore, all of these are unforgeable, showing that the conditions we state in our theorem are satisfied in practice.

Result 1: The unforgeability check LSBUnforgeability is efficient (feasible for large neural networks) and effective. It outputs conclusively that forgeries are impossible on all evaluated cases.

[9]We want to stress out that we introduce this convention only because we want to reuse the *floating*-point *source* gradients output by the PyTorch training process (currently, it supports only floats), however, if the training had been conducted in fixed-point precision, the amount of precision bits would have been uniquely determined.

[10]In fact, any sufficiently large amount can be taken.

Epoch	Step	Time (s)	Unforgeable
0	839	22	✓
1	402	22	✓
1	447	24	✓
2	0	23	✓
2	42	23	✓
2	194	22	✓
2	232	23	✓
2	361	24	✓
2	481	22	✓
2	505	23	✓
2	534	22	✓
3	187	22	✓
3	401	22	✓
3	410	22	✓
3	722	23	✓
3	736	22	✓
4	186	23	✓
4	217	24	✓
4	295	23	✓
4	296	22	✓
4	610	23	✓
4	695	23	✓
4	827	22	✓
4	936	22	✓
5	332	23	✓

Table 5.1. All 25 evaluated checkpoints for LeNet5 on MNIST with fixed precision of 26 bits are unforgeable.

One may ask what happens if multiple datasets are concatenated, or if the size of the dataset were larger. We thus additionally consider larger number of batches M for LeNet5 on MNIST, and ResNet-mini on CIFAR10. We find that the tests are conclusive for both LeNet5 and ResNet-mini at precision 26 when we vary the number of batches (Table 5.4).

Result 2: The unforgeability check LSBUnforgeability is conclusive even on larger sample sizes compared to [226].

5.6.2 Precision at which LSB Checks are Conclusive?

One might ask what precision is enough to prove unforgeability with our LSB

Epoch	Step	Time (s)	Unforgeable
0	98	1561	✓
0	259	1202	✓
0	646	1555	✓
0	480	1129	✓
1	84	1408	✓
1	416	1183	✓
2	182	1465	✓
2	249	1449	✓
2	286	1448	✓
2	743	1254	✓
2	750	1124	✓
3	115	1401	✓
3	130	1186	✓
3	215	952	✓
3	250	1522	✓
3	261	1266	✓
3	275	1154	✓
3	317	1393	✓
4	714	1423	✓
4	28	1154	✓
4	677	1337	✓
5	74	965	✓
5	452	1380	✓
5	541	1319	✓
5	644	1056	✓

Table 5.2. All 25 evaluated checkpoints for ResNet-mini on CIFAR10 with fixed precision of 26 bits are unforgeable.

Epoch	Step	Time (s)	Unforgeable
1	209	7078	✓
1	262	9945	✓
1	257	10254	✓
3	11	10259	✓
3	450	10406	✓

Table 5.3. All 5 evaluated checkpoints for VGG-mini on CIFAR10 with fixed precision of 26 bits are unforgeable.

check? We focus on scenarios with lower number of precision bits, that may potentially allow approximate forgeries. We take 5 different LeNet5 checkpoints and vary the selection of precision bits from 1 to 24 from the *source* gradients. In Table 5.5, we give the exact ranks of the systems obtained for these 5 checkpoints, and gray out the full ranks which essentially correspond to unforgeability. For each of the 5 checkpoints, none of the systems for bits below 14 have full rank, i.e.,

Architecture	M	Avg. Time (s)	Unforgeable
LeNet5	600	45.84	25 / 25
	800	91.56	25 / 25
ResNet-mini	600	5810.2	5 / 5
	800	8817	5 / 5

Table 5.4. Even when varying the number of minibatches M, LSBUnforgeability remains conclusive for both LeNet5 and ResNet-mini at precision 26.

LSBUnforgeability cannot exclude forgery for such cases. Indirectly, this means that potential approximate forgeries with precision up to 2^{-13} (around 10^{-4} in L_∞) are still possible. *This range already covers all approximate forgeries we present in Section 5.6.4.* On the other hand, with around 20-bit precision the checkpoints transition to unforgeable. Hence, it would be unlikely to produce approximate forgeries with a precision higher than 2^{-21} (around 10^{-6}) at any of these checkpoints, regardless of the technique used to generate them. We cannot completely exclude the possibility, as carries from the lower (beyond taken precision) bits may still propagate, however, such carries will only randomize the LSBs, thus yielding similar, full-rank matrices.

Result 3: LSBUnforgeability is conclusively finds unforgeability for precision over 20 bits in all evaluated cases.

Therefore, when interpreting results of approximate forgery even in fixed-point arithmetic, the precision considered plays a significant role in determining whether forgery works at all.

5.6.3 Divergence with Approximate Forgery?

Our next experiments provide evidence that approximate forgeries at an intermediate step in training leads to large differences in the final model after training more steps. We argue that while obtaining a limited precision (e.g., up to $\delta = 3$ decimals) forgeries at a particular step is feasible, in subsequent training steps these errors will once again increase. We test this hypothesis by implementing the search for approximate forgeries [226].

	Checkpoint 1	**Checkpoint 2**	**Checkpoint 3**	**Checkpoint 4**	**Checkpoint 5**
Bit					
1	3121	6794	2727	4422	2542
2	4277	9336	3717	6093	3457
3	5794	12091	5048	8175	4624
4	7645	14882	6708	10657	6108
5	9873	17859	8756	13394	7843
6	12491	20533	11207	16241	9943
7	15105	22588	13848	19127	12306
8	17487	24049	16544	21400	14837
9	19457	24978	18810	23031	17284
10	21224	25342	20720	24206	19417
11	22709	25477	22344	25015	21304
12	23930	25561	23638	25414	22778
13	24785	25594	24547	25565	23991
14	25276	25600	25123	25593	24852
15	25496	25600	25399	25598	25313
16	25581	25600	25518	25599	25539
17	25595	25600	25575	25600	25593
18	25597	25600	25592	25600	25598
19	25599	25600	25598	25600	25599
20	25600	25600	25598	25600	25600
21	25600	25600	25600	25600	25600
22	25600	25600	25600	25600	25600
23	25600	25600	25600	25600	25600
24	25600	25600	25600	25600	25600

Table 5.5. Even at smaller precision on LeNet5 checkpoints, our LSB check can determine unforgeability. The grayed out cells correspond to full rank, i.e., the checkpoints at this bit precision are not forgeable.

We randomly sample 25 saved checkpoints for LeNet5, ResNet-miniand VGG-mini from the first 5 epochs with θ_t model parameters (same ones as in Section 5.6.1). The target checkpoint that we want to forge is θ_{t+1}. Then, according to the previously proposed strategy, for each checkpoint θ_t, we sample $M = 400$ forgery candidate batches with size 64. Then, we perform one training step from θ_t using these samples

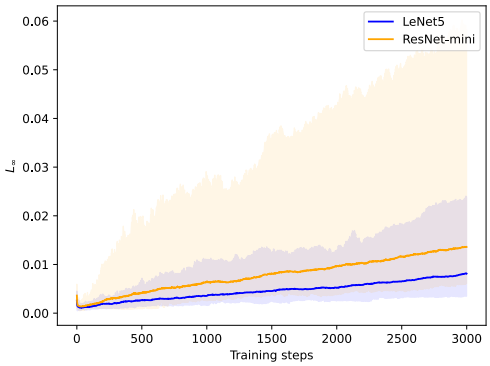


Figure 5.2. The approximately forged model parameters diverge after subsequent training of LeNet5 on MNIST and ResNet-mini on CIFAR10. The solid line indicates the mean L_∞ distance over the 25 checkpoints while the translucent region indicates the maximum and the minimum L_∞ distance boundaries for the corresponding architecture.

and greedily select the one with the smallest L_2 and, respectively, L_∞ distances from the target checkpoint: $argmin_M||\theta_{t+1} - \theta'_{t+1}||_p, p \in \{2, \infty\}$. Then, to test potential divergence, we keep training the forged model parameters with the same data as the target trace. We train for $3,000$ additional steps for LeNet5 and ResNet-mini, and $10,000$ more for VGG-mini. In Figure 5.2, we show for LeNet5 and ResNet-mini that the distance between the initial and forged models' parameters increases. For VGG-mini, the L_∞ difference between the training run and the forged run initially decreases, but, as with LeNet5 and ResNet-mini, it eventually diverges (Figure 5.3). We observe that the larger the model (number of parameters), the slower the divergence. It takes about $8,000$ training steps for the L_∞ distance to be greater than the initial distance, with respect to the forged parameters. This is due to the gradient descent updates being smaller in magnitude than the initial error for VGG-mini.

We run one more series of experiments with longer training. More precisely, we train only LeNet5 (for efficiency reasons) for $17,000$ training steps. The propagation of distance is given in Figure 5.4 and we can observe a similar outcome. Therefore, based on these two experiments (refer to Appendix B.1 for L_2 results), it is clear that even not-so close approximate forgeries will diverge in the successive training steps.

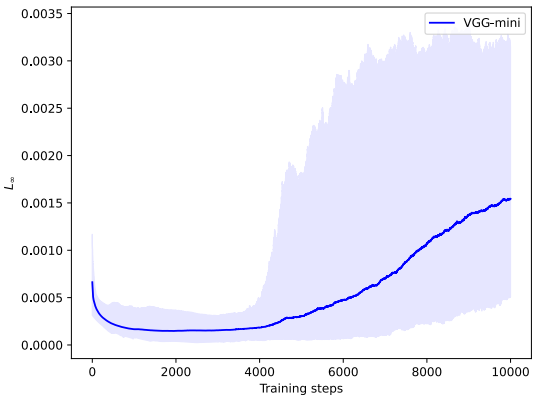


Figure 5.3. The approximately forged model parameters diverge after subsequent training of VGG-mini on CIFAR10. The solid line indicates the mean L_∞ distance over the 25 checkpoints while the translucent region indicates the maximum and the minimum L_∞ distance boundaries for the corresponding architecture.

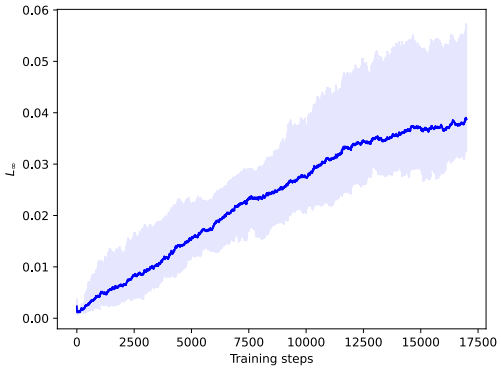


Figure 5.4. L_∞ distance between the forged batch and the benign training for LeNet5 on MNIST over 5 checkpoints. The solid line indicates the mean L_∞ distance over the 5 checkpoints while the translucent region indicates the maximum and the minimum L_∞ distance boundaries for the corresponding architecture.

Result 4: Approximate forgeries in floating-point precision eventually diverge and always result in clearly distinct model parameters by the end of training.

Impact of Larger M on Forgeries. We check whether increasing the number of candidate minibatches helps with obtaining better approximate forgeries. We increase the number of candidate minibatches and run the approximate forgery attack for $M \in \{600, 800\}$ by sampling without replacement, on all models, including VGG-mini. When increasing the number of candidate batches, there is no real improvement

in the obtained approximate forgeries, for all evaluated models. At the same time, doubling the number of batches means that the running time of the attack also doubles. Specifically, we run the approximate forgery, as in the evaluation of [226] for 25 checkpoints, to find the best L_∞ norm for LeNet5, ResNet-mini and VGG-mini for all $M \in \{400, 600, 800\}$. Among all of the 25 evaluated checkpoints the attacks' best approximate forgery among all 25 checkpoints was 1.56×10^{-4} in L_∞ norm on VGG-mini when $M = 800$. However, increasing the number of batches did not improve the average L_∞ distance by much, i.e., by less than 10^{-5}. For the largest number of batches $M = 800$, the time taken for VGG-mini is 40× larger than for LeNet5 and 2× larger than for ResNet-mini. We give the detailed results in Appendix B.2.

5.6.4 Divergence due to Floating-point Errors?

Recall from Section 5.4.3 that forgeries are not well-defined for reals implemented with floating-point precision. We first confirm that additions of gradients in floating-point lead to non-zero rounding errors. For this purpose, we run a series of experiments at different training epochs. At each epoch, we sample uniformly at random a minibatch of 1024. Then we sum up the gradients in 1000 different random orders, and check the number of different sums we get. The final results show that at each of the tested epochs, 1000 out of 1000 sums are different, i.e., each shuffle leads to a distinct sum. The sums differ on L_∞ errors in the range 10^{-12} to 10^{-17}. Hence, we can conclude that even when one considers the same minibatches but in different summation order, one may not produce this trivial forgery on all n bits. In our experiments, in 100% of the cases, different order resulted in different values. Thus, in floating-point precision, forgeries resist standard definition, rather, they will require at minimum an additional specification of the order of summation.

We investigate what happens if the above produced errors are kept small throughout training. If such is the case then one may argue that approximate forgeries with such precision (i.e., equal on almost all n bits) should be accepted as valid under the premise that the errors may have originated from minor hardware or library discrepancies. To test the propagation of small errors, among all of the previously generated potential pairs of sums, we sample a random pair at 5 different

	Shuffle Error	Epoch 1	Epoch 2	Epoch 3	Epoch 4	Epoch 5
(epoch 1, step 100)	4.62e-14	2.83e-04	1.33e-03	1.66e-03	2.33e-03	3.16e-03
(epoch 2, step 100)	2.49e-14	8.25e-04	1.47e-03	1.59e-03	4.27e-03	5.67e-03
(epoch 3, step 100)	3.02e-14	8.93e-04	1.98e-03	4.22e-03	4.88e-03	4.49e-03
(epoch 4, step 100)	3.38e-14	1.12e-03	3.35e-03	3.53e-03	4.73e-03	4.52e-03
(epoch 5, step 100)	4.62e-14	2.13e-03	4.27e-03	3.75e-03	5.86e-03	4.33e-03

Table 5.6. Even very small differences of 10^{-14} (called shuffle errors) due to the summation order in floating-point produce divergence over 5 epochs of training.

checkpoints. For each pair, we produce the resulting parameters (at the sample epoch/step), and then train independently each of them for 5 additional epochs on the same randomly sampled minibatches of size 1024. At the end of the training, we compare the differences between the final parameters, i.e., we compute the L_∞ norm between the parameter vectors. The results at different training epochs are presented in Table 5.6. We can see that the small initial errors (differences between the parameters), quickly expand and even after the first training epoch (around 1000 training steps) become pronounced, large errors. This means that even if we consider very close approximate forgeries (as close as a rounding error produced during floating-point addition), the subsequent training process will rapidly expand the small, initial difference and the almost identical pair of parameters produced by the approximate forgery will diverge into clearly distinct parameters. Therefore, based on these experiments, we can make two key observations. First, approximate forgeries clearly lead to distinguishable final output models in training, and therefore, are ill-suited for use in formal definitions. Second, even exact forgeries in floating-point precision diverge due to rounding errors of additions. Additionally, we have shown that small errors introduced due to hardware and library discrepancies lead to clear parameter discrepancies after a few rounds of training.

5.7 Discussion

In this chapter, we have argued for refining the definitions of forgeability, irrespective of applications and datasets. Next, we discuss the setup limitations, and the complexity of the proposed test.

5.7.1 Limitations

Our tests were conclusive for all of the experimental setups considered in prior work. These setups assume that the adversary is constrained to use samples from the dataset that the ML trainer has released for verification. The verifier is honest, and the adversary can only modify the choice of minibatches from the fixed dataset at a training step (see Section 5.2.2). This is a **post-deployment** adversary, that aims to construct forged gradient updates for data samples after the execution traces and their logs have been released. If the adversary can modify the dataset, then they could form $\hat{\boldsymbol{b}}_t$ from samples outside the training dataset such as synthetically generated samples [261], or sampling more points from the distribution, until the test fails. Despite recent works proposing adversaries that synthesize data, there is no evidence that such attacks are possible at a higher precision (not approximate) [53]. Moreover, these attacks should be efficient enough to be mounted for every instance where a data point is used, not just one training step (whereas verifying impossibility for one is enough). Without breaking the requirement that the test be run with respect to a dataset, we can ask how often is the test conclusive if one were to consider a larger dataset. We evaluate this in Section 5.7.2.

A **pre-deployment** adversary, on the other hand, can manipulate both the training hyperparameters and the dataset. A more powerful test is required, one that does not assume that the adversary cannot manipulate the setup phase. The verifier therefore can only check that the update is valid, i.e., given some minibatch $\hat{\boldsymbol{b}}_t \neq \boldsymbol{b}_t$, the model parameters $\hat{\theta}_{t+1}$ satisfy the forgery condition $\theta_{t+1} = \hat{\theta}_{t+1}$. In theory, such an adversary can construct a valid execution trace that is unforgeable with respect to D, but forgeable with respect to a trace obtained from a different dataset and hyperparameters. Formal statements about the existence of such adversaries at high precision would unlock more practical applications that rely on proof-of-learning logs [105].

One limitation of our check is that it works under the condition that the size of the dataset is less than the model parameters, which is typically the case for many deep learning models and datasets. However, future unforgeability checks can consider the scenario where the number of model parameters is less than the dataset size, and design checks that are conclusive. These scenarios might be more relevant

to large language models, such as a T5-base transformer model [183, 187] (around 220 million parameters[11]) trained the C4 crawl dataset (estimated at 365 million records) [183, 41].

5.7.2 Theoretical Complexity

Attacks proposed in [226] that find approximate forgeries (greedy) search the space of minibatches with the smallest distance in parameter space to the targeted model parameters. Instead, one can try running such search algorithm until an exact forgery is found, if one wants to utilize them to answer the decision problem of forgeability. However, as our evaluation points out, there might not be a solution for exact forgery, in which case the search would exhaust all possible $\binom{m}{k}$ minibatches to decide that there is no solution. Our test is rank computation whose worst running time is $\mathcal{O}(n \cdot m^2)$ where n and m are the number of model parameters and, respectively, the dataset size. The computational difficulty of other search procedures when exact forgery is possible is unknown; we briefly alluded to these approaches in Section 5.3.2.

5.7.3 Repudiation Game

Let us assume that at time t, the ML trainer has trained an ML model with some training dataset and releases publicly the ML model parameters θ at time t for everyone to use, along with the claim that training dataset D_1 was used. They have logged every training step of the training process, along with making the training reproducible. At time $t+1$, the claimant becomes suspicious of the ML trainer's claim that D_1 is the dataset they have trained on, and believes that they have actually, at time t, trained on the training dataset D_2 that contains their data. For simplicity and without lack of generality, assume $D_2 = D_1 \cup \{x\}$. Given the claimant's suspicion, can we verify if the ML trainer used x at time t? For that, at time $t+2$, the ML trainer releases their training log with all the information required to replay the execution of SGD and check that the training dataset D_1 was indeed used. A trusted third-party replays the execution of SGD using the information released by the ML trainer and concludes that the final model parameters are the

[11] https://github.com/google-research/text-to-text-transfer-transformer/blob/main/released_checkpoints.md#t511

same as the parameters θ released at time t, hence the trace is *valid*. We call this the validity check. Is the validity check of the execution trace sufficient to resolve this dispute? We have to consider that two worlds are possible: one where the ML trainer is honest and one where the ML trainer is not honest. The main conundrum is that we do not know in which world we are in.

(World 1) ML trainer is honest. If the ML trainer is honest, they have trained on D_1 at time t. Thus, if this is the world we live in, providing a valid execution trace and checking if $x \in D_1$ is sufficient to convince the judge and the claimant.

(World 2) ML trainer is dishonest. However, consider the alternative. If the ML trainer is dishonest, then at time t they have lied about training with D_1 and have obtained θ from a different dataset $D_2 = D_1 \cup \{x\}$, containing the claimant's data x. There are two potential scenarios here. In the first scenario, the ML trainer cannot produce a valid trace since their actual training log at time t is a log from D_2. Such a dishonest trainer is immediately caught by the validity check. The second (and more interesting) scenario is that, at time $t + 2$, the dishonest ML trainer can obtain a valid trace that results in θ using D_1 since that is what they claimed to have used at time t. In this case, the trace validity mechanism based on simply replaying the executing of SGD would wrongly certify that the ML trainer has not used sample x at time t. A valid execution trace and a simple membership query $x \in D_1$ is thus insufficient to determine whether x was used or not at time t.

What is released at time t matters. What if at time t the ML trainer only releases θ and does not divulge which training dataset was used? Then, a dishonest trainer has more freedom in creating a valid trace that results in θ. At time $t + 2$, the ML trainer fabricates a valid trace and training dataset that does not contain x such that the final model parameters are θ. We refer to this as a pre-deployment adversary (see Section 5.7.1). What if at time t the ML trainer not only releases θ but also the intermediate set of checkpoints, along with the claim that D_1 was used. If we are in world 2, then the model was trained on D_2, and a validity-based check would not be able to distinguish between an honest trainer and a dishonest one. Then, at time $t + 2$, the ML trainer can come up with a different selection of minibatches for each training step that leads to exactly the same trace. This is the adversary we model in this work.

Relation to the one-step forgery game. The adversary in Section 5.2.2 is the ML trainer whose aim is to modify only one training step of SGD to construct the same trace starting from the claimed D_1. Under this setup, with our proposed check, LSBUnforgeability, if we are in the first world (ML trainer is honest), we can soundly attest it. This is what we describe above as the post-deployment adversary (Section 5.7.1). What remains for future work is to check if the ML trainer can obtain a different valid trace (i.e., not the same intermediate checkpoints) whose final model parameters are θ starting from D_1.

5.8 Related Work

Approximate Forgery & Applications. Prior work has shown that different minibatches can produce similar model parameters using SGD, with direct implication to applications such as unlearning, proof-of-learning, and membership inference tests. A recent work [226] argues that approximate unlearning is not refutable or auditable because forgery of minibatches that contain the to-be-unlearned samples (say $\boldsymbol{x}$) is possible. This implies that we could have obtained a similar model parameter state had we used a different minibatch (without $\boldsymbol{x}$) from the training dataset. Thus, one cannot distinguish whether these execution traces correspond to the training dataset with the $\boldsymbol{x}$ samples. Another recent work [117] proposes using approximate forgery for repudiating membership inference tests. In this application, the authors have considered forging multiple checkpoints throughout the training process in order to find similar execution traces for D and $D - \boldsymbol{x}$. The resulting models have similar parameters up to some error in vector norms due to forging at multiple checkpoints. On these models, membership inference attacks are not able to distinguish whether $\boldsymbol{x}$ has been used. We motivated forgery using the proof-of-learning logs introduced in [105], which allows a verifier or a third-party auditor to check that 1) the computation was done over a given dataset and 2) that all the steps of the computation have been done correctly to obtain a final set of parameters. The verifier asks the ML owner / adversary to produce a sequence of batch indices and intermediate model updates such that starting from the initialization one can replicate the path to the final model parameters. Their proposed approach is to select only a subset of checkpoints to verify the model parameters for. However,

there is no guarantee that an adversary is not able to forge the minibatch (find a different from the one used by the model owner) to produce the desired model parameters via SGD. In light of our results, we argue that one should consider exact forgery under fixed-point precision, since white-box verifiers are able to examine the forgery under all available precision, whereas previously demonstrated forgeries (i.e., with different samples) have as high an approximation error as 10^{-3} (in L_∞). Such approximation errors in forgeries are much higher than ones that could be attributed to floating-point errors—which we evaluated to introduce differences of the order of 10^{-14}. In addition, [226] describes forgery under other setups, e.g., when one considers *similar* datasets to the one used for training or where the initial model parameters are not the same, i.e., not forgery at a checkpoint but rather across multiple training steps at a time. These proposed problems are beyond the scope of our results but are interesting future work.

Algebraic Precision. Our work highlights the role of algebraic precision in specifying properties and drawing refutable conclusions about experimental observations about training with SGD. This issue is shared with other prior works that are not concerned with forgery as well. For instance, data reordering attacks on SGD distort the training execution trace to an adversary's advantage (e.g., longer convergence times, drop in task performance) [209]. These attacks use alternative minibatches from the same dataset reshuffled or reordered to produce similar but different model parameters after some training steps. In our experiments, we also show that it is possible to obtain this type of training divergence under minibatch reshuffling to change the training execution trace because of floating-point errors that propagate (Section 5.6.4). We pinpoint that these phenomena are due to the non-associativity of floating-point computations. If one did not have the *exact* order they would not be able to reproduce the execution trace of the training algorithm. This observation is also related to the problem of reproducibility in machine learning research which has been a known issue in creating artifacts [180, 223, 176]. To this end, our approach works under fixed-point precision, where additions have the required algebraic properties such as associativity and commutativity. There is on-going research into making training available in lower or fixed-point precision [71]. Quantization techniques are commonly used to accelerate inference of deep learning models but these do

not apply to our setup since the gradient update computation is still being done in floating-point [96, 97] or in bifurcation/mixed precision (both floating-point and 8-bit integer) [11]. Other techniques propose complete fixed-point precision training pipelines that achieve good task performance [137, 70, 39]. On the other hand, these errors that accumulate because of choosing a different order of samples during SGD introduce some noise that helps making training data less "distinguishable". This is in line with recent work on repudiating membership inference attacks using approximate forgery [117, 116] but other types of noise have been purposefully added to the gradient computation to add privacy. For instance, it is common to add Laplace or Gaussian noise to gradients in order to achieve differential privacy [1]. Noise is also added to gradients in order to defend against bias attacks in inverting gradients [5, 52].

Finding Pre-images for Neural Networks. Our work considers finding collisions in the gradient update step due to freedom of choosing the minibatches in training. There has been research in understanding collisions at inference time rather than during training, for instance, when two different inputs produce similar [134] or the same activations or logits with a given ML model [163]. The problem of exact forgeries in this thesis is that of finding a second pre-image in gradient descent, whereas prior work on gradient inversion considers the problem of finding any pre-image—finding the input to the model from the gradients [263]. Without bias in the model architecture, one work shows that recovering input data points can be uniquely determined from the gradients [52]. More advanced gradient inversion techniques deal with different types of neural networks [68, 103, 262]. Gradient inversion considers recovering the input data sample that results in a given gradient vector. However, recovering the set of data samples used in a minibatch given a gradient update vector has yielded much lesser success thus far, though attacks exist [255, 94].

5.9 Summary

In this chapter, we identified mild and sufficient conditions under which gradient updates at one step of standard SGD training are unforgeable. Ours is the first result on proving unforgeability to the best of our knowledge. We found that these

conditions are satisfied for the same benchmarking setup as prior work, i.e., single-step forging is not possible for LeNet5 and ResNet-mini neural networks on MNIST and CIFAR10, respectively. Our work explains that algebraic precision plays a crucial role in making refutable claims about model comparison. We believe these aspects matter practically to forgery-based security arguments and beyond.

5.10 Future Work

There are a number of interesting open problems that stem from the forgery property of SGD traces. The work laid out in this chapter highlights the need for algebraic precision in definitions of forgery. In particular, a key insight is that forgeability of traces is well-defined under fixed-point arithmetic where the addition is associative. However, most of the machine learning frameworks use IEEE 754 floating-point arithmetic. This is one of the sources of non-determinism that prior work mentions, among others. In order to have any meaningful properties, we have to be careful when analyzing the rounding errors introduced by computing in floating-point. We can bound the tolerance δ_{FP} that results from summing the vectors in any order in floating-point (e.g., via semi-numerical analysis). If that bound is smaller than the bound derived by the attack ($\delta_a \approx 10^{-4}$), it prompts the question of approximate unforgeability: Are forgeries possible which are within the tolerance of δ_{FP}? We require novel techniques that look at geometric properties of the gradient matrix that allow us to reason about distances between vectors. Future work should consider such questions to extend our ability to reason about traces in more practical setting.

Besides the issue of computer representation, there are storage and privacy that hinder the practicality of our approach based on recording the training logs. The ML trainer has to store all training steps and share all the inputs with the verifier, including hyperparameters and training dataset. While it is common practice to store intermediate checkpoints every X training steps (commonly used are values for X are 100 or $1,000$), the forgery game presented in this chapter assumes all training steps are recorded in the trace. The verifier has to recompute and check every training step of the trace in order to verify that the trace is unforgeable. Moreover, the verifier is trusted since all the trainer's information (including training

dataset) is required. Thus, if the training dataset contains private and sensitive information the ML trainer may not be willing to share their traces. Future work can consider the more practical setting of verifying intermediate checkpoints rather than the whole trace. In this case, a probabilistic forgery game could be derived based on a security trade-off parameter. Another interesting line of work is to prove unforgeability without revealing the training dataset.

We refer the reader to Section 5.7 where we discuss the distinction whether the claim of using dataset D_1 happens at t (in which case the adversary acts post-deployment) or not (in which case the adversary acts pre-deployment). Even under the post-deployment model, can the ML trainer manipulate only the training procedure to obtain a forged trace? Thus, without making changes to D_1 or the final model parameters θ, can they recreate a trace that does not use a target sample x? This is an interesting future work direction. While prior work has shown that training logs are forgeable, they do so either by considering approximate forgeries that are far larger than rounding off errors of floating-point computation [53, 226], or by giving the adversary the ability to change the dataset D_1 [261]. Future work can consider improving forgery attacks to settle the question of how hard it is to forge traces. The existence of an efficient adversary with fewer assumptions that can forge traces given a set of model parameters is consequential to membership inference tests, unlearning auditing, and model ownership. As a result, future work should consider this direction, and search for techniques that resolve these issues systematically.

Chapter 6

Conclusion

In this thesis, we identified gaps several foundational aspects of security in machine learning, from security definitions, models of reasoning about the training process, to sound procedures for statistically verifying security. Our research closes such gaps by putting forward precise definitions of forgery, i.e., when two different sets of samples lead to the same model. Our work is the first to show that stochastic gradient descent steps are collision-resistant computations under mild checkable conditions, which is in contrast with claims of existing works on the same experimental setup. Our approach reasons about execution traces of stochastic gradient descent, and considers forgeries of one training step. We show why existing approximate forgery definitions suffer from a divergence phenomenon, namely after subsequent training steps, the differences in the model parameters increase by 100× between the approximately forged execution trace and the original traces. To reason about the forgeability of traces, we proposed LSBUnforgeability, a test that is efficient and conclusively shows unforgeability for all the considered experimental setups. LSBUnforgeability is a first step towards practical mechanisms for solving disputes over data non-repudiation.

This thesis proposes the first causal model for stochastic gradient descent for studying memorization versus generalization. Memorization has been highlighted as a privacy weakness in training machine learning models via membership inference attacks, whose goal is to infer if a sample belonged to the training dataset. It is *not* well understood, however, why they arise. Are they a natural consequence of imperfect generalization only? Which underlying causes should we address during training to mitigate these attacks? Towards answering such questions, we propose the first approach to explain membership inference attacks and their connection to

generalization based on principled *causal reasoning*. We offer causal graphs that quantitatively explain the observed MI attack performance achieved for 6 attack variants. We refute several prior non-quantitative hypotheses that over-simplify or over-estimate the influence of underlying causes, thereby failing to capture the complex interplay between several factors. Our causal models also show a new connection between generalization and MI attacks via their shared causal factors. Our causal models have high predictive power (0.90), i.e., their analytical predictions match with observations in unseen experiments often, which makes analysis via them a pragmatic alternative.

Finally, this thesis proposes the first frameworks for statistical verification of properties with soundness guarantees in both white-box and black-box access to the machine learning models. We demonstrate the utility of these sound procedures in several security applications for which we formalize counting queries. In summary, we contribute to precise formalisms and algorithmic tools for rigorous security analysis, and give concrete security applications where our frameworks show utility.

Bibliography

[1] M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang, "Deep learning with differential privacy", in *Conference on Computer and Communications Security (CCS)*, 2016, pp. 308–318.

[2] I. Abío, R. Nieuwenhuis, A. Oliveras, and E. Rodríguez-Carbonell, "BDDs for pseudo-boolean constraints–revisited", in *International Conference on Theory and Applications of Satisfiability Testing (SAT)*, 2011.

[3] I. Abío, R. Nieuwenhuis, A. Oliveras, and E. Rodríguez-Carbonell, "A parametric approach for smaller and better encodings of cardinality constraints", in *Principles and Practice of Constraint Programming (CP)*, 2013.

[4] A. Albarghouthi, L. D'Antoni, S. Drews, and A. V. Nori, "Fairsquare: Probabilistic verification of program fairness", in *International Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA)*, 2017.

[5] Y. Aono, T. Hayashi, L. Wang, S. Moriai, *et al.*, "Privacy-preserving deep learning via additively homomorphic encryption", *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 5, pp. 1333–1345, 2017.

[6] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein generative adversarial networks", in *International conference on machine learning*, PMLR, 2017, pp. 214–223.

[7] R. Asín, R. Nieuwenhuis, A. Oliveras, and E. Rodríguez-Carbonell, "Cardinality networks: A theoretical and empirical study", *Constraints*, 2011.

[8] I. S. Association *et al.*, "754-2019-IEEE Standard for Floating-Point Arithmetic", 2019.

[9] A. Athalye, N. Carlini, and D. Wagner, "Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples", in *International Conference on Machine Learning (ICML)*, 2018.

[10] M. Balunović, M. Baader, G. Singh, T. Gehr, and M. Vechev, "Certifying geometric robustness of neural networks", in *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

[11] R. Banner, I. Hubara, E. Hoffer, and D. Soudry, "Scalable methods for 8-bit training of neural networks", in *Advances in neural information processing systems (NeurIPS)*, 2018.

[12] D. Barber and C. M. Bishop, "Ensemble learning in bayesian neural networks", *Nato ASI Series F Computer and Systems Sciences*, vol. 168, pp. 215–238, 1998.

[13] E. Bareinboim and J. Pearl, "Controlling selection bias in causal inference", in *Artificial Intelligence and Statistics*, PMLR, 2012, pp. 100–108.

[14] K. E. Batcher, "Sorting networks and their applications", in *AFIPS'1968*, 1968.

[15] J. Berkson, "Limitations of the application of fourfold table analysis to hospital data", *Biometrics Bulletin*, vol. 2, no. 3, pp. 47–53, 1946.

[16] A. Boopathy, T.-W. Weng, P.-Y. Chen, S. Liu, and L. Daniel, "Cnn-cert: An efficient framework for certifying robustness of convolutional neural networks", in *Conference on Artificial Intelligence (AAAI)*, 2019.

[17] H. E. Brady, "Causation and explanation in social science", na, 2008.

[18] Z. Cai and M. Kuroki, "On identifying total effects in the presence of latent variables and selection bias", in *Proceedings of the Twenty-Fourth Conference on Uncertainty in Artificial Intelligence (UAI)*, 2008, pp. 62–69.

[19] N. Carlini, A. Athalye, N. Papernot, W. Brendel, J. Rauber, D. Tsipras, I. Goodfellow, and A. Madry, "On evaluating adversarial robustness", *arXiv preprint arXiv:1902.06705*, 2019.

[20] N. Carlini, C. Liu, Ú. Erlingsson, J. Kos, and D. Song, "The secret sharer: Evaluating and testing unintended memorization in neural networks", in *USENIX Security Symposium (USENIX Security)*, 2019, pp. 267–284.

[21] N. Carlini, C. Liu, J. Kos, Ú. Erlingsson, and D. Song, "The secret sharer: Measuring unintended neural network memorization & extracting secrets", in *USENIX Security Symposium (USENIX Security)*, 2019.

[22] N. Carlini, P. Mishra, T. Vaidya, Y. Zhang, M. Sherr, C. Shields, D. Wagner, and W. Zhou, "Hidden voice commands." In *USENIX Security Symposium (USENIX Security)*, 2016.

[23] N. Carlini, F. Tramer, E. Wallace, M. Jagielski, A. Herbert-Voss, K. Lee, A. Roberts, T. Brown, D. Song, Ú. Erlingsson, *et al.*, "Extracting training data from large language models", in *USENIX Security Symposium (USENIX Security)*, 2021.

[24] N. Carlini and D. Wagner, "Towards evaluating the robustness of neural networks", in *Symposium on Security and Privacy (S&P)*, 2017.

[25] S. Chakraborty, D. J. Fremont, K. S. Meel, S. A. Seshia, and M. Y. Vardi, "Distribution-aware sampling and weighted model counting for SAT", in *Conference on Artificial Intelligence (AAAI)*, 2014.

[26] S. Chakraborty, D. Fried, K. S. Meel, and M. Y. Vardi, "From weighted to unweighted model counting." In *IJCAI'15*, 2015.

[27] S. Chakraborty, K. S. Meel, and M. Y. Vardi, "A scalable approximate model counter", in *Principles and Practice of Constraint Programming (CP)*, 2013.

[28] S. Chakraborty, K. S. Meel, and M. Y. Vardi, "Balancing scalability and uniformity in SAT witness generator", in *Annual Design Automation Conference (DAC)*, 2014.

[29] S. Chakraborty, K. S. Meel, and M. Y. Vardi, "Algorithmic improvements in approximate counting for probabilistic inference: From linear to logarithmic SAT calls", in *IJCAI'16*, 2016.

[30] Y. Chen, C. Li, and Z. Lu, "Computing wasserstein-p distance between images with linear cost", in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 519–528.

[31] C.-H. Cheng, G. Nührenberg, and H. Ruess, "Maximum resilience of artificial neural networks", in *ATVA'17*, 2017.

[32] D. Chickering, D. Geiger, and D. Heckerman, "Learning bayesian networks: Search methods and experimental results", in *proceedings of fifth conference on artificial intelligence and statistics*, 1995, pp. 112–128.

[33] B. Chmiel, L. Ben-Uri, M. Shkolnik, E. Hoffer, R. Banner, and D. Soudry, "Neural gradients are near-lognormal: Improved quantized and sparse training", *arXiv preprint arXiv:2006.08173*, 2020.

[34] C. A. Choquette-Choo, F. Tramer, N. Carlini, and N. Papernot, "Label-only membership inference attacks", in *International Conference on Machine Learning (ICML)*, 2021.

[35] J. Cohen, E. Rosenfeld, and Z. Kolter, "Certified adversarial robustness via randomized smoothing", in *International Conference on Machine Learning (ICML)*, 2019.

[36] E. Creamer, "Authors file a lawsuit against openai for unlawfully 'ingesting' their books", https://www.theguardian.com/books/2023/jul/05/authors-file-a-lawsuit-against-openai-for-unlawfully-ingesting-their-books, Accessed: 2023-08-13, 2023.

[37] R. Daly and Q. Shen, "Methods to accelerate the learning of bayesian network structures", in *Proceedings of the 2007 UK Workshop on Computational Intelligence*, Citeseer, 2007.

[38] A. Datta, M. Fredrikson, G. Ko, P. Mardziel, and S. Sen, "Use privacy in data-driven systems: Theory and experiments with machine learnt programs", in *Conference on Computer and Communications Security (CCS)*, 2017.

[39] C. De Sa, M. Leszczynski, J. Zhang, A. Marzoev, C. R. Aberger, K. Olukotun, and C. Ré, "High-accuracy low-precision training", *arXiv preprint arXiv:1803.03383*, 2018.

[40] G. S. Dhillon, K. Azizzadenesheli, Z. C. Lipton, J. Bernstein, J. Kossaifi, A. Khanna, and A. Anandkumar, "Stochastic activation pruning for robust adversarial defense", *arXiv preprint arXiv:1803.01442*, 2018.

[41] J. Dodge, M. Sap, A. Marasović, W. Agnew, G. Ilharco, D. Groeneveld, M. Mitchell, and M. Gardner, "Documenting large webtext corpora: A case study on the colossal clean crawled corpus", *arXiv preprint arXiv:2104.08758*, 2021.

[42] E. Dohmatob, "Limitations of adversarial robustness: Strong no free lunch theorem", *arXiv*, 2018.

[43] J. M. Dudek, K. S. Meel, and M. Y. Vardi, "Combining the k-cnf and xor phase-transitions", in *IJCAI'16*, 2016.

[44] J. M. Dudek, K. S. Meel, and M. Y. Vardi, "The hard problems are almost everywhere for random CNF-XOR formulas", in *IJCAI'17*, 2017.

[45] K. Dvijotham, R. Stanforth, S. Gowal, T. Mann, and P. Kohli, "A Dual Approach to Scalable Verification of Deep Networks", in *Uncertainty in Artificial Intelligence (UAI)*, 2018.

[46] C. Dwork, M. Hardt, T. Pitassi, O. Reingold, and R. Zemel, "Fairness through awareness", in *ITCS'12*, 2012.

[47] G. K. Dziugaite, A. Drouin, B. Neal, N. Rajkumar, E. Caballero, L. Wang, I. Mitliagkas, and D. M. Roy, "In search of robust measures of generalization", in *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

[48] N. Eén and N. Sörensson, "Translating pseudo-Boolean constraints into SAT." *JSAT*, vol. 2, no. 1-4, pp. 1–26, 2006.

[49] R. Ehlers, "Formal verification of piece-wise linear feed-forward neural networks", in *ATVA'17*, 2017.

[50] S. Ermon, C. P. Gomes, A. Sabharwal, and B. Selman, "Taming the Curse of Dimensionality: Discrete Integration by Hashing and Optimization", in *International conference on machine learning (ICML)*, 2013.

[51] O. J. of the European Union, "Regulation (eu) 2016/679 of the european parliament and of the council of 27 april 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing directive 95/46/ec (general data protection regulation)",, vol. L119, pp. 1–88, 2016.

[52] L. Fan, K. W. Ng, C. Ju, T. Zhang, C. Liu, C. S. Chan, and Q. Yang, "Rethinking privacy preserving deep learning: How to evaluate and thwart privacy attacks", *Federated Learning: Privacy and Incentive*, pp. 32–50, 2020.

[53] C. Fang, H. Jia, A. Thudi, M. Yaghini, C. A. Choquette-Choo, N. Dullerud, V. Chandrasekaran, and N. Papernot, "Proof-of-learning is currently more broken than you think", in *European Symposium on Security and Privacy (EuroS&P)*, IEEE, 2023.

[54] M. Fazlyab, M. Morari, and G. J. Pappas, "Probabilistic verification and reachability analysis of neural networks via semidefinite programming", in *2019 IEEE 58th Conference on Decision and Control (CDC)*, IEEE, 2019, pp. 2726–2731.

[55] M. Feldman, S. A. Friedler, J. Moeller, C. Scheidegger, and S. Venkatasubramanian, "Certifying and removing disparate impact", in *International Conference on Knowledge Discovery and Data Mining (KDD)*, 2015.

[56] V. Feldman, "Does learning require memorization? a short tale about a long tail", in *Symposium on Theory of Computing (STOC)*, 2020.

[57] J. K. Fichte, M. Hecher, and F. Hamiti, "The model counting competition 2020", *ACM J. Exp. Algorithmics*, vol. 26, Oct. 2021, ISSN: 1084-6654.

[58] N. Ford, J. Gilmer, N. Carlini, and D. Cubuk, "Adversarial examples are a natural consequence of test error in noise", in *International conference on machine learning (ICML)*, 2019.

[59] M. Fredrikson, S. Jha, and T. Ristenpart, "Model inversion attacks that exploit confidence information and basic countermeasures", in *Conference on Computer and Communications Security (CCS)*, 2015.

[60] N. Friedman, M. Goldszmidt, and A. Wyner, "Data analysis with bayesian networks: A bootstrap approach", *arXiv preprint arXiv:1301.6695*, 2013.

[61] N. Friedman and D. Koller, "Being bayesian about network structure. a bayesian approach to structure discovery in bayesian networks", *Machine learning*, vol. 50, no. 1, pp. 95–125, 2003.

[62] C. Frogner, C. Zhang, H. Mobahi, M. Araya, and T. A. Poggio, "Learning with a wasserstein loss", *Advances in neural information processing systems*, vol. 28, 2015.

[63] S. Galhotra, Y. Brun, and A. Meliou, "Fairness testing: Testing software for discrimination", in *Joint Meeting on Foundations of Software Engineering (FSE/ESEC)*, 2017.

[64] A. Galloway, G. W. Taylor, and M. Moussa, "Attacking binarized neural networks", in *International Conference on Learning Representations (ICLR)*, 2018.

[65] Y. Gao, C. Xu, D. Wang, S. Chen, D. C. Ranasinghe, and S. Nepal, "Strip: A defence against trojan attacks on deep neural networks", *arXiv*, 2019.

[66] M. R. Gary and D. S. Johnson, "Computers and intractability: A guide to the theory of NP-completeness", 1979.

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

[68] J. Geiping, H. Bauermeister, H. Dröge, and M. Moeller, "Inverting gradients-how easy is it to break privacy in federated learning?" In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

[69] S. Geman, E. Bienenstock, and R. Doursat, "Neural networks and the bias/-variance dilemma", *Neural computation*, vol. 4, no. 1, pp. 1–58, 1992.

[70] A. Ghaffari, M. S. Tahaei, M. Tayaranian, M. Asgharian, and V. Partovi Nia, "Is integer arithmetic enough for deep learning training?" In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

[71] A. Gholami, S. Kim, Z. Dong, Z. Yao, M. W. Mahoney, and K. Keutzer, "A survey of quantization methods for efficient neural network inference", *arXiv preprint arXiv:2103.13630*, 2021.

[72] J. Gilmer, R. P. Adams, I. Goodfellow, D. Andersen, and G. E. Dahl, "Motivating the rules of the game for adversarial example research", *arXiv*, 2018.

[73] J. Gilmer, L. Metz, F. Faghri, S. S. Schoenholz, M. Raghu, M. Wattenberg, and I. Goodfellow, "Adversarial spheres", *arXiv*, 2018.

[74] C. Glymour, K. Zhang, and P. Spirtes, "Review of causal discovery methods based on graphical models", *Frontiers in genetics*, vol. 10, p. 524, 2019.

[75] D. Goldberg, "What every computer scientist should know about floating-point arithmetic", *ACM computing surveys (CSUR)*, vol. 23, no. 1, pp. 5–48, 1991.

[76] M. Goldblum, D. Tsipras, C. Xie, X. Chen, A. Schwarzschild, D. Song, A. Mądry, B. Li, and T. Goldstein, "Dataset security for machine learning: Data poisoning, backdoor attacks, and defenses", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 45, no. 2, pp. 1563–1580, 2022.

[77] C. P. Gomes, A. Sabharwal, and B. Selman, "Model counting: A new strategy for obtaining good bounds", in *Conference on Artificial Intelligence (AAAI)*, 2006.

[78] I. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples", in *International Conference on Learning Representations (ICLR)*, 2015.

[79] D. Gopinath, H. Converse, C. Pasareanu, and A. Taly, "Property inference for deep neural networks", in *International Conference on Automated Software Engineering (ASE)*, 2019.

[80] S. Gowal, S.-A. Rebuffi, O. Wiles, F. Stimberg, D. A. Calian, and T. A. Mann, "Improving robustness using generated data", *Advances in Neural Information Processing Systems*, vol. 34, pp. 4218–4233, 2021.

[81] R. Grosu and S. A. Smolka, "Monte carlo model checking", in *TACAS'05*, 2005.

[82] W. Guo, L. Wang, X. Xing, M. Du, and D. Song, "Tabor: A highly accurate approach to inspecting and restoring trojan backdoors in ai systems", *arXiv preprint arXiv:1908.01763*, 2019.

[83] M. Hardt, E. Price, and N. Srebro, "Equality of opportunity in supervised learning", *Advances in neural information processing systems*, vol. 29, 2016.

[84] W. K. Hastings, "Monte carlo sampling methods using markov chains and their applications",, 1970.

[85] J. Hayes, L. Melis, G. Danezis, and E. De Cristofaro, "Logan: Membership inference attacks against generative models", *Proceedings on Privacy Enhancing Technologies (PoPETs)*, vol. 2019, no. 1, pp. 133–152, 2019.

[86] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition", in *Computer Vision and Pattern Recognition (CVPR)*, 2016.

[87] X. He, R. Wen, Y. Wu, M. Backes, Y. Shen, and Y. Zhang, "Node-level membership inference attacks against graph neural networks", *arXiv preprint arXiv:2102.05429*, 2021.

[88] J. J. Heckman, "Econometric causality", *International statistical review*, vol. 76, no. 1, pp. 1–27, 2008.

[89] M. Hein and M. Andriushchenko, "Formal guarantees on the robustness of a classifier against adversarial manipulation", in *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

[90] J. Hoffstein, J. Pipher, and J. H. Silverman, "Ntru: A ring-based public key cryptosystem", in *Algorithmic Number Theory: Third International Symposiun, ANTS-III Portland, Oregon, USA, June 21–25, 1998 Proceedings*, Springer, 2006, pp. 267–288.

[91] S. Holtzen, G. Broeck, and T. Millstein, "Sound abstraction and decomposition of probabilistic programs", in *International conference on machine learning (ICML)*, 2018.

[92] R. Houthooft, X. Chen, Y. Duan, J. Schulman, F. De Turck, and P. Abbeel, "Curiosity-driven exploration in deep reinforcement learning via bayesian neural networks", *arXiv preprint arXiv:1605.09674*, 2016.

[93] X. Huang, M. Kwiatkowska, S. Wang, and M. Wu, "Safety verification of deep neural networks", in *International Conference on Computer Aided Verification (CAV)*, 2017.

[94] Y. Huang, S. Gupta, Z. Song, K. Li, and S. Arora, "Evaluating gradient inversion attacks and defenses in federated learning", in *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

[95] Y. Huang and M. Valtorta, "Pearl's calculus of intervention is complete", in *Proceedings of the Twenty-Second Conference on Uncertainty in Artificial Intelligence (UAI)*, 2006, pp. 217–224.

[96] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks", in *Advances in Neural Information Processing Systems (NeurIPS)*, 2016.

[97] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Quantized neural networks: Training neural networks with low precision weights and activations", *The Journal of Machine Learning Research*, vol. 18, no. 1, pp. 6869–6898, 2017.

[98] B. Hui, Y. Yang, H. Yuan, P. Burlina, N. Z. Gong, and Y. Cao, "Practical blind membership inference attack via differential comparisons", in *Network and Distributed Systems Security (NDSS)*, 2021.

[99] A. Ilyas, S. Santurkar, D. Tsipras, L. Engstrom, B. Tran, and A. Madry, "Adversarial examples are not bugs, they are features", in *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

[100] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros, "Image-to-image translation with conditional adversarial networks", in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 1125–1134.

[101] A. Jaber, J. Zhang, and E. Bareinboim, "Causal identification under markov equivalence", in *Proceedings of the 34th Conference on Uncertainty in Artificial Intelligence (UAI)*, 2018.

[102] K. Jamieson, M. Malloy, R. Nowak, and S. Bubeck, "Lil'ucb: An optimal exploration algorithm for multi-armed bandits", in *Conference on Learning Theory*, PMLR, 2014, pp. 423–439.

[103] J. Jeon, K. Lee, S. Oh, J. Ok, *et al.*, "Gradient inversion with generative image prior", in *Advances in neural information processing systems (NeurIPS)*, 2021.

[104] M. R. Jerrum and A. Sinclair, "The Markov chain Monte Carlo method: An approach to approximate counting and integration", *Approximation algorithms for NP-hard problems*, pp. 482–520, 1996.

[105] H. Jia, M. Yaghini, C. A. Choquette-Choo, N. Dullerud, A. Thudi, V. Chandrasekaran, and N. Papernot, "Proof-of-learning: Definitions and practice", in *Symposium on Security and Privacy (S&P)*, 2021.

[106] J. Jia, A. Salem, M. Backes, Y. Zhang, and N. Z. Gong, "Memguard: Defending against black-box membership inference attacks via adversarial examples", in *Computer and Communications Security (CCS)*, 2019.

[107] K. Jia and M. Rinard, "Exploiting verified neural networks via floating point numerical error", in *Static Analysis: 28th International Symposium, SAS 2021, Chicago, IL, USA, October 17–19, 2021, Proceedings 28*, Springer, 2021, pp. 191–205.

[108] Y. Jiang, D. Krishnan, H. Mobahi, and S. Bengio, "Predicting the generalization gap in deep networks with margin distributions", in *International Conference on Learning Representations (ICLR)*, 2019.

[109] N. Kalra and S. M. Paddock, "Driving to safety: How many miles of driving would it take to demonstrate autonomous vehicle reliability?" *Transportation Research Part A: Policy and Practice*, vol. 94, 2016.

[110] T. Karim, T. Furon, and M. Rousset, "Gradient-informed neural network statistical robustness estimation", in *International Conference on Artificial Intelligence and Statistics*, PMLR, 2023, pp. 323–334.

[111] G. Katz, C. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer, "Reluplex: An efficient smt solver for verifying deep neural networks", in *International Conference on Computer Aided Verification (CAV)*, 2017.

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

[113] M. Kocaoglu, A. Jaber, K. Shanmugam, and E. Bareinboim, "Characterization and learning of causal graphs with latent variables from soft interventions." In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

[114] R. Kohavi, D. H. Wolpert, *et al.*, "Bias plus variance decomposition for zero-one loss functions", in *International Conference on Machine Learning (ICML)*, 1996.

[115] S. Kolouri, Y. Zou, and G. K. Rohde, "Sliced wasserstein kernels for probability distributions", in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 5258–5267.

[116] Z. Kong, A. R. Chowdhury, and K. Chaudhuri, "Can membership inferencing be refuted?" *arXiv preprint arXiv:2303.03648*, 2023.

[117] Z. Kong, A. Roy Chowdhury, and K. Chaudhuri, "Forgeability and membership inference attacks", in *Workshop on Artificial Intelligence and Security (AISec)*, 2022.

[118] P. Koopman and B. Osyk, "Safety argument considerations for public road testing of autonomous vehicles", SAE Technical Paper, Tech. Rep., 2019.

[119] K. Krishna, G. S. Tomar, A. P. Parikh, N. Papernot, and M. Iyyer, "Thieves on Sesame street! model extraction of BERT-based apis", in *International Conference on Learning Representations (ICLR)*, 2019.

[120] A. Krizhevsky, G. Hinton, *et al.*, "Learning multiple layers of features from tiny images",, 2009.

[121] J. Kung, D. Zhang, G. van der Wal, S. Chai, and S. Mukhopadhyay, "Efficient object detection using embedded binarized neural networks", *Journal of Signal Processing Systems*, 2018.

[122] M. Kwiatkowska, G. Norman, and D. Parker, "Stochastic model checking", in *International School on Formal Methods for the Design of Computer, Communication and Software Systems*, 2007, pp. 220–270.

[123] M. Kwiatkowska, G. Norman, and D. Parker, "Prism: Probabilistic model checking for performance and reliability analysis", *ACM SIGMETRICS Performance Evaluation Review*, vol. 36, no. 4, pp. 40–45, 2009.

[124] K. G. Larsen and A. Legay, "Statistical model checking past, present, and future", in *International Symposium On Leveraging Applications of Formal Methods, Verification and Validation*, Springer, 2014, pp. 135–142.

[125] Y. LeCun, "The mnist database of handwritten digits", *http://yann. lecun. com/exdb/mnist/*, 1998.

[126] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition", *Neural computation*, vol. 1, no. 4, pp. 541–551, 1989.

[127] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition", *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[128] Y. LeCun and C. Cortes, "MNIST handwritten digit database",, 2010.

[129] M. Lecuyer, V. Atlidakis, R. Geambasu, D. Hsu, and S. Jana, "Certified robustness to adversarial examples with differential privacy", *arXiv preprint arXiv:1802.03471*, 2018.

[130] M. Lecuyer, V. Atlidakis, R. Geambasu, D. Hsu, and S. Jana, "Certified robustness to adversarial examples with differential privacy", in *Symposium on Security and Privacy (SP)*, 2019, pp. 656–672.

[131] S. Lee and E. Bareinboim, "Causal identification with matrix equations", in *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

[132] K. Leino and M. Fredrikson, "Stolen memories: Leveraging model memorization for calibrated white-box membership inference", in *USENIX Security Symposium (USENIX Security)*, 2020.

[133] J. Li, N. Li, and B. Ribeiro, "Membership inference attacks and defenses in classification models", in *Conference on Data and Application Security and Privacy (CODASPY)*, 2021.

[134] K. Li, T. Zhang, and J. Malik, "Approximate feature collisions in neural nets", in *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

[135] Z. Li and Y. Zhang, "Membership leakage in label-only exposures", in *Conference on Computer and Communications Security (CCS)*, 2021.

[136] F. Liao, M. Liang, Y. Dong, T. Pang, X. Hu, and J. Zhu, "Defense against adversarial attacks using high-level representation guided denoiser", in *Computer Vision and Pattern Recognition (CVPR)*, 2018.

[137] D. Lin, S. Talathi, and S. Annapureddy, "Fixed point quantization of deep convolutional networks", in *International conference on machine learning (ICML)*, 2016.

[138] W. Lin, Z. Yang, X. Chen, Q. Zhao, X. Li, Z. Liu, and J. He, "Robustness verification of classification deep neural networks via linear programming", in *Computer Vision and Pattern Recognition (CVPR)*, 2019.

[139] S. Linton, G. Nebe, A. Niemeyer, R. Parker, and J. Thackray, "A parallel algorithm for gaussian elimination over finite fields", *arXiv preprint arXiv:1806.04211*, 2018.

[140] H. Liu, J. Jia, W. Qu, and N. Z. Gong, "Encodermi: Membership inference against pre-trained encoders in contrastive learning", in *Computer and Communications Security (CCS)*, 2021, pp. 2081–2095.

[141] Y. Liu, S. Ma, Y. Aafer, W.-C. Lee, J. Zhai, W. Wang, and X. Zhang, "Trojaning attack on neural networks", in *Network and Distributed Systems Security (NDSS)*, 2018.

[142] Y. Liu, R. Wen, X. He, A. Salem, Z. Zhang, M. Backes, E. De Cristofaro, M. Fritz, and Y. Zhang, "Ml-doctor: Holistic risk assessment of inference attacks against machine learning models", *arXiv preprint arXiv:2102.02551*, 2021.

[143] Y. Liu, Y. Xie, and A. Srivastava, "Neural trojans", in *International Conference on Computer Design (ICCD)*, 2017.

[144] Y. Long, V. Bindschaedler, L. Wang, D. Bu, X. Wang, H. Tang, C. A. Gunter, and K. Chen, "Understanding membership inferences on well-generalized learning models", *arXiv preprint arXiv:1802.04889*, 2018.

[145] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks", *arXiv preprint arXiv:1706.06083*, 2017.

[146] S. Mahloujifar, D. I. Diochnos, and M. Mahmoody, "The curse of concentration in robust learning: Evasion and poisoning attacks from concentration of measure", in *Conference on Artificial Intelligence (AAAI)*, 2019.

[147] M. M. Marini and B. Singer, "Causality in the social sciences", *Sociological methodology*, vol. 18, pp. 347–409, 1988.

[148] F. J. Massey Jr, "The kolmogorov-smirnov test for goodness of fit", *Journal of the American statistical Association*, vol. 46, no. 253, pp. 68–78, 1951.

[149] B. McDanel, S. Teerapittayanon, and H. T. Kung, "Embedded Binarized Neural Networks", in *EWSN'17*, 2017.

[150] K. S. Meel, "Constrained counting and sampling: Bridging the gap between theory and practice", PhD thesis, Rice University, 2017.

[151] N. Mehrabi, F. Morstatter, N. Saxena, K. Lerman, and A. Galstyan, "A survey on bias and fairness in machine learning", *arXiv preprint arXiv:1908.09635*, 2019.

[152] F. Mireshghallah, K. Goyal, A. Uniyal, T. Berg-Kirkpatrick, and R. Shokri, "Quantifying privacy risks of masked language models using membership inference attacks", *arXiv preprint arXiv:2203.03929*, 2022.

[153] M. Mirman, G. Singh, and M. Vechev, "A provable defense for deep residual networks", *arXiv preprint arXiv:1903.12519*, 2019.

[154] M. Mitzenmacher and E. Upfal, "Probability and computing: randomization and probabilistic techniques in algorithms and data analysis", Cambridge university press, 2017.

[155] S. K. Murakonda and R. Shokri, "Ml privacy meter: Aiding regulatory compliance by quantifying the privacy risks of machine learning", *arXiv preprint arXiv:2007.09339*, 2020.

[156] N. Narodytska, S. P. Kasiviswanathan, L. Ryzhyk, M. Sagiv, and T. Walsh, "Verifying properties of binarized deep neural networks", in *Conference on Artificial Intelligence (AAAI)*, 2018.

[157] N. Narodytska, A. Shrotri, K. S. Meel, A. Ignatiev, and J. Marques-Silva, "Assessing heuristic machine learning explanations with model counting", in *International Conference on Theory and Applications of Satisfiability Testing (SAT)*, 2019.

[158] M. Nasr, R. Shokri, and A. Houmansadr, "Machine learning with membership privacy using adversarial regularization", in *Computer and Communications Security (CCS)*, 2018, pp. 634–646.

[159] M. Nasr, R. Shokri, and A. Houmansadr, "Comprehensive privacy analysis of deep learning: Passive and active white-box inference attacks against centralized and federated learning", in *Symposium on Security and Privacy (S&P)*, 2019.

[160] B. Neal, C.-W. Huang, and S. Raghupathi, "Realcause: Realistic causal inference benchmarking", *arXiv preprint arXiv:2011.15007*, 2020.

[161] B. Neal, S. Mittal, A. Baratin, V. Tantia, M. Scicluna, S. Lacoste-Julien, and I. Mitliagkas, "A modern take on the bias-variance tradeoff in neural networks", *arXiv preprint arXiv:1810.08591*, 2018.

[162] R. M. Neal, "Probabilistic inference using markov chain monte carlo methods",, 1993.

[163] U. Ozbulak, M. Gasparyan, S. Rao, W. De Neve, and A. Van Messem, "Exact feature collisions in neural networks", *arXiv preprint arXiv:2205.15763*, 2022.

[164] N. Papernot, F. Faghri, N. Carlini, I. Goodfellow, R. Feinman, A. Kurakin, C. Xie, Y. Sharma, T. Brown, A. Roy, A. Matyasko, V. Behzadan, K. Hambardzumyan, Z. Zhang, Y.-L. Juang, Z. Li, R. Sheatsley, A. Garg, J. Uesato, W. Gierke, Y. Dong, D. Berthelot, P. Hendricks, J. Rauber, and R. Long, "Technical report on the cleverhans v2.1.0 adversarial examples library", *arXiv preprint arXiv:1610.00768*, 2018.

[165] N. Papernot, P. McDaniel, and I. Goodfellow, "Transferability in machine learning: From phenomena to black-box attacks using adversarial samples", *arXiv*, 2016.

[166] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami, "The limitations of deep learning in adversarial settings", in *European Symposium on Security and Privacy (EuroS&P)*, 2016.

[167] N. Papernot, P. McDaniel, X. Wu, S. Jha, and A. Swami, "Distillation as a defense to adversarial perturbations against deep neural networks", in *SP'16*, 2016.

[168] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch", in *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

[169] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library", in *Advances in Neural Information Processing Systems (NeurIPS)*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[170] J. Pearl, "Causality", Cambridge university press, 2009.

[171] J. Pearl, "On a class of bias-amplifying variables that endanger effect estimates", *arXiv preprint arXiv:1203.3503*, 2012.

[172] J. Pearl *et al.*, "Models, reasoning and inference", *Cambridge, UK: CambridgeUniversityPress*, vol. 19, p. 2, 2000.

[173] K. Pei, Y. Cao, J. Yang, and S. Jana, "Deepxplore: Automated whitebox testing of deep learning systems", in *SOSP'17*, 2017.

[174] S. Peng, W. Xu, C. Cornelius, M. Hull, K. Li, R. Duggal, M. Phute, J. Martin, and D. H. Chau, "Robust principles: Architectural design principles for adversarially robust cnns", *arXiv preprint arXiv:2308.16258*, 2023.

[175] T. Philipp and P. Steinke, "Pblib – a library for encoding pseudo-boolean constraints into cnf", in *Theory and Applications of Satisfiability Testing – SAT 2015*, ser. Lecture Notes in Computer Science, 2015.

[176] J. Pineau, P. Vincent-Lamarre, K. Sinha, V. Larivière, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and H. Larochelle, "Improving reproducibility in machine learning research (a report from the neurips 2019 reproducibility program)", *The Journal of Machine Learning Research*, vol. 22, no. 1, pp. 7459–7478, 2021.

[177] Y. Pote, S. Joshi, and K. S. Meel, "Phase transition behavior of cardinality and xor constraints", in *International Joint Conferences on Artificial Intelligence*, 2019.

[178] L. Pulina and A. Tacchella, "An abstraction-refinement approach to verification of artificial neural networks", in *International Conference on Computer Aided Verification (CAV)*, 2010.

[179] PyTorch, "Reference implementation of lenet5 for forging", https://github.com/cleverhans-lab/Forging, Accessed: 2023-02-28, 2022.

[180] PyTorch Contributors, "PyTorch reproducibility documentation", https://pytorch.org/docs/1.13/notes/randomness.html?highlight=reproducibility, Accessed: 2023-04-28, 2022.

[181] J. Queen, "Sarah silverman sues meta, openai for copyright infringement", hhttps://www.reuters.com/legal/sarah-silverman-sues-meta-openai-copyright-infringement-2023-07-09/, Accessed: 2023-08-13, 2023.

[182] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks", *arXiv preprint arXiv:1511.06434*, 2015.

[183] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu, "Exploring the limits of transfer learning with a unified text-to-text transformer", *The Journal of Machine Learning Research*, vol. 21, no. 1, pp. 5485–5551, 2020.

[184] A. Raghunathan, J. Steinhardt, and P. Liang, "Certified defenses against adversarial examples", in *International Conference on Learning Representations (ICLR)*, 2018.

[185] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "Xnor-net: Imagenet classification using binary convolutional neural networks", in *European Conference on Computer Vision*, 2016.

[186] G. Repo, "Reference implementation of resnet-mini for forging", https://github.com/nikhilbarhate99/Image-Classifiers, Accessed: 2023-04-10, 2018.

[187] A. Roberts, H. W. Chung, A. Levskaya, G. Mishra, J. Bradbury, D. Andor, S. Narang, B. Lester, C. Gaffney, A. Mohiuddin, C. Hawthorne, A. Lewkowycz, A. Salcianu, M. van Zee, J. Austin, S. Goodman, L. B. Soares, H. Hu, S. Tsvyashchenko, A. Chowdhery, J. Bastings, J. Bulian, X. Garcia, J. Ni, A. Chen, K. Kenealy, J. H. Clark, S. Lee, D. Garrette, J. Lee-Thorp, C. Raffel, N. Shazeer, M. Ritter, M. Bosma, A. Passos, J. Maitin-Shepard, N. Fiedel, M. Omernick, B. Saeta, R. Sepassi, A. Spiridonov, J. Newlan, and A. Gesmundo, "Scaling up models and data with `t5x` and `seqio`", *arXiv preprint arXiv:2203.17189*, 2022. [Online]. Available: https://arxiv.org/abs/2203.17189.

[188] J. M. Robins, A. Rotnitzky, and D. O. Scharfstein, "Sensitivity analysis for selection bias and unmeasured confounding in missing data and causal inference models", in *Statistical models in epidemiology, the environment, and clinical trials*, Springer, 2000, pp. 1–94.

[189] R. W. Robinson, "Counting unlabeled acyclic digraphs", in *Combinatorial mathematics V*, Springer, 1977, pp. 28–43.

[190] P. R. Rosenbaum, "Sensitivity analysis in observational studies", *Encyclopedia of statistics in behavioral science*, 2005.

[191] W. Ruan, X. Huang, and M. Kwiatkowska, "Reachability analysis of deep neural networks with provable guarantees", in *IJCAI'18*, 2018.

[192] Y. Rubner, C. Tomasi, and L. J. Guibas, "The earth mover's distance as a metric for image retrieval", *International journal of computer vision*, vol. 40, pp. 99–121, 2000.

[193] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge", *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

[194] A. Sablayrolles, M. Douze, C. Schmid, Y. Ollivier, and H. Jégou, "White-box vs black-box: Bayes optimal strategies for membership inference", in *International Conference on Machine Learning (ICML)*, 2019.

[195] A. Salem, G. Cherubin, D. Evans, B. Köpf, A. Paverd, A. Suri, S. Tople, and S. Zanella-Béguelin, "Sok: Let the privacy games begin! a unified treatment of data inference privacy in machine learning", *arXiv preprint arXiv:2212.10986*, 2022.

[196] A. Salem, Y. Zhang, M. Humbert, M. Fritz, and M. Backes, "Ml-leaks: Model and data independent membership inference attacks and defenses on machine learning models", in *Network and Distributed Systems Security Symposium (NDSS)*, Internet Society, 2019.

[197] A. Sauer, K. Schwarz, and A. Geiger, "Stylegan-xl: Scaling stylegan to large diverse datasets", in *ACM SIGGRAPH 2022 conference proceedings*, 2022, pp. 1–10.

[198] R. Scheines, "An introduction to causal inference",, 1997.

[199] G. Schwarz, "Estimating the dimension of a model", *The annals of statistics*, pp. 461–464, 1978.

[200] M. Scutari, "Learning bayesian networks with the bnlearn R package", *Journal of Statistical Software*, vol. 35, no. 3, pp. 1–22, 2010.

[201] M. Scutari and R. Nagarajan, "On identifying significant edges in graphical models of molecular networks", *arXiv preprint arXiv:1104.0896*, 2011.

[202] K. Shanmugam, M. Kocaoglu, A. G. Dimakis, and S. Vishwanath, "Learning causal graphs with small interventions", in *Advances in Neural Information Processing Systems (NeurIPS)*, 2015.

[203] A. Sharma and E. Kiciman, "Dowhy: An end-to-end library for causal inference", *arXiv preprint arXiv:2011.04216*, 2020.

[204] A. Sharma, E. Kiciman, *et al.*, "DoWhy: A Python package for causal inference", https://github.com/microsoft/dowhy, 2019.

[205] A. Shih, A. Darwiche, and A. Choi, "Verifying binarized neural networks by angluin-style learning", in *International Conference on Theory and Applications of Satisfiability Testing (SAT)*, 2019.

[206] R. Shokri, M. Strobel, and Y. Zick, "On the privacy risks of model explanations", in *AAAI/ACM Conference on AI, Ethics, and Society*, 2021, pp. 231–241.

[207] R. Shokri, M. Stronati, C. Song, and V. Shmatikov, "Membership inference attacks against machine learning models", in *Symposium on Security and Privacy (S&P)*, 2017.

[208] I. Shpitser and J. Pearl, "Identification of joint interventional distributions in recursive semi-markovian causal models", in *Proceedings of the National Conference on Artificial Intelligence*, Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, vol. 21, 2006, p. 1219.

[209] I. Shumailov, Z. Shumaylov, D. Kazhdan, Y. Zhao, N. Papernot, M. A. Erdogdu, and R. J. Anderson, "Manipulating SGD with data ordering attacks", in *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

[210] E. H. Simpson, "The interpretation of interaction in contingency tables", *Journal of the Royal Statistical Society: Series B (Methodological)*, vol. 13, no. 2, pp. 238–241, 1951.

[211] G. Singh, T. Gehr, M. Mirman, M. Püschel, and M. T. Vechev, "Fast and effective robustness certification." In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

[212] G. Singh, T. Gehr, M. Püschel, and M. Vechev, "An abstract domain for certifying neural networks", *Principles of Programming Languages (POPL)*, 2019.

[213] G. Singh, T. Gehr, M. Püschel, and M. Vechev, "Boosting robustness certification of neural networks", in *International Conference on Learning Representations (ICLR)*, 2019.

[214] C. Sinz, "Towards an optimal cnf encoding of boolean cardinality constraints", in *Principles and Practice of Constraint Programming (CP)*, 2005.

[215] L. Song and P. Mittal, "Systematic evaluation of privacy risks of machine learning models", in *USENIX Security Symposium (USENIX Security)*, 2021.

[216] L. Song, R. Shokri, and P. Mittal, "Membership inference attacks against adversarially robust deep learning models", in *IEEE Security and Privacy Workshops (SPW)*, IEEE, 2019, pp. 50–56.

[217] L. Song, R. Shokri, and P. Mittal, "Privacy risks of securing machine learning models against adversarial examples", in *Computer and Communications Security (CCS)*, 2019.

[218] M. Soos and K. S. Meel, "BIRD: Engineering an efficient CNF-XOR SAT solver and its applications to approximate model counting", in *Conference on Artificial Intelligence (AAAI)*, 2019.

[219] N. A. Spielberg, M. Brown, N. R. Kapania, J. C. Kegelman, and J. C. Gerdes, "Neural network vehicle models for high-performance automated driving", *Science Robotics*, vol. 4, no. 28, eaaw1975, 2019.

[220] P. Spirtes, C. Glymour, and R. Scheines, "Causation, prediction, and search", MIT press, 2001.

[221] L. Stockmeyer, "The complexity of approximate counting", in *Symposium on Theory of Computing (STOC)*, 1983.

[222] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks", *arXiv preprint arXiv:1312.6199*, 2013.

[223] R. Tatman, J. VanderPlas, and S. Dane, "A practical taxonomy of reproducibility for machine learning research", 2018.

[224] S. Teerapittayanon, B. McDanel, and H.-T. Kung, "Branchynet: Fast inference via early exiting from deep neural networks", in *ICPR'16*, 2016.

[225] E. Thorn, S. C. Kimmel, M. Chaka, B. A. Hamilton, *et al.*, "A framework for automated driving system testable cases and scenarios", United States. Department of Transportation. National Highway Traffic Safety, Tech. Rep., 2018.

[226] A. Thudi, H. Jia, I. Shumailov, and N. Papernot, "On the necessity of auditable algorithmic definitions for machine unlearning", in *USENIX Security Symposium (USENIX Security)*, 2022.

[227] K. Tit, T. Furon, and M. Rousset, "Efficient statistical assessment of neural network corruption robustness", *Advances in Neural Information Processing Systems*, vol. 34, pp. 9253–9263, 2021.

[228] V. Tjeng, K. Xiao, and R. Tedrake, "Evaluating robustness of neural networks with mixed integer programming",, 2019.

[229] S. Toda, "On the computational power of pp and (+) p", in *FOCS'1989*, 1989.

[230] S. Tople, A. Sharma, and A. Nori, "Alleviating privacy attacks via causal learning", in *International Conference on Machine Learning (ICML)*, 2020.

[231] F. Tramèr, A. Kurakin, N. Papernot, I. Goodfellow, D. Boneh, and P. McDaniel, "Ensemble adversarial training: Attacks and defenses", in *International Conference on Learning Representations (ICLR)*, 2018.

[232] H.-D. Tran, S. Choi, H. Okamoto, B. Hoxha, G. Fainekos, and D. Prokhorov, "Quantitative verification for neural networks using probstars", in *Proceedings of the 26th ACM International Conference on Hybrid Systems: Computation and Control*, 2023, pp. 1–12.

[233] S. Triantafillou, V. Lagani, C. Heinze-Deml, A. Schmidt, J. Tegner, and I. Tsamardinos, "Predicting causal relationships from biological data: Applying automated causal discovery on mass cytometry data of human immune cells", *Scientific reports*, vol. 7, no. 1, pp. 1–11, 2017.

[234] S. Truex, L. Liu, M. E. Gursoy, L. Yu, and W. Wei, "Demystifying membership inference attacks in machine learning as a service", *IEEE Transactions on Services Computing*, 2019.

[235] M. C. Tschantz, S. Sen, and A. Datta, "Sok: Differential privacy as a causal property", in *Symposium on Security and Privacy (S&P)*, 2020.

[236] G. S. Tseitin, "On the complexity of derivation in propositional calculus", in *Automation of reasoning*, 1983.

[237] D. Tsipras, S. Santurkar, L. Engstrom, A. Turner, and A. Madry, "Robustness may be at odds with accuracy", *arXiv preprint arXiv:1805.12152*, 2018.

[238] "UCI machine learning repository", http://archive.ics.uci.edu/ml, 2017.

[239] J. Uesato, B. O'Donoghue, P. Kohli, and A. Oord, "Adversarial risk and the dangers of evaluating against weak attacks", in *International Conference on Machine Learning (ICML)*, 2018.

[240] L. G. Valiant, "The complexity of enumeration and reliability problems", *SIAM Journal on Computing*, 1979.

[241] S. Verma and J. Rubin, "Fairness definitions explained", in *Proceedings of the international workshop on software fairness*, 2018, pp. 1–7.

[242] A. Wald, "Sequential tests of statistical hypotheses", *The annals of mathematical statistics*, vol. 16, no. 2, pp. 117–186, 1945.

[243] S. Wang, K. Pei, J. Whitehouse, J. Yang, and S. Jana, "Formal security analysis of neural networks using symbolic intervals", in *USENIX Security Symposium (USENIX Security)*, 2018.

[244] Z. Wang, T. Pang, C. Du, M. Lin, W. Liu, and S. Yan, "Better diffusion models further improve adversarial training", in *International Conference on Machine Learning*, PMLR, 2023, pp. 36 246–36 263.

[245] S. Webb, T. Rainforth, Y. W. Teh, and M. P. Kumar, "A statistical approach to assessing neural network robustness", in *International Conference on Learning Representations (ICLR)*, 2019.

[246] T.-W. Weng, H. Zhang, H. Chen, Z. Song, C.-J. Hsieh, D. Boning, I. S. Dhillon, and L. Daniel, "Towards fast computation of certified robustness for relu networks", *arXiv preprint arXiv:1804.09699*, 2018.

[247] T.-W. Weng, H. Zhang, P.-Y. Chen, J. Yi, D. Su, Y. Gao, C.-J. Hsieh, and L. Daniel, "Evaluating the robustness of neural networks: An extreme value theory approach", *arXiv preprint arXiv:1801.10578*, 2018.

[248] E. Wong and Z. Kolter, "Provable defenses against adversarial examples via the convex outer adversarial polytope", in *International Conference on Machine Learning (ICML)*, 2018.

[249] H. Wu and X. Liu, "Dynamic bayesian networks modeling for inferring genetic regulatory networks by search strategy: Comparison between greedy hill climbing and mcmc methods", in *Proc. of World Academy of Science, Engineering and Technology*, Citeseer, vol. 34, 2008, pp. 224–234.

[250] J. Yang and K. S. Meel, "Engineering an efficient pb-xor solver", in *27th International Conference on Principles and Practice of Constraint Programming (CP 2021)*, Schloss-Dagstuhl-Leibniz Zentrum für Informatik, 2021.

[251] Z. Yang, Y. Yu, C. You, J. Steinhardt, and Y. Ma, "Rethinking bias-variance trade-off for generalization of neural networks", in *International Conference on Machine Learning (ICML)*, PMLR, 2020, pp. 10 767–10 777.

[252] J. Ye, A. Maddi, S. K. Murakonda, V. Bindschaedler, and R. Shokri, "Enhanced membership inference attacks against machine learning models", in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, 2022, pp. 3093–3106.

[253] X. Ye, P. Dai, J. Luo, X. Guo, Y. Qi, J. Yang, and Y. Chen, "Accelerating cnn training by pruning activation gradients", in *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XXV 16*, Springer, 2020, pp. 322–338.

[254] S. Yeom, I. Giacomelli, M. Fredrikson, and S. Jha, "Privacy risk in machine learning: Analyzing the connection to overfitting", in *Computer Security Foundations Symposium (CSF)*, 2018.

[255] H. Yin, A. Mallya, A. Vahdat, J. M. Alvarez, J. Kautz, and P. Molchanov, "See through gradients: Image batch recovery via gradinversion", in *Computer Vision and Pattern Recognition (CVPR)*, 2021.

[256] H. L. Younes, "Verification and planning for stochastic processes with asynchronous events", PhD thesis, Carnegie Mellon University, 2005.

[257] M. B. Zafar, I. Valera, M. G. Rodriguez, and K. P. Gummadi, "Fairness constraints: Mechanisms for fair classification", in *Artificial Intelligence and Statistics (AISTATS)*, 2015.

[258] S. Zanella-Béguelin, L. Wutschitz, S. Tople, V. Rühle, A. Paverd, O. Ohrimenko, B. Köpf, and M. Brockschmidt, "Analyzing information leakage of updates to natural language models", in *Conference on Computer and Communications Security (CCS)*, 2020, pp. 363–375.

[259] H. Zhang, T.-W. Weng, P.-Y. Chen, C.-J. Hsieh, and L. Daniel, "Efficient neural network robustness certification with general activation functions", in *NeurIPS'18*, 2018.

[260] M. Zhang, Z. Ren, Z. Wang, P. Ren, Z. Chen, P. Hu, and Y. Zhang, "Membership inference attacks against recommender systems", in *Computer and Communications Security (CCS)*, 2021, pp. 864–879.

[261] R. Zhang, J. Liu, Y. Ding, Z. Wang, Q. Wu, and K. Ren, ""adversarial examples" for proof-of-learning", in *Symposium on Security and Privacy (S&P)*, 2022.

[262] J. Zhu and M. Blaschko, "R-gap: Recursive gradient attack on privacy", *arXiv preprint arXiv:2010.07733*, 2020.

[263] L. Zhu, Z. Liu, and S. Han, "Deep leakage from gradients", in *Advances in neural information processing systems (NeurIPS)*, 2019.

Appendix A

Causal Analysis of Membership Inference Attacks Appendix

A.1 Detailed Analysis of Causal Queries

We provide more detailed explanations of our results analyzing membership inference (MI) attack causes in this section. Our causal models are available in Appendix A.2.

In Table A.1, we show the different configurations we trained.

A.1.1 Analysis of MI Attacks

Variance and Bias. We link the variance from the bias-variance decomposition with the "closeness" of the shadow models' prediction. We find that the variance generally plays a role in the MI attack, as our causal graphs identify a path from it to the MI attack accuracy. In particular, the variance on unseen samples has generally a larger impact for the multiple shadow model attack and the single shadow model attack with the label feature as input.

CE vs. MSE-trained models. The MI attacks have different mechanisms not just per attack but also depending on the loss function. We find that the variances for MSE models do not have a significant impact on the MI attack performance.

Table A.1. Different configurations of models we trained and analyzed.

Dataset	CE		MSE	
	With Scheduler	Without Scheduler	With Scheduler	Without Scheduler
CIFAR10	✓	✓	✓	✓
CIFAR100	✓	✓	-	-
MNIST	✓	✓	✓	✓

We observe that for MSE-trained models, the variances of both the training data (members) and testing data (non-members) are typically smaller than their CE counterpart. This means that there is less variance among models and, thus, for MSE-trained models, shadow models' prediction vectors would have a similar distribution to that of the target model. This case explains the prior works' intuition that one does not require multiple shadow models—even one shadow model captures the behaviour of the target model closely.

Training set size and model complexity. For all of the evaluated attacks and loss functions, we find that larger training set size causes a lower MI attack accuracy. We also validate that a larger model complexity causes a better MI attack performance. The changes in these features are related to generalization, not only the MI attack performance. Such findings validate the prior work hypotheses (Q3 and Q4 in Table 4.3). Our analysis, though, singles out the causal effect of the training size on the MI attack accuracy, when it is independent of the model complexity. If we simultaneously changed both of them, we would be able to find a sweet spot of the best MI attack accuracy and how well the model generalizes. While this has been studied in prior work, with our method we can confirm how these two factors independently influence the privacy leakage.

Multiple Shadow Model Attack. For CE-trained models, a larger overfitting gap (`AccDiff`) causes the MI attack accuracy to increase (`ShadowAcc`), even when controlling for bias. This validates Q1 from prior work (Table A.2). The differences in the behaviour of the model, e.g., its unique distribution of the prediction vector, influences the attack performance. We find that the variance of the prediction vectors for the training set (members) causes the accuracy of the multiple shadow model attack to increase very slightly. The differences in the prediction vector of the non-members (as measured by `TestVar`) has a greater effect on the MI attack accuracy. This validates the prior work hypothesis (Q2) that the differences in the shadow and target model affect the MI attack. The estimated ATE of `TrainVar` on `ShadowAcc` is 0.02, whereas that of `TestVar` is 0.94 (Table A.2). For MSE-trained models, the gap in `AccDiff` does not cause an increase in the multiple shadow model attack accuracy–the inferred model does not have a causal path to the MI attack. The variance in the non-member predictions has a causal effect on the MI attack

accuracy, though it is less than for CE-trained models (Table A.3). There is no causal effect of the variance of the members on the MI attack accuracy. These two findings invalidate prior work hypotheses for MSE-trained models.

Table A.2. We compute the average effect on the 6 evaluated MI attacks of the causes mentioned in prior works for CE-trained models.

Attack	Feature	ATE	p-value
MLLeakAcc	AccDiff	0.1798	4.82E-58
MLLeakAcc	CentroidDist	0.0000	0.00E+00
MLLeakAcc	LossDiff	0.0000	0.00E+00
MLLeakAcc	NumParams	0.1609	1.42E-04
MLLeakAcc	TestBias	3.9476	3.55E-06
MLLeakAcc	TestVar	0.8342	1.26E-06
MLLeakAcc	TrainBias	0.0000	0.00E+00
MLLeakAcc	TrainSize	-0.0890	1.24E-06
MLLeakAcc	TrainVar	-0.3385	4.25E-02
MLLeakAcc-l	AccDiff	0.1817	2.03E-95
MLLeakAcc-l	CentroidDist	-0.2877	1.53E-07
MLLeakAcc-l	LossDiff	0.0000	0.00E+00
MLLeakAcc-l	NumParams	0.1607	1.64E-05
MLLeakAcc-l	TestBias	3.3946	5.15E-05
MLLeakAcc-l	TestVar	0.8382	3.30E-04
MLLeakAcc-l	TrainBias	0.0000	0.00E+00
MLLeakAcc-l	TrainSize	-0.1052	2.10E-09
MLLeakAcc-l	TrainVar	-0.2395	1.95E-02
MemGuardAcc	AccDiff	0.0805	5.04E-11
MemGuardAcc	CentroidDist	0.1410	6.87E-16
MemGuardAcc	LossDiff	0.0000	0.00E+00
MemGuardAcc	NumParams	0.0393	1.91E-02
MemGuardAcc	TestBias	1.7732	4.67E-03
MemGuardAcc	TestVar	0.1690	5.56E-02
MemGuardAcc	TrainBias	0.0000	0.00E+00

Attack	Feature	ATE	p-value
MemGuardAcc	TrainSize	-0.0587	1.77E-09
MemGuardAcc	TrainVar	-0.0345	1.99E-01
ShadowAcc	AccDiff	0.2949	1.54E-136
ShadowAcc	LossDiff	0.0000	0.00E+00
ShadowAcc	NumParams	0.1536	1.03E-05
ShadowAcc	TestBias	2.8590	3.14E-05
ShadowAcc	TestVar	0.9483	1.40E-04
ShadowAcc	TrainBias	0.0000	0.00E+00
ShadowAcc	TrainSize	-0.1137	2.03E-13
ShadowAcc	TrainVar	0.0243	3.20E-02
ThreshAcc	AccDiff	0.2715	5.63E-83
ThreshAcc	LossDiff	1.4711	2.48E-01
ThreshAcc	NumParams	0.1374	1.33E-01
ThreshAcc	TestBias	1.6541	1.85E-04
ThreshAcc	TestVar	1.0284	1.44E-03
ThreshAcc	TrainBias	0.0000	0.00E+00
ThreshAcc	TrainSize	-0.0893	2.64E-09
ThreshAcc	TrainVar	0.2026	2.19E-03
MLLeakTop3Acc	AccDiff	0.1773	3.14E-59
MLLeakTop3Acc	CentroidDist	0.2741	8.65E-21
MLLeakTop3Acc	LossDiff	0.0000	0.00E+00
MLLeakTop3Acc	NumParams	0.1415	8.46E-02
MLLeakTop3Acc	TestBias	3.8692	1.36E-06
MLLeakTop3Acc	TestVar	0.7784	6.85E-07
MLLeakTop3Acc	TrainBias	0.0000	0.00E+00
MLLeakTop3Acc	TrainSize	-0.0937	8.03E-08
MLLeakTop3Acc	TrainVar	-0.3431	3.55E-02
MLLeakTop3Acc-l	AccDiff	0.1578	4.96E-54
MLLeakTop3Acc-l	CentroidDist	0.2753	6.41E-19
MLLeakTop3Acc-l	LossDiff	0.0000	0.00E+00
MLLeakTop3Acc-l	NumParams	0.1602	6.13E-05

Attack	Feature	ATE	p-value
MLLeakTop3Acc-l	TestBias	3.9796	9.55E-07
MLLeakTop3Acc-l	TestVar	0.7538	4.85E-07
MLLeakTop3Acc-l	TrainBias	0.0000	0.00E+00
MLLeakTop3Acc-l	TrainSize	-0.0971	5.57E-08
MLLeakTop3Acc-l	TrainVar	-0.3991	2.93E-02

Table A.3. We compute the average effect on the 6 evaluated attacks of each features over the MSE-trained models.

Attack	Feature	ATE	p-value
MLLeakAcc	AccDiff	0.0000	0.00E+00
MLLeakAcc	CentroidDist	0.0000	0.00E+00
MLLeakAcc	LossDiff	0.2719	7.31E-63
MLLeakAcc	NumParams	0.1787	1.18E-01
MLLeakAcc	TestBias	0.0000	0.00E+00
MLLeakAcc	TestVar	0.0000	0.00E+00
MLLeakAcc	TrainBias	0.0000	0.00E+00
MLLeakAcc	TrainSize	-0.0857	3.96E-03
MLLeakAcc	TrainVar	-0.0472	9.27E-01
MLLeakAcc-l	AccDiff	0.0000	0.00E+00
MLLeakAcc-l	CentroidDist	0.0000	0.00E+00
MLLeakAcc-l	LossDiff	0.3416	1.25E-103
MLLeakAcc-l	NumParams	0.1647	6.61E-02
MLLeakAcc-l	TestBias	0.0000	0.00E+00
MLLeakAcc-l	TestVar	0.0000	0.00E+00
MLLeakAcc-l	TrainBias	0.0000	0.00E+00
MLLeakAcc-l	TrainSize	-0.1015	2.11E-03
MLLeakAcc-l	TrainVar	0.0618	5.31E-01
MemGuardAcc	AccDiff	0.0000	0.00E+00
MemGuardAcc	CentroidDist	0.0903	9.28E-01
MemGuardAcc	LossDiff	0.1278	6.82E-03

Attack	Feature	ATE	p-value
MemGuardAcc	NumParams	0.0469	1.50E-01
MemGuardAcc	TestBias	0.0000	0.00E+00
MemGuardAcc	TestVar	0.0000	0.00E+00
MemGuardAcc	TrainBias	0.0000	0.00E+00
MemGuardAcc	TrainSize	-0.0371	9.67E-03
MemGuardAcc	TrainVar	0.0023	9.04E-01
ShadowAcc	AccDiff	0.0000	0.00E+00
ShadowAcc	LossDiff	0.6194	2.86E-01
ShadowAcc	NumParams	-0.0055	9.43E-03
ShadowAcc	TestBias	0.1960	2.39E-01
ShadowAcc	TestVar	0.2057	1.25E-01
ShadowAcc	TrainBias	0.6174	1.28E-05
ShadowAcc	TrainSize	-0.0881	4.25E-03
ShadowAcc	TrainVar	0.0277	3.94E-01
ThreshAcc	AccDiff	0.0000	0.00E+00
ThreshAcc	LossDiff	-0.6702	1.20E-12
ThreshAcc	NumParams	0.0111	3.30E-02
ThreshAcc	TestBias	0.0631	3.61E-01
ThreshAcc	TestVar	0.2835	3.88E-01
ThreshAcc	TrainBias	-0.2563	3.44E-05
ThreshAcc	TrainSize	-0.0954	2.91E-03
ThreshAcc	TrainVar	0.0450	2.47E-01
MLLeakTop3Acc	AccDiff	0.0000	0.00E+00
MLLeakTop3Acc	CentroidDist	0.0000	0.00E+00
MLLeakTop3Acc	LossDiff	0.2147	8.43E-10
MLLeakTop3Acc	NumParams	0.1107	6.76E-02
MLLeakTop3Acc	TestBias	-0.0580	9.64E-02
MLLeakTop3Acc	TestVar	0.2350	5.08E-02
MLLeakTop3Acc	TrainBias	0.0000	0.00E+00
MLLeakTop3Acc	TrainSize	-0.0888	1.72E-03
MLLeakTop3Acc	TrainVar	-0.1551	7.91E-01

Attack	Feature	ATE	p-value
MLLeakTop3Acc-l	AccDiff	0.0000	0.00E+00
MLLeakTop3Acc-l	CentroidDist	0.0000	0.00E+00
MLLeakTop3Acc-l	LossDiff	0.2783	1.46E-65
MLLeakTop3Acc-l	NumParams	0.1645	7.26E-02
MLLeakTop3Acc-l	TestBias	0.0000	0.00E+00
MLLeakTop3Acc-l	TestVar	0.0000	0.00E+00
MLLeakTop3Acc-l	TrainBias	0.0000	0.00E+00
MLLeakTop3Acc-l	TrainSize	-0.0935	6.53E-03
MLLeakTop3Acc-l	TrainVar	-0.0627	8.80E-01

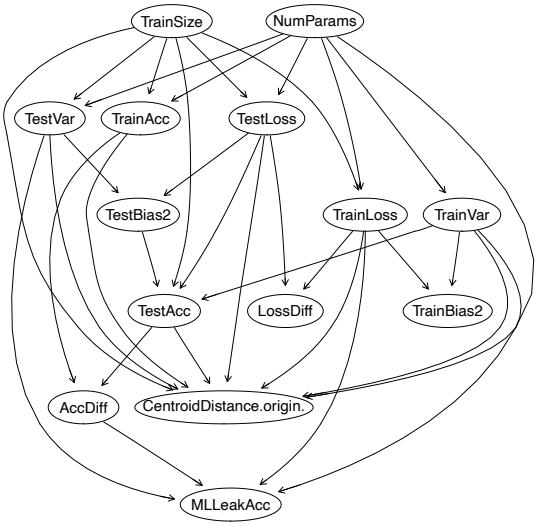
Single Shadow Model Attacks. The largest influence on the single shadow model accuracy (MLLeakTop3Acc (-l)) is the centroid distance between members and non-members (CentroidDist), thus confirming prior work hypothesis (Q8). Our approach singles out the effect of the CentroidDist from other variables such as NumParams and TrainSize which indirectly affect the CentroidDist itself. In Table A.2, the estimated ATE of the centroid distance on the MLLeakTop3Acc is 0.27. We find that the variance of the outputs of the models is a cause for the single shadow model attack, to various degrees depending on the type of attack. Prior work overlooks the differences in the prediction vectors between the target and shadow model. Thus, our analysis refutes prior work (Q5). We also refute the hypothesis that there are no differences in taking only the top-3 prediction vs. the whole prediction vector (Q7). There are a number of key differences in the causes of these variations of the single shadow model attack. The variance in the non-members' prediction vectors (TestVar) is a cause for the single shadow model attack that uses the whole prediction vector and the label as input features (MLLeakAcc-l). Interestingly, the accuracy of the attacks that take the top-3 predictions (MLLeakTop3Acc and MLLeakTop3Acc-l) is less sensitive to the variance of the prediction vectors compared to the single shadow model attack that uses the whole prediction vector, as well as the multiple shadow model attack (Table A.2). Our observation is that the variance influences the attacks that consider the whole prediction vector compared to ones that take only the top predictions as models agree on top predictions more than on

the last predictions. A larger causal effect of the variance means that the attack is sensitive to the specific changes in the prediction vector influenced by the dataset or randomness. Thus, attacks that are robust to these changes on average can more readily transfer membership information beyond the dataset and architecture of the target model.

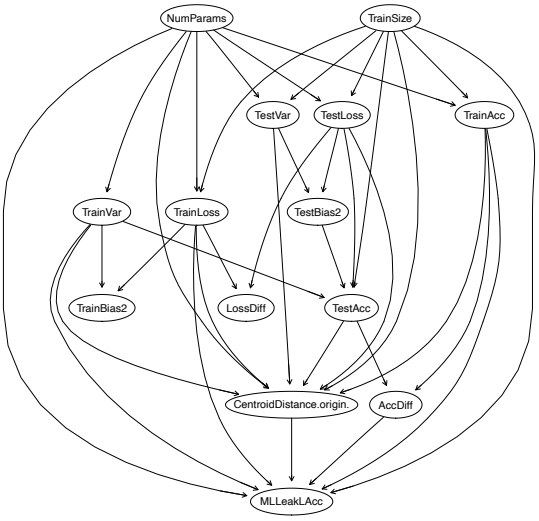
Threshold-based Attack. `LossDiff` is a significant cause for the threshold attack accuracy, as expected. On average, the variance of the prediction vectors significantly influence the average performance of the loss-based attack. We find the ATE of the `LossDiff` on the `ThreshAcc` to be close to 1.37 (Table A.2). On closer inspection, beyond the prior work hypothesis, we find that there are other causes. For instance, the variance of the prediction vectors causes the MI attack accuracy. Both `TrainVar` and `TestVar` have an estimated ATE of around 0.20 and 1.02, respectively. For models trained with MSE, the train-to-test loss difference (`LossDiff`) consistently has a causal effect on the MI attack performance rather than the train-to-test accuracy gap.

A.2 Causal Models

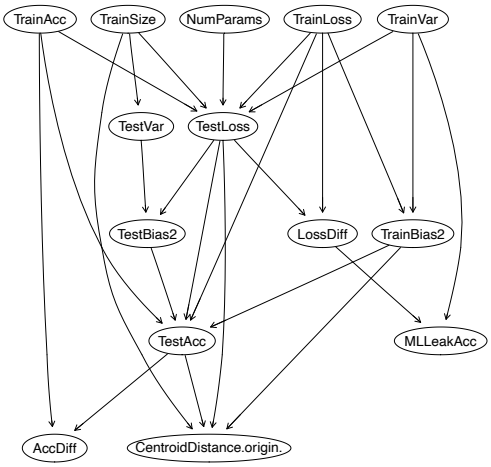
For each attack of the 6 attack variants (with and without a defense), we show the resulting graphs below.



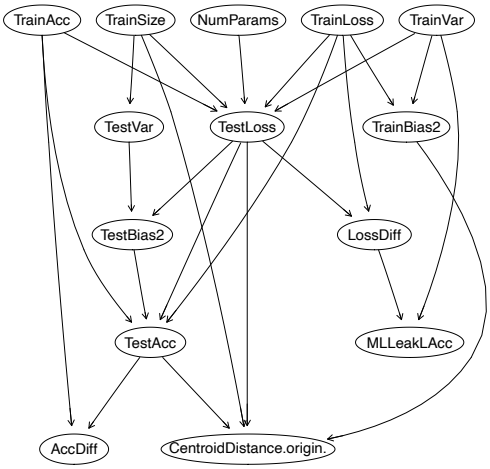
(a) The causal model Etio infers for the target `MLLeakAcc` (CE-trained models).



(b) The causal model Etio infers for the target `MLLeakAcc-l` (CE-trained models).

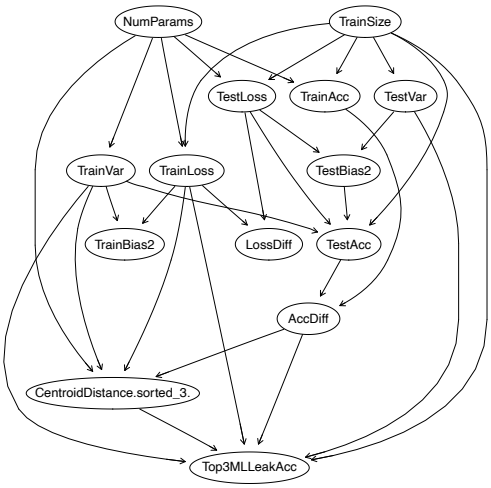


(c) The causal model Etio infers for the target `MLLeakAcc` (MSE-trained models).

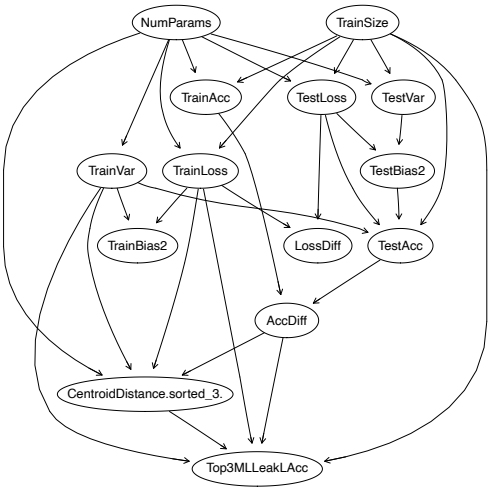


(d) The causal model Etio infers for the target `MLLeakAcc-l` (MSE-trained models).

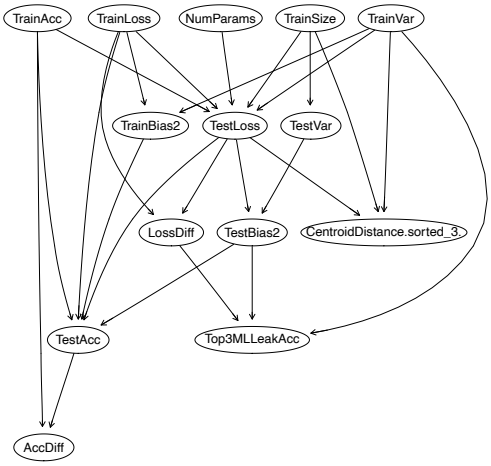
Figure A.1. Etio graphs for the single shadow model with top-10 prediction vector (with and without label) as input to the attack model.



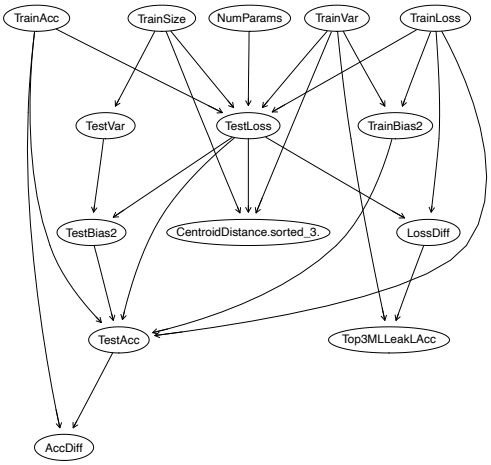
(a) The causal model Etio infers for the target `MLLeakTop3Acc` (CE-trained models).



(b) The causal model Etio infers for the target `MLLeakTop3Acc-l` (CE-trained models).

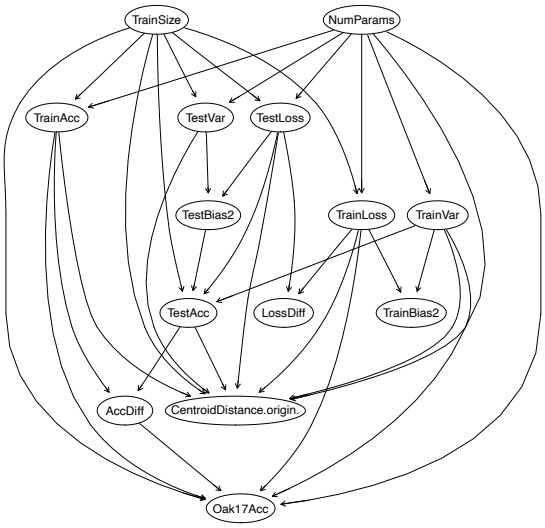


(c) The causal model Etio infers for the target `MLLeakTop3Acc` (MSE-trained models).

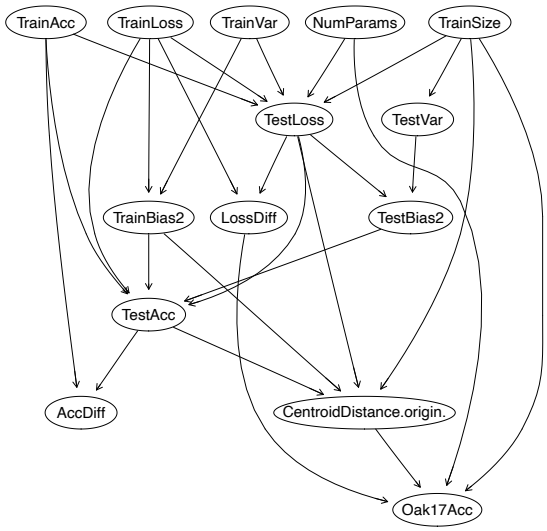


(d) The causal model Etio infers for the target `MLLeakTop3Acc-l` (MSE-trained models).

Figure A.2. Etio graphs for the single shadow model that takes the top-3 prediction vector (with and without label) as input to the attack model.



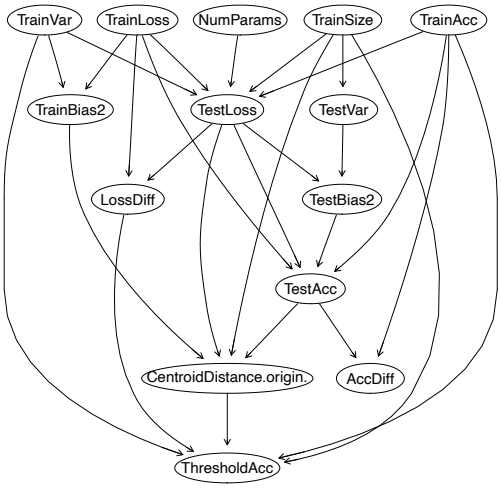
(a) The causal model Etio infers for the target `ShadowAcc` (CE-trained models).



(b) The causal model Etio infers for the target `ShadowAcc` (MSE-trained models).

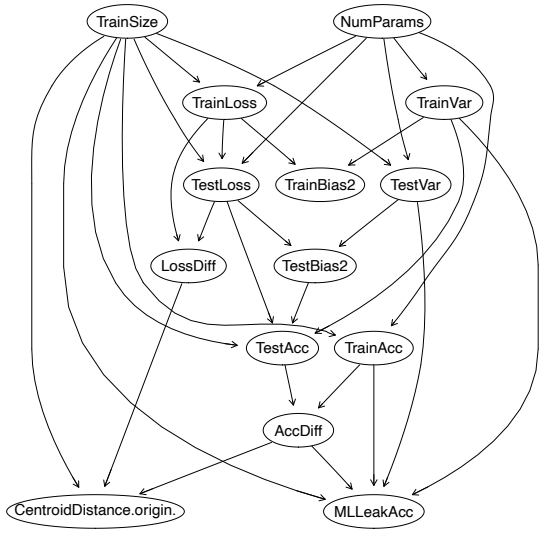
Figure A.3. The causal model Etio infers for the multiple shadow model attack for CE and MSE-trained models, where the target node is `ShadowAcc`.

(a) The causal model Etio infers for the target `ThreshAcc` (CE-trained models).

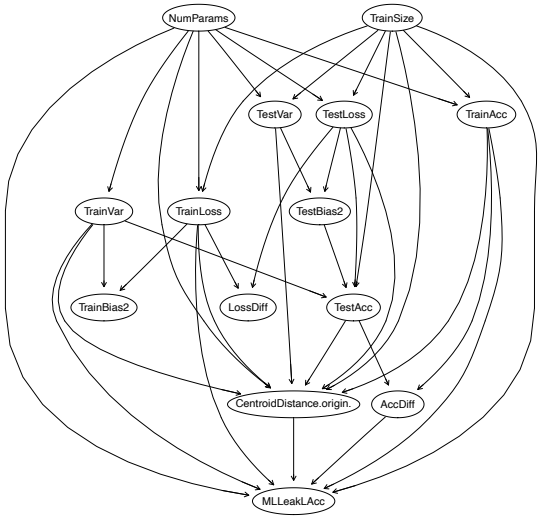


(b) The causal model Etio infers for the target `ThreshAcc` (MSE-trained models).

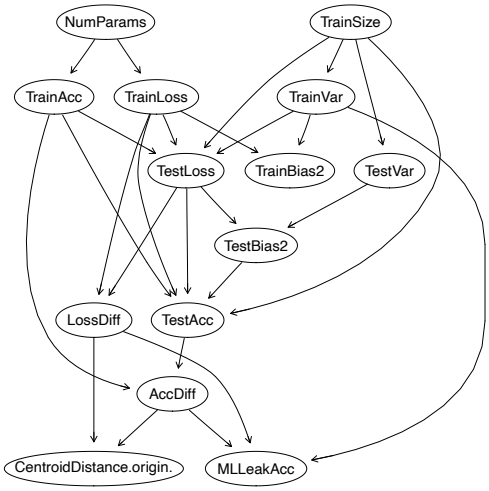
Figure A.4. The causal model Etio infers for the multiple shadow model attack for CE and MSE-trained models, where the target node is `ShadowAcc`.



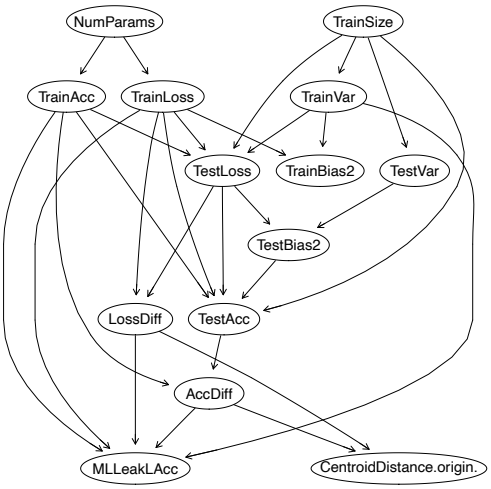
(a) The causal model Etio infers for the target `MLLeakAcc` (CE-trained models with regularization).



(b) The causal model Etio infers for the target `MLLeakAcc-l` (CE-trained models with regularization).

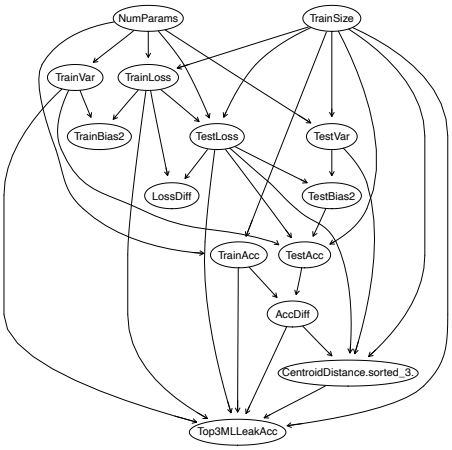


(c) The causal model Etio infers for the target `MLLeakAcc` (MSE-trained models with regularization).



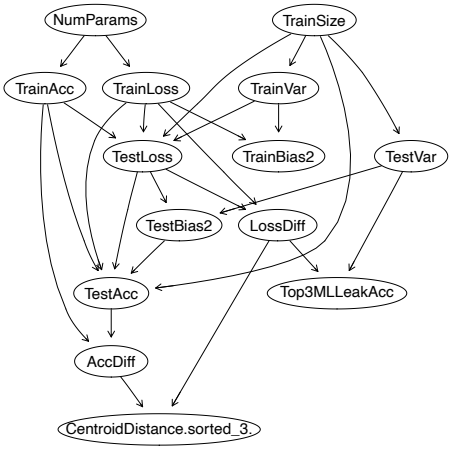
(d) The causal model Etio infers for the target `MLLeakAcc-l` (MSE-trained models with regularization).

Figure A.5. Etio graphs for the single shadow model with top-10 prediction vector (with and without label) as input to the attack model. The models have been trained with L2-regularization (weight decay=$5 \times 10^{-}3$).



(a) The causal model Etio infers for the target `MLLeakTop3Acc` (CE-trained models with regularization).

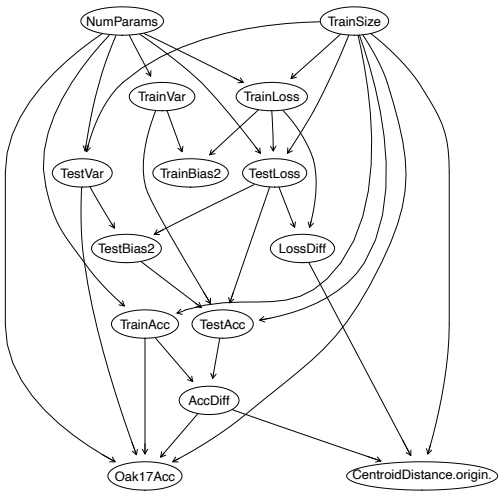
(b) The causal model Etio infers for the target `MLLeakTop3Acc-l` (CE-trained models with regularization).



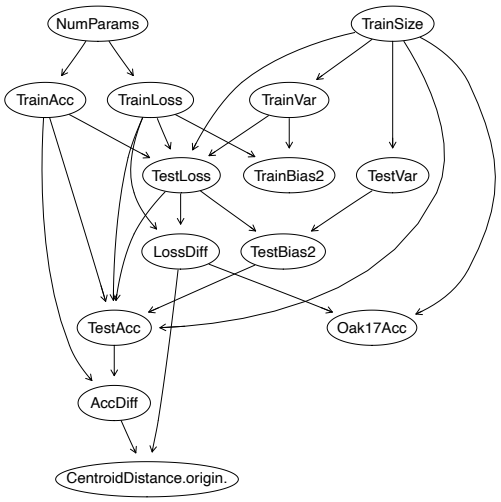
(c) The causal model Etio infers for the target `MLLeakTop3Acc` (MSE-trained models with regularization).

(d) The causal model Etio infers for the target `MLLeakTop3Acc-l` (MSE-trained models with regularization).

Figure A.6. Etio graphs for the single shadow model that takes the top-3 prediction vector (with and without label) as input to the attack model. The models have been trained with L2-regularization (weight decay=$5 \times 10^{-}3$).

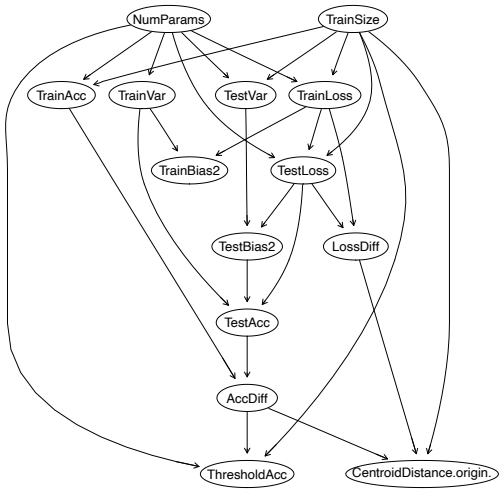


(a) The causal model Etio infers for the target `ShadowAcc` (CE-trained models with regularization).

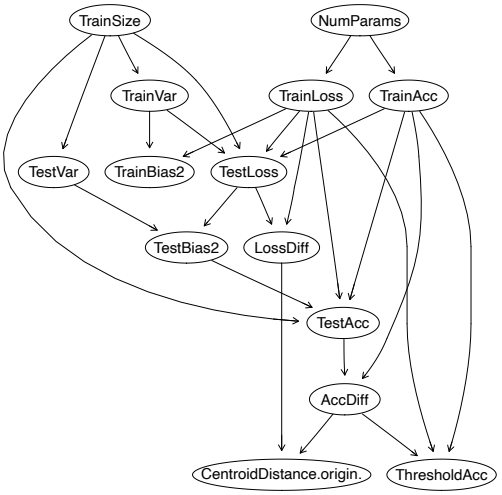


(b) The causal model Etio infers for the target `ShadowAcc` (MSE-trained models with regularization).

Figure A.7. The causal model Etio infers for the multiple shadow model attack for CE and MSE-trained models, where the target node is `ShadowAcc`. The models have been trained with L2-regularization (weight decay=$5 \times 10^{-}3$).

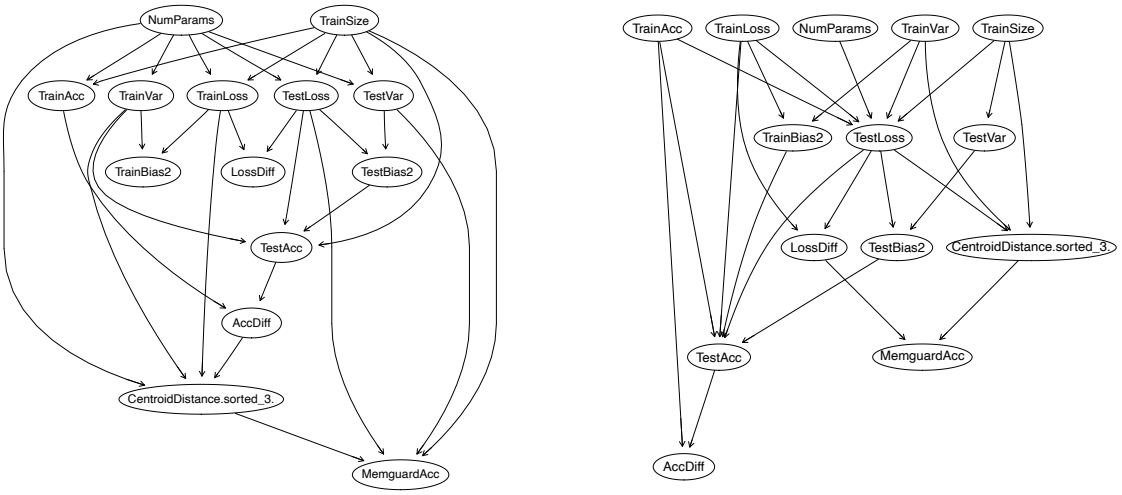


(a) The causal model Etio infers for the target `ThreshAcc` (CE-trained models with regularization).



(b) The causal model Etio infers for the target `ThreshAcc` (MSE-trained models with regularization).

Figure A.8. The causal model Etio infers for the multiple shadow model attack for CE and MSE-trained models, where the target node is `ThreshAcc`. The models have been trained with L2-regularization (weight decay=$5 \times 10^{-}3$).



(a) The causal model Etio infers for the target `MemGuardAcc` (CE-trained models).

(b) The causal model Etio infers for the target `MemGuardAcc` (MSE-trained models).

Figure A.9. The causal model Etio infers for the single shadow model on MemGuard defended models. The target node in this case `MemGuardAcc` represents the accuracy of the MLLeak attack with top-3 predictions on defended models.

Appendix B

Forgeability Evaluation Appendix

B.1 Divergence Results

We ran the approximate forgery [226] to select batches ($M = 400$) that minimize the L_2 norm for both ResNet-mini and LeNet5 on CIFAR10 and MNIST, respectively (Figure B.1). We observe the same trend for the L_2 norm as we did for the L_∞ norm. The training diverges in L_2 norm in less than 100 training steps. This suggests that approximate forgeries are detectable for a verifier that compares the benign training and the forged run. In Figure B.2, we demonstrate that with extended training of LeNet5 on MNIST, the divergence (in L_2) keeps increasing. This suggests that a single approximate forgery determines a significant change in the model parameters for subsequent training steps.

B.2 Approximate Forgery Scalability

We ran approximate forgery [226] procedure and selected a larger number of batches, i.e., $M \in \{400, 600, 800\}$. Our aim was to evaluate if the approximate forgery attack can find minibatches which result in closer model parameters in L_∞ distance. We find that the improvement is marginal for all of the evaluated models (MNIST, ResNet-mini and VGG-mini), while the time taken doubles as the batch size also doubles (Table B.1).

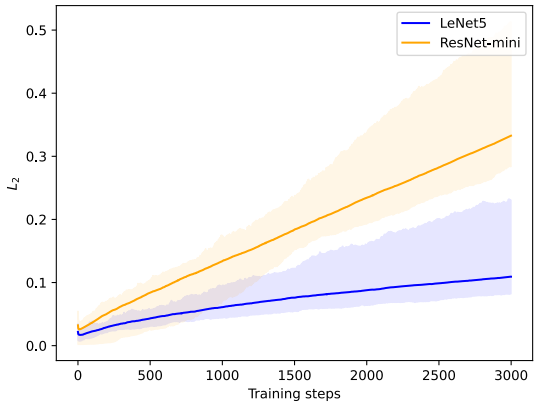


Figure B.1. The forged model parameters diverge after subsequent training (in L_2 distance) on 25 checkpoints for ResNet-mini on CIFAR10 and LeNet5 on MNIST. The solid line indicates the mean L_2 distance over the 25 checkpoints while the translucent region indicates the maximum and the minimum L_2 distance boundaries for the corresponding architecture.

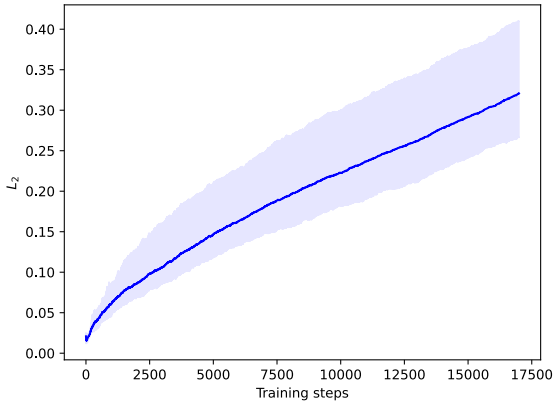


Figure B.2. Extended training shows even larger divergence in L_2 distance between for LeNet5 on MNIST over 5 checkpoints. The solid line indicates the mean L_2 distance over the 5 checkpoints while the translucent region indicates the maximum and the minimum L_2 distance boundaries for the corresponding architecture.

Archictecture	M	Time (s)	L_∞ (Avg, Max, Min)	Avg. L_2 (Avg, Max, Min)
LeNet5	400	205.58	(6.62e-04, 1.16e-03, 3.12e-04)	(9.43e-03, 2.21e-02, 8.42e-04)
	600	165.85	(6.38e-04, 1.11e-03, 3.12e-04)	(9.26e-03, 2.10e-02, 7.09e-04)
	800	206.21	(6.23e-04, 1.02e-03, 1.56e-04)	(9.10e-03, 2.33e-02, 4.93e-04)
ResNet-mini	400	2010.92	(1.55e-03, 2.28e-03, 3.12e-04)	(3.21e-02, 5.12e-02, 1.94e-03)
	600	2817.59	(1.51e-03, 2.28e-03, 3.12e-04)	(3.16e-02, 5.12e-02, 1.91e-03)
	800	3768.89	(1.48e-03, 2.23e-03, 3.13e-04)	(3.13e-02, 5.18e-02, 1.93e-03)
VGG-mini	400	4314.00	(6.62e-04, 1.16e-03, 3.12e-04)	(9.43e-03, 2.21e-02, 8.42e-04)
	600	6711.57	(6.38e-04, 1.11e-03, 3.12e-04)	(9.26e-03, 2.10e-02, 7.09e-04)
	800	8489.83	(6.23e-04, 1.02e-03, 1.56e-04)	(9.10e-03, 2.33e-02, 4.93e-04)

Table B.1. The approximate forgery attacks only marginally improves with increasing the number of candidate minibatches (M) considered from 400 to 800. The time represents the total time to load the model, sample the minibatches and pick the best update for all 25 checkpoints considered.

Publications during PhD Study

[1] T. Baluta, S. Shen, S. Shinde, K. S. Meel, and P. Saxena, "Quantitative verification of neural networks and its security applications", in *Conference on Computer and Communications Security (CCS)*, 2019.

[2] Z. L. Chua, Y. Wang, T. Baluta, P. Saxena, Z. Liang, and P. Su, "One engine to serve'em all: Inferring taint rules without architectural semantics." In *Network and Distributed System Security Symposium (NDSS)*, 2019.

[3] T. Baluta, Z. L. Chua, K. S. Meel, and P. Saxena, "Scalable quantitative verification for deep neural networks", in *International Conference on Software Engineering (ICSE)*, 2021.

[4] A. Kolluri, T. Baluta, and P. Saxena, "Private hierarchical clustering in federated networks", in *Conference on Computer and Communications Security (CCS)*, 2021.

[5] B. Wang, T. Baluta, A. Kolluri, and P. Saxena, "SynGuar: Guaranteeing generalization in programming by example", in *Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*, 2021.

[6] T. Baluta, S. Shen, S. Hitarth, S. Tople, and P. Saxena, "Membership inference attacks and generalization: A causal perspective", in *Conference on Computer and Communications Security (CCS)*, 2022.

[7] A. Kolluri, T. Baluta, B. Hooi, and P. Saxena, "LPGNet: Link private graph networks for node classification", in *Conference on Computer and Communications Security (CCS)*, 2022.

[8] B. Wang, A. Kolluri, I. Nikolić, T. Baluta, and P. Saxena, "User-customizable transpilation of scripting languages", in *International Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA)*, 2023.

[9] T. Baluta, I. Nikolić, R. Jain, D. Aggarwal, and P. Saxena, "Unforgeability in stochastic gradient descent", in *Conference on Computer and Communications Security (CCS)*, 2023.

[10] J. Yang, A. Shaw, T. Baluta, M. Soos, and K. S. Meel, "Explaining SAT solving using causal reasoning", in *International Conference on Theory and Applications of Satisfiability Testing (SAT)*, 2023.