Journal of Machine Learning Research 27 (2026) 1-58 Submitted 9/25; Revised 3/26; Published 4/26

Abstract Gradient Training: A Unified Certification Framework for Data Poisoning, Unlearning, and Differential Privacy

Philip Sosnin P.SOSNIN23@IMPERIAL.AC.UK
Department of Computing
Imperial College London
London, SW7 2AZ, United Kingdom

Matthew Wicker M.WICKER@IMPERIAL.AC.UK
Department of Computing
Imperial College London
London, SW7 2AZ, United Kingdom

Josh Collyer JCOLLYER@TURING.AC.UK
The Alan Turing Institute
London, NW1 2DB, United Kingdom

Calvin Tsay C.TSAY@IMPERIAL.AC.UK
Department of Computing
Imperial College London
London, SW7 2AZ, United Kingdom

Editor: Shiqian Ma

Abstract

The impact of inference-time data perturbation (e.g., adversarial attacks) has been extensively studied in machine learning, leading to well-established certification techniques for adversarial robustness. In contrast, certifying models against training data perturbations remains a relatively under-explored area. These perturbations can arise in three critical contexts: adversarial data poisoning, where an adversary manipulates training samples to corrupt model performance; machine unlearning, which requires certifying model behavior under the removal of specific training data; and differential privacy, where guarantees must be given with respect to substituting individual data points. This work introduces Abstract Gradient Training (AGT), a unified framework for certifying robustness of a given model and training procedure to training data perturbations, including bounded perturbations, the removal of data points, and the addition of new samples. By bounding the reachable set of parameters, i.e., establishing provable parameter-space bounds, AGT provides a formal approach to analyzing the behavior of models trained via first-order optimization methods.

Keywords: Formal Verification, Data Poisoning, Differential Privacy.

1. Introduction

The proliferation of machine learning models in critical applications, ranging from healthcare to autonomous driving, has raised significant concerns regarding their safety and privacy (Bommasani et al., 2021). The scale of modern datasets, while enabling unprecedented model performance, introduces significant vulnerabilities. For example, the impracticality of

exhaustive quality checks exposes systems to *adversarial data poisoning* (Tian et al., 2022), while the sensitive nature of user-collected data requires rigorous guarantees on *privacy leakage* (Dwork and Roth, 2014). The changing landscape of data protection regulations further complicates this picture, as model owners may be required to honor individuals' *right to be forgotten* (Bourtoule et al., 2021). These challenges present an urgent need for methods to analyze how various training-time perturbations—including data poisoning, data removal, or data substitution—impact model performance and security. Understanding these effects is crucial for developing machine learning systems that remain both effective and trustworthy in sensitive applications.

The vulnerability of deep learning models to inference-time adversarial attacks is well-documented (Szegedy et al., 2013), leading to significant advancements in certification and robust training methodologies (König et al., 2024). In contrast, the problem of certifying model robustness against training-time perturbations remains comparatively under-explored. Unlike inference-time attacks, which manipulate inputs without altering model parameters, training-time perturbations directly affect the learning process, potentially influencing the entire model architecture and all subsequent model predictions (e.g., as in Figure 1). Existing techniques, such as those for data poisoning (Rosenfeld et al., 2020; Steinhardt et al., 2017; Xie et al., 2022) or differentially private prediction (Papernot et al., 2016a; van der Maaten and Hannun, 2020), often rely on overly conservative analysis or require modification of the training procedure itself (e.g., requiring large ensembles of models).

This work introduces our novel framework, termed **Abstract Gradient Training (AGT)**, for efficiently analyzing the effect of data perturbation on training pipelines. Inspired by inference-time certification techniques, AGT allows for the computation of *valid bounds on the reachable set of parameters*, that is, the region of parameter space that contains all possible learned model parameters resulting from any allowable perturbation to the training data. By shifting the focus from dataset perturbations to parameter perturbations, we show that AGT offers a more tractable path to certification. We presented preliminary versions of this work in Sosnin et al. (2024) and Wicker et al. (2025). More recently, Lorenz et al. (2024a,b) apply similar methodologies to certify against data poisoning attacks. This work builds upon these previous works by introducing a comprehensive framework of perturbation models and certification techniques. The contributions of this work are threefold:

- We present a comprehensive framework that unifies certification for data poisoning, differentially private prediction, and certified machine unlearning, building upon and extending the foundational work of Sosnin et al. (2024) and Wicker et al. (2025).
- We generalize the AGT framework to alternative bounding mechanisms such as mixed-integer programming (MIP), yielding tighter and more precise parameter-space bounds.
- We provide an extensive empirical evaluation, comparing the efficacy of various bound computation methods, including MIP-based approaches, against baselines of interval bound propagation (IBP) and linear programming (LP).

The remainder of this paper is structured as follows. We begin with related work and mathematical preliminaries in Sections 2 and 3, followed by a formal problem definition in Section 4. We then introduce Abstract Gradient Training in Section 5, detailing its interval bound propagation instantiation in Section 6 and refinement via mixed-integer

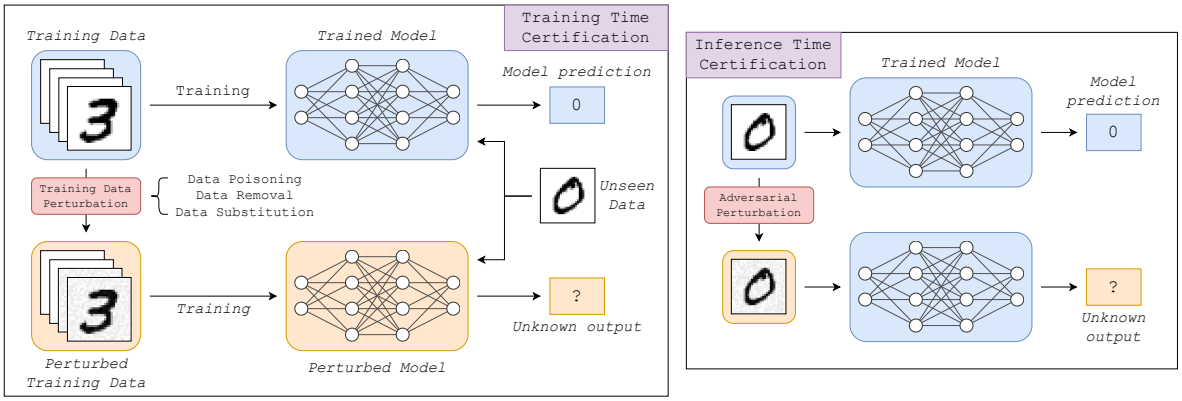


Figure 1: Illustration of training-time certification (left) vs inference-time certification (right). The goal of training-time certification is to verify the behavior of the perturbed model for any possible training data perturbation. Unlike inference-time certification, training-time certification requires reasoning over the entire training process.

programming in Section 7. Sections 8- 10 demonstrates the application of our bounds to certify properties in data poisoning, unlearning, and privacy, along with a comprehensive experimental evaluation.

2. Background and Related Works

Our methodology builds on established certification techniques developed for inference-time adversarial robustness (also known as evasion attacks), instead leveraging these methods to certify against training-time perturbations. In addition to inference-time certification, we examine prior work on analyzing, certifying, and defending against training-time perturbations in three key areas: adversarial data poisoning, machine unlearning, and differential privacy. This section reviews existing approaches in these domains and identifies open challenges that motivate our work.

2.1 Certification of Adversarial Robustness

Machine learning models, particularly neural networks, are susceptible to adversarial examples – carefully designed inputs intended to deceive the model, often by introducing changes imperceptible to humans (Szegedy et al., 2013). Even well-trained networks that achieve high test-time accuracy may fail to generalize to new scenarios, leaving them vulnerable to adversarial attacks (Papernot et al., 2016b). In applications of machine learning for which incorrect predictions can lead to significant consequences, it is crucial to develop provable verification techniques to complement empirical performance testing.

Formally, verification aims to determine whether a machine learning model adheres to a specified input-output relationship over a given input domain (Liu et al., 2021). Methods can be categorized based on the strength of their guarantees:

- *Sound* algorithms are those that will only declare a property to be true if it is true.
- *Complete* algorithms are guaranteed to confirm a property is true whenever it is true.

While soundness and completeness are both desirable, to improve computational efficiency, some approaches sacrifice completeness by employing approximations, potentially leading to *incomplete* results. In this work, we will use the following terminology: *Verification* will refer to strictly methods that are both sound and complete, and *certification* will describe methods that are sound but incomplete.

Existing Approaches. Certification techniques can be broadly categorized based on the techniques they use. Those based on propositional logic encode the network and property as a satisfiability problem. If the formula is satisfiable, it implies that a counterexample exists, meaning the property is violated (Katz et al., 2017). Alternatively, those that use domain propagation or abstract interpretation propagate reachable domains through the network's layers. This technique over-approximates the set of possible outputs for a given input domain. If the propagated output domain satisfies the desired property, it therefore holds over the input domain (Gowal et al., 2018; Xiang et al., 2018; Zhang et al., 2018; Singh et al., 2019).

Finally, optimization-based methods encode neural networks as a set of constraints, transforming the certification task into an optimization problem (Ehlers, 2017; Fischetti and Jo, 2018; Bunel et al., 2018). These techniques commonly leverage mixed-integer programming solvers to either certify robustness or identify counterexamples. We refer the reader to Huchette et al. (2026) for a review of these encodings and problem formulations.

Relation to This Work. Existing certification algorithms are not directly applicable to certifying models with respect to training-time perturbations. Our certification problem specifically involves analyzing the entire training trajectory while accounting for perturbations at each iteration. Viewing the training process as a non-linear dynamical system, our approach aligns with existing literature on reachability analysis for such systems (Althoff et al., 2021)

In our case, the reachability analysis must encompass not only the neural network itself, but also the computation of gradients and parameter updates at each training step. The most closely related certification approaches are those that treat both the model's inputs and parameters as variables. Such formulations appear in the certification of Bayesian neural networks (Wicker et al., 2020) and in preliminary work on robust explanations (Wicker et al., 2022). However, despite these methodological connections, none of these existing approaches directly extend to the general training setting considered in this work.

2.2 Adversarial Data Poisoning

Data poisoning attacks represent a significant threat to the integrity of machine learning models, wherein malicious actors manipulate training data to degrade model performance or induce specific, undesirable behaviors (Biggio and Roli, 2018; Biggio et al., 2014; Newsome et al., 2006). A backdoor attack, for example, introduces a hidden vulnerability where a specific input pattern, or trigger, forces the model to act in an unexpected way at inference time, while the model behaves normally on all other inputs. For example, a lane detection system with a backdoor might be tricked into misclassifying lane markings if it sees a traffic cone (Han et al., 2022).

Previous research has revealed that attacks affecting even a small proportion of training data can lead to catastrophic model failures (Carlini et al., 2023). For example, poisoning can readily manipulate recommender systems on platforms such as YouTube, eBay, and Yelp (Yang et al., 2017). Other research has shown that poisoning just 1% of training data can force targeted misclassifications (Zhu et al., 2019), or introduce targeted backdoor vulnerabilities in lane detection systems (Han et al., 2022).

Poisoning attacks can be broadly categorized based on their objectives: *untargeted poisoning* aims to generally corrupt model performance, potentially leading to denial-of-service (Muñoz-González et al., 2017). *Targeted poisoning* focuses on misclassifying specific inputs while maintaining overall performance, and *backdoor attacks* introduce hidden triggers that activate only when particular patterns are present at inference time (Chen et al., 2017; Gu et al., 2017; Han et al., 2022; Zhu et al., 2019). In this work, we adopt the classification of attack goals proposed in Tian et al. (2022), and refer readers to that source for a more detailed review of data poisoning attacks.

Adversary Capabilities. Modeling the capabilities of a poisoning adversary is critical in assessing the safety of machine learning systems. In this work, we consider two distinct threat models: bounded and unbounded attacks. In *bounded attacks*, adversaries are constrained in the magnitude of perturbations they can apply to training data, typically defined by norms in feature and label spaces. In contrast, *unbounded attacks* allow adversaries to inject arbitrary data points, potentially exploiting data collection vulnerabilities. The latter assumes a more powerful adversary capable of significantly altering the training distribution. In both cases, we assume a white-box setting where the adversary possesses comprehensive knowledge of the training process, including model architecture, initialization, data, and hyperparameters, to establish worst-case robustness guarantees.

Poisoning Attack Defenses. Defending against poisoning attacks is a complex challenge. Traditional defenses often focus on identifying and mitigating specific attack strategies. For example, Li et al. (2020) train classifiers to detect and reject potentially poisoned inputs by leveraging datasets generated from known attack strategies. Other approaches apply noise or clipping techniques to limit the impact of certain perturbations (Hong et al., 2020). However, these targeted defenses may not generalize to novel or more sophisticated attacks.

To address the limitations of attack-specific defenses, researchers have explored methods for *certified robustness*, which aim to provide provable guarantees against a wider range of poisoning attacks. For linear models, Steinhardt et al. (2017) and Rosenfeld et al. (2020) establish upper bounds on the effectiveness of gradient-based and ℓ_2 perturbation attacks, respectively. Differential privacy has also been investigated as a means to provide statistical guarantees in limited poisoning scenarios (Xie et al., 2022). Deterministic certification methods, such as those employing *aggregation* techniques, involve partitioning the dataset and training multiple models to achieve robustness guarantees (Levine and Feizi, 2020; Wang et al., 2022; Rezaei et al., 2023). These methods offer strong guarantees but often incur substantial computational costs given the need to train and evaluate numerous models.

Most similar to this work is the recent work of Lorenz et al. (2024b) and Lorenz et al. (2024a). In these works, they apply interval bound propagation and mixed-integer programming to certify training pipelines against data poisoning attacks. While methodologically similar, these works are more limited in scope and thus can be considered as a subset of the

more general approach we present here. Their work focuses on a reduced set of threat models (specifically, bounded perturbations applied to all training samples), while we consider a wider range of adversaries, including those limited to poisoning a small number of samples with unbounded perturbations. Furthermore, their analysis is confined to a single training iteration using a mixed-integer bilinear formulation, whereas our work examines multiple training iterations and a wide range of formulations and relaxations, including linear and quadratic approximations. Finally, we also extend the certification framework to a wider family of parameter domains, such as polyhedral domains.

Relation to This Work. Unlike many prior works, our approach aims to certify a given model and training algorithm and therefore can be used to analyze and certify the poisoning robustness of various proposed defenses and benchmarks. Like existing approaches, our method computes a sound (but potentially incomplete) certificate that bounds the impact of the poisoning attack. Importantly, our work is complementary to prior work on aggregation methods, suggesting potential avenues for future research to combine both techniques for enhanced, certified robustness.

2.3 Differential Privacy

The deployment of machine learning in sensitive domains, such as healthcare and finance, requires robust privacy safeguards. As these models increasingly rely on personal data, ensuring the confidentiality of individuals' information becomes paramount (Cummings et al., 2021). Differential privacy (DP) has emerged as the primary tool in the development of privacy-preserving machine learning algorithms. DP provides a rigorous mathematical framework to quantify and limit the leakage of sensitive information from datasets. Formally, differential privacy is defined in terms of the probability of obtaining a specific set of outcomes from a randomized algorithm when applied to adjacent datasets.

Definition 1 ((ϵ, δ)-Differential Privacy (Dwork and Roth, 2014))**.** *A randomized mechanism $\mathcal{M}$ is (ϵ, δ)-differentially private if, for all pairs of adjacent datasets A, B and any $S \subseteq$ Range$(\mathcal{M})$,*

$$\mathbb{P}\big(\mathcal{M}(A) \in S\big) \leq e^{\epsilon}\mathbb{P}\big(\mathcal{M}(B) \in S\big) + \delta \tag{1}$$

Here, ϵ is known as the privacy leakage, while δ allows for a small probability of failure.

In this work, we define distances between datasets using the Hamming distance $d(A, B)$, and adjacent datasets are those for which $d(A, B) = 1$.

Differential Privacy in Machine Learning. Current approaches to achieve differential privacy in machine learning broadly fall into two categories: private training and private prediction. Private training methods, exemplified by DP-SGD (Abadi et al., 2016), introduce randomness into the training process to ensure that it satisfies Definition 1, thereby enabling the release of model parameters with DP guarantees. This approach, while effective, incurs costs due to the required pre-specification of privacy parameters, and it risks introducing harmful biases during training, such as discriminatory effects (Fioretto et al., 2022).

Private prediction, on the other hand, focuses on privatizing the model's output predictions. It allows for dynamic adjustment of privacy budgets and is applicable to complex training scenarios, such as federated learning. Existing techniques for private prediction, e.g. the

subsample-and-aggregate approach (Papernot et al., 2016a; van der Maaten and Hannun, 2020), add noise to the model's predictions calibrated to the *global sensitivity*. This approach, while providing a straightforward guarantee of Definition 1, often leads to the addition of excessive noise, as global sensitivity considers the worst case across all possible datasets. Consequently, the utility of the model's predictions can be significantly and unnecessarily diminished, particularly when the model's behavior is locally stable.

In practice, private prediction empirically exhibits an unfavorable privacy-utility trade-off compared to private training (van der Maaten and Hannun, 2020), and recent audits have indicated a lack of tightness in their privacy analyses (Chadha et al., 2024).

Limitations of Current Approaches to Differential Privacy. Despite DP's widespread adoption, *post-hoc* audits have revealed gaps between attacker strength and guarantees offered by DP (Carlini et al., 2022; Yu et al., 2022). As a result, several works seek more specific, and thus sharper, privacy guarantees by leveraging specific information about the learning algorithm and dataset. For example, Nissim et al. (2007) and Liu et al. (2022) use notions of local sensitivity to produce tighter bounds. In Ligett et al. (2017), the authors privately search the space of privacy-preserving parameters to tune performance on a given dataset, while in Yu et al. (2022) the authors propose individual differential privacy, which can compute tighter privacy bounds for given individuals in the dataset. Unlike this work, the above works rely solely on private training, e.g., DP-SGD (Abadi et al., 2016).

Relation to This Work. An application of the methodology proposed in this work is proving tighter bounds for the private prediction setting. Specifically, we aim to use certification to prove the stability of given predictions of machine learning models, thus providing data-specific bounds on the (local) sensitivity of model predictions. We then build upon the approach of Nissim et al. (2007) to derive bounds on the *smooth sensitivity* of model predictions, which we use to improve the privacy-utility tradeoffs inherent in the private prediction setting. This represents a significant step towards bridging the performance gap between private prediction and private training.

2.4 Machine Unlearning.

Related to differential privacy is the concept of machine unlearning, which addresses complementary privacy concerns. While DP focuses on preventing any information leakage from a model's training data, machine unlearning is concerned with only the removal of specific data points' influence (Cao and Yang, 2015). This capability has become increasingly important due to regulations such as GDPR in the EU, which codify the "right to be forgotten" (Bourtoule et al., 2021). As models grow in size and complexity, naive retraining approaches become prohibitively expensive, requiring more efficient unlearning methods.

Unlearning guarantees can be broadly categorized into two types. *Exact unlearning* (Bourtoule et al., 2021) requires that the unlearned model be identical to a model retrained from scratch without the forgotten data. Since this is often computationally infeasible, *approximate unlearning* (Guo et al., 2019) relaxes this requirement by instead limiting statistical similarity between the unlearned model and the retrained one, formalized using bounds similar to differential privacy. In fact, differentially private learning algorithms inherently satisfy approximate unlearning guarantees, though often with excessive performance penalties

(Neel et al., 2021). Within this category, certified unlearning (Guo et al., 2019) aims to provide verifiable certificates that bound the statistical distance between the unlearned and retrained models, i.e. to provide guarantees in the context of approximate unlearning. These approaches provide provable bounds on the statistical distance between unlearned models and those trained from scratch without the forgotten data point.

Existing Approaches. Current certified unlearning techniques include influence function-based methods (Koh and Liang, 2017; Guo et al., 2019), which approximate the effect of removing training points through first-order Taylor expansions of the model's loss function. While computationally efficient, these methods can produce inaccurate certificates for non-convex models (Basu et al., 2020). Another prominent approach is SISA (Sharded, Isolated, Sliced, and Aggregated) training (Bourtoule et al., 2021), which partitions the training data across multiple shards and only retrains affected shards when unlearning requests arise. This reduces computational overhead of retraining but may compromise model performance due to data fragmentation.

Despite progress in the field, existing unlearning methods face significant challenges. Most critically, current certification methods focus on bounding the distance between model parameters, which does not directly translate to guarantees on model predictions (Thudi et al., 2022). This parameter-space focus can lead to overly conservative certificates that do not accurately reflect the actual privacy risks posed by unlearned models.

Relation to This Work. While our work does not fit within either the exact or approximate unlearning settings, our approach extends the analysis of the influence of unlearning on the predictions of a machine learning model. Unlike approaches based on influence functions, our parameter-space bounds give formal (non-approximate) guarantees on the impacts of data removal. Furthermore, rather than working solely in parameter space, we use our bounds to certify aspects of model behavior, such as how predictions change when data is removed. This allows for a precise characterization of model behavior under unlearning.

3. Preliminaries

Before introducing our certification framework, we first establish the necessary mathematical background and notation.

Notation. We define a machine learning model as a function $f(x, \theta)$ with parameters θ mapping inputs x from a feature space to outputs y in a target space. We consider both regression and classification settings, allowing the target space to be discrete, continuous, and/or multivariate. Our focus is on supervised learning, where the model is trained on a labeled dataset $\mathcal{D} = \{(x^{(i)}, y^{(i)})\}_{i=1}^{N}$ consisting of N input-output pairs used to optimize the parameters θ. Key notation is defined throughout and summarized in Table 1.

Neural networks. While the methods introduced in this work are applicable to a broad class of machine learning models, we primarily focus on feed-forward neural networks. A feed-forward neural network if a machine learning model f defined as a composition of K layers with parameters $\theta = \left\{(W^{(i)}, b^{(i)})\right\}_{i=1}^{K}$ given by

$$\hat{z}^{(k)} = W^{(k)} z^{(k-1)} + b^{(k)}, \quad z^{(k)} = \sigma\left(\hat{z}^{(k)}\right), \tag{2}$$

where $z^{(0)} = x$, $f(x, \theta) = \hat{z}^{(K)}$, and σ is the activation function, which we take to be the rectified linear unit (ReLU).

Gradient-based training. Training neural networks typically relies on *gradient-based optimization*, where model parameters are iteratively updated to minimize a loss function. Let $\mathcal{L}$ denote a loss function that measures model quality, e.g. how different the model predictions are from the ground truth labels. Given a training dataset $\mathcal{D}$, the model parameters θ are optimized by minimizing the empirical risk:

$$\theta^* = \arg\min_{\theta} \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}\left(f\left(x^{(i)}, \theta\right), y^{(i)}\right). \tag{3}$$

The predominant approach to solving this optimization problem is through **stochastic gradient descent (SGD)**, where the model parameters are iteratively updated using a randomly sampled mini-batch $\mathcal{B} \subset \mathcal{D}$. The update rule is given by:

$$\theta^{(t)} = \theta^{(t-1)} - \frac{\alpha}{|\mathcal{B}|} \sum_{(x^{(i)}, y^{(i)}) \in \mathcal{B}} \nabla_\theta \mathcal{L}\left(f\left(x^{(i)}, \theta^{(t-1)}\right), y^{(i)}\right), \tag{4}$$

where $\alpha > 0$ is the learning rate, and the gradient is averaged over the mini-batch $\mathcal{B}$. We will denote the parameters obtained by following this procedure for a model f with a dataset $\mathcal{D}$ as $\theta = M(\mathcal{D})$.

Several extensions of SGD introduce adaptive learning mechanisms to enhance convergence, such as Adam (Kingma and Ba, 2014). While the techniques developed here apply to any first-order optimization algorithm, we focus on standard SGD (4) to simplify our exposition.

Table 1: Summary of notation.

Symbol	Description
$f(x, \theta)$	Model with parameters θ evaluated at input x
$\mathcal{D}$, $\tilde{\mathcal{D}}$	Nominal and perturbed training datasets
$\mathcal{T}(\mathcal{D})$	Perturbation model: set of feasible perturbed datasets
$M(\mathcal{D})$	Training mechanism: maps a dataset to learned parameters
Θ, $\Theta^{(t)}$	Valid parameter-space domain (at iteration t)
$\mathcal{B}^{(t)}$, $\mathcal{I}^{(t)}$	Training batch and its index set at iteration t
α	Learning rate
$\delta^{(t,i)}$	Per-sample loss gradient for sample i at iteration t
$\Delta\theta^{(t)}$	Descent direction (averaged loss gradient) at iteration t
$\boldsymbol{\theta} = [\theta^L, \theta^U]$	Interval domain over parameters
n	Maximum number of perturbed/removed/substituted samples
ϵ, ν	Perturbation radii in feature and label space
κ	Gradient clipping threshold

4. Problem Formulation

In this section, we formalize the problem of certifying robustness to training data perturbations and define perturbation models relevant to the applications of data poisoning, machine

unlearning and differential privacy. These formulations serve as the basis for our certification framework, which we describe in the subsequent sections.

4.1 Certification Problem

In the context of training data perturbations, our certification objective is to formally certify whether a machine learning model trained with a specified algorithm satisfies a given property despite modifications to its training data. Given an initial dataset $\mathcal{D}$, a training procedure M, and a perturbation model $\mathcal{T}$ that defines feasible training data modifications, we seek to certify that a property P holds for any trained parameter $\tilde{\theta}$ obtained from a perturbed dataset $\tilde{\mathcal{D}} \in \mathcal{T}(\mathcal{D})$. Formally, we aim to prove:

$$\forall \tilde{\mathcal{D}} \in \mathcal{T}(\mathcal{D}), \quad \tilde{\theta} = M(\tilde{\mathcal{D}}) \implies P(\tilde{\theta}). \tag{5}$$

If this condition holds, the model is guaranteed to satisfy property P across all valid training data perturbations, establishing a formal guarantee under the specified perturbation model.

Following standard approaches in the machine learning robustness literature (see Section 2.1), we reformulate this certification problem as an optimization problem:

$$\max_{\tilde{\mathcal{D}}} \; J(\tilde{\theta}) \quad \text{s.t.} \quad \tilde{\theta} = M(\tilde{\mathcal{D}}), \quad \tilde{\mathcal{D}} \in \mathcal{T}(\mathcal{D}). \tag{6}$$

where $J(\theta)$ is an objective function related to the property P via a specification on the optimal solution of the above problem. For instance, if P is the correctness of the model's prediction on a particular input x, one can reformulate the certification problem via $J(\theta) = c^T f(x, \theta)$ for some vector c, i.e. a linear specification on the output logits of the network (Gowal et al., 2018). If the optimal value of the optimization problem is $\leq b$ for a chosen constant b then we can say that P is guaranteed to hold.

In this work, we focus on *sound but incomplete* certification, meaning that we aim compute a provably valid upper bound on the above optimization problem. If this bound remains within some specification determined by P, we establish a formal certification guarantee.

In the following subsection, we outline specific choices for $\mathcal{T}$ that correspond to key applications in data poisoning robustness, machine unlearning and differential privacy. Choices of objective function J, and the practical implications of post-training certificates are discussed in detail in Sections 8- 10.

4.2 Characterizing Training Data Perturbations

A *perturbation model* $\mathcal{T}(\mathcal{D})$ specifies the set of all possible modified datasets that could result, e.g., from adversarial manipulation or privacy-driven modifications. We categorize training data perturbations into three primary types:

1) Bounded manipulation of training data. In the bounded perturbation setting, we assume that any subset of up to n training samples may be modified within a predefined constraint, such as an ℓ_p-norm ball. Given a nominal dataset $\mathcal{D}$, we define the perturbed dataset as $\tilde{\mathcal{D}} \in \mathcal{T}^{n,p,\epsilon,q,\nu}_{\text{bounded}}(\mathcal{D})$, where perturbations are constrained by:

$$\|x^{(i)} - \tilde{x}^{(i)}\|_p \leq \epsilon, \quad \|y^{(i)} - \tilde{y}^{(i)}\|_q \leq \nu, \quad \forall i \in \mathcal{I}. \tag{7}$$

Here, ϵ and ν represent the maximum allowable perturbations in the feature and label spaces, respectively, measured using the ℓ_p and ℓ_q norms. The index set $\mathcal{I}$ specifies the subset of up to n training samples that can be modified, which we take to be any set satisfying $\mathcal{I} \subset \{1, \ldots, N\}$ subject to $|\mathcal{I}| \leq n$. This perturbation model encompasses any poisoning adversary capable of modifying up to n training data samples within the specified bounds.

2) Removal of training data. In training data removal, the perturbation model considers datasets $\tilde{\mathcal{D}} \in \mathcal{T}^n_{\text{removal}}(\mathcal{D})$, where up to n samples are removed from the original dataset:

$$\tilde{\mathcal{D}} = \mathcal{D} \setminus \mathcal{S}, \quad |\mathcal{S}| \leq n, \tag{8}$$

where $\mathcal{S} \subset \mathcal{D}$ is the set of removed data-points. This setting is particularly relevant for machine unlearning applications, where certification guarantees must be given with respect to the deletion of specific training data points.

3) Replacement of training data. The substitution model generalizes both the bounded poisoning and removal settings by allowing an adversary to replace a subset of the training data with new points, leading to a dataset $\tilde{\mathcal{D}} \in \mathcal{T}^n_{\text{subs.}}(\mathcal{D})$ of the form:

$$\tilde{\mathcal{D}} = (\mathcal{D} \setminus \mathcal{S}) \cup \tilde{\mathcal{S}}, \quad |\mathcal{S}| = |\tilde{\mathcal{S}}| \leq n, \tag{9}$$

where $\mathcal{S}$ is a set of up to n removed samples, and $\tilde{\mathcal{S}}$ is a set of introduced samples. This type of perturbation allows consideration of both general poisoning attacks (where adversaries inject maliciously crafted data that is not necessarily related to the removed samples) or differential privacy (where guarantees must be given with respect to the substitution of data).

5. A Unified Certification Framework

Analyzing the behavior of machine learning models under training data perturbations poses a significant theoretical and computational challenge. Unlike standard inference-time certification, which considers perturbation to a single data-point, our setting considers the worst-case outcome in the space of all possible datasets that could have been used to train the model. This significantly expands the complexity of the analysis, as it requires reasoning over a combinatorially large space of possible training datasets, each of which can lead to a different set of learned model parameters. To overcome this challenge, we introduce the concept of valid parameter-space domains, which provide a structured way to efficiently certify properties of the trained model. We then present Abstract Gradient Training (AGT), a framework designed to compute these bounds and enable scalable certification.

5.1 Valid Parameter-Space Domains

Certification problems of the form (6) present a significant computational challenge, as they require optimization over the space of all possible training datasets. Rather than directly optimizing over this combinatorially large space, we instead seek to establish a *valid parameter-space domain*, which defines the reachable set in parameter space that contains all possible learned model parameters resulting from any training with any allowable perturbation to the training data. This reformulation allows us to shift our certification problem from dataset perturbations to parameter perturbations, enabling more efficient analysis.

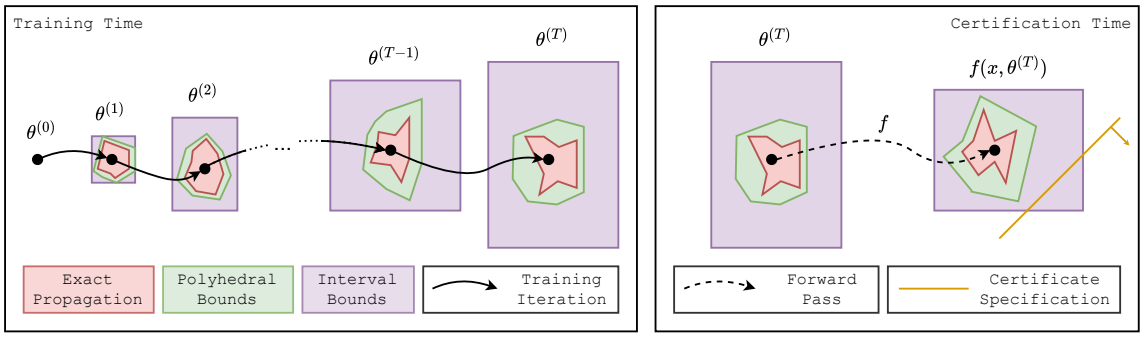


Figure 2: Outline of our certification framework. During training, parameter-space bounds are propagated through each iteration of gradient descent. At each step, AGT computes a sound over-approximation of all reachable parameters, with some over-approximation introduced at each iteration due to the use of abstract domains. Post-training, the final parameter-space bounds are used to certify the network's predictions against a given specification. While we illustrate certification in logit space, our framework supports certification with respect to any desired criterion.

Formally, let $\mathcal{D}$ be a nominal training dataset, and let $\mathcal{T}(\mathcal{D})$ denote the set of all reachable datasets under a given perturbation model (modification, removal, or substitution). We define a set Θ as a **valid parameter-space domain** with respect to a training algorithm M if:

$$\tilde{\theta} = M(\tilde{\mathcal{D}}) \in \Theta, \quad \forall \tilde{\mathcal{D}} \in \mathcal{T}\big(\mathcal{D}\big), \tag{10}$$

where $\tilde{\theta} = M(\tilde{\mathcal{D}})$ represents the parameters obtained by training on a perturbed dataset $\tilde{\mathcal{D}}$. Thus, Θ serves as a certified enclosure that bounds all possible learned parameters resulting from any feasible modification to the training dataset.

Any parameter-space domain satisfying (10) enables us to upper-bound our certification objective in (6). Specifically, we establish the following key result:

Theorem 2. *Let Θ be a valid parameter-space domain for a given perturbation model $\mathcal{T}$. Then, for any objective function J, the worst-case impact of training data perturbations can be bounded by optimizing over the parameter space instead of the dataset space:*

$$\max_{\tilde{\mathcal{D}} \in \mathcal{T}(\mathcal{D})} J\big(M\big(\tilde{\mathcal{D}}\big)\big) \;\leq\; \max_{\tilde{\theta} \in \Theta} J\big(\tilde{\theta}\big). \tag{11}$$

Theorem 2 establishes that we can shift certification problems over the adversarial dataset perturbation into the parameter space, where it can be more tractably analyzed. Unlike the intractable dataset-space maximization on the left-hand side of (11), the equivalent optimization problem over the parameter-space domain on the right-hand side can be efficiently upper-bounded using established certification techniques (Adams et al., 2023; Wicker et al., 2020, 2023). This transformation constitutes the basis for our approach to certified robustness and privacy guarantees.

5.2 Abstract Gradient Training for Parameter-Space Bounds

In this section, we outline our framework for computing valid parameter-space domains. Rather than attempting to track the exact optimization trajectories of model parameters under possible training data perturbations, we construct an **abstract representation** of the training dynamics that provides a sound over-approximation of all reachable parameter values, i.e., bounds on the reachable set. This approach, which we term **Abstract Gradient Training (AGT)**, effectively allows us to compute provable bounds on the set of parameters that a model can reach under a given perturbation model.

To begin, we impose the following assumptions on our analysis.:

- **Assumption 1: Fixed data ordering.** The sequence in which training samples are processed remains unchanged between the nominal and perturbed datasets
- **Assumption 2: Fixed parameter initialization.** The training process starts from the same parameter initialization for both the nominal and perturbed datasets.
- **Assumption 3: Fixed training hyperparameters.** The perturbation model cannot influence hyperparameters such as learning rate, batch size, or number of training iterations.

These assumptions allow us to isolate the effect of training data perturbations from factors such as stochastic parameter initialization or data shuffling. As a result, our certification framework provides guarantees that specifically account for adversarial modifications to the training data. Certificates derived under this framework are valid only within the scope of these assumptions. We note these assumptions may be relaxed within our framework with the trade-off of increased computational cost and reduced certification tightness.

We now briefly discuss the practical implications of each assumption. Assumption 1 (fixed data ordering) requires that the sequence in which training samples are processed be determined before training. In practice, most training pipelines shuffle data at the start of each epoch; however, once a particular shuffle has been realized, the ordering is deterministic. Our certificates therefore apply to a *specific realization* of any such random ordering—namely, the one that was used during training. If the adversary were able to influence the data ordering itself, certification would need to account for all possible orderings, which is beyond the scope of our current framework. For the data removal perturbation model, Assumption 1 applies to the assignment of data to training batches. Specifically, if sample i appears in batch j in the original ordering, this remains unchanged after perturbation. Removed samples are simply excluded from their respective batches, resulting in potentially varied batch sizes.

Assumption 2 (fixed parameter initialization) ensures that the starting point of optimization is shared between the nominal and perturbed training runs. If initialization is randomized in practice, one may either certify a specific realization (the one that was used, which is deterministic post-hoc) or start AGT from an interval encompassing all possible initializations, at the cost of looser bounds. Assumption 3 (fixed hyperparameters) is standard in the certification literature; relaxing this assumption would require an additional layer of abstraction over the hyperparameter space, which is an interesting direction for future work.

Abstract Training Dynamics. Let $\theta^{(t)}$ denote the model parameters at training step t under standard gradient-based optimization. Given a loss function $\mathcal{L}$, the parameters are

updated using the stochastic gradient descent (SGD) rule, reproduced from (4):

$$\theta^{(t)} = \theta^{(t-1)} - \frac{\alpha}{|\mathcal{B}|} \sum_{(x^{(i)},y^{(i)})\in\mathcal{B}} \nabla_\theta \mathcal{L}\left(f\left(x^{(i)}, \theta^{(t-1)}\right), y^{(i)},\right), \tag{12}$$

where α is the learning rate, and $\nabla_\theta \mathcal{L}$ is gradient of the loss function with respect to the model parameters. When the training dataset is perturbed, the loss landscape and resulting gradient updates also shift, causing deviations in the optimization trajectory at both the *current iteration* and *all subsequent iterations*.

To obtain sound bounds on the parameters under such perturbations, we define a valid parameter-space domain $\Theta^{(t)}$ at each training step t such that:

$$\theta^{(t)}(\tilde{\mathcal{D}}) \in \Theta^{(t)}, \quad \forall \tilde{\mathcal{D}} \in \mathcal{T}(\mathcal{D}), \tag{13}$$

where we use $\theta^{(t)}(\tilde{\mathcal{D}})$ to denote the value of the t-th parameter iterate trained using the dataset $\tilde{\mathcal{D}}$. That is, at each step of training, the set $\Theta^{(t)}$ provides an enclosure containing all possible parameter values that could be reached when training on any perturbed dataset $\tilde{\mathcal{D}}$. By propagating these bounds through the optimization process, we obtain a final parameter-space domain $\Theta^{(T)}$ that satisfies (10).

A key challenge in AGT is propagating the parameter-space domains through successive training iterations. Each parameter update must account for both: (1) the cumulative effect of perturbations from all previous iterations and (2) the direct effect of perturbations in the current iteration. Formally, given $\Theta^{(t-1)}$ (the parameter-space domain at step $t-1$), the domain at step t is defined as:

$$\Theta^{(t)} = \left\{ \theta^{(t-1)} - \frac{\alpha}{|\tilde{\mathcal{B}}|} \sum_{\tilde{\mathcal{B}}} \nabla_\theta \mathcal{L}\left(f\left(x^{(i)}, \theta^{(t-1)}\right), y^{(i)}\right) \mid \tilde{\mathcal{B}} \subset \mathcal{T}(\mathcal{D}),\ \theta^{(t-1)} \in \Theta^{(t-1)} \right\}. \tag{14}$$

Here, the constraint $\theta^{(t-1)} \in \Theta^{(t-1)}$ ensures that the domain captures the effect of training data perturbations across all previous iterations, while $\tilde{\mathcal{B}} \subset \mathcal{T}(\mathcal{D})$ represents perturbations in the current mini-batch. The former is necessary to represent the worst case $\theta^{(t-1)} \in \Theta^{(t-1)}$ in terms of the latter step. In general, the exact parameter-space bound $\Theta^{(t)}$ has a complicated form, as it encapsulates the entire history of training dynamics, including past gradients, dataset perturbations, and optimization steps. Directly maintaining and propagating this exact set is computationally intractable for all but the simplest models.

To address this, we restrict our analysis to specific classes of **abstract domains**, which over-approximate $\Theta^{(t)}$ in a practical manner while maintaining soundness. These abstract domains balance the trade-off between *tightness*, which ensures meaningful certification guarantees, and *tractability*, which enables efficient computation. This trade-off has been thoroughly investigated in the literature on certifying adversarial robustness, for example in Huchette et al. (2026). In particular, we will explore the following abstract domains:

- **Intervals:** A simple yet computationally efficient abstraction, where each parameter $\theta_j^{(t)}$ is independently bounded within an interval $[\underline{\theta}_j^{(t)}, \overline{\theta}_j^{(t)}]$. This approach enables fast propagation of parameter-space bounds (i.e., a box domain) through successive training iterations, but may introduce significant over-approximation error.

- **Polytopes:** A more expressive abstraction that captures dependencies among parameters by over-approximating $\Theta^{(t)}$ with a convex polytope $\{\theta \mid A\theta \leq b\}$. This domain can capture dependencies between parameters but requires solving a set of convex optimization problems at each step.

- **Mixed-Integer Domains:** The most precise domain, where $\Theta^{(t)}$ is encoded as a set of linear, quadratic, and integer constraints. This formulation can provide tight parameter-space bounds, but is computationally expensive, requiring the solution of NP-Hard non-convex optimization problems.

Each of these domains provides a different balance between computational efficiency and certification tightness, which we explore in subsequent sections.

5.3 Abstract Gradient Training as Constrained Optimization

At each iteration of AGT, the exact set of reachable parameters can be expressed as the feasible region of a constrained optimization problem. Given an initial parameter set $\Theta^{(0)}$, we characterize the set of all possible parameters at iteration T as:

$$
\begin{aligned}
&\theta^{(0)} \in \Theta^{(0)} && && \text{(Initial Conditions)}\\
&\widetilde{\mathcal{D}} = \left\{ \left(\tilde{x}^{(i)}, \tilde{y}^{(i)} \right) \right\}_{i=1}^{N} \in \mathcal{T}(\mathcal{D}) && && \text{(Dataset Perturbation)}\\
&\mathcal{B}^{(t)} = \left\{ \left(\tilde{x}^{(i)}, \tilde{y}^{(i)} \right) \right\}_{i \in \mathcal{I}^{(t)}} \subset \widetilde{\mathcal{D}} && t \in [T] && \text{(Batch Sampling)}\\
&\hat{y}^{(t,i)} = f\left(\tilde{x}^{(i)}, \theta^{(t-1)} \right) && i \in \mathcal{I}^{(t)}, t \in [T] && \text{(Forward Pass)}\\
&\delta^{(t,i)} = \nabla_\theta \mathcal{L}\left(\hat{y}^{(t,i)}, \tilde{y}^{(i)} \right) && i \in \mathcal{I}^{(t)}, t \in [T] && \text{(Backward Pass)}\\
&\theta^{(t)} = \theta^{(t-1)} - \frac{\alpha}{|\mathcal{B}^{(t)}|} \sum_{i \in \mathcal{I}^{(t)}} \delta^{(t,i)} && t \in [T] && \text{(Parameter Update)}
\end{aligned}
$$

Here, the fixed index sets $\mathcal{I}^{(t)} \subset [N]$ capture the data ordering assumption by assigning training samples to each batch $\mathcal{B}^{(t)}$.

The constraint corresponding to the forward pass has been extensively studied in the optimization literature, particularly through mixed-integer encodings of neural networks and their activation functions. In contrast, the backward pass has received significantly less attention in the context of constrained optimization. We provide exact formulations for both components of the optimization problem in Section 7.

The formulation above captures all possible parameter values reachable at time step T under the given training data perturbations. As mentioned above, directly computing the exact reachable set $\Theta^{(t)}$ is intractable for large-scale models, as it requires reasoning over all possible perturbations and optimization trajectories. Instead, this optimization-based formulation serves as a foundation from which *tractable relaxations* can be derived, which will be the subject of the following sections.

6. Efficient Relaxation via Interval Bound Propagation

Interval arithmetic is a well-established technique that favors computational efficiency over tightness in the certification of machine learning models (Gowal et al., 2018). Interval domains provide a simple yet scalable method for over-approximating the space of possible model parameters throughout training. By bounding each parameter independently within an interval, this approach allows for sound propagation of bounds while maintaining computational tractability.

6.1 The Interval Domain

In an interval-based representation, each parameter θ_j is independently bounded within an interval. For ease of notation, we denote a full set of parameter intervals as the interval domain $\boldsymbol{\theta} = \left[\theta^L, \theta^U\right]$, where membership is defined as:

$$\theta \in \boldsymbol{\theta} \implies \theta_j^L \leq \theta_j \leq \theta_j^U, \quad \forall j. \tag{15}$$

Interval domains offer a computationally efficient alternative to more complex abstractions, such as polytopes or mixed-integer domains. However, they can introduce significant *over-approximation error* due to the inherent independence assumption between parameters. Despite this limitation, interval bounds serve as a useful baseline for certifying robustness, particularly in large-scale models where finer-grained representations may be intractable.

Interval Arithmetic. To efficiently propagate interval-based bounds, we leverage the well-established framework of **interval arithmetic** Rump (1999), which extends standard arithmetic operations to interval-valued inputs. Let us denote intervals over matrices as $\boldsymbol{A} := [A^L, A^U] \subseteq \mathbb{R}^{n \times m}$ such that $A \in \boldsymbol{A} \implies A^L \leq A \leq A^U$. We define the following interval matrix arithmetic operations[1]:

$$\begin{aligned}
\text{Addition:} \quad & A + B \in [\boldsymbol{A} \oplus \boldsymbol{B}] \; \forall A \in \boldsymbol{A}, B \in \boldsymbol{B} \\
\text{Subtraction:} \quad & A - B \in [\boldsymbol{A} \ominus \boldsymbol{B}] \; \forall A \in \boldsymbol{A}, B \in \boldsymbol{B} \\
\text{Matrix multiplication:} \quad & A \times B \in [\boldsymbol{A} \otimes \boldsymbol{B}] \; \forall A \in \boldsymbol{A}, B \in \boldsymbol{B} \\
\text{Elementwise multiplication:} \quad & A \circ B \in [\boldsymbol{A} \odot \boldsymbol{B}] \;\; \forall A \in \boldsymbol{A}, B \in \boldsymbol{B} \\
\text{Scalar multiplication:} \quad & \alpha A \in [\alpha \boldsymbol{A}] \; \forall A \in \boldsymbol{A}
\end{aligned}$$

Each of these operations can be computed (or over-approximated) using standard interval arithmetic in at most 4x the computational cost of its non-interval counterpart. For example, interval matrix multiplication can be computed efficiently using Rump or Nguyen's algorithms (Rump, 1999; Nguyen, 2012). We denote interval vectors as $\boldsymbol{a} := \left[a^L, a^U\right]$ with analogous operations.

6.2 Interval Decomposition of Abstract Gradient Training.

Interval domains provide a natural representation for model parameters, perturbed data, model activations, and gradient updates within the Abstract Gradient Training framework.

1. We note the absence of interval division, which poses challenges when the divisor interval contains zero; we will not require this operation in our subsequent analysis.

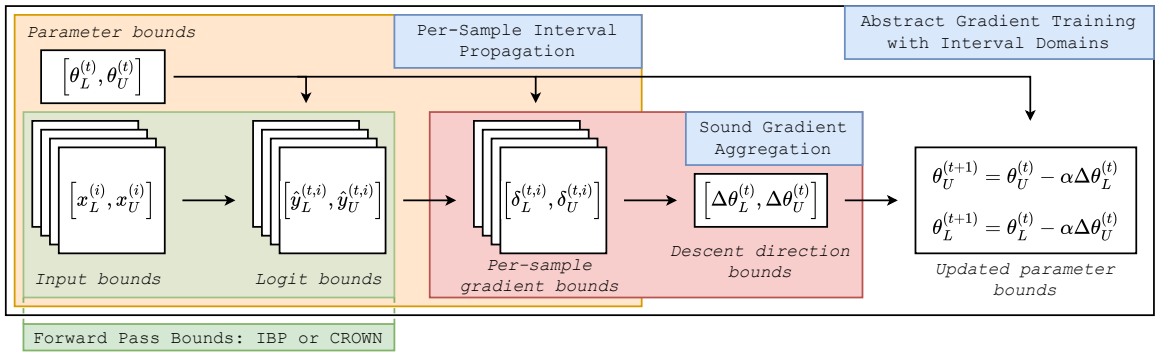


Figure 3: Summary of AGT with interval domains: (1) Forward pass bounds are computed using an interval method. (2) Per-sample gradient bounds are computed using interval backpropagation. (3) Descent direction bounds are computed using sound aggregation of per-sample gradients with respect to a given perturbation model. (4) The updated parameter interval is computed using interval arithmetic.

We begin the training process with the parameter interval $\boldsymbol{\theta^{(0)}} = [\theta^{(0)}, \theta^{(0)}]$, and seek to compute a valid parameter-space interval, $\boldsymbol{\theta^{(t)}}$, for each subsequent training iteration.

Each iteration updates model parameters based on the gradient of the loss function. In standard SGD, the parameter update at iteration t given by (4) can be re-written as:

$$\hat{y}^{(t,i)} = f\left(x^{(i)}, \theta^{(t-1)}\right), \quad \delta^{(t,i)} = \nabla_\theta \mathcal{L}\left(\hat{y}^{(t,i)}, y^{(i)}\right), \tag{16}$$

$$\Delta\theta^{(t)} = \frac{1}{|\mathcal{I}^{(t)}|} \sum_{i \in \mathcal{I}^{(t)}} \delta^{(t,i)}, \quad \theta^{(t)} = \theta^{(t-1)} - \alpha\Delta\theta^{(t)}, \tag{17}$$

where again the index set $\mathcal{I}^{(t)} \subset [N]$ captures the data ordering (i.e. the indices of the data-points included in the t-th batch). We will refer to $\Delta\theta^{(t)}$ as the average gradient, or *descent direction*, computed over the training batch $\mathcal{B}^{(t)}$. With interval domains, our objective is to compute sound interval enclosures over each of the operations above:

$$\hat{y}^{(t,i)} = \left[\hat{y}_L^{(t,i)}, \hat{y}_U^{(t,i)}\right], \quad \delta^{(t,i)} = \left[\delta_L^{(t,i)}, \delta_U^{(t,i)}\right], \tag{18}$$

$$\Delta\theta^{(t)} = \left[\Delta\theta_L^{(t)}, \Delta\theta_U^{(t)}\right], \quad \theta^{(t)} = \left[\theta_L^{(t)}, \theta_U^{(t)}\right]. \tag{19}$$

We address this challenge via a two-stage approach, as illustrated in Figure 3:

1. **Per-sample gradient bounds:** We first derive valid interval bounds for each *per-sample* gradient term $\delta^{(t,i)}$, by propagating bounds through both the forward and backward passes of the neural network. We instantiate two strategies for this step: interval bound propagation, and an extension of the CROWN algorithm adapted to the interval-weight setting.

2. **Sound aggregation:** We then apply a *sound aggregation mechanism* to bound the summation of these individual gradient terms, taking into account the specific

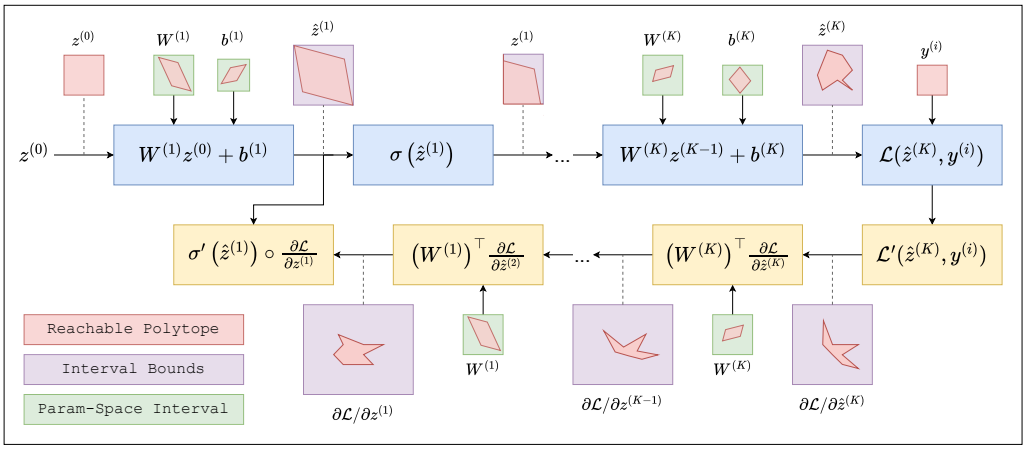


Figure 4: Bounding per-sample gradients using interval bound propagation. Top: Interval propagation through the forward pass, bounding each layer's activations with respect to both input and parameter intervals. Bottom: Interval propagation through the backward pass, bounding the gradient of the loss with respect to all model parameters. Gradients with respect to model parameters are omitted for clarity. The key challenge, compared to standard IBP for adversarial robustness, is that both the inputs *and* the network weights are interval-valued.

perturbation model $\mathcal{T}$, and ultimately producing a sound interval enclosure of the descent direction $\Delta\theta^{(t)}$. This step ensures that the resulting bounds reflect the fact that *up to* n data points may be altered.

The remainder of this section specifies the exact computations involved in each of these steps.

6.3 Computing Per-Sample Gradient Bounds

At a high level, bounding per-sample gradients requires propagating intervals through three stages (Figure 3): first, a forward pass through the network to bound the model's predictions; second, differentiation of the loss function to compute the initial (bounded) gradient signal; and third, a backward pass to bound the gradients with respect to model parameters. Each stage uses interval arithmetic to maintain sound enclosures, with the key challenge being that both the network inputs *and* the network parameters may be interval-valued.

Let $\delta^{(t,i)}$ denote the gradient of the loss function $\mathcal{L}$ with respect to the model parameters θ, evaluated for the i-th sample at the t-th training iteration:

$$\delta^{(t,i)} = \nabla_\theta \mathcal{L}\left(f\left(x^{(i)}, \theta^{(t-1)}\right), y^{(i)}\right). \tag{20}$$

Our objective is to derive sound interval bounds for $\delta^{(t,i)}$ that capture all perturbations introduced by the perturbation model $\mathcal{T}$. Depending on the specific choice of $\mathcal{T}$, these bounds must be sound with respect to:

- Only the parameter interval $\boldsymbol{\theta^{(t-1)}}$ (e.g., when considering sample removal/substitution).

- Both the parameter interval $\boldsymbol{\theta^{(t-1)}}$ and intervals representing perturbations of the input $x^{(i)} \in \boldsymbol{x^{(i)}}$ and label $y^{(i)} \in \boldsymbol{y^{(i)}}$ (e.g., when considering bounded input and label modifications).

For generality, we focus on the second, more comprehensive case. The first case can be treated as a special instance by setting $\boldsymbol{x^{(i)}}$ and $\boldsymbol{y^{(i)}}$ to zero-width intervals, effectively eliminating their contribution to the bounds and reducing the problem to one with parameter-space perturbations only.

Remark 3. *While the bounded perturbation model allows for general ℓ_p- norm perturbations, in this section we focus exclusively on their over-approximations using the ℓ_∞ norm. This approach provides a sound, though potentially looser, bound. For instance, the interval over the input feature space is expressed as $\boldsymbol{x^{(i)}} = \left[x^{(i)} - \epsilon, x^{(i)} + \epsilon\right]$.*

To compute sound gradient bounds, we decompose the gradient computation process into three independent stages, which we detail below.

1) Bounding the Forward Pass. Computing bounds on the forward pass of a neural network is straightforwardly defined using interval arithmetic. The parameter interval $\boldsymbol{\theta^{(t-1)}}$ defines intervals on the weights and biases of the network, represented as $W^{(k)} \in \boldsymbol{W^{(k)}}$ and $b^{(k)} \in \boldsymbol{b^{(k)}}$ for each layer $k = 1, \ldots, K$. Given these interval parameters, and an interval $\boldsymbol{z^{(0)}} = \boldsymbol{x}$ on the (perturbed) input to the network, we recursively propagate bounds through the network layers. The pre-activation and post-activation interval bounds at each layer are computed as:

$$\hat{\boldsymbol{z}}^{(\boldsymbol{k})} = \boldsymbol{W^{(k)}} \otimes \boldsymbol{z^{(k-1)}} \oplus \boldsymbol{b^{(k)}}, \quad \boldsymbol{z^{(k)}} = \sigma\left(\hat{\boldsymbol{z}}^{(\boldsymbol{k})}\right). \tag{21}$$

Here, $\hat{\boldsymbol{z}}^{(\boldsymbol{k})}$ represents the interval bounds on the pre-activation values, while $\boldsymbol{z}^{(k)}$ contains the bounds on the post-activation values. For a monotonic activation function σ, interval propagation is performed by independently applying σ to the lower and upper bounds of its input, i.e., $\sigma\left(\left[a^L, a^U\right]\right) = \left[\sigma\left(a^L\right), \sigma\left(a^U\right)\right]$.

Remark 4. *We note that it is possible to employ more precise (but computationally heavier) approaches in this step to compute tighter intervals over the forward pass. While this step differs from standard adversarial robustness certification due to the inclusion of intervals over the model parameters, many existing methods can be adapted to this more general setting. As an example, we provide an extension of the well-known CROWN algorithm to handle interval-valued parameters in Appendix B.*

2) Bounding the Loss Function. The first step of the backward pass involves computing the partial derivative of the loss $\mathcal{L}(\hat{y}, y)$ with respect to the network logits $\hat{y}$. The interval bounds must be valid with respect to both the logit interval $\hat{\boldsymbol{y}} = \hat{\boldsymbol{z}}^{(\boldsymbol{K})}$ (obtained from the forward pass) and the label interval $\boldsymbol{y}$ (given by the perturbation model).

For common loss functions such as cross-entropy and squared error loss, these gradients can be explicitly derived, allowing interval arithmetic to compute sound enclosures over the required terms. Consider the squared error loss, defined as $\mathcal{L}\left(\hat{y}, y\right) = \|\hat{y} - y\|_2^2$, which has a partial derivative given by $\partial\mathcal{L}/\partial\hat{y} = 2(\hat{y} - y)$. Applying interval arithmetic to this gradient expression yields a sound enclosure for the loss gradient:

$$\frac{\boldsymbol{\partial\mathcal{L}}}{\boldsymbol{\partial\hat{y}}} = 2(\hat{\boldsymbol{y}} \ominus \boldsymbol{y}). \tag{22}$$

Computing interval bounds on derivatives of other loss functions, e.g., cross-entropy loss, requires propagation through additional non-linear terms including the softmax function. Interval propagation for the cross-entropy and hinge loss functions are detailed in Appendix A.

3) Bounding the Backward Pass. Finally, to establish interval bounds on the full backward pass of the network, we extend the interval arithmetic-based approach of Wicker et al. (2022), which provides bounds on derivatives of the form $\partial\mathcal{L}/\partial z^{(k)}$. Our extension additionally computes sound bounds on gradients with respect to the network parameters (i.e., $\partial\mathcal{L}/\partial W^{(k)}$ and $\partial\mathcal{L}/\partial b^{(k)}$).

For a feed-forward neural network, the backward pass using standard back-propagation follows the recursive update rule:

$$\frac{\partial\mathcal{L}}{\partial z^{(k-1)}} = \left(W^{(k)}\right)^\top \frac{\partial\mathcal{L}}{\partial\hat{z}^{(k)}}, \quad \frac{\partial\mathcal{L}}{\partial\hat{z}^{(k)}} = \sigma'\left(\hat{z}^{(k)}\right) \circ \frac{\partial\mathcal{L}}{\partial z^{(k)}} \tag{23}$$

$$\frac{\partial\mathcal{L}}{\partial W^{(k)}} = \frac{\partial\mathcal{L}}{\partial\hat{z}^{(k)}}\left(z^{(k-1)}\right)^\top, \quad \frac{\partial\mathcal{L}}{\partial b^{(k)}} = \frac{\partial\mathcal{L}}{\partial\hat{z}^{(k)}}, \tag{24}$$

where $\sigma'(\cdot)$ denotes the derivative of the activation function, and $\circ$ represents element-wise multiplication. To bound the backward pass, we leverage the following intervals:

1. Interval bounds on intermediate activations $z^{(k)}$ and pre-activations $\hat{z}^{(k-1)}$ obtained from the forward pass.
2. Bounds on the initial loss gradient, $\frac{\partial\mathcal{L}}{\partial\hat{z}^{(K)}}$, computed using the interval propagation procedure described previously.

Using these components, we derive sound interval bounds for the gradients through the following interval matrix operations:

$$\frac{\partial\mathcal{L}}{\partial\boldsymbol{z}^{(k-1)}} = \left(\boldsymbol{W}^{(k)}\right)^\top \otimes \frac{\partial\mathcal{L}}{\partial\hat{\boldsymbol{z}}^{(k)}}, \quad \frac{\partial\mathcal{L}}{\partial\hat{\boldsymbol{z}}^{(k)}} = \sigma'\left(\hat{\boldsymbol{z}}^{(k)}\right) \odot \frac{\partial\mathcal{L}}{\partial\boldsymbol{z}^{(k)}} \tag{25}$$

$$\frac{\partial\mathcal{L}}{\partial\boldsymbol{W}^{(k)}} = \frac{\partial\mathcal{L}}{\partial\hat{\boldsymbol{z}}^{(k)}} \otimes \left(\boldsymbol{z}^{(k-1)}\right)^\top, \quad \frac{\partial\mathcal{L}}{\partial\boldsymbol{b}^{(k)}} = \frac{\partial\mathcal{L}}{\partial\hat{\boldsymbol{z}}^{(k)}}, \tag{26}$$

where all interval operations follow previously defined rules. Interval propagation through the activation function derivative $\sigma'(\cdot)$ depends on the specific activation function employed. For ReLU networks, the derivative is the Heaviside function, for which interval bounds can be soundly propagated as $\sigma'([a^L, a^U]) = [\sigma'(a^L), \sigma'(a^U)]$.

The resulting gradient intervals provide a complete interval enclosure of all gradients of the model for all $W^{(k)} \in \boldsymbol{W}^{(k)}$, $b^{(k)} \in \boldsymbol{b}^{(k)}$ given a single sample $\tilde{x} \in \tilde{\boldsymbol{x}}$, $\tilde{y} \in \tilde{\boldsymbol{y}}$.

6.4 Computing Descent Direction Bounds via Sound Aggregation

The main idea behind sound aggregation is that, when only n out of b samples in a batch can be perturbed, the descent direction is not as uncertain as a naive interval sum would suggest. By exploiting the constraint that at most n samples are affected, we can compute tighter bounds by selecting the worst-case contribution of the n most impactful perturbations at each parameter index independently. The index-wise approach is still an over-approximation (since different parameter indices may have different worst-case samples), but it yields bounds

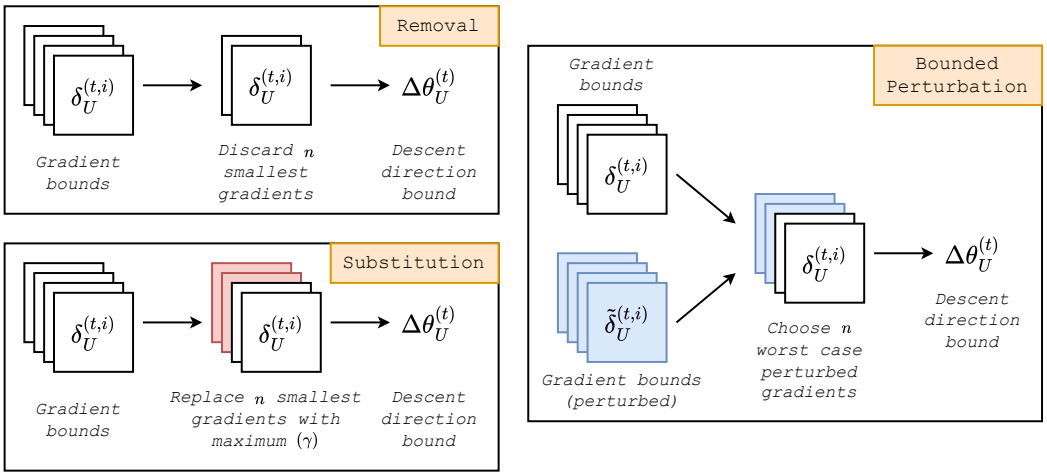


Figure 5: Sound aggregation of per-sample gradients for our three perturbation models. We illustrate the procedure for upper-bounding a single parameter, with the full bounds obtained by independently repeating the process at each parameter index. The lower bound is not shown but follows analogous operations. Each perturbation model implies different structural constraints on the descent direction.

that are significantly tighter than naively treating all samples as potentially perturbed. This step requires careful consideration of the aggregation process across the per-sample gradients, while accounting for the specific perturbation model $\mathcal{T}$.

The descent direction in standard SGD is computed as an average of individual gradients across the batch, $\Delta\theta^{(t)} = \sum_i \delta^{(t,i)}/|\mathcal{B}^{(t)}|$. When considering an interval bound on this term, a naive approach would be to simply compute the interval sum of per-sample gradient bounds and divide by the batch size. However, this approach yields overly conservative bounds, as it ignores other dependencies implied by the constraints of the perturbation model $\mathcal{T}$, such as being able to modify only up to n training samples in the dataset. Instead, we develop tailored procedures for bounding the descent direction that exploit the structure of specific perturbation models to obtain valid and tighter bounds.

Before presenting the aggregation procedures for each perturbation model, we establish the following notation:

- Let $\delta_L^{(t,i)}$ and $\delta_U^{(t,i)}$ be bounds capturing the effect of perturbations in previous iterations, satisfying: $\delta_L^{(t,i)} \leq \delta^{(t,i)} \leq \delta_U^{(t,i)}$, $\forall \delta^{(t,i)} \in \left\{ \nabla_\theta \mathcal{L}\left(f\left(x^{(i)}, \tilde{\theta}^{(t-1)}\right), y^{(i)}\right) \mid \tilde{\theta}^{(t-1)} \in \boldsymbol{\theta^{(t-1)}} \right\}$.

- Let $\tilde{\delta}_L^{(t,i)}$ and $\tilde{\delta}_U^{(t,i)}$ be bounds on the *combined* effect of previous perturbations and worst-case bounded perturbations of the current batch, satisfying: $\tilde{\delta}_L^{(t,i)} \leq \tilde{\delta}^{(t,i)} \leq \tilde{\delta}_U^{(t,i)}$ for all $\tilde{\delta}^{(t,i)}$ in

$$\left\{ \nabla_\theta \mathcal{L}\left(f\left(\tilde{x}, \tilde{\theta}^{(t-1)}\right), \tilde{y}\right) \mid \tilde{\theta}^{(t-1)} \in \boldsymbol{\theta^{(t-1)}}, \|x^{(i)} - \tilde{x}\|_p \leq \epsilon, \|y^{(i)} - \tilde{y}\|_q \leq \nu \right\}.$$

These bounds can be computed using the interval propagation procedure outlined in the previous section. For the second case, we over-approximate the norm constraints using the intervals $\tilde{\boldsymbol{x}} = [x^{(i)} - \epsilon, x^{(i)} + \epsilon]$ and $\tilde{\boldsymbol{y}} = [y^{(i)} - \nu, y^{(i)} + \nu]$.

Throughout our analysis, we assume that the perturbation model $\mathcal{T}$ can act independently on each batch $\mathcal{B}^{(t)}$. While this is a relaxation compared to $\mathcal{T}$ acting once on the entire dataset, it remains sound and facilitates tractable computation of bounds. Consequently, our guarantees hold for up to n modified points per batch. However, any certificates defined over the entire dataset remain valid only for up to n modified points per dataset, reflecting the worst-case scenario in which all modifications occur within a single batch.

Remark 5. *In poisoning scenarios, if we adopt a more relaxed attack model – assuming the adversary lacks access to data ordering (i.e., is not an online adversary as in Zhang et al. (2020)) and that poisoned data is uniformly distributed across batches – we can specify our guarantees with respect to a "safe" proportion of poisoned samples per batch by applying statistical bounds on the likelihood of exceeding a certain number of poisoned samples within any batch. This approach yields less conservative, though probabilistic, guarantees compared to the worst-case deterministic bounds presented here.*

A summary of our sound aggregation mechanisms for each perturbation model is shown in Figure 5. Below we detail the specifics of each mechanism.

6.4.1 Bounding the Descent Direction under Removal.

The most straightforward perturbation model is the *removal* of n training samples from the batch. To bound this case, we observe that for any particular parameter index, the average gradient is maximized by removing the n samples with the smallest gradients, and minimized by removing the n samples with the largest gradients.

The effect of the removal perturbation model can be bounded by applying this reasoning at *each parameter index* independently. This represents an over-approximation of the actual effects, since any realizable training data perturbation must select the same n points to remove across all parameter indices. However, this trade-off between parameter independence and computational efficiency is necessary when working within the interval domain.

The following theorem formalizes how to compute bounds on the descent direction under the data removal perturbation model:

Theorem 6. *Given a nominal batch $\mathcal{B}^{(t)} = \left\{\left(\tilde{x}^{(i)}, \tilde{y}^{(i)}\right)\right\}_{i=1}^{b}$ of size b, an interval parameter domain $\boldsymbol{\theta^{(t-1)}}$, the descent direction*

$$\Delta\theta^{(t)} = \frac{1}{|\tilde{\mathcal{B}}^{(t)}|} \sum_{\left(\tilde{x}^{(i)}, \tilde{y}^{(i)}\right) \in \tilde{\mathcal{B}}^{(t)}} \nabla_\theta \mathcal{L}\left(f\left(\tilde{x}^{(i)}, \theta^{(t-1)}\right), \tilde{y}^{(i)}\right) \tag{27}$$

is bounded element-wise by

$$\Delta\theta_L^{(t)} = \frac{1}{b-n}\left(\underset{b-n}{\mathrm{SEMin}}\left\{\delta_L^{(t,i)}\right\}_{i=1}^{b}\right), \quad \Delta\theta_U^{(t)} = \frac{1}{b-n}\left(\underset{b-n}{\mathrm{SEMax}}\left\{\delta_U^{(t,i)}\right\}_{i=1}^{b}\right) \tag{28}$$

for any perturbed batch $\tilde{\mathcal{B}}^{(t)}$ derived from $\mathcal{B}^{(t)}$ by removing up to n data points.

In this theorem, the operations SEMax_a and SEMin_a correspond to taking the sum of the top-a and bottom-a elements at each index, sorted by magnitude, respectively.

6.4.2 Bounding the Descent Direction under Substitution.

Extending this approach to the case of data substitution introduces a challenge in bounding the descent direction. Specifically, the inclusion of an arbitrary data point can have an unbounded effect on the model's gradient, making it impossible to enclose the descent direction within intervals without modifying the SGD update step. To address this, we introduce a *clipped* SGD update, defined as

$$\Delta\theta^{(t)} = \frac{1}{|\mathcal{B}^{(t)}|} \sum_{(\tilde{x}^{(i)},\tilde{y}^{(i)})\in\mathcal{B}^{(t)}} \text{Clip}_\kappa \left[\nabla_\theta \mathcal{L}\left(f\big(\tilde{x}^{(i)}, \theta^{(t-1)}\big), \tilde{y}^{(i)}\right)\right], \tag{29}$$

where Clip_κ is a truncation operator that clamps all elements of its input to be between $-\kappa$ and κ, while leaving those within the range unchanged. This allows consideration of *arbitrary substitutions*, but the resulting reachable parameter sets are only valid with respect to the training process that applies the same clipping procedure to the nominal parameters.

Remark 7. *Per-sample gradient clipping is widely used in machine learning to restrict the impact of individual training examples, such as in differentially private training (Abadi et al., 2016). Unlike the commonly used ℓ_2-norm clipping, here we opt for the truncation operator, as it is better suited for interval bound propagation.*

Since the clipping procedure is applied element-wise, we can conclude that the worst-case addition of n datapoints can only have an effect of up to $\pm n\kappa$ on the sum of the gradients. Specifically, we have the following result:

Theorem 8. *Given a nominal batch $\mathcal{B}^{(t)} = \left\{\big(x^{(i)}, y^{(i)}\big)\right\}_{i=1}^{b}$, and an interval parameter domain $\boldsymbol{\theta^{(t-1)}}$, the descent direction*

$$\Delta\theta^{(t)} = \frac{1}{|\tilde{\mathcal{B}}^{(t)}|} \sum_{\left(x^{(i)},y^{(i)}\right)\in\tilde{\mathcal{B}}^{(t)}} \text{Clip}_\kappa \left[\nabla_\theta \mathcal{L}\left(f\big(x^{(i)}, \theta^{(t-1)}\big), y^{(i)}\right)\right] \tag{30}$$

is bounded element-wise by

$$\Delta\theta_L^{(t)} = \frac{1}{b}\left(\underset{b-n}{\text{SEMin}}\left\{\delta_L^{(t,i)}\right\}_{i=1}^{b} - n\kappa\mathbf{1}\right), \quad \Delta\theta_U^{(t)} = \frac{1}{b}\left(\underset{b-n}{\text{SEMax}}\left\{\delta_U^{(t,i)}\right\}_{i=1}^{b} + n\kappa\mathbf{1}\right) \tag{31}$$

for any perturbed batch $\tilde{\mathcal{B}}^{(t)}$ derived from $\mathcal{B}^{(t)}$ removing up to n data-points and adding up to n arbitrary data-points.

6.4.3 Bounding the Descent Direction under Bounded Perturbation.

Finally, we analyze the effect of bounded perturbations to existing training data samples. We begin by noting that perturbing the i-th data point impacts our bounds by an additional contribution of $\left(\tilde{\delta}_U^{(t,i)} - \delta_U^{(t,i)}\right)$ to the upper bound and $\left(\tilde{\delta}_L^{(t,i)} - \delta_L^{(t,i)}\right)$ to the lower bound. By independently selecting the worst-case n contributions at each parameter index, we derive an efficient method for computing a valid interval over the overall descent direction. Formally, we state the following result:

Algorithm 1 Abstract Gradient Training with Interval Domains

input: f - model, $\mathcal{D}$ - dataset, $\{\mathcal{I}\}_{t=1}^{T}$ - data ordering, $\theta^{(0)}$ - parameter initialization, α - learning rate, $\mathcal{T}$ - perturbation model.
output: $[\theta_L^{(T)}, \theta_U^{(T)}]$ - valid parameter space domain
$\left[\theta_L^{(0)}, \theta_U^{(0)}\right] \leftarrow \left[\theta^{(0)}, \theta^{(0)}\right]$
for $t = 1, \ldots, T$ **do**
  for $i \in \mathcal{I}^{(t)}$ **do**
    $[\hat{y}_L^{(t,i)}, \hat{y}_U^{(t,i)}] \leftarrow$ BOUND_FORWARD_PASS$(x^{(i)}, [\theta_L^{(t)}, \theta_U^{(t)}], \mathcal{T})$ ▷ IBP §6.3 or CROWN §B
    $[\delta_L^{(t,i)}, \delta_U^{(t,i)}] \leftarrow$ BOUND_BACKWARD_PASS$([\hat{y}_L^{(t,i)}, \hat{y}_U^{(t,i)}], [\theta_L^{(t)}, \theta_U^{(t)}], y^{(i)}, \mathcal{T})$ ▷ IBP §6.3
  end for
  $[\Delta\theta_L^{(t)}, \Delta\theta_U^{(t)}] \leftarrow$ SOUND_AGGREGATION$(\{[\delta_L^{(t,i)}, \delta_U^{(t,i)}]\}_i, \mathcal{T})$ ▷ Aggregation Mechanism §6.4
  $[\theta_L^{(t)}, \theta_U^{(t)}] \leftarrow [\theta_L^{(t-1)} - \alpha\Delta\theta_U^{(t)}, \theta_U^{(t-1)} - \alpha\Delta\theta_L^{(t)}]$ ▷ Parameter Interval Update
end for
return $[\theta_L^{(T)}, \theta_U^{(T)}]$

Theorem 9. *Given a nominal batch* $\mathcal{B}^{(t)} = \left\{\left(x^{(i)}, y^{(i)}\right)\right\}_{i=1}^{b}$*, and an interval parameter domain* $\boldsymbol{\theta^{(t-1)}}$*, the descent direction*

$$\Delta\theta^{(t)} = \frac{1}{|\tilde{\mathcal{B}}^{(t)}|} \sum_{\left(\tilde{x}^{(i)}, \tilde{y}^{(i)}\right) \in \tilde{\mathcal{B}}^{(t)}} \nabla_\theta \mathcal{L}\left(f\left(x^{(i)}, \theta^{(t-1)}\right), y^{(i)}\right) \tag{32}$$

is bounded element-wise by

$$\Delta\theta^{L} = \frac{1}{b}\left(\underset{n}{\text{SEMin}}\left\{\tilde{\delta}_L^{(t,i)} - \delta_L^{(t,i)}\right\}_{i=1}^{b} + \sum_{i=1}^{b} \delta_L^{(t,i)}\right), \tag{33}$$

$$\Delta\theta^{U} = \frac{1}{b}\left(\underset{n}{\text{SEMax}}\left\{\tilde{\delta}_U^{(t,i)} - \delta_U^{(t,i)}\right\}_{i=1}^{b} + \sum_{i=1}^{b} \delta_U^{(t,i)}\right). \tag{34}$$

for any batch $\tilde{\mathcal{B}}^{(t)}$ *derived from* $\mathcal{B}^{(t)}$ *by bounded perturbation within* $\mathcal{T}_{bounded}^{n,p,\epsilon,q,\nu}$ *given by* (7)*.*

As before, the above result soundly bounds the descent direction for all parameters, with the update being applied independently at each parameter index. This approach, while computationally efficient for propagating interval enclosures, introduces a potentially loose over-approximation, as the n worst-case points for a given parameter index are unlikely to be the same as those for the parameters at all other indices.

6.5 Summary of Abstract Gradient Training with Interval Domains

The instantiation of Abstract Gradient Training using interval arithmetic facilitates computationally efficient certification of large-scale training pipelines. However, this efficiency comes at the cost of potentially significant over-approximation of the reachable parameter space at each training iteration, which may reduce the tightness of the resulting guarantees. Algorithm 1 summarizes the key steps of this approach in pseudo-code; we now outline additional computational considerations that arise in practice.

Computational Complexity. The computational cost of AGT with interval domains is dependent on the forward-pass bounding method. Simple IBP incurs a cost roughly four times that of standard forward and backward passes. Similarly, our CROWN-based bounds increase the cost by a factor of four compared to the original CROWN algorithm, which has a complexity of $\mathcal{O}(m^2n^3)$ for an m-layer network with n neurons per layer and n outputs (Zhang et al., 2018). The SEMin / SEMax operations, while $\mathcal{O}(b)$ per index, are efficiently parallelizable on GPUs. Empirically, we observe that AGT with interval domains typically adds less than a 2x runtime overhead to standard training.

Limitations. While AGT with interval domains provides valid parameter space bounds for any gradient-based training algorithm, the tightness of these bounds varies based on architecture, hyperparameters, and training procedures. Notably, the algorithm assumes simultaneous worst-case perturbation at each parameter index, in addition to our relaxation of worst-case batch perturbations. This approach, while computationally efficient for propagating interval enclosures, introduces a potentially loose over-approximation, as the n worst-case perturbed points for any parameter index are unlikely to be the same for all indices. Consequently, we found that achieving meaningful guarantees with this approach often necessitates larger batch sizes or fewer training epochs. Furthermore, loss functions like multi-class cross-entropy exhibit loose interval relaxations, leading to weaker guarantees compared to those for regression or binary classification.

7. Tight Analysis via Optimization-Based Certification

In the previous section, we established a sound and computationally efficient approach for bounding the training dynamics using interval propagation. While efficient, this approach introduces over-approximations at each iteration, which can lead to loose or even vacuous certification guarantees. To achieve a tighter certification, we revisit the optimization-based formulation of AGT, aiming for a more precise characterization of training perturbations.

This section begins with a mixed-integer programming (MIP) formulation that provides an exact representation of our abstract training dynamics. We then explore convex and linear relaxations that balance computational efficiency and certification tightness. Finally, we introduce decomposition strategies that partition the AGT problem into smaller sub-problems, potentially enabling more scalable certification for larger-scale training settings.

7.1 Mixed-Integer Programming

To obtain an exact representation of our abstract training dynamics, we formulate the AGT constrained optimization problem as a **mixed-integer program (MIP)**. Unlike interval propagation, which provides a single lower and upper bound for each parameter, MIP enables an exact characterization of the worst-case perturbations by explicitly encoding discrete constraints and non-linear dependencies over the entire training trajectory (Sosnin et al., 2026). However, this precision comes at the cost of increased computational complexity, with mixed-integer programs being NP-Hard, in general.

Mixed-integer programming is a powerful optimization framework that extends linear programming by incorporating both continuous and discrete variables. A standard mixed-

integer program takes the form:

$$\min\ h(a,b) \tag{35}$$
$$\text{s.t. } g_j(a,b) \leq 0, \qquad j = 1,\ldots,m, \tag{36}$$
$$a \in \mathbb{R}^{n_c}, b \in \mathbb{Z}^{n_i} \tag{37}$$

where a represents n_c continuous decision variables, b represents n_i integer decision variables, and $g_i(\cdot)$ are m constraint functions that may include both linear and non-linear components. The inclusion of integer variables allows MIP to model complex non-convex decision problems, such as those including neural networks, logical constraints or piecewise functions.

The structure of the objective and constraint functions defines the specific class of optimization problem. When the objective function h is linear and the constraints g_i include quadratic or bilinear terms in the decision variables, the problem falls under the category of a *mixed-integer quadratically constrained program (MIQCP)*. If both the objective function and all constraints are purely linear, the formulation simplifies to a *mixed-integer linear program (MILP)*, which generally allows for more efficient solving methods. Finally, if the integer variables are relaxed to continuous domains, then the problem may be a *quadratically constrained program (QCP)* or *linear program (LP)*.

In the context of AGT, MIQCPs offer the greatest expressive power, allowing for exact characterization of the training dynamics. However, MIQCPs often require the most expensive solvers, whereas relaxations can provide a practical trade-off between precision and efficiency.

7.2 Mixed-Integer Quadratically Constrained Representations of AGT

To leverage MIP for AGT certification, we must express the constraints governing training perturbations, model structure, and parameter updates in a form compatible with mixed-integer programming. This requires reformulating key components of the training process, such as activation functions, gradient computations, and adversarial modifications, using mixed-integer quadratic constraints.

Relation to previous works. The formulations in this section are closely related to those of Lorenz et al. (2024a), which introduced a MIQCP formulation for certification against data poisoning. While that work also considers training data perturbations, the forward pass, and gradient computations as mixed-integer programs, our approach offers a more comprehensive framework. Specifically, while Lorenz et al. (2024a) consider only bounded perturbations to all training examples, we expand the scope to a general set of perturbation models. Moreover, their formulation is restricted to single training iteration per optimization problem, whereas our methodology considers the entire training procedure as a unified optimization problem.

7.2.1 Dataset and Perturbation Models

In our framework, training data perturbations arise from adversarial modifications or constraints imposed by differential privacy and machine unlearning. To certify properties of the model with respect to these perturbation models, we must encode these modifications as feasible constraint sets. We provide an explicit mixed-integer quadratic formulation for each of the perturbation models below:

Bounded Feature and Label Perturbation. An adversary is allowed to modify a subset of up to n training samples within predefined norms (here we present the formulation for the $p = \infty$ and $q = 0$ norms). To begin, we add a binary indicator variable $s \in \{0,1\}^{|\mathcal{D}|}$ to represent the adversary's choice of poisoning targets. Then, the feature and label space data poisoning constraints can be represented by the following linear constraints:

$$\begin{aligned} x^{(i)} - \epsilon s_i \leq \tilde{x}^{(i)} \leq x^{(i)} + \epsilon s_i, \qquad & i = 1, \ldots, |\mathcal{D}| \\ y^{(i)}(1 - s_i) \leq \tilde{y}^{(i)} \leq y^{(i)}(1 - s_i) + s_i, \quad & i = 1, \ldots, |\mathcal{D}| \\ \tilde{x}^{(i)} \in \mathbb{R}^{n_x}, \tilde{y}^{(i)} \in \{0,1\} \qquad & i = 1, \ldots, |\mathcal{D}| \\ \sum_{i=1}^{|\mathcal{D}|} s_i \leq n, & \end{aligned} \tag{38}$$

The set of constraints above results in $s_i = 0 \implies (\tilde{x}^{(i)}, \tilde{y}^{(i)}) = (x^{(i)}, y^{(i)})$, while $s_i = 1$ indicates that the i-th training sample may be poisoned. The above formulation covers binary classification only, but may be adapted to suit regression and multi-class problems.

Removal of Training Data. We can encode for removal of training data similarly to the above, by again introducing an indicator variable $s \in \{0,1\}^{|\mathcal{D}|}$ indicating the exclusion of a particular training sample. Then, by adding the following constraint to each parameter update step, we can encode for the removal of n or fewer training points:

$$\sum_{i=1}^{|\mathcal{D}|} s_i \leq n, \qquad \theta^{(t)} = \theta^{(t-1)} - \frac{\alpha}{|\mathcal{I}^{(t)}| - \sum_{i \in \mathcal{I}^{(i)}} s_i} \sum_{i \in \mathcal{I}^{(i)}} (1 - s_i)\delta^{(t,i)}, \qquad t \in [T]. \tag{39}$$

Here $s_i = 1$ denotes the exclusion of the i-th training point from all training batches.

Substitution of Training Data. Finally, the substitution of existing training data by arbitrary training data can be handled by considering some bounded domain $\mathcal{X}^a$ from which the substituted data may be drawn. This assumes the existence of such a set, though in practice such a set can be constructed by imposing bounds on the allowable feature space. We can encode the added data through additional variables $\tilde{x}^{(j)} \in \mathcal{X}^a, \tilde{y}^{(j)} \in \{0,1\}$ for $j \in 1, \ldots, n$. The appropriate forward and backward passes encoded via the model constraints discussed below to obtain variables for the gradients of the model, denoted $\tilde{\delta}^{(t,j)}$. Then, similar to the data removal case above we introduce binary decision variables $s \in \{0,1\}^{|\mathcal{D}|}$ indicating the inclusion of each original data-point in the dataset, and $\tilde{s} \in \{0,1\}^n$ indicating the index in the dataset which the perturbed point replaces. The substituted data can then be expressed with the following constraints on the parameter update:

$$\sum_{i=1}^{|\mathcal{D}|} s_i = \sum_{j=1}^{n} \tilde{s}_j \leq n, \quad \theta^{(t)} = \theta^{(t-1)} - \frac{\alpha}{|\mathcal{I}^{(t)}|} \left[\sum_{i \in \mathcal{I}^{(i)}} (1 - s_i)\delta^{(t,i)} + \sum_{j=1}^{n} \tilde{s}_j \tilde{\delta}^{(t,j)} \right], \quad t \in [T]. \tag{40}$$

This formulation requires that when $s_i = 0$, the original training sample remains unchanged, while when $s_i = 1$, the sample is substituted with a new valid point from the given domain, with n controlling the maximum number of substitutions allowed.

7.2.2 Encoding the Forward Pass

Representation of neural network models as mixed-integer programs is well established within the certification literature (Huchette et al., 2026). By encoding activation functions and linear layers as mixed-integer constraints, MIP formulations enable precise characterization of a network's behavior under adversarial perturbations. Unlike standard certification settings, in our case the network parameters $\theta = \left\{(W^{(i)}, b^{(i)})\right\}_{i=1}^{K}$ are also decision variables, leading to quadratic constraints. In this work, we focus on the ReLU activation, which is representable as a mixed-integer linear program. Several such MILP representations exist, such as those based on variable partitions (Tsay et al., 2021) or the convex hull (Anderson et al., 2020), but here we present the simplest formulation, known as the 'Big-M' formulation. The k-th layer in the network, given by $z^{(k)} = \text{ReLU}\left(W^{(k)} z^{(k-1)} + b^{(k)}\right)$, can be exactly represented as

$$z^{(k)} \geq W^{(k)} z^{(k-1)} + b^{(k)}, \tag{41}$$

$$z^{(k)} \leq W^{(k)} z^{(k-1)} + b^{(k)} - M(1-s), \tag{42}$$

$$0 \leq z^{(k)} \leq Ms, \tag{43}$$

$$s \in \{0,1\}^{n_k}. \tag{44}$$

where M is a sufficiently large constant and s is a binary decision variable denoting the activation state of each neuron. We highlight that constraints (41) and (42) are *quadratic* constraints, since $W^{(k)}$ and $z^{(k-1)}$ are both decision variables.

7.2.3 Encoding the Backward Pass

To integrate the loss function into our optimization formulation, we must represent the partial derivative of the loss with respect to the model outputs. Since not all loss functions are exactly MIQCP-representable, we focus only on the squared error and hinge loss here. The more common cross entropy loss function cannot be encoded exactly, though relaxations are possible; we leave this possibility to future works.

Squared error loss: The simplest loss function is again the squared error loss, which can be encoded via the linear constraint $\frac{\partial \mathcal{L}}{\partial \hat{y}} = 2(\hat{y} - y)$.

Hinge loss: The hinge loss, given by $\mathcal{L}(\hat{y}, y) = \max\{0, 1 - \hat{y}y\}$, can be re-written in terms of a ReLU activation function $\mathcal{L}(\hat{y}, y) = \text{ReLU}(1 - \hat{y}y)$. Let the intermediate term by denoted by a variable $u = 1 - \hat{y}y$. Then, we first encode $\text{ReLU}(u)$ using the standard formulation above, noting that the gradient of the ReLU term is given by the binary variable s present in the big-M formulation. The gradient of the loss with respect to the logits $\hat{y}$ is then represented by the bilinear equality constraint $\partial \mathcal{L} / \partial \hat{y} = -ys$.

The backwards pass through the neural network is straightforward to encode using bilinear constraints. We again note that the gradient of each ReLU activation is already encoded via the binary variable s in the Big-M formulation above.

7.2.4 Parameter Updates

Finally, we can complete our formulation of the AGT training dynamics by including the updated model parameters. For this constraint, we include a copy of the network parameters at each training iteration as a decision variable in our optimization problem. With the

exception of the removal perturbation model (which is discussed separately above), the linear constraints linking each iteration are simply

$$\theta^{(t)} = \theta^{(t-1)} - \frac{\alpha}{|\mathcal{I}^{(t)}|} \sum_{i \in \mathcal{I}^{(t)}} \delta^{(t,i)}, \qquad t = 1, \ldots, T. \tag{45}$$

The decision variables $\delta^{(t,i)}$ are defined via the forward and backward pass constraints above.

7.2.5 Choosing the MIP Objective Function

The formulations outlined above allow us to encode the full training trajectory as a single MIQCP, capturing the constraints governing the evolution of model parameters. To derive a final valid parameter-space domain for inference-time certification, we must compute bounds on the final parameter state, $\theta^{(T)}$.

There are multiple ways to define this final parameter domain. A straightforward approach is to optimize for the minimum and maximum values of each parameter $\theta_j^{(T)}$ individually. By setting the objective function to $\min / \max \theta_j^{(T)}$ for each index j and solving the corresponding optimization problems, we obtain a valid interval enclosure for $\theta^{(T)}$. This approach yields an interval domain that is significantly tighter than those obtained via interval bound propagation, as it fully accounts for dependencies among parameters during training.

For an even more refined representation of the final parameter space, we can compute the faces of a bounded polytope enclosing $\theta^{(T)}$. Specifically, by selecting a set of face directions A and maximizing each linear projection $A_i\theta^{(T)}$ as an objective function, we can construct a polyhedral domain of the form:

$$\left\{\theta^{(T)} \mid A\theta^{(T)} \leq c\right\} \tag{46}$$

This polytope representation provides a structured enclosure of the final parameter space, which can lead to tighter and more informative inference-time certification guarantees.

7.3 Convex Relaxations of AGT

While the MIQCP formulation provides a precise characterization of the training dynamics, its computational complexity can be prohibitive, especially for large-scale training pipelines. The presence of integer variables in the optimization formulation combined with bilinear constraints significantly increases the difficulty of obtaining solutions.

To improve scalability while preserving the soundness of our certification guarantees, we explore relaxations that transform the original MIQCP into potentially more tractable linear optimization problems, which admit polynomial-time solutions. By replacing non-convex constraints with convex outer approximations and relaxing integer constraints into continuous domains, we can formulate optimization problems that are more efficiently solvable using standard convex programming techniques.

Linearization of Integer Constraints. In our MIQCP formulation, integer variables play a crucial role in encoding discrete decisions, such as the selection of perturbed training samples or the activation behavior of piecewise-linear functions. However, the inclusion of integer constraints significantly increases the computational complexity of solving the optimization problem. Therefore, we investigate the *continuous relaxation* of our formulation,

which replaces any binary variables $s_i \in \{0, 1\}$ with a continuous counterpart $s_i \in [0, 1]$. This relaxation weakens the formulation, for example by allowing the bounded threat model to partially poison more than n points, but with the same total poisoning magnitude.

Linearization of Quadratic Constraints. Many of the constraints in our MIQCP formulation involve bilinear terms, such the product of model activations with network weight matrices. While this exactly encodes the model constraints, these bilinear terms introduce non-convexity into the optimization problem. To improve tractability, we apply linearization techniques that replace bilinear terms by their over-approximate linear envelopes.

A standard approach for linearizing a bilinear term $c = ab$, where $a \in [a_L, a_U]$ and $b \in [b_L, b_U]$, is to impose McCormick envelope constraints (McCormick, 1976):

$$c \geq a_L b + a b_L - a_L b_L, \tag{47}$$
$$c \geq a_U b + a b_U - a_U b_U, \tag{48}$$
$$c \leq a_L b + a b_U - a_L b_U, \tag{49}$$
$$c \leq a_U b + a b_L - a_U b_L. \tag{50}$$

By applying this linearization to all bilinear terms in a formulation, we can form a convex outer approximation of the optimization problem, losing exactness but potentially leading to faster solve times.

Bounds Tightening. The effectiveness of convex relaxations hinges on the tightness of the intermediate variable bounds used to construct the original formulation. When linearizing mixed-integer or quadratic constraints, the gap between the mixed-integer optimal solution and its convex relaxation directly characterizes the "tightness" of the relaxation. Big-M formulations are particularly sensitive to the choice of M, with excessively large values leading to weak relaxations (Tsay et al., 2021). Similarly, McCormick envelopes require tight intermediate bounds (a_L, a_U, b_L, b_U) to generate meaningful convex approximations (McCormick, 1976). Recent advances have demonstrated substantial improvements in certification performance, particularly in deep problems, through iterative refinement of these bounds (Sosnin and Tsay, 2024). In this work, we employ interval arithmetic for bounding intermediate variables, but more sophisticated bound tightening procedures, such as optimization-based bound tightening, represent promising directions for future work.

7.4 Decomposition of the AGT Optimization Problem

The formulations discussed above capture the entire training trajectory within a single large-scale optimization problem. While this provides a precise characterization of the parameter evolution, the resulting problem requires T copies of the model parameters, as well as a copy of the entire training dataset. To improve scalability, we explore decomposition strategies that break the problem into smaller sub-problems that require only a subset of variables and constraints to be included.

Rolling-Horizon Decomposition. Instead of optimizing the entire training trajectory at once, rolling-horizon decomposition reformulates the AGT optimization problem as a sequence of smaller subproblems, each considering a limited time horizon. This approach

Table 2: Comparison of AGT formulation size for a neural network with L layers of W neurons per layer, trained on a dataset of size N with M features per entry for T iterations of batch-size B.

	# Binary Variables	**# Continuous Variables**	**# Quadratic Constraints**
MIQCP	$\mathcal{O}(WLBT + N)$	$\mathcal{O}(W^2LT+WLBT+NM)$	$\mathcal{O}(W^2LT + WLBT)$
MILP	$\mathcal{O}(WLBT + N)$	$\mathcal{O}(W^2LBT + NM)$	0
QCP	0	$\mathcal{O}(W^2LT+WLBT+NM)$	$\mathcal{O}(W^2LT + WLBT)$
LP	0	$\mathcal{O}(W^2LBT + NM)$	0

mitigates the computational complexity and memory constraints associated with tracking the full trajectory while still capturing the dependencies between successive updates.

At a given training iteration t, we compute bounds on the parameters $\theta^{(t)}$ using only a fixed-length window of the most recent updates, rather than the entire training history. Specifically, we consider only the past w iterations, where w is a chosen window size. In the next step, we compute bounds at iteration $t + p$, where $p \leq w$ is the window step size. This sliding window approach captures recent updates while ignoring older ones, whose influence on the current parameters naturally diminishes over time.

This decomposition strikes a balance between computational efficiency and bound tightness. By limiting the horizon of analysis, we enable incremental and scalable computation of certified bounds, making reachability analysis tractable over the course of training.

Sample-Wise Decomposition. While the rolling-horizon decomposition above can significantly reduce the size of the AGT sub-problems, it still requires consideration of an entire batch of training data within a single optimization problem. For large models, this may still be computationally prohibitive, so another natural decomposition is a sample-wise decomposition, where the bounding problem is broken down across individual training samples, and per-sample gradients are bounded independently. This requires the solution to many more sub-problems (at least an upper and lower bound for each parameter gradient at each training sample) but these smaller problems are solved independently and are readily parallelizable. We leave this possibility to future works.

7.5 Summary of Optimization-Based AGT

This section established a hierarchy of optimization approaches, ranging from exact MIQCP formulations to computationally efficient relaxations and decompositions. Table 2 provides a comparison based on variable and constraint complexity. We observe that relaxing quadratic constraints, despite usually simplifying optimization, dramatically increases the formulation's size. The practical implications of these trade-offs are explored empirically in Section 8.

Choice of Formulation. The formulation can be chosen to balance between computational cost and certificate tightness. We summarize practical recommendations as follows:

- *Interval-based AGT* (Section 6) is the method of choice when computational scalability is the primary concern: it incurs roughly a 2-4× runtime overhead over standard training

and is applicable to models with thousands of parameters trained over many iterations. However, interval bounds can degrade significantly over long training horizons due to the accumulation of over-approximation error at each iteration.

- *LP relaxations* can offer a middle ground, providing tighter bounds than interval arithmetic while remaining solvable in polynomial time. They are most practical for moderate-sized models and short-to-medium training horizons.
- *MIP-based formulations* (MILP and MIQCP) provide the tightest bounds but are computationally expensive (NP-Hard in general) and are most practical for small models or short training horizons where precise guarantees are critical, e.g., see Sosnin et al. (2026). Rolling-horizon decomposition (Section 7.4) can substantially reduce the computational burden, at the expense of some tightness.

8. Core Certification Objectives and Experimental Evaluation

This section establishes the fundamental certification objectives underlying our target applications and outlines their certification via AGT. We then provide an experimental validation of the certification approach on an illustrative 2d dataset. Code to reproduce all experiments is available at `github.com/psosnin/AbstractGradientTraining`, with complete experimental details provided in Appendix C.

8.1 Basic Certification Objectives

We introduce two point-wise certification objectives for a model's prediction on an input-label pair (x, y) with respect to perturbations to the training data. Both are defined with respect to Θ, the set containing all possible model parameters resulting from training on a dataset consistent with a perturbation model $\mathcal{T}$, which can be computed using AGT.

A prediction of a model f is *certified correct* if $f(x, \tilde{\theta}) = y$ for all $\tilde{\theta} \in \Theta$. This certificate implies that no feasible perturbation to the training data would cause the model to make an incorrect prediction on x. This certificate can only hold if the nominal model already predicts the correct label. A prediction is *certified stable* if $f(x, \theta) = f(x, \tilde{\theta})$ for all $\tilde{\theta} \in \Theta$. This weaker notion guarantees the prediction remains the same, but not necessarily correct, isolating the effect of training data perturbations from the model's baseline accuracy.

These certificates can be computed by deriving a sound interval bound, $[\hat{y}_L, \hat{y}_U]$, on the model's output for all $\tilde{\theta} \in \Theta$ using IBP, CROWN, or optimization formulations. If the lower bound of the logit for the predicted class t exceeds the upper bounds of all other class logits (i.e., $[\hat{y}_L]_t \geq [\hat{y}_U]_{t'}$ for all $t' \neq t$) then the prediction is certified stable. If the original prediction was also correct, it is certified correct.

While the certificates discussed above apply to individual predictions, AGT's parameter-space bounds can be used to compute collective certificates over batches of data points, which provide improved tightness (cf. point-wise certificates) at a computational cost (Chen et al., 2022). Our subsequent analysis of AGT will therefore focus on point-wise certification.

Computational Cost Overview. For interval-based AGT, the dominant cost is the forward and backward pass bound propagation at each training iteration, which incurs roughly a 2–4× overhead compared to standard training (Section 6). Practically, we observe

that interval-based AGT typically adds less than a 2× end-to-end runtime overhead over standard training. For optimization-based AGT, the dominant cost is the MIP or LP solve. As shown in Table 3, solve times range from seconds for LP formulations to hours for MIQCP, scaling with the number of training iterations and model parameters included. Rolling-horizon decomposition (Section 7.4) can mitigate this cost by limiting the number of iterations per subproblem, at the expense of some bound tightness.

8.2 Illustrative 2d Example

We first illustrate our approach on a binary classification task using a two-dimensional half-moons dataset with 128 training samples. We train a support vector machine (SVM) with cubic feature expansions, resulting in a 10-parameter model. We evaluate robustness in a label-flipping poisoning setting, where a single training label ($n = 1$) can be flipped to the opposite class. This setup allows us to compare our certified bounds with an enumeration-based ground truth. All optimization-based bounds are solved in Gurobi on a machine with 2x AMD EPYC 9334 CPU, using a per-subproblem timeout of 3600 seconds.

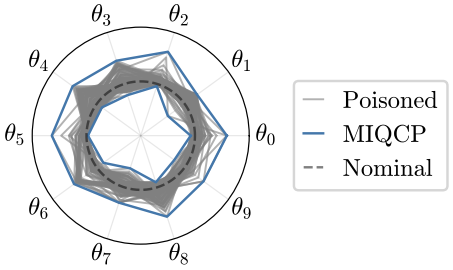
Comparison of MIQCP with Empirical Attacks. We evaluate our exact MIQCP formulation against an exhaustive enumeration of all possible label flips on the dataset. Figure 6a illustrates the element-wise parameter bounds along with the parameters obtained under label flipping. Axes such are scaled for easy visual comparison. We observe that for each parameter, the bound is tight, i.e., there is at least one label flip that causes the parameter at that index to attain its lower / upper bound.

In Figure 6b, we plot the decision boundaries of all models trained with a single label flip. Given the small size of this example dataset, a single label flip can notably perturb the model's decision boundary. The red and blue regions visualize where point-wise robustness certificates hold, based on the final certified parameter interval obtained from the exact MIQCP bounds. We observe only a small gap between the certified regions and the true poisoning vulnerabilities. This gap is attributed to the independence assumption within the interval domain, which, despite the individual upper and lower parameter bounds being exact (i.e., achievable by an actual label flip), introduces some additional looseness.

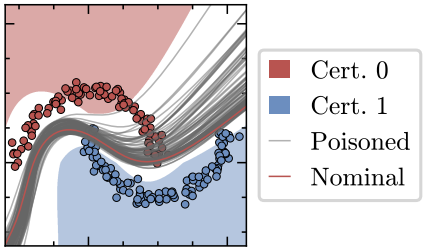
Comparison of Bounding Methods. We now compare four optimization-based formulations (LP, QP, MILP, MIQCP) and an interval-based approach (IBP). For each method, we construct interval bounds over the final model parameters by solving independent minimization and maximization problems for each parameter.

For each optimization-based method, we compare both the formulation of the full training pipeline, as well as certification via decomposition into smaller subproblems based on a *rolling horizon* strategy. Specifically, we solve non-overlapping optimization subproblems spanning w training iterations. Larger horizons are more precise and involve less sub-problems, but each individual solve is more costly. We also compare against IBP, which is efficient but extremely loose in this low-data setting.

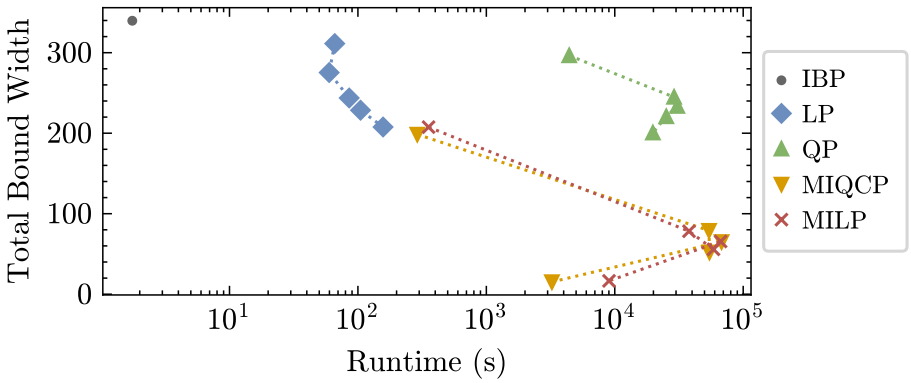
Figure 6c plots the total certified bound width $\|\theta_U - \theta_L\|_1$ against runtime for each method and horizon size. We observe that all optimization-based methods outperform IBP in bound tightness, though they come with a significant runtime cost (2–5 orders of magnitude higher). Additionally, the choice of relaxation significantly impacts both runtime and tightness. LP-



(a) Element-wise parameter bounds vs empirical label flipping attacks.



(b) Decision boundaries of poisoned models vs certified regions from MIQCP.



(c) Runtime vs. tightness trade-off for bounding methods. Bounds using the same relaxation but different horizons are grouped; Table 3 provides a comprehensive list of results for each formulation.

Figure 6: Certification on the half-moons dataset and comparison with empirical label-flipping attacks. Note that MIP-based methods produce relatively tighter bounds over longer horizons, while interval-based bounds are independent of horizon length.

and MIQCP-based bounds offer a favorable balance compared to QP and MILP formulations. MILP and MIQCP are the tightest methods, with MIQCP being the best overall in terms of both runtime and tightness for the "Full" formulation.

Comparing Perturbation Models. Finally, we evaluate the impact of various perturbation models using a larger half-moons dataset of $N = 3000$ samples. We train a logistic regression model using interval-based AGT using the same cubic features as in the previous analysis. We visualize these trained models in Figure 7 for different values of perturbation budget n. Each colored region indicates the area for which we cannot provide certificates of prediction stability for the given perturbation model. As expected, points far from the decision boundary of the model exhibit stronger certificates than those close to the boundary.

Of the four perturbation models considered, we observe that unlearning (data removal) yields the tightest bounds. This is due to the fact that the data removal aggregation mechanism operates only on the observed gradient bounds within each given batch. In contrast,

Table 3: Certified bound width and corresponding runtime (in seconds) across methods and horizon length (in batches). Each cell reports the certified bound width, with runtime shown in parentheses. For IBP, bound width accumulates across iterations, while optimization-based methods can capture inter-iteration dependencies, leading to significantly tighter bounds over the full training trajectory (the "Full" row).

Horizon	**IBP**	**LP**	**QP**	**MILP**	**MIQCP**
1	339.7 (2.2)	311.3 (66)	296.4 (4418)	207.5 (355)	198.5 (290)
3	–	275.4 (59.7)	244.8 (28954)	78.2 (37849)	79.0 (54192)
5	–	243.7 (85.5)	233.5 (30689)	55.8 (58724)	52.2 (54634)
7	–	228.5 (105)	220.7 (25170)	65.5 (66705)	65.1 (67847)
14 (Full)	–	207.7 (157)	201.9 (33157)	16.6 (9020)	**15.3** (3240)

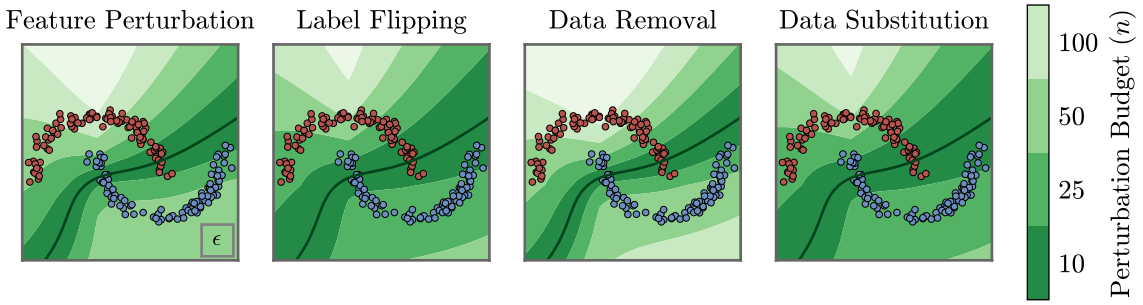


Figure 7: Regions where our certification holds for various perturbation models on a half-moons dataset. Bounds computed using interval-based AGT.

the feature and label perturbation models must account for the worst-case adversarial modification, which necessarily admits more over-approximation. Similarly, the data substitution model, which encompasses all other perturbation models, produces strictly looser bounds.

9. Practical Case Studies: Data Poisoning

This section explores certified guarantees specifically in the context of adversarial data poisoning. In such attacks, an adversary manipulates a subset of the training data to degrade a model's performance. Our goal is to certify robustness against these attacks by establishing sound upper bounds on their success.

We will begin by outlining the specific certification objectives for data poisoning. We then apply them to larger-scale problems using case studies from our previous work (Sosnin et al., 2024), focusing exclusively on interval-based Abstract Gradient Training.

Remark 10. *All guarantees reported in this section are valid under the assumptions stated in Section 5: fixed data ordering, parameter initialization, and training hyperparameters. In practice, this corresponds to certifying a specific realization of the training pipeline.*

9.1 Certification Objectives and Perturbation Models

The choice of the perturbation model $\mathcal{T}$ defines the adversarial capabilities that we are certifying against. For example, a **bounded adversary** that can only make bounded modifications, such as adding imperceptible triggers for backdoor attacks, is modeled by $\mathcal{T}^{n,p,\epsilon,q,\nu}_{\text{bounded}}$. In contrast, the more powerful **substitution model**, $\mathcal{T}^{n}_{\text{subs.}}$, represents an adversary that can arbitrarily replace entire training samples. Certification is only valid within the specific constraints of the chosen perturbation model.

We outline certified bounds for common attack goals:

- *Untargeted Poisoning:* These attacks aim to make the model unusable by preventing training convergence. The objective is to maximize the average loss over a dataset $\mathcal{D}_{\text{cert}}$:

$$J(\tilde{\theta}) = \frac{1}{|\mathcal{D}_{\text{cert}}|} \sum_{\mathcal{D}_{\text{cert}}} \mathcal{L}\big(f(x^{(i)}, \tilde{\theta}), y^{(i)}\big). \tag{51}$$

  An upper bound on $J(\tilde{\theta})$ provides a certificate on the attack's success.

- *Targeted Poisoning:* The goal is to make predictions fall outside a **safe set** of outputs $S(x^{(i)}, y^{(i)})$. The objective is the fraction of data points where this occurs:

$$J(\tilde{\theta}) = \frac{1}{|\mathcal{D}_{\text{cert}}|} \sum_{\mathcal{D}_{\text{cert}}} \mathbb{1}\big(f(x^{(i)}, \tilde{\theta}) \notin S(x^{(i)}, y^{(i)})\big). \tag{52}$$

  A sound upper bound on this objective limits the attack's success rate. When the safe set is the ground-truth label, a lower bound on $1 - J(\tilde{\theta})$ is called the **certified accuracy**.

- *Backdoor Poisoning:* These attacks introduce a latent vulnerability by assuming the adversary can also manipulate inputs at test time. The objective is to make the model produce predictions outside the safe set for a manipulated input $x^\star \in V(x^{(i)})$ (e.g., an ℓ_∞ ball around the input):

$$J(\tilde{\theta}) = \frac{1}{|\mathcal{D}_{\text{cert}}|} \sum_{i=1}^{k} \mathbb{1}\big(\exists x^\star \in V(x^{(i)}) \; s.t. \; f^M(x^\star) \notin S(x^{(i)}, y^{(i)})\big). \tag{53}$$

  The certification for this objective must account for both training data perturbations and test-time triggers to ensure soundness.

Each of the poisoning objectives above can be bounded using the certificates described in Section 8.1. This can be achieved through a point-wise decomposition of the batch certificates (e.g., using IBP or CROWN) or through joint consideration of $\mathcal{D}_{\text{cert}}$ using optimization formulations. We note that in the backdoor case, the inference-time perturbation $x^\star \in V(x^{(i)})$ must be included in any bound propagation or optimization formulation to ensure soundness with respect to trigger manipulations.

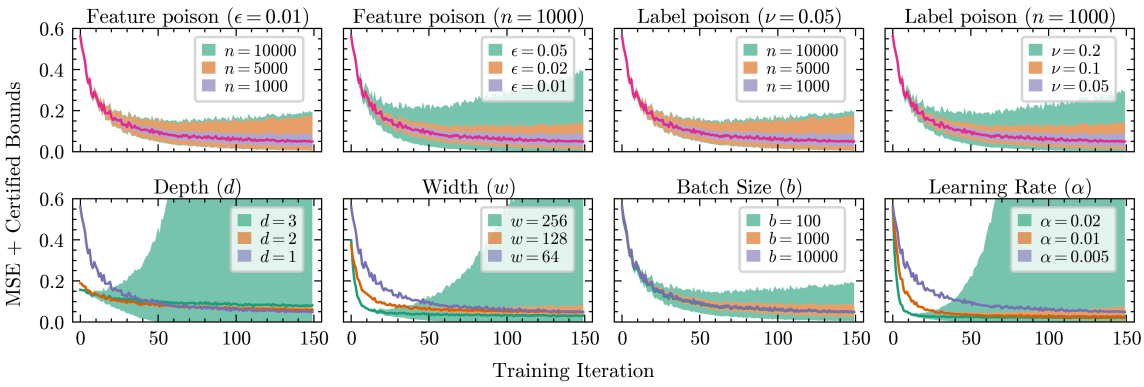


Figure 8: Mean squared error bounds on the UCI-houseelectric dataset for a bounded adversary. Top: Effect of adversary strength. Bottom: Effect of model/training hyperparameters (with $n = 100, \epsilon = 0.01, \nu = 0$). Where not stated, $p = q = \infty, d = 1, w = 50, b = 10000, \alpha = 0.02$.

9.2 UCI Regression

We begin by examining a straightforward regression model for the household electric power consumption dataset ('houseelectric') from the UCI repository (Hebrail and Berard, 2012). We train fully connected neural networks using interval-based AGT and track the evolution of our certified bounds on the model's mean squared error throughout training, as shown in Figure 8. Specifically, the colored regions indicate certified upper and lower bounds on the model's mean squared error under the given training configuration and poisoning adversary.

Figure 8 (top) illustrates the evolution of our MSE certificates for a neural network with one hidden layer of 50 neurons under various poisoning attack configurations. As expected, increasing any of the parameters n, ϵ, or ν leads to looser certified performance bounds. We note that we are able to obtain non-vacuous bounds for even $n = 10000$, which corresponds to the entire dataset being potentially poisoned.

Figure 8 (bottom) shows the progression of our bounds during training under a fixed poisoning attack for various hyperparameter settings. Generally, we find that increasing the number of model parameters (increasing either in width or depth) results in weaker guarantees. Larger batch sizes yield tighter bounds, as the fixed number of poisoned samples n is distributed across more training data, reducing their relative impact. Increasing the learning rate accelerates convergence but causes the certified bounds to degrade faster.

9.3 Comparison with Baselines (MNIST Digit Classification)

We evaluate our method in the multi-class setting of MNIST digit classification, focusing specifically on label-flipping attacks. Unlike in binary classification or regression, interval-based AGT struggles to produce tight guarantees in multi-class problems (see Section 6). To address this, we apply PCA to project the input data into a 32-dimensional feature space. Since we assume the features are clean and only the labels may be corrupted, this

dimensionality reduction is certifiably sound. Similar preprocessing steps are also common in prior works on certified label poisoning, such as SS-DPA (Levine and Feizi, 2020).

Figure 9 compares the certified accuracy of our method to two existing approaches: DPA (Levine and Feizi, 2020) and randomized smoothing (RS) (Rosenfeld et al., 2020). While these baselines serve similar goals, they differ significantly in scope and efficiency. RS applies only to linear models and label-flipping attacks, whereas our approach generalizes to a wider range of models and poisoning capabilities. DPA is more flexible in the types of adversaries it handles, but at the cost of substantially higher computational overhead and major changes to the training pipeline.

To illustrate this overhead, the DPA results in Figure 9 (reproduced from Levine and Feizi (2020)) use an ensemble of 3000 models. This alone introduces a 3000× increase in memory cost compared to standard training. In addition, each ensemble member is larger than our base model and trained under a less limited setup. The total runtime for generating certified guarantees with DPA is approximately 20 hours (0.4 minutes per model), whereas AGT computes certified bounds in just a few seconds. In terms of certified performance, AGT can outperform RS when the number of flipped labels is small. However, across all tested scenarios, DPA consistently provides stronger guarantees than AGT.

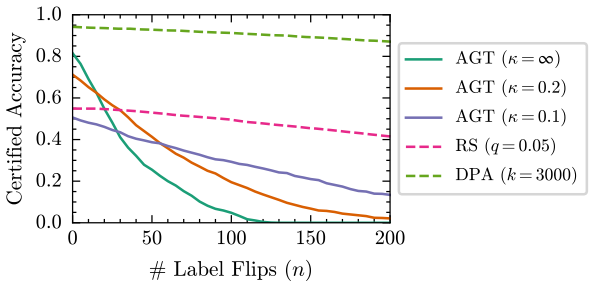


Figure 9: Certified accuracy on the MNIST dataset under label-flipping. Results for DPA and RS reproduced from Figures 1 of Levine and Feizi (2020) and Rosenfeld et al. (2020), respectively.

9.4 MedMNIST Image Classification

We next explore a fine-tuning scenario using the retinal OCT dataset (OCTMNIST) (Yang et al., 2021), which includes four classes: one normal and three abnormal. We focus on the binary classification task of distinguishing normal from abnormal samples.

Our experiments use the "small" architecture from Gowal et al. (2018), consisting of two convolutional layers (with widths 16 and 32) followed by a fully connected layer with 100 units. The fine-tuning procedure proceeds as follows: we pre-train the model using only three of the four classes, excluding the rarest class (Drusen), via the robust training method from Wicker et al. (2022). The robust pre-training ensures the base model is resilient to feature perturbations during subsequent fine-tuning. We then simulate an attack scenario in which Drusen samples are potentially poisoned and introduced during fine-tuning as an additional abnormal class. During this phase, we train only the final dense layer using mini-batches containing a mix of 50% Drusen samples (i.e., 3000 out of 6000 samples per batch).

Figure 10 presents how certified accuracy bounds degrade as the strength of the poisoning attack increases. In the case of feature-only poisoning, we find that attacks with $\epsilon > 0.02$ on roughly $n \approx 500$ samples are sufficient to produce certified bounds that fall below the accuracy of the original pre-trained model. When allowing both feature and label poisoning,

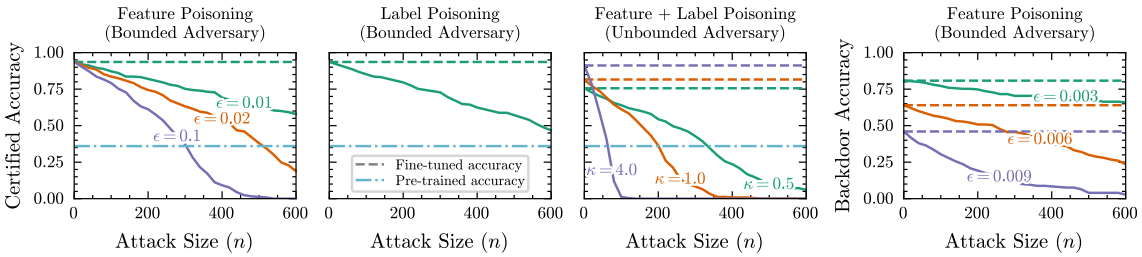


Figure 10: Certified accuracy (left) and backdoor accuracy (right) for a binary classifier fine-tuned on the Drusen class of OCTMNIST for an attack size up to 10% poisoned data per batch ($b = 6000, p = \infty, q = 0, \nu = 1$). Dashed lines show the nominal accuracy of each fine-tuned model.

the bounds degrade more rapidly, as expected. Increasing the clipping parameter κ can improve certified accuracy in this setting, at the expense of nominal model performance (Figure 10, center right). Under label-only poisoning, the model proves more robust: $n \geq 600$ mislabeled samples are required for the certified accuracy bound to drop to that of the original pre-trained model, corresponding to 20% of Drusen samples per batch being mislabeled.

Finally, we consider a backdoor attack scenario in which the adversary's strength at training and inference time is matched. Even without poisoning, the model is vulnerable to small adversarial perturbations at inference time: an ϵ as low as 0.009 is sufficient to reduce certified accuracy on backdoored inputs to below 50%. As the adversary's strength increases, the certified accuracy continues to decline.

9.5 Data Poisoning in Self-Driving Applications

Finally, we apply our method to a steering angle prediction task for autonomous driving, where the goal is to predict the vehicle's steering angle from an input image (Bojarski et al., 2016). The model architecture consists of convolutional layers with 24, 36, 48, and 64 filters, followed by fully connected layers with 100, 50, and 10 units.

Our fine-tuning setup mirrors the previous experiment. The model is first pre-trained on videos 2 through 6 of the Udacity self-driving car dataset[2]. We then fine-tune only the dense layers using data from video 1, which features challenging lighting conditions, while allowing for the possibility of label poisoning during this phase.

Figure 11 presents certified bounds on mean squared error (MSE) for the video 1 data and illustrates how these bounds impact predicted steering angles. As in previous experiments, fine-tuning improves nominal performance on the new data. However, the certified MSE bounds degrade as the number of potentially poisoned samples increases (Figure 11, left), with the rate of degradation highly sensitive to the poisoning strength parameter ν.

2. `github.com/udacity/self-driving-car/tree/master`

Figure 11: Left: Fine-tuning PilotNet on unseen data with a bounded label-poisoning attack ($q = \infty$). Right: Steering angle prediction before and after fine-tuning ($n = 300, q = \infty, \nu = 0.01$), indicated by the angle of the lines.

10. Applications in Unlearning and Differential Privacy

10.1 Machine Unlearning

Certification with respect to the removal of data points is of particular interest in the field of machine unlearning. While our method does not fit precisely within existing machine unlearning settings, our bounds deterministically bound the distance between a trained model and an unlearned model. By employing our parameter space bounds, we can characterize aspects of model behavior under unlearning, such as prediction stability. State-of-the-art approaches in certified machine unlearning, such as SISA (Bourtoule et al., 2021), typically operate by training an ensemble of classifiers on disjoint sets of user data. Upon receiving an unlearning request, the affected model is taken offline to undergo partial re-training.

A potential application of our bounds is to allow certification that individual model predictions would remain unchanged under re-training with up to n data points removed. Specifically, we can achieve this by first training a model using AGT under the perturbation model $\mathcal{T}^n_{\text{removal}}$; at inference time, we can then compute certificates of prediction stability with respect to the resulting parameter-space domain. This capability has the following practical implications: if such certificates are provided for all queries to the model, the system may potentially remain online with no risk to user data privacy, provided that the number of unlearning requests remains below n.

10.2 Differentially Private Prediction

Finally, we outline the usage of our parameter-space bounds in differentially private prediction. Here we provide an overview of a procedure to bound the *smooth sensitivity* of model predictions; for further details on this quantity and a comprehensive privacy analysis of our approach, we refer the reader to our previous work (Wicker et al., 2025).

10.2.1 Prediction Sensitivity

In differential privacy, the *sensitivity* of a prediction is a measure of how much the output of a function, such as a model's prediction, can change when a single individual's data is added

to or removed from the dataset. The sensitivity can be used to calibrate additive noise that can ensure that a model's output satisfies Definition 1 for a given (ϵ, δ).

The *global sensitivity* represents the largest possible change in a function's output when comparing any two datasets that differ by a single data point. For instance, in binary classification, the maximum possible change in the classifier's output is 1.

However, for many predictions, global sensitivity significantly overestimates the actual change in the model's output that results from a single data change. This overestimation leads to adding an excessive amount of noise to the prediction to preserve privacy, which can degrade the model's accuracy. This has led to the development of tighter, data-dependent notions of sensitivity, namely the *smooth sensitivity*:

Definition 11 (Smooth Sensitivity, Nissim et al. (2007))**.** *The β-smooth*[3] *sensitivity of the model prediction $f(x, \theta)$ trained on a dataset $\mathcal{D}$ via training mechanism M is*

$$\mathrm{SS}^{\beta}(f(x,\cdot),\mathcal{D}) = \max_{n\in\mathbb{N}^{+}} e^{-\beta n} A^{n}(f(x,\cdot),\mathcal{D}) \tag{54}$$

where $A^n(f, \mathcal{D})$ is the maximum local sensitivity, defined as

$$A^{n}(f(x,\cdot),\mathcal{D}) = \max_{\tilde{\mathcal{D}}\in\mathcal{T}^{n}_{subs.}(\mathcal{D})} \mathrm{LS}(f(x,\cdot),\tilde{\mathcal{D}}). \tag{55}$$

Here, $\mathrm{LS}(f(x,\cdot),\mathcal{D})$ *is the local sensitivity, given by*

$$\mathrm{LS}(f(x,\cdot),\mathcal{D}) = \max_{\tilde{\mathcal{D}}\in\mathcal{T}^{1}_{subs.}(\mathcal{D})} \|f(x,\theta) - f(x,\tilde{\theta})\|_1 \ \textit{s.t.}\ \theta = M(f,\mathcal{D}), \tilde{\theta} = M(f,\tilde{\mathcal{D}}). \tag{56}$$

Our previous work established the following result that constitutes a sound upper bound on the smooth sensitivity in terms of point-wise certificates of prediction stability.

Theorem 12 (Certified Upper Bound on Smooth Sensitivity, Wicker et al. (2025))**.** *Let $f(x, \theta)$ denote the prediction of a binary classifier with parameters θ trained on a dataset $\mathcal{D}$ at a point x. If $f(x, \theta)$ is stable at a distance of n' with respect to the perturbation model $\mathcal{T}^{n'}_{subs.}$, then the following provides an upper bound on the β-smooth sensitivity:*

$$\mathrm{SS}^{\beta}(f(x,\cdot),D) = \max_{n\in\mathbb{N}^{+}} e^{-\beta n} A^{n}(f(x,\cdot),D) \leq e^{-\beta n'} \tag{57}$$

To use the above, we first compute valid parameter space bounds for a fixed set of n values. Then, for a given query x, we attempt to compute certificates of prediction stability for the largest possible n within our set. Then Theorem 12 can be applied to upper-bound the smooth sensitivity.

Experimental Validation of Smooth Sensitivity Bounds. Figure 12 depicts the upper bound on smooth sensitivity of a model trained on the halfmoons dataset ($N = 3000$). For each bounding method, parameter-space bounds are computed for $n = 1, \ldots, 15$ during model training. These bounds are then used to upper-bound the smooth sensitivity according to Theorem 12. Noting that the global sensitivity for binary classification takes a value of 1, we observe that along the decision boundary, the smooth sensitivity bound cannot improve

3. Here β is a parameter that takes any value $\leq \epsilon/6$.

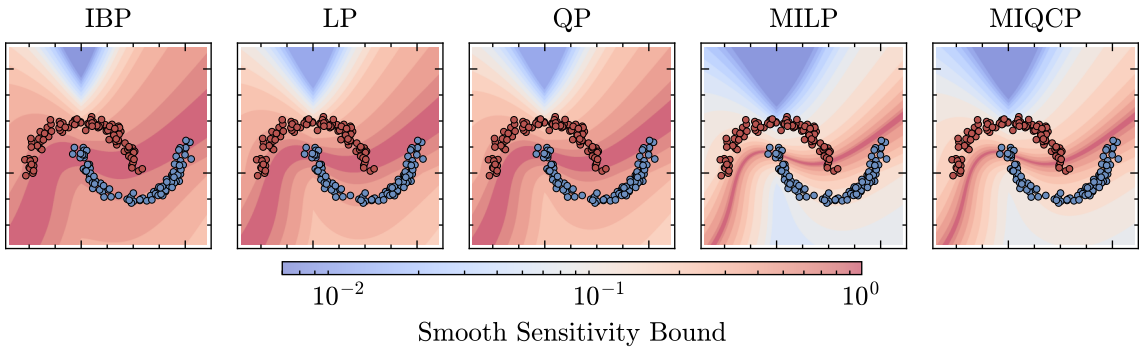


Figure 12: Visualization of smooth sensitivity bounds obtained using various AGT bounding strategies for privacy parameters $\epsilon = 2.0, \delta = 0$. A horizon length of 1 is used for all optimization-based methods.

over the global bound. However, for predictions further from the decision boundary the smooth sensitivity can have smooth sensitivity bounds several orders of magnitude lower than the global sensitivity, with tighter bounding achieving the tightest bounds. As we will demonstrate below, having tighter bounds on the smooth sensitivity leads to improved downstream performance in the private prediction setting.

10.2.2 Private Prediction using Smooth Sensitivity

In order to ensure that the the prediction of a model satisfies differential privacy, it is necessary to add noise to the model outputs before releasing them. In binary classification, for example, we can privatize predictions using the following response mechanism:

$$R(x) = \begin{cases} 1, & \text{if } f(x, \theta) + z > 0.5, \\ 0, & \text{otherwise}, \end{cases} \tag{58}$$

where z is a noise term drawn from a distribution chosen to satisfy Definition 1.

A common approach is the *Laplace mechanism*, which adds noise from $\text{Lap}(1/\epsilon)$, ensuring $(\epsilon, 0)$-differential privacy. However, this approach relies on global sensitivity, leading to excessive noise in many cases.

To improve privacy-utility tradeoffs, Nissim et al. (2007) showed that drawing

$$z \sim \text{Cauchy}\left(\frac{6\,\text{SS}^{\beta}(f(x, \cdot), \mathcal{D})}{\epsilon}\right) \tag{59}$$

ensures $(\epsilon, 0)$-differential privacy for $\beta \leq \epsilon/6$. Moreover, any upper bound on $\text{SS}^{\beta}(f(x, \cdot), \mathcal{D})$ can be used to calibrate noise.

We use AGT to compute a certified upper bound on smooth sensitivity, which we then use to adjust the noise level. This significantly reduces the magnitude of noise required to meet Definition 1, thereby enhancing the privacy-utility tradeoff in private prediction.

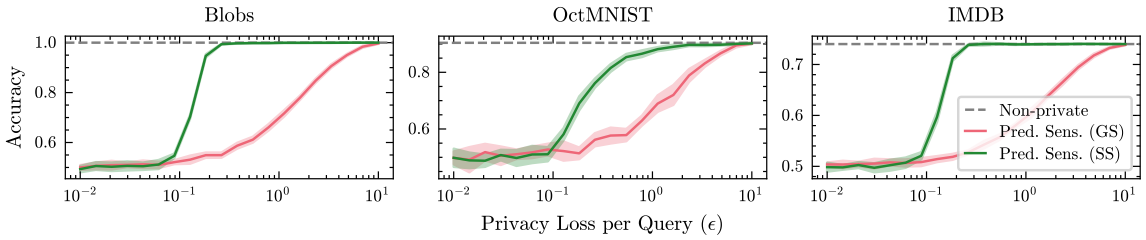


Figure 13: Private prediction accuracy for a single model using global sensitivity vs our smooth sensitivity bounds derived using interval-based AGT.

Experimental Results in Private Prediction. Here we reproduce the results from Wicker et al. (2025) that studies the effectiveness of our proposed mechanism. As in the preceding section, all certificates reported here hold only under the assumptions of Section 5. We evaluate our proposed method across three tasks: (i) Blobs, training a logistic regression model on a synthetic dataset of isotropic Gaussian clusters, (ii) Medical imaging: fine-tuning a model on retinal OCT images analogous to Section 9.4, and (iii) Sentiment Classification, training a neural network for sentiment analysis on the IMDB movie reviews dataset, using GPT-2 embeddings as input features.

Figure 13 compares the performance of the prediction sensitivity mechanism using global sensitivity versus smooth sensitivity bounds obtained using interval-based AGT. Our method maintains near noise-free accuracy at privacy budgets (ϵ) up to an order of magnitude smaller than those required when using global sensitivity. This improvement is especially pronounced in settings where the data are well-separated (as in the "Blobs" task) or when the training dataset is large (as in the "IMDB" task).

11. Conclusions

In this work, we introduced a mathematical framework for computing rigorous parameter-space bounds on the impact of training data perturbations, including data poisoning, data removal, and data substitution. We presented two instantiations of our framework: an efficient but conservative over-approximation using interval bound propagation and an exact analysis via mixed-integer programming. Through our experiments, we analyzed the trade-offs between these approaches, highlighting the ongoing need for tighter yet computationally efficient bounding techniques. Finally, we demonstrated the practical effectiveness of our method on real-world applications, including autonomous driving and medical image classification. This work lays a foundation for future certification-based techniques aimed at enhancing the robustness and privacy of machine learning models.

An important direction for future work is scaling to larger models. While certifying complete training of large-scale models remains beyond the reach of current formal methods, parameter-efficient fine-tuning strategies such as LoRA (Hu et al., 2022) are a promising development. In these settings, only a small number of (low-rank adapter) parameters are updated during fine-tuning, which significantly reduces the dimensionality of the parameter space that AGT must track. This could enable the certification of fine-tuning pipelines for foundation models, where poisoning and privacy concerns are particularly pressing.

Acknowledgments

This work was supported by the Turing's Defence and Security programme through a partnership with the UK government in accordance with the framework agreement between GCHQ & The Alan Turing Institute. CT was also supported by a BASF/Royal Academy of Engineering Senior Research Fellowship. The authors thank Fraser Kennedy and Jodie Knapp for helpful discussions and feedback, especially in the final stages of this project.

References

Martin Abadi, Andy Chu, Ian Goodfellow, H Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 308–318, 2016.

Steven Adams, Andrea Patane, Morteza Lahijanian, and Luca Laurenti. BNN-DP: robustness certification of Bayesian neural networks via dynamic programming. In *International Conference on Machine Learning*, pages 133–151. PMLR, 2023.

Matthias Althoff, Goran Frehse, and Antoine Girard. Set propagation techniques for reachability analysis. *Annual Review of Control, Robotics, and Autonomous Systems*, 4(1): 369–395, 2021.

Ross Anderson, Joey Huchette, Will Ma, Christian Tjandraatmadja, and Juan Pablo Vielma. Strong mixed-integer programming formulations for trained neural networks. *Mathematical Programming*, 183(1):3–39, 2020.

Samyadeep Basu, Philip Pope, and Soheil Feizi. Influence functions in deep learning are fragile. *arXiv preprint arXiv:2006.14651*, 2020.

Battista Biggio and Fabio Roli. Wild patterns: Ten years after the rise of adversarial machine learning. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 2154–2156, 2018.

Battista Biggio, Konrad Rieck, Davide Ariu, Christian Wressnegger, Igino Corona, Giorgio Giacinto, and Fabio Roli. Poisoning behavioral malware clustering. In *Artificial Intelligent and Security Workshop*, pages 27–36, 2014.

Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.

Rishi Bommasani, Drew A Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, et al. On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258*, 2021.

Lucas Bourtoule, Varun Chandrasekaran, Christopher A Choquette-Choo, Hengrui Jia, Adelin Travers, Baiwu Zhang, David Lie, and Nicolas Papernot. Machine unlearning. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 141–159. IEEE, 2021.

Rudy R Bunel, Ilker Turkaslan, Philip Torr, Pushmeet Kohli, and Pawan K Mudigonda. A unified view of piecewise linear neural network verification. *Advances in Neural Information Processing Systems*, 31, 2018.

Yinzhi Cao and Junfeng Yang. Towards making systems forget with machine unlearning. In *IEEE Symposium on Security and Privacy (SP)*, pages 463–480. IEEE, 2015.

Nicholas Carlini, Steve Chien, Milad Nasr, Shuang Song, Andreas Terzis, and Florian Tramer. Membership inference attacks from first principles. In *IEEE Symposium on Security and Privacy (SP)*, pages 1897–1914. IEEE, 2022.

Nicholas Carlini, Matthew Jagielski, Christopher A Choquette-Choo, Daniel Paleka, Will Pearce, Hyrum Anderson, Andreas Terzis, Kurt Thomas, and Florian Tramèr. Poisoning web-scale training datasets is practical. *arXiv preprint arXiv:2302.10149*, 2023.

Karan Chadha, Matthew Jagielski, Nicolas Papernot, Christopher Choquette-Choo, and Milad Nasr. Auditing private prediction. *arXiv preprint arXiv:2402.09403*, 2024.

Ruoxin Chen, Zenan Li, Jie Li, Junchi Yan, and Chentao Wu. On collective robustness of bagging against data poisoning. In *International Conference on Machine Learning*, pages 3299–3319. PMLR, 2022.

Xinyun Chen, Chang Liu, Bo Li, Kimberly Lu, and Dawn Song. Targeted backdoor attacks on deep learning systems using data poisoning. *arXiv preprint arXiv:1712.05526*, 2017.

Rachel Cummings, Gabriel Kaptchuk, and Elissa M Redmiles. " i need a better description": An investigation into user expectations for differential privacy. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, pages 3037–3052, 2021.

Cynthia Dwork and Aaron Roth. The algorithmic foundations of differential privacy. *Foundations and Trends® in Theoretical Computer Science*, 9(3–4):211–407, 2014.

Ruediger Ehlers. Formal verification of piece-wise linear feed-forward neural networks. *arXiv preprint arXiv:1705.01320*, 2017.

Ferdinando Fioretto, Cuong Tran, Pascal Van Hentenryck, and Keyu Zhu. Differential privacy and fairness in decisions and learning tasks: A survey. *arXiv preprint arXiv:2202.08187*, 2022.

Matteo Fischetti and Jason Jo. Deep neural networks and mixed integer linear optimization. *Constraints*, 23(3):296–309, 2018.

Sven Gowal, Krishnamurthy Dvijotham, Robert Stanforth, Rudy Bunel, Chongli Qin, Jonathan Uesato, Relja Arandjelovic, Timothy Mann, and Pushmeet Kohli. On the effectiveness of interval bound propagation for training verifiably robust models. *arXiv preprint arXiv:1810.12715*, 2018.

Tianyu Gu, Brendan Dolan-Gavitt, and Siddharth Garg. Badnets: Identifying vulnerabilities in the machine learning model supply chain. *arXiv preprint arXiv:1708.06733*, 2017.

Chuan Guo, Tom Goldstein, Awni Hannun, and Laurens Van Der Maaten. Certified data removal from machine learning models. *arXiv preprint arXiv:1911.03030*, 2019.

Xingshuo Han, Guowen Xu, Yuan Zhou, Xuehuan Yang, Jiwei Li, and Tianwei Zhang. Physical backdoor attacks to lane detection systems in autonomous driving. In *30th ACM International Conference on Multimedia*, pages 2957–2968, 2022.

Georges Hebrail and Alice Berard. Individual Household Electric Power Consumption. UCI Machine Learning Repository, 2012. DOI: https://doi.org/10.24432/C58K54.

Sanghyun Hong, Varun Chandrasekaran, Yiğitcan Kaya, Tudor Dumitraş, and Nicolas Papernot. On the effectiveness of mitigating data poisoning attacks with gradient shaping. *arXiv preprint arXiv:2002.11497*, 2020.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Liang Wang, Weizhu Chen, et al. Lora: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022.

Joey Huchette, Gonzalo Muñoz, Thiago Serra, and Calvin Tsay. When deep learning meets polyhedral theory: A survey. *INFORMS Journal on Computing*, 2026.

Guy Katz, Clark Barrett, David L Dill, Kyle Julian, and Mykel J Kochenderfer. Reluplex: An efficient SMT solver for verifying deep neural networks. In *Computer Aided Verification (CAV)*, pages 97–117. Springer, 2017.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Pang Wei Koh and Percy Liang. Understanding black-box predictions via influence functions. In *International conference on machine learning*, pages 1885–1894. PMLR, 2017.

Matthias König, Annelot W Bosman, Holger H Hoos, and Jan N van Rijn. Critically assessing the state of the art in neural network verification. *Journal of Machine Learning Research*, 25(12):1–53, 2024.

Alexander Levine and Soheil Feizi. Deep partition aggregation: Provable defense against general poisoning attacks. *arXiv preprint arXiv:2006.14768*, 2020.

Suyi Li, Yong Cheng, Wei Wang, Yang Liu, and Tianjian Chen. Learning to detect malicious clients for robust federated learning. *arXiv preprint arXiv:2002.00211*, 2020.

Katrina Ligett, Seth Neel, Aaron Roth, Bo Waggoner, and Steven Z Wu. Accuracy first: Selecting a differential privacy level for accuracy constrained ERM. *Advances in Neural Information Processing Systems*, 30, 2017.

Changliu Liu, Tomer Arnon, Chris Lazarus, Christopher Strong, Clark Barrett, and Mykel J Kochenderfer. Algorithms for verifying deep neural networks. *Foundations and Trends in Optimization*, 4(3-4):244–404, 2021.

Xiyang Liu, Weihao Kong, and Sewoong Oh. Differential privacy and robust statistics in high dimensions. In *Conference on Learning Theory*, pages 1167–1246. PMLR, 2022.

Tobias Lorenz, Marta Kwiatkowska, and Mario Fritz. Bicert: A bilinear mixed integer programming formulation for precise certified bounds against data poisoning attacks. *arXiv preprint arXiv:2412.10186*, 2024a.

Tobias Lorenz, Marta Kwiatkowska, and Mario Fritz. Fullcert: Deterministic end-to-end certification for training and inference of neural networks. In *DAGM German Conference on Pattern Recognition*, pages 71–85. Springer, 2024b.

Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 142–150, Portland, Oregon, USA, June 2011. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/P11-1015`.

Garth P McCormick. Computability of global solutions to factorable nonconvex programs: Part I—convex underestimating problems. *Mathematical Programming*, 10(1):147–175, 1976.

Luis Muñoz-González, Battista Biggio, Ambra Demontis, Andrea Paudice, Vasin Wongrassamee, Emil C Lupu, and Fabio Roli. Towards poisoning of deep learning algorithms with back-gradient optimization. In *10th ACM Workshop on Artificial Intelligence and Security*, pages 27–38, 2017.

Seth Neel, Aaron Roth, and Saeed Sharifi-Malvajerdi. Descent-to-delete: Gradient-based methods for machine unlearning. In *Algorithmic Learning Theory*, pages 931–962. PMLR, 2021.

James Newsome, Brad Karp, and Dawn Song. Paragraph: Thwarting signature learning by training maliciously. In *International Symposium on Recent Advances in Intrusion Detection (RAID)*, pages 81–105. Springer, 2006.

Hong Diep Nguyen. Efficient implementation of interval matrix multiplication. In *Applied Parallel and Scientific Computing: 10th International Conference, PARA 2010, Reykjavík, Iceland, June 6-9, 2010, Revised Selected Papers, Part II 10*, pages 179–188. Springer, 2012.

Kobbi Nissim, Sofya Raskhodnikova, and Adam Smith. Smooth sensitivity and sampling in private data analysis. In *ACM Symposium on Theory of Computing*, pages 75–84, 2007.

Nicolas Papernot, Martín Abadi, Ulfar Erlingsson, Ian Goodfellow, and Kunal Talwar. Semi-supervised knowledge transfer for deep learning from private training data. *arXiv preprint arXiv:1610.05755*, 2016a.

Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z Berkay Celik, and Ananthram Swami. The limitations of deep learning in adversarial settings. In *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 372–387. IEEE, 2016b.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

Keivan Rezaei, Kiarash Banihashem, Atoosa Chegini, and Soheil Feizi. Run-off election: Improved provable defense against data poisoning attacks. In *International Conference on Machine Learning*, pages 29030–29050. PMLR, 2023.

Elan Rosenfeld, Ezra Winston, Pradeep Ravikumar, and Zico Kolter. Certified robustness to label-flipping attacks via randomized smoothing. In *International Conference on Machine Learning*, pages 8230–8241. PMLR, 2020.

Siegfried M Rump. Fast and parallel interval arithmetic. *BIT Numerical Mathematics*, 39: 534–554, 1999.

Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin Vechev. An abstract domain for certifying neural networks. *Proc. ACM Program. Lang.*, 3(POPL), jan 2019. doi: 10.1145/3290354.

Philip Sosnin and Calvin Tsay. Scaling mixed-integer programming for certification of neural network controllers using bounds tightening. In *2024 IEEE 63rd Conference on Decision and Control (CDC)*, pages 1645–1650. IEEE, 2024.

Philip Sosnin, Mark N Müller, Maximilian Baader, Calvin Tsay, and Matthew Wicker. Certified robustness to data poisoning in gradient-based training. *arXiv preprint arXiv:2406.05670*, 2024.

Philip Sosnin, Jodie Knapp, Fraser Kennedy, Josh Collyer, and Calvin Tsay. Exact certification of data-poisoning attacks using mixed-integer programming. *arXiv preprint arXiv:2602.16944*, 2026.

Jacob Steinhardt, Pang Wei W Koh, and Percy S Liang. Certified defenses for data poisoning attacks. *Advances in Neural Information Processing Systems*, 30, 2017.

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

Anvith Thudi, Hengrui Jia, Ilia Shumailov, and Nicolas Papernot. On the necessity of auditable algorithmic definitions for machine unlearning. In *31st USENIX security symposium (USENIX Security 22)*, pages 4007–4022, 2022.

Zhiyi Tian, Lei Cui, Jie Liang, and Shui Yu. A comprehensive survey on poisoning attacks and countermeasures in machine learning. *ACM Computing Surveys*, 55(8):1–35, 2022.

Calvin Tsay, Jan Kronqvist, Alexander Thebelt, and Ruth Misener. Partition-based formulations for mixed-integer optimization of trained ReLU neural networks. *Advances in Neural Information Processing Systems*, 34:3068–3080, 2021.

Laurens van der Maaten and Awni Hannun. The trade-offs of private prediction. *arXiv preprint arXiv:2007.05089*, 2020.

Wenxiao Wang, Alexander J Levine, and Soheil Feizi. Improved certified defenses against data poisoning with (deterministic) finite aggregation. In *International Conference on Machine Learning*, pages 22769–22783. PMLR, 2022.

Dennis Wei, Haoze Wu, Min Wu, Pin-Yu Chen, Clark Barrett, and Eitan Farchi. Convex bounds on the softmax function with applications to robustness verification. *arXiv preprint arXiv:2303.01713*, 2023.

Matthew Wicker, Luca Laurenti, Andrea Patane, and Marta Kwiatkowska. Probabilistic safety for Bayesian neural networks. In *Conference on Uncertainty in Artificial Intelligence*, pages 1198–1207. PMLR, 2020.

Matthew Wicker, Juyeon Heo, Luca Costabello, and Adrian Weller. Robust explanation constraints for neural networks. *arXiv preprint arXiv:2212.08507*, 2022.

Matthew Wicker, Andrea Patane, Luca Laurenti, and Marta Kwiatkowska. Adversarial robustness certification for bayesian neural networks. *arXiv preprint arXiv:2306.13614*, 2023.

Matthew Robert Wicker, Philip Sosnin, Igor Shilov, Adrianna Janik, Mark Niklas Mueller, Yves-Alexandre De Montjoye, Adrian Weller, and Calvin Tsay. Certification for differentially private prediction in gradient-based training. In *International Conference on Machine Learning*, pages 66726–66745. PMLR, 2025.

Weiming Xiang, Hoang-Dung Tran, and Taylor T Johnson. Output reachable set estimation and verification for multilayer neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 29(11):5777–5783, 2018.

Chulin Xie, Yunhui Long, Pin-Yu Chen, and Bo Li. Uncovering the connection between differential privacy and certified robustness of federated learning against poisoning attacks. *arXiv preprint arXiv:2209.04030*, 2022.

Guolei Yang, Neil Zhenqiang Gong, and Ying Cai. Fake co-visitation injection attacks to recommender systems. In *NDSS*, 2017.

Jiancheng Yang, Rui Shi, and Bingbing Ni. MedMNIST classification decathlon: A lightweight AutoML benchmark for medical image analysis. In *IEEE 18th International Symposium on Biomedical Imaging (ISBI)*, pages 191–195, 2021.

Da Yu, Gautam Kamath, Janardhan Kulkarni, Tie-Yan Liu, Jian Yin, and Huishuai Zhang. Individual privacy accounting for differentially private stochastic gradient descent. *arXiv preprint arXiv:2206.02617*, 2022.

Huan Zhang, Tsui-Wei Weng, Pin-Yu Chen, Cho-Jui Hsieh, and Luca Daniel. Efficient neural network robustness certification with general activation functions. *Advances in Neural Information Processing Systems*, 31, 2018.

Xuezhou Zhang, Xiaojin Zhu, and Laurent Lessard. Online data poisoning attacks. In *Learning for Dynamics and Control*, pages 201–210. PMLR, 2020.

Chen Zhu, W Ronny Huang, Hengduo Li, Gavin Taylor, Christoph Studer, and Tom Goldstein. Transferable clean-label poisoning attacks on deep neural nets. In *International Conference on Machine Learning*, pages 7614–7623. PMLR, 2019.

Appendix A. Interval Propagation Through Common Loss Functions

Cross Entropy Loss. The cross-entropy loss, defined as

$$\mathcal{L}(\hat{y}, y) = -\sum_i y_i^t \log p_i, \tag{60}$$

with gradient

$$\frac{\partial \mathcal{L}(\hat{y}, y)}{\partial \hat{y}} = p - y^t \tag{61}$$

measures the distance between the predicted probability distribution and the true labels. Here, $p = \text{softmax}(\hat{y})$ are the predicted class probabilities and y^t is the one-hot encoding of the true class y.

To compute a sound enclosure on the loss gradient, we first derive interval bounds on the output probabilities p_i, which are obtained by passing the model output $\hat{y}$ through the softmax function, given by

$$p_i = \frac{1}{1 + \sum_{j \neq i} \exp(\hat{y}_j - \hat{y}_i)}. \tag{62}$$

Sound bounds on the output probabilities are obtained by reasoning over the output interval $\hat{\boldsymbol{y}}$:

$$\boldsymbol{p_i} = \left[\frac{1}{1 + \sum_{j \neq i} \exp\left(\hat{y}_j^U - \hat{y}_i^L\right)}, \frac{1}{1 + \sum_{j \neq i} \exp\left(\hat{y}_j^L - \hat{y}_i^U\right)} \right] \tag{63}$$

Now, taking $\boldsymbol{y^t}$ to be the interval over the one-hot encoding of the (potentially poisoned) true label, the interval over the loss gradient is given by

$$\frac{\boldsymbol{\partial \mathcal{L}(\hat{y}, y)}}{\boldsymbol{\partial \hat{y}}} = \boldsymbol{p} \ominus \boldsymbol{y^t} \tag{64}$$

Hinge Loss. The hinge loss, commonly used for training support vector machines (SVMs), is defined as $\mathcal{L}(\hat{y}, y) = \max(0, 1 - y\hat{y})$ The gradient of the hinge loss with respect to the predicted output $\hat{y}$ is given by:

$$\frac{\partial \mathcal{L}}{\partial \hat{y}} = \begin{cases} -y, & \text{if } 1 - y\hat{y} > 0 \\ 0, & \text{otherwise.} \end{cases} \tag{65}$$

To compute an interval enclosure for this gradient, we first note that both $\hat{y}$ and y may be perturbed under the given threat model. We define the interval representations:

- $\hat{\boldsymbol{y}} = [\hat{y}^L, \hat{y}^U]$, obtained from forward pass bounds.
- $\boldsymbol{y} = [y, y]$ without label perturbation, $\boldsymbol{y} = [0, 1]$ with label perturbation.

Appendix B. Improved Forward Pass Bounds using an Extended CROWN Algorithm

In Section 6.3, we extended the interval-based certification framework of Gowal et al. (2018) for computing forward pass bounds to consider intervals over the parameters of the network. A natural progression is to explore the applicability to other certification algorithms to this setting. To address this, we present a new, explicit extension of the CROWN algorithm (Zhang et al., 2018) that handles interval-bounded weights and biases. Here we present only a summary of the proposed method. For a more comprehensive overview along with proofs of all results we refer the reader to Sosnin et al. (2024).

The standard CROWN algorithm bounds the outputs of a neural network's m-th layer by back-propagating linear bounds through each intermediate activation function to the input layer. In the case of interval parameters, the sign of a particular weight may be ambiguous (when the interval spans zero), making it impossible to determine which linear bound (upper or lower) to back-propagate.

Our approach below generalizes the CROWN algorithm to the case of interval parameters, where the weights and biases involved in each linear relaxation are themselves represented as intervals. We note that linear bound propagation with interval parameters has been studied previously in the context of floating-point sound certification (Singh et al., 2019).

Following the notation of the original CROWN algorithm (Zhang et al., 2018), we present our result for an extended CROWN algorithm that bounds outputs for neural networks with interval-bounded parameters:

Proposition 13 (Explicit output bounds of neural network f with interval parameters, Sosnin et al. (2024))**.** *Given an m-layer neural network function $f : \mathbb{R}^{n_{in}} \to \mathbb{R}^{n_{out}}$ whose unknown parameters lie in the intervals $b^{(k)} \in \boldsymbol{b}^{(\boldsymbol{k})}$ and $W^{(k)} \in \boldsymbol{W}^{(\boldsymbol{k})}$ for $k = 1, \ldots, m$, there exist two explicit functions*

$$f_j^L\left(x, \Omega^{(0:m)}, \Theta^{(1:m)}, b^{(1:m)}\right) = \Omega_{j,:}^{(0)} x + \sum_{k=1}^{m} \Omega_{j,:}^{(k)} \left(b^{(k)} + \Theta_{:,j}^{(k)}\right) \tag{66}$$

$$f_j^U\left(x, \Lambda^{(0:m)}, \Delta^{(1:m)}, b^{(1:m)}\right) = \Lambda_{j,:}^{(0)} x + \sum_{k=1}^{m} \Lambda_{j,:}^{(k)} \left(b^{(k)} + \Delta_{:,j}^{(k)}\right) \tag{67}$$

such that $\forall x \in \boldsymbol{x}$

$$f_j(x) \geq \min\left\{ f_j^L\left(x, \Omega^{(0:m)}, \Theta^{(1:m)}, b^{(1:m)}\right) \mid \Omega^{(k)} \in \boldsymbol{\Omega}^{(\boldsymbol{k})}, b^k \in \boldsymbol{b}^{(\boldsymbol{k})} \right\}$$
$$f_j(x) \leq \max\left\{ f_j^U\left(x, \Lambda^{(0:m)}, \Delta^{(1:m)}, b^{(1:m)}\right) \mid \Lambda^{(k)} \in \boldsymbol{\Lambda}^{(\boldsymbol{k})}, b^k \in \boldsymbol{b}^{(\boldsymbol{k})} \right\},$$

where $\boldsymbol{x}$ is a closed input domain and $\Lambda^{(0:m)}, \Delta^{(1:m)}, \Omega^{(0:m)}, \Theta^{(1:m)}$ are the equivalent weights and biases of the upper and lower linear bounds, respectively. The bias terms $\Delta^{(1:m)}, \Theta^{(1:m)}$ are explicitly computed based on the linear relaxation of the activation functions. The weights $\Lambda^{(0:m)}, \Omega^{(0:m)}$ lie in intervals $\boldsymbol{\Lambda}^{(\boldsymbol{0:m})}, \boldsymbol{\Omega}^{(\boldsymbol{0:m})}$, which are computed in an analogous way to standard (non-interval) CROWN.

Computing Equivalent Weights and Biases. Our instantiation of the CROWN algorithm in Proposition 13 relies on the computation of the equivalent bias terms $\Delta^{(1:m)}, \Theta^{(1:m)}$ and interval enclosures over the equivalent weights $\Omega^{(0:m)}, \Lambda^{(0:m)}$. This proceeds similarly to the standard CROWN algorithm, but now accounts for intervals over the parameters $b^{(1:m)}, W^{(1:m)}$ of the network.

Closed Form Output Bounds. Given the two functions $f_j^L(\cdot)$, $f_j^U(\cdot)$ as defined above and intervals over all the relevant variables, we can compute the following closed-form global bounds:

$$\gamma_j^L = \min \left\{ \boldsymbol{\Omega}_{\boldsymbol{j},:}^{(\mathbf{0})} \otimes \boldsymbol{x} \oplus \sum_{k=1}^{m} \boldsymbol{\Omega}_{\boldsymbol{j},:}^{(\boldsymbol{k})} \otimes \left[\boldsymbol{b}^{(\boldsymbol{k})} \oplus \Theta_{:,j}^{(k)} \right] \right\}$$
$$\gamma_j^U = \max \left\{ \boldsymbol{\Lambda}_{j,:}^{(0)} \otimes \boldsymbol{x} \oplus \sum_{k=1}^{m} \boldsymbol{\Lambda}_{\boldsymbol{j},:}^{(\boldsymbol{k})} \otimes \left[\boldsymbol{b}^{(\boldsymbol{k})} \oplus \Delta_{:,j}^{(k)} \right] \right\}$$

where min / max are performed element-wise and return the lower / upper bounds of each interval enclosure. Then, we have $\gamma_j^L \leq f_j(x) \leq \gamma_j^U$ for all $x \in \boldsymbol{x}$, $b^{(k)} \in \boldsymbol{b}^{(\boldsymbol{k})}$ and $W^{(k)} \in \boldsymbol{W}^{(\boldsymbol{k})}$, which suffices to bound the output of the neural network. These intermediate bounds can then be used to further bound the gradient of the network as in Section 6.

Bounding the Backward Pass using CROWN. The CROWN algorithm can be applied to any composition of functions that can be upper- and lower-bounded by linear equations. Therefore, it is possible to consider both the forward and backward passes in a single combined CROWN pass. However, linear bounds on the gradient of the loss function tend to be relatively loose, e.g., linear bounds on the softmax function may be orders-of-magnitude looser than constant $[0, 1]$ bounds (Wei et al., 2023). As a result, we found that the tightest bounds are obtained by using either IBP / CROWN to bound the forward pass and IBP to bound the backward pass.

Appendix C. Experimental Details

This section details the experimental set-up used for the experiments detailed in Sections 8-10. All experiments were run on a server equipped with 2x AMD EPYC 9334 CPUs and 2x NVIDIA L40 GPUs using an implementation of Abstract Gradient Training written in Python available at `github.com/psosnin/AbstractGradientTraining`. All optimization-based bounds are solved in Gurobi with a limit of 16 threads per subproblem with a timeout of 3600 seconds.

Tables 4 and 5 detail the specific dataset and hyperparameter settings used in our experimental evaluation. For each experimental result, we provide the number of epochs, learning rate, learning rate decay factor (η), and batchsize. Learning rate decay is applied using a standard learning rate schedule $\alpha_n = \alpha/(1 + \eta n)$. Below we provide further details for specific settings not covered in the main text.

Dataset	# Features	# Samples
Halfmoons (small)	$9^{\dagger}$	128
Halfmoons (large)	$9^{\dagger}$	3000
Blobs	$6^{\dagger}$	3000
UCI House-electric	11	2049280
MNIST	784	60000
OCT-MNIST	784	97477
Udacity Self-Driving	39600	31573
IMDB	$768^{\star}$	40000

Table 4: Datasets used in experimental evaluations. † : feature dimension including polynomial feature expansion. * : feature dimension post GPT-2 embedding.

Figure & Dataset)	Model	Loss	Learn. rate	Batch Size	# Epochs	Learn. rate decay	Gradient Clipping
Figure 6 Halfmoons (small)	Linear	Hinge	5.0	64	7	–	–
Figure 7 Halfmoons (large)	Linear	BCE	3.0	3000	10	0.6	0.5
Figure 8 UCI Houseelectric	FC	MSE	0.005-0.02	100-10000	1	–	–
Figure 9 MNIST	PCA + Linear	BCE	5.0	60000	3	1.0	–
Figure 10 OCT-MNIST	Conv. Small (Gowal et al., 2018)	BCE	0.05	6000	2	5.0	–
Figure 11 Udacity Self-Driving	PilotNet (Bojarski et al., 2016)	MSE	0.25	10000	2	10.0	–
Figure 12 Halfmoons (large)	Linear	Hinge	2.0	3000	9	–	–
Figure 13 Blobs	Linear	BCE	1.0	3000	4	0.6	0.06
Figure 13 OCT-MNIST	Conv. Small (Gowal et al., 2018)	BCE	0.06	20000	4	0.5	0.9
Figure 13 IMDB	FC	BCE	0.2	100000	3	0.5	0.02

Table 5: Hyperparameter settings used in experimental evaluations.

C.1 Additional Experimental Details: Differentially Private Prediction

In Figure 13, we examine the performance of AGT in differentially private prediction for three distinct binary classification tasks. Below we provide additional experimental details describing each setting in detail.

Blobs Dataset. In Figure 13 (left), we examine a dataset consisting of 3,000 samples drawn from an isotropic Gaussian distribution. The objective is to predict the distribution of origin for each sample in a supervised learning task. We train a single-layer neural network with 128 hidden neurons using interval-based AGT for values of n in $\{1, \ldots, 100\}$. Using these parameter-space bounds, we determine the maximum stable $n^\star \in \{1, \ldots, 100\}$ for each point in a grid over the domain. This value is used to compute a bound on the smooth sensitivity, which is further used to calibrate noise for the downstream private prediction task.

Retinal OCT Image Classification. Next, we consider another dataset with larger scale inputs: classification of medical images from the retinal OCT (OctMNIST) dataset of MEDMNIST (Yang et al., 2021). We consider binary classification over this dataset, where a model is tasked with predicting whether an image is normal or abnormal (the latter combines three distinct abnormal classes from the original dataset).

The model comprises two convolutional layers of 16 and 32 filters and an ensuing 100-node dense layer, corresponding to the 'small' architecture from Gowal et al. (2018). To demonstrate our framework, we consider a base model pre-trained on public data and then fine-tuned on new, privacy-sensitive data, corresponding to the 7754 Drusen samples (a class of abnormality omitted from initial training). First, we train the complete model excluding this using standard stochastic gradient descent. We then fine-tune only the dense layer weights to recognise the new class, with a mix of 50% Drusen samples per batch. We aim to ensure privacy only with respect to the fine-tuning data.

IMDB Movie Reviews Finally, we consider fine-tuning GPT-2 (Radford et al., 2019) for sentiment analysis on the large-scale (40,000 samples) IMDb movie review dataset (Maas et al., 2011). In this setup, we assume that GPT-2 was pre-trained on publicly available data, distinct from the data used for fine-tuning, which implies no privacy risk from the pre-trained embeddings themselves. Under this assumption, we begin by encoding each movie review into a 768-dimensional vector using GPT-2's embeddings. We then train a fully connected neural network consisting of 1×100 nodes, to perform binary sentiment classification (positive vs negative reviews).

Appendix D. Proofs

D.1 Proof of Theorem 2 (Parameter Space Equivalence)

Proof Without loss of generality, take the objective function we wish to optimize to be denoted simply by J. By definition, there exists a parameter, $\theta^\circ = M(\tilde{\mathcal{D}})$ resulting from a particular dataset $\tilde{\mathcal{D}} \in \mathcal{T}(\mathcal{D})$ such that θ° provides a (potentially non-unique) optimal solution to the optimization problem we wish to bound, i.e., the left hand side of the inequality. Given a valid parameter space domain Θ satisfying (10), we have that necessarily, $\theta^\circ \in \Theta$. Therefore, the result of optimizing over Θ can provide at a minimum the bound realized

by θ°; however, due to approximation, this bound might not be tight, so optimizing over Θ provides an upper-bound, thus proving the inequalities above. ■

D.2 Proof of Theorem 6 (Sound Aggregation for Data Removal)

The nominal descent direction for a parameter θ is the averaged gradient over a training batch $\mathcal{B}$, defined as

$$\Delta\theta = \frac{1}{b}\sum_{i=1}^{b} \delta^{(i)}$$

where each gradient term is given by $\delta^{(i)} = \nabla_\theta \mathcal{L}\left(f^\theta\left(x^{(i)}\right), y^{(i)}\right)$. Our goal is to upper bound this descent direction when up to n points from $\mathcal{B}$ are removed. We wish to compute this bound to additionally be sound with respect to any $\theta \in [\theta_L, \theta_U]$. We will begin by bounding the descent direction for a fixed θ, then generalize to all $\theta \in [\theta_L, \theta_U]$. We present only the upper bounds here, though corresponding results for the lower bound can be shown by reversing the inequalities and replacing SEMax with SEMin.

Bounding the descent direction for a fixed, scalar θ. Consider the effect of removing up to n data points from batch $\mathcal{B}$. Without loss of generality, assume the gradient terms are sorted in descending order, i.e., $\delta^{(1)} \geq \delta^{(2)} \geq \cdots \geq \delta^{(b)}$. Then, the average gradient over all points can be bounded above by the average over the largest $b - n$ terms:

$$\Delta\theta = \frac{1}{b}\sum_{i=1}^{b} \delta^{(i)} \leq \frac{1}{b-n}\sum_{i=1}^{b-n} \delta^{(i)}$$

This bound corresponds to removing the n points with the smallest gradients.

Bounding the effect of a variable parameter interval. We extend this bound to any $\theta \in [\theta_L, \theta_U]$. Assume the existence of upper bounds $\delta_U^{(i)}$ on the gradients for each data point over the interval, such that

$$\delta_U^{(i)} \geq \nabla_{\theta'} \mathcal{L}\left(f^{\theta'}(x^{(i)}), y^{(i)}\right) \quad \forall\, \theta' \in [\theta_L, \theta_U].$$

Then, using these upper bounds, we further bound $\Delta\theta$ as

$$\Delta\theta \leq \frac{1}{b-n}\left(\sum_{i=1}^{b} \delta_U^{(i)}\right)$$

where, as before, we assume $\delta_U^{(i)}$ are indexed in descending order.

Extending to the multi-dimensional case. To generalize to the multi-dimensional case, we apply the above bound component-wise. Since gradients are not necessarily ordered for each parameter component, we introduce the SEMax_n operator, which selects and sums the largest n terms at each index. This yields the following bound on the descent direction:

$$\Delta\theta \leq \frac{1}{b-n}\left(\underset{b-n}{\text{SEMax}}\left\{\delta_U^{(i)}\right\}_{i=1}^{b}\right)$$

which holds for any $\theta \in [\theta_L, \theta_U]$ and up to n removed points. □

We have established the upper bound on the descent direction. The corresponding lower bound can be derived by reversing the inequalities and substituting SEMax with the analagous minimization operator, SEMin.

D.3 Proof of Theorem 8 (Sound Aggregation for Data Substitution)

The nominal clipped descent direction for a parameter θ is the averaged, clipped gradient over a training batch $\mathcal{B}$, defined as

$$\Delta\theta = \frac{1}{b}\sum_{i=1}^{b} \text{Clip}_\kappa\left[\delta^{(i)}\right]$$

where each gradient term is given by $\delta^{(i)} = \nabla_\theta \mathcal{L}\left(f^\theta\left(x^{(i)}\right), y^{(i)}\right)$. Our goal is to bound this descent direction for the case when (up to) n points are removed or added to the training data, for any $\theta \in [\theta_L, \theta_U]$. We begin by bounding the descent direction for a fixed, scalar θ, then generalize to all $\theta \in [\theta_L, \theta_U]$ and to the multi-dimensional case (i.e., multiple parameters). We present only the upper bounds here; analogous results apply for lower bounds.

Bounding the descent direction for a fixed, scalar θ. Consider the effect of removing up to n data points from batch $\mathcal{B}$. Without loss of generality, assume the gradient terms are sorted in descending order, i.e., $\delta^{(1)} \geq \delta^{(2)} \geq \cdots \geq \delta^{(b)}$. Then, the average clipped gradient over all points can be bounded above by the average over the largest $b - n$ terms:

$$\Delta\theta = \frac{1}{b}\sum_{i=1}^{b} \text{Clip}_\kappa\left[\delta^{(i)}\right] \leq \frac{1}{b-n}\sum_{i=1}^{b-n} \text{Clip}_\kappa\left[\delta^{(i)}\right]$$

This bound corresponds to removing the n points with the smallest gradients.

Next, consider adding n arbitrary points to the training batch. Since each added point contributes at most κ due to clipping, the descent direction with up to n removals and n additions is bounded by

$$\frac{1}{b}\sum_{i=1}^{b} \text{Clip}_\kappa\left[\delta^{(i)}\right] \leq \frac{1}{b-n}\sum_{i=1}^{b-n} \text{Clip}_\kappa\left[\delta^{(i)}\right] \leq \frac{1}{b}\left(n\kappa + \sum_{i=1}^{b} \text{Clip}_\kappa\left[\delta^{(i)}\right]\right)$$

where the bound now accounts for replacing the n smallest gradient terms with the maximum possible value of κ from the added samples.

Bounding the effect of a variable parameter interval. We extend this bound to any $\theta \in [\theta_L, \theta_U]$. Assume the existence of upper bounds $\delta_U^{(i)}$ on the clipped gradients for each data point over the interval, such that

$$\delta_U^{(i)} \geq \text{Clip}_\kappa\left[\nabla_{\theta'} \mathcal{L}\left(f^{\theta'}(x^{(i)}), y^{(i)}\right)\right] \quad \forall\, \theta' \in [\theta_L, \theta_U].$$

Then, using these upper bounds, we further bound $\Delta\theta$ as

$$\Delta\theta \leq \frac{1}{b}\left(n\kappa + \sum_{i=1}^{b} \text{Clip}_\kappa\left[\delta_U^{(i)}\right]\right)$$

where, as before, we assume $\delta_U^{(i)}$ are indexed in descending order.

Extending to the multi-dimensional case. To generalize to the multi-dimensional case, we apply the above bound component-wise. Since gradients are not necessarily ordered for each parameter component, we introduce the SEMax_n operator, which selects and sums the largest n terms at each index. This yields the following bound on the descent direction:

$$\Delta\theta \leq \frac{1}{b}\left(\underset{b-n}{\text{SEMax}}\left\{\delta_U^{(i)}\right\}_{i=1}^{b} + n\kappa\mathbf{1}_d\right)$$

which holds for any $\theta \in [\theta_L, \theta_U]$ and up to n removed and replaced points. □

We have established the upper bound on the descent direction. The corresponding lower bound can be derived by reversing the inequalities and substituting SEMax with the analagous minimization operator, SEMin.

D.4 Proof of Theorem 9 (Sound Aggregation for Bounded Perturbation)

The nominal descent direction for a parameter θ is the averaged gradient over a training batch $\mathcal{B}$, defined as

$$\Delta\theta = \frac{1}{b}\sum_{i=1}^{b}\delta^{(i)}$$

where each gradient term is given by $\delta^{(i)} = \nabla_\theta \mathcal{L}\left(f^\theta\left(x^{(i)}\right), y^{(i)}\right)$. Our goal is to upper bound this descent direction when up to n points from $\mathcal{B}$ are poisoned by up to ϵ in the feature space and ν in label space. The bound is additionally computed with respect to any $\theta \in [\theta_L, \theta_U]$. We again begin by bounding the descent direction for a fixed θ, then generalize to all $\theta \in [\theta_L, \theta_U]$. We present only the upper bounds here, though corresponding results for the lower bound can be shown by reversing the inequalities and replacing SEMax with SEMin.

Bounding the descent direction for a fixed θ. Consider the effect of poisoning either the features or the labels of a data point. For a given data point, an adversary may choose to poison its features, its labels, or both. In total, at most n points may be influenced by the poisoning adversary. We assume that $n \leq b$, otherwise take at most $\min(n, b)$ points to be poisoned.

Assume that we have access to sound gradient upper bounds

$$\delta^{(i)} \leq \tilde{\delta}_U^{(i)} \quad \forall \delta^{(i)} \in \left\{\nabla_{\theta'}\mathcal{L}\left(f^{\theta'}\left(\tilde{x}\right), \tilde{y}\right) \mid \|x^{(i)} - \tilde{x}\|_p \leq \epsilon, \|y^{(i)} - \tilde{y}\|_q \leq \nu\right\}.$$

where the inequalities are interpreted element-wise. Here, $\tilde{\delta}_U^{(i)}$ corresponds to an upper bound on the maximum possible gradient achievable at the data-point $(x^{(i)}, y^{(i)})$ through poisoning.

The adversary's maximum possible impact on the descent direction at any point i is given by $\tilde{\delta}_U^{(i)} - \delta^{(i)}$. To maximize an upper bound on $\Delta\theta$, we consider the n points with the largest possible adversarial contributions. Therefore, we obtain

$$\Delta\theta = \frac{1}{b}\sum_{i=1}^{b}\delta^{(i)} \leq \frac{1}{b}\left(\underset{n}{\text{SEMax}}\left\{\tilde{\delta}_U^{(i)} - \delta^{(i)}\right\}_{i=1}^{b} + \sum_{i=1}^{b}\delta^{(i)}\right),$$

where the SEMax operation corresponds to taking the sum of the largest n elements of its argument at each element. This bound captures the maximum increase in $\Delta\theta$ that an adversary can induce by poisoning up to n data points.

Bounding the effect of a variable parameter interval. Now, we wish to compute a bound on $\Delta\theta$ for any $\theta \in [\theta_L, \theta_U]$. To achieve this, we extend our previous gradient bounds to account for the interval over our parameters. Specifically, we define upper bounds on the nominal and adversarially perturbed gradients that hold across the entire parameter interval:

$$\delta \leq \delta_U^{(i)} \quad \forall \delta \in \left\{ \nabla_{\theta'} \mathcal{L} \left(f^{\theta'}(x^{(i)}), y^{(i)} \right) \mid \theta' \in [\theta^L, \theta^U] \right\},$$
$$\tilde{\delta} \leq \tilde{\delta}_U^{(i)} \quad \forall \tilde{\delta} \in \left\{ \nabla_{\theta'} \mathcal{L} \left(f^{\theta'}(\tilde{x}), \tilde{y} \right) \mid \theta' \in [\theta^L, \theta^U], \|x^{(i)} - \tilde{x}\|_p \leq \epsilon, \|y^{(i)} - \tilde{y}\|_q \leq \nu \right\}.$$

Thus, the descent direction is upper bounded by

$$\Delta\theta \leq \Delta\theta^U = \frac{1}{b} \left(\underset{n}{\text{SEMax}} \left\{ \tilde{\delta}_U^{(i)} - \delta_U^{(i)} \right\}_{i=1}^{b} + \sum_{i=1}^{b} \delta_U^{(i)} \right)$$

for all $\theta \in [\theta_L, \theta_U]$, where the appropriate bounds with respect to the parameter interval have been substituted in.