Provable Gradient Editing of Deep Neural Networks

Zhe Tao
University of California, Davis
Davis, CA 95616, USA
zhetao@ucdavis.edu
zhe@zhe-tao.com

Aditya V. Thakur
University of California, Davis
Davis, CA 95616, USA
avthakur@ucdavis.edu

Abstract

In explainable AI, DNN gradients are used to interpret the prediction; in safety-critical control systems, gradients could encode safety constraints; in scientific-computing applications, gradients could encode physical invariants. While recent work on provable editing of DNNs has focused on input-output constraints, the problem of enforcing hard constraints on DNN gradients remains unaddressed. We present ProGrad, the first efficient approach for editing the parameters of a DNN to provably enforce hard constraints on the DNN gradients. Given a DNN $\mathcal{N}$ with parameters θ, and a set $\mathcal{S}$ of pairs $(\mathbf{x}, \mathrm{Q})$ of input $\mathbf{x}$ and corresponding linear gradient constraints Q, ProGrad finds new parameters $\widehat{\boldsymbol{\theta}}$ such that $\bigwedge_{(\mathbf{x},\mathrm{Q})\in\mathcal{S}} \frac{\partial}{\partial\mathbf{x}}\mathcal{N}(\mathbf{x};\widehat{\boldsymbol{\theta}}) \in \mathrm{Q}$ while minimizing the changes $\|\widehat{\boldsymbol{\theta}} - \theta\|$. The key contribution is a novel *conditional variable gradient* of DNNs, which relaxes the NP-hard provable gradient editing problem to a linear program (LP), enabling ProGrad to use an LP solver to efficiently and effectively enforce the gradient constraints. We experimentally evaluated ProGrad via enforcing **(i)** hard Grad-CAM constraints on IMAGENET ResNet DNNs; **(ii)** hard Integrated Gradients constraints on Llama 3 and Qwen 3 LLMs; **(iii)** hard gradient constraints in training a function-approximation DNN as a proxy for safety constraints in control systems and physical invariants in scientific applications. The results highlight the unique capability of ProGrad in enforcing hard constraints on DNN gradients.

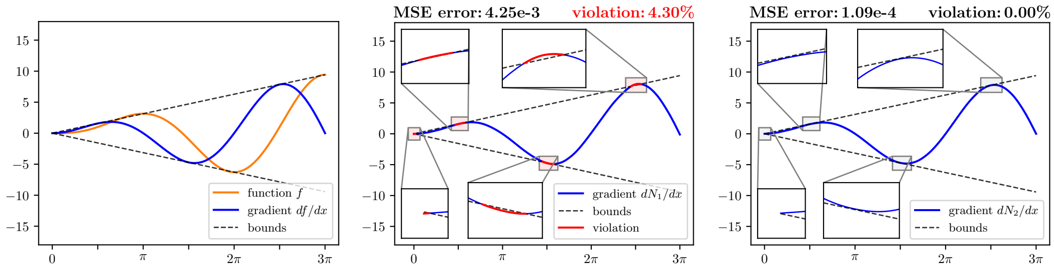
1 Introduction

Incorporating constraints into deep neural networks (DNNs) can enable learning with less data (e.g., in scientific domains) and provide guarantees about their behavior (e.g., in safety-critical applications). This has led to many recent works on provable editing of DNNs [41, 40], which enforce hard constraints on input-output behavior of DNNs. However, the problem of enforcing hard constraints on the gradients of DNNs remains unaddressed. This paper presents ProGrad, which addresses the provable gradient editing problem defined below:

Definition 1.1. Given a DNN $\mathcal{N} : \mathbb{R}^n \to \mathbb{R}^m$ with parameters θ, a set $\mathcal{S} \subseteq \left\{\mathbf{x} \mid \mathbf{x} \in \mathbb{R}^n\right\} \times \left\{\mathrm{Q} \mid \mathrm{Q} \subseteq \mathbb{R}^{m\times n}\right\}$ of pairs $(\mathbf{x}, \mathrm{Q})$, where $\mathbf{x} \in \mathbb{R}^n$ is an input and $\mathrm{Q} \stackrel{\text{def}}{=} \left\{\mathbf{J} \in \mathbb{R}^{m\times n} \mid \mathbf{A}\,\mathrm{vec}(\mathbf{J}) \le \mathbf{b}\right\}$ is the corresponding linear constraints on the Jacobian $\frac{\partial}{\partial\mathbf{x}}\mathcal{N}(\mathbf{x};\widehat{\boldsymbol{\theta}}) \in \mathbb{R}^{m\times n}$ of the DNN $\mathcal{N}$ with respect to the input $\mathbf{x}$, we use $\mathrm{vec}(\mathbf{J})$ to denote the Jacobian matrix $\mathbf{J}$ flattened as a vector. The **provable gradient editing problem** is to find new parameters $\widehat{\boldsymbol{\theta}}$ such that

$$\min \|\widehat{\boldsymbol{\theta}} - \theta\| \quad \text{s.t.} \quad \bigwedge_{(\mathbf{x},\mathrm{Q})\in\mathcal{S}} \frac{\partial}{\partial\mathbf{x}}\mathcal{N}(\mathbf{x};\widehat{\boldsymbol{\theta}}) \in \mathrm{Q} \tag{1}$$ ■

39th Conference on Neural Information Processing Systems (NeurIPS 2025).



(a) Target function f, whose gradient $\frac{d}{dx}f$ is bounded.

(b) Unsafe DNN $\mathcal{N}_1$, whose gradient $\frac{d}{dx}\mathcal{N}_1$ is *not bounded*.

(c) Safe $\mathcal{N}_2$ repaired by ProGrad, whose gradient $\frac{d}{dx}\mathcal{N}_2$ is *bounded*.

Figure 1: **Enforcing hard constraints on the gradient of a DNN. (Left) 1(a)** shows the target function $f(x) = -x\cos(x) + \sin(x)$ over the input domain $[0, 3\pi]$, whose gradient $\frac{d}{dx}f(x) = x\sin(x)$ is bounded by $g_u(x) = x$ and $g_l(x) = -x$. **(Middle) 1(b)** shows a DNN $\mathcal{N}_1$ trained on both the output and the gradient of f. Although $\mathcal{N}_1$ has good (low) output and gradient errors, its gradient $\frac{d}{dx}\mathcal{N}_1$ violates the hard constraints—as shown in the zoom-ins, its gradient $\frac{d}{dx}\mathcal{N}_1$ is *not bounded* by g_u and g_l. **(Right) 1(c)** shows a DNN $\mathcal{N}_2$ edited by ProGrad to satisfy the hard gradient constraints – its gradient $\frac{d}{dx}\mathcal{N}_2$ is *bounded* by g_u and g_l. DNN $\mathcal{N}_2$ also achieves better (lower) output and gradient errors. See Section 5.3 for more details.

(a) Original image.

(b) Reference Grad-CAM on the original image.

(c) Misclassified Gaussian-noise corrupted image.

(d) Deviated Grad-CAM on the corrupted image.

(e) Edited Grad-CAM on the corrupted image.

Figure 2: **Enforcing hard Grad-CAM constraints on an IMAGENET ResNet152 DNN.** Figure 2(a) shows a correctly-classified image with an appropriate Grad-CAM (2(b)) focusing on the object of interest. Figure 2(c) shows a Gaussian-noise-corrupted version of the original image, which is misclassified by the DNN, and the focus of its Grad-CAM (2(d)) deviates from the object of interest. We use ProGrad to edit the DNN to enforce the reference Grad-CAM (2(b)) on the corrupted image (2(c)). Figure 2(e) shows the Grad-CAM of the corrupted image on the edited DNN, which is now aligned with the reference Grad-CAM (2(b)). See Section 5.1 for more details.

In other words, provable gradient editing aims to make minimal changes to the parameters of a DNN to ensure that the DNN is guaranteed to satisfy any affine constraints involving the gradients of the DNN with respect to its input. Hard constraints on the gradient of a function with respect to its input are common in many scientific applications and safety-critical control systems. As an illustrative example, consider the function $f(x)$ in Figure 1(a) whose gradient $\frac{d}{dx}f$ should be bounded by $g_u(x) = x$ and $g_l(x) = -x$ over the domain $[0, 3\pi]$. Prior regularization-based training approaches can be used to enforce soft constraints on the gradient. However, as shown in Figure 1(b), although such a DNN $\mathcal{N}_1$ achieves good (low) output and gradient errors, its gradient $\frac{d}{dx}\mathcal{N}_1$ exceeds the bounds g_u and g_l and violates the hard gradient constraints, hence *unsafe*. In contrast, Figure 1(c) shows a *safe* DNN $\mathcal{N}_2$ edited by our method ProGrad to satisfy the hard constraints on the gradient, which also achieves better (lower) output and gradient errors.

Another natural application of gradient constraints is to ensure that the DNN has the appropriate gradient-based interpretation for a given input, which is important for explainable AI. Figure 2 shows a use case where we enforce an expected Grad-CAM attribution (Definition 3.5) for an IMAGENET ResNet-152 DNN; Figure 3 shows a use case where we enforce an expected Integrated Gradients attribution (Definition 3.4) for a Llama 3 LLM [9].

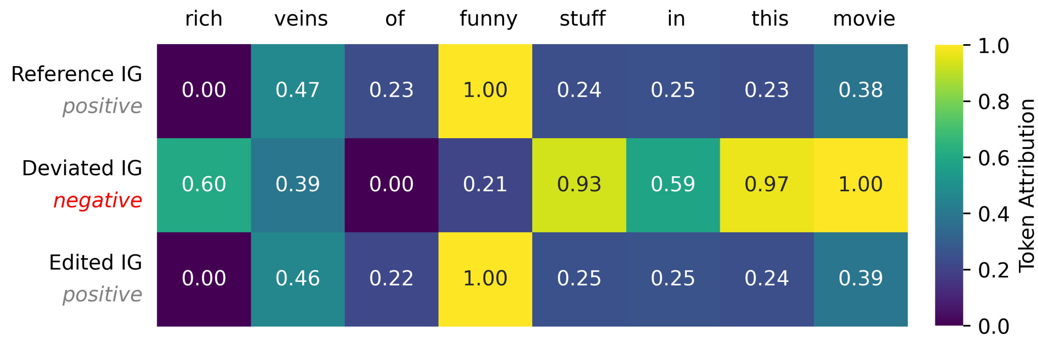


Figure 3: **Enforcing hard Integrated Gradients (IG) constraints on a Llama 3 LLM [9].** The sentence is from the Stanford Sentiment Treebank 2 (SST-2) dataset [36]. **The first row** shows the *reference IG* from a larger LLM, which correctly classifies the sentiment of the sentence as *positive*. **The second row** shows the *deviated IG* from the smaller LLM, which *misclassifies* the sentiment of the sentence as *negative*. **The third row** shows the *edited IG* from the repaired smaller LLM edited by `ProGrad`, whose IG is enforced to be close to the reference IG from the larger LLM, and correctly classifies the sentiment of the sentence as *positive*. See Section 5.2 for more details.

To the best of our knowledge, `ProGrad` is the *first* efficient approach for enforcing hard constraints on the gradients of a DNN that runs in polynomial time in the size of the edited layers (Theorem 4.3). Our **key contribution** is a novel *conditional variable gradient* of DNNs (Definition 4.2), which relaxes the NP-hard provable gradient editing problem to a **linear programming (LP)** problem, enabling `ProGrad` to use an LP solver to efficiently and effectively enforce the gradient constraints.

We evaluate `ProGrad` by enforcing **(i)** hard Grad-CAM constraints on IMAGENET ResNet DNNs; **(ii)** hard Integrated Gradients constraints on Llama 3 and Qwen 3 LLMs; **(iii)** hard gradient constraints in training a function-approximation DNN, which acts as a proxy for safety constraints in control systems and physical invariants in scientific applications. *The results highlight the unique capability of* `ProGrad` *in enforcing hard constraints on DNN gradients and gradient-based explanations.*

2 Related Work

There have been many recent works on incorporating constraints into deep learning. These can be categorized into four categories based on *whether they treat constraints as soft or hard constraints*, and *whether they support input-output constraints or gradient constraints*.

For **soft constraints**, regularized training modifies the loss function to incorporate constraints as regularization and does not guarantee constraint satisfaction. There have been many recent such approaches for incorporating *input-output constraints* [14, 45, 23, 5, 12, 17, 47, 38]. Certified training [21, 7, 31, 1, 24, 22, 20] is a type of regularized training geared towards adversarial robustness. For *gradient constraints*, Park et al. [25] present a technique for preserving the Grad-CAM attribution when performing network compression. They proposed a new loss function that incorporates the match between the attribution maps during the fine-tuning stage of compression, and present three variations of this matching loss function: `EWA`, `SWA` and `SSWA`. Similar approaches are also used in visual grounding for visual question answering tasks [30]. Physics-informed neural networks (PINNs) [28, 3] is another type of regularized training that incorporates DNN gradients to encode physics constraints to solve differential equations using DNNs.

For **hard constraints**, directly using an SMT solver to incorporate *input-output constraints* is inefficient and does not scale beyond small DNNs [6, 19, 8]. More efficient approaches are based on relaxing the problem to solving an LP problem [41, 37, 40]. `APRNN` [41] is efficient in enforcing arbitrary affine output constraints for input points. The state-of-the-art provable editing approaches `APRNN` and `PREPARED` [40] are also able to handle affine output constraints for input polytopes. However, they do not handle gradient constraints.

To the best of our knowledge, ours is the first approach to efficiently enforce hard constraints on the gradients of DNNs. Though the current work focuses on handling input points, future work could enforce gradient constraints on input polytopes by adapting ideas from APRNN and PREPARED.

DNN verification aims to determine whether a DNN satisfies a given input-output [34, 35, 50, 43, 33, 49, 46, 4, 2, 44] or gradient property such as monotonicity and Lipschitz robustness [15, 16, 32, 13].

3 Preliminaries

We use $\boldsymbol{x} \in \mathbb{R}$ to denote a scalar, $\mathbf{x} \in \mathbb{R}^m$ to denote a column vector, and $\mathbf{W} \in \mathbb{R}^{n\times m}$ to denote a matrix. Variables in blue with a hat denote LP decision variables, e.g., $\widehat{\mathbf{x}}$, $\widehat{\mathbf{z}}$, $\widehat{\mathbf{W}}$. Variables in blue with a tilde denote the conditional variables, e.g., $\widetilde{\mathbf{x}}$, $\widetilde{\mathbf{z}}$.

Definition 3.1 (DNN)**.** Consider an L-layer feed-forward ReLU deep neural network (DNN) $\mathcal{N}$ with parameters $\theta \overset{\text{def}}{=} \{\mathbf{W}^{(0)}, \mathbf{W}^{(1)}, \dots, \mathbf{W}^{(L-1)}, \mathbf{b}^{(0)}, \mathbf{b}^{(1)}, \dots, \mathbf{b}^{(L-1)}\}$. For each layer $0 \le \ell < L$, we use

$$\mathbf{z}^{(\ell)} \overset{\text{def}}{=} \mathbf{W}^{(\ell)}\mathbf{x}^{(\ell)} + \mathbf{b}^{(\ell)} \tag{2}$$

to denote the pre-activation output $\mathbf{z}^{(\ell)}$ after the affine transformation. For a non-last layer $\ell < L-1$,

$$\mathbf{x}^{(\ell+1)} \overset{\text{def}}{=} \operatorname{diag}\big(\mathbf{1}_{\mathbf{z}^{(\ell)}>0}\big)\, \mathbf{z}^{(\ell)} \tag{3}$$

denotes the layer output $\mathbf{x}^{(\ell+1)}$ after the ReLU activation, where $\operatorname{diag}\big(\mathbf{1}_{\mathbf{z}^{(\ell)}>0}\big)$ is a diagonal matrix of the indicator vector $\mathbf{1}_{\mathbf{z}^{(\ell)}>0}$, whose i-th element is 1 if $\mathbf{z}_i^{(\ell)} > 0$, or 0 otherwise. For the last layer $\ell = L-1$, we assume no ReLU activation and use $\mathbf{x}^{(L)} \overset{\text{def}}{=} \mathbf{z}^{(L-1)}$ to denote the layer output. Given the DNN input $\mathbf{x} \in \mathbb{R}^n$, we have the first-layer input $\mathbf{x}^{(0)} \overset{\text{def}}{=} \mathbf{x}$ and the network output $\mathcal{N}(\mathbf{x};\theta) \overset{\text{def}}{=} \mathbf{x}^{(L)}$. ■

Definition 3.2 (Gradient of DNNs)**.** For an L-layer ReLU DNN $\mathcal{N} : \mathbb{R}^n \to \mathbb{R}^m$ with parameters θ, the gradient (Jacobian) $\frac{\partial}{\partial \mathbf{x}}\mathcal{N}(\mathbf{x};\theta)$ of the DNN output $\mathcal{N}(\mathbf{x};\theta) \in \mathbb{R}^m$ with respect to the input $\mathbf{x} \in \mathbb{R}^n$ is defined by the chain rule as

$$\frac{\partial}{\partial \mathbf{x}}\mathcal{N}(\mathbf{x};\theta) \overset{\text{def}}{=} \prod_{\ell=L-1}^{0} \frac{\partial \mathbf{x}^{(\ell+1)}}{\partial \mathbf{z}^{(\ell)}}\frac{\partial \mathbf{z}^{(\ell)}}{\partial \mathbf{x}^{(\ell)}} \tag{4}$$

For the ReLU activation $\mathbf{x}^{(\ell+1)} \overset{\text{def}}{=} \operatorname{diag}\big(\mathbf{1}_{\mathbf{z}^{(\ell)}>0}\big)\, \mathbf{z}^{(\ell)}$ of a non-last layer $\ell < L-1$, the gradient $\frac{\partial \mathbf{x}^{(\ell+1)}}{\partial \mathbf{z}^{(\ell)}}$ of the ℓ-th layer post-activation output $\mathbf{x}^{(\ell+1)}$ with respect to the pre-activation output $\mathbf{z}^{(\ell)}$ is

$$\frac{\partial \mathbf{x}^{(\ell+1)}}{\partial \mathbf{z}^{(\ell)}} \overset{\text{def}}{=} \operatorname{diag}\big(\mathbf{1}_{\mathbf{z}^{(\ell)}>0}\big) \tag{5}$$

and for the last layer $\ell = L-1$ without ReLU activation, $\frac{\partial \mathbf{x}^{(L)}}{\partial \mathbf{z}^{(L-1)}} \overset{\text{def}}{=} \operatorname{diag}(\mathbf{1})$ is the identity matrix.

For the affine transformation $\mathbf{z}^{(\ell)} \overset{\text{def}}{=} \mathbf{W}^{(\ell)}\mathbf{x}^{(\ell)} + \mathbf{b}^{(\ell)}$, the gradient $\frac{\partial \mathbf{z}^{(\ell)}}{\partial \mathbf{x}^{(\ell)}}$ of the pre-activation output $\mathbf{z}^{(\ell)}$ with respect to the ℓ-th layer input $\mathbf{x}^{(\ell-1)}$ is defined as

$$\frac{\partial \mathbf{z}^{(\ell)}}{\partial \mathbf{x}^{(\ell)}} \overset{\text{def}}{=} \mathbf{W}^{(\ell)} \tag{6}$$ ■

3.1 Conditional variable output of DNNs for provable output editing

Our method ProGrad extends the conditional variable output of DNNs, introduced by APRNN [41], to the conditional variable gradient of DNNs. We now present how APRNN focuses on the forward pass and constructs the conditional variable output of DNNs. For clarity, we assume the first-layer weight and all-layer biases are variables. In practice, APRNN can freeze the first few layers and only make the rest of the DNN have variable parameters.

Definition 3.3 (Conditional variable output of DNNs)**.** Consider an L-layer feed-forward ReLU DNN $\mathcal{N}$ with parameters $\widehat{\boldsymbol{\theta}} \overset{\text{def}}{=} \{\widehat{\mathbf{W}}^{(0)}, \mathbf{W}^{(1)}, \dots, \mathbf{W}^{(L-1)}, \widehat{\mathbf{b}}^{(0)}, \widehat{\mathbf{b}}^{(1)}, \dots, \widehat{\mathbf{b}}^{(L-1)}\}$, where the first-layer weight $\widehat{\mathbf{W}}^{(0)}$ and all-layer biases $\widehat{\mathbf{b}}^{(\ell)}$ are variables. For an input $\mathbf{x} \in \mathbb{R}^n$, the conditional variable output $\widetilde{\mathcal{N}}(\mathbf{x};\widehat{\boldsymbol{\theta}}) \in \mathbb{R}^m$ with linear activation condition $\widetilde{\varphi}$ is a linear expression over $\widehat{\boldsymbol{\theta}}$ that is sound if the condition $\widetilde{\varphi}$ is satisfied. In other words, for any assignment θ to the variable parameters $\widehat{\boldsymbol{\theta}}$ that satisfies the activation condition $\widetilde{\varphi}$, $\widetilde{\mathcal{N}}(\mathbf{x};\theta) = \mathcal{N}(\mathbf{x};\theta)$. Next we define the conditional variable output and condition $\widetilde{\varphi}^{(\ell)}$ for each layer ℓ.

The pre-activation conditional variable output $\widetilde{\mathbf{z}}^{(0)}$ for the first layer $\ell = 0$ with variable weight is

$$\widetilde{\mathbf{z}}^{(0)} \stackrel{\text{def}}{=} \widehat{\mathbf{W}}^{(0)}\mathbf{x}^{(0)} + \widehat{\mathbf{b}}^{(0)} \tag{7}$$

The pre-activation conditional variable output $\widetilde{\mathbf{z}}^{(\ell)}$ for other layers $0 < \ell < L$ with constant weight is

$$\widetilde{\mathbf{z}}^{(\ell)} \stackrel{\text{def}}{=} \mathbf{W}^{(\ell)}\widetilde{\mathbf{x}}^{(\ell)} + \widehat{\mathbf{b}}^{(\ell)} \tag{8}$$

The conditional variable layer output $\widehat{\mathbf{x}}^{(\ell+1)}$ for a non-last layer $\ell < L-1$ is defined as

$$\widetilde{\mathbf{x}}^{(\ell+1)} \stackrel{\text{def}}{=} \operatorname{diag}\left(\mathbf{1}_{\mathbf{z}^{(\ell)}>0}\right) \widetilde{\mathbf{z}}^{(\ell)} \tag{9}$$

with the ℓ-th layer activation condition $\widetilde{\varphi}^{(\ell)}$

$$\widetilde{\varphi}^{(\ell)} \stackrel{\text{def}}{=} \operatorname{diag}\left(\mathbf{1}_{\mathbf{z}^{(\ell)}>0}\right) \widetilde{\mathbf{z}}^{(\ell)} > 0 \ \wedge\ \operatorname{diag}\left(\mathbf{1}_{\mathbf{z}^{(\ell)}\leq 0}\right) \widetilde{\mathbf{z}}^{(\ell)} \leq 0 \tag{10}$$

The activation condition $\widetilde{\varphi}^{(\ell)}$ is constraining the ReLU activation pattern of the pre-activation conditional variable output $\widetilde{\mathbf{z}}^{(\ell)}$ to be the same as a constant pre-activation output $\mathbf{z}^{(\ell)}$, so that if $\widetilde{\varphi}^{(\ell)}$ is satisfied, $\operatorname{diag}\left(\mathbf{1}_{\widetilde{\mathbf{z}}^{(\ell)}>0}\right) = \operatorname{diag}\left(\mathbf{1}_{\mathbf{z}^{(\ell)}>0}\right)$. The choice of the constant $\mathbf{z}^{(\ell)}$ is arbitrary, the default choice is the pre-activation output of the original DNN $\mathcal{N}$. For the last layer $\ell = L-1$ without ReLU activation, $\widetilde{\mathbf{x}}^{(L)} \stackrel{\text{def}}{=} \widetilde{\mathbf{z}}^{(L-1)}$ and $\widetilde{\varphi}^{(L-1)} \stackrel{\text{def}}{=} \top$.

Given the DNN input $\mathbf{x}^{(0)} \stackrel{\text{def}}{=} \mathbf{x}$, we have the conditional variable network output $\widetilde{\mathcal{N}}(\mathbf{x};\widehat{\boldsymbol{\theta}}) \stackrel{\text{def}}{=} \widetilde{\mathbf{x}}^{(L)}$ with the activation condition $\widetilde{\varphi} \stackrel{\text{def}}{=} \bigwedge_\ell \widetilde{\varphi}^{(\ell)}$. ■

3.2 Gradient-based interpretation methods

Definition 3.4 (Integrated Gradients [39]). Given an input $\mathbf{x}$ and a baseline input $\mathbf{x}^0$. Let $\mathbf{x}^\alpha \stackrel{\text{def}}{=} \mathbf{x}^0 + \alpha\mathbf{d}$ be the linearly interpolated points between the baseline $\mathbf{x}^0$ and the input $\mathbf{x}$, where $\alpha \in [0,1]$ and $\mathbf{d} \stackrel{\text{def}}{=} \mathbf{x} - \mathbf{x}^0$. The *integrated gradients* (IG) from $\mathbf{x}^0$ to $\mathbf{x}$ is defined as the integral of the gradients $\frac{\partial}{\partial \mathbf{x}}\mathcal{N}(\mathbf{x}^\alpha)$ along the path, then multiplied by the difference vector $\mathbf{d}$:

$$\mathtt{IG}(\mathbf{x}) \stackrel{\text{def}}{=} \mathbf{d} \odot \int_{\alpha=0}^{1} \frac{\partial}{\partial \mathbf{x}}\mathcal{N}(\mathbf{x}^\alpha) d\alpha \tag{11}$$

where $\odot$ denotes the element-wise product. In practice, this integral is approximated by a left-Riemann sum with m steps:

$$\mathtt{IG}^{\text{approx}}(\mathbf{x}) \stackrel{\text{def}}{=} \mathbf{d} \odot \frac{1}{m}\sum_{k=0}^{m} \frac{\partial}{\partial \mathbf{x}}\mathcal{N}(\mathbf{x}^{\frac{k}{m}}) \tag{12}$$ ■

Definition 3.5 (Grad-CAM [29]). Given a convolutional DNN $\mathcal{N}$ and input $\mathbf{x}$, let $\mathbf{y} \in \mathbb{R}^C$ denote the DNN output, and $\mathbf{z} \in \mathbb{R}^{K\times H\times W}$ denote the pre-activation output of the last convolutional layer of the DNN where K is the number of channels, and H and W are the height and width of the feature map. The Grad-CAM localization map $\mathbf{L}^c \in \mathbb{R}^{H\times W}$ is defined as

$$\mathbf{L}^c \stackrel{\text{def}}{=} \sum_k \boldsymbol{\alpha}_k^c \mathbf{z}_k \quad \text{where} \quad \boldsymbol{\alpha}_k^c \stackrel{\text{def}}{=} \frac{1}{H\times W}\sum_i\sum_j \frac{\partial \mathbf{y}_c}{\partial \mathbf{z}_{k,i,j}} \tag{13}$$

where the neuron importance weights $\boldsymbol{\alpha}^c \in \mathbb{R}^K$ for $\mathbf{z}$ is the average gradient of the output class c with respect to each channel k of the pre-activation output $\mathbf{z}$. Optionally, one can simplify the Grad-CAM localization map $\mathbf{L}^c$ by applying a ReLU activation to ignore the negative attributions. In this paper, we consider the Grad-CAM attribution with any sign, hence don't apply the ReLU activation to $\mathbf{L}^c$. ■

4 Approach

This section presents ProGrad, which solves the provable gradient editing problem of DNNs using linear programming (LP). For ease of exposition, we consider the following provable gradient editing problem for a scalar-valued DNN $\mathcal{N} : \mathbb{R}^n \to \mathbb{R}$ with gradient $\frac{\partial}{\partial \mathbf{x}}\mathcal{N}(\mathbf{x};\theta) \in \mathbb{R}^n$, and assume the first-layer weight and all-layer biases are variables. *In practice,* ProGrad *allows editing only the last few layers and freezing the rest of the DNN, enabling efficient editing because the size of the LP only depends on the edited layers instead of the entire DNN.* We defer the extension to the general provable editing problem of Definition 1.1 for vector-valued DNNs, as well as handling multiple inputs and editing only the last few layers to Appendix B. The proofs for all theorems can be found in Appendix A.

Definition 4.1. Given a DNN $\mathcal{N} : \mathbb{R}^n \to \mathbb{R}$ with parameters θ, and an input $\mathbf{x} \in \mathbb{R}^n$. The **provable gradient editing problem** is to find new parameters $\widehat{\boldsymbol{\theta}}$ that

$$\min \|\widehat{\boldsymbol{\theta}} - \theta\| \quad \text{s.t.} \quad \frac{\partial}{\partial \mathbf{x}}\mathcal{N}(\mathbf{x};\widehat{\boldsymbol{\theta}}) \in \mathrm{Q} \tag{14}$$

where $\frac{\partial}{\partial \mathbf{x}}\mathcal{N}(\mathbf{x};\widehat{\boldsymbol{\theta}}) \in \mathbb{R}^n$ denotes the gradient of the scalar output of $\mathcal{N}(\mathbf{x};\widehat{\boldsymbol{\theta}}) \in \mathbb{R}$ with respect to the input $\mathbf{x}$ with new parameters $\widehat{\boldsymbol{\theta}}$. $\mathrm{Q} \stackrel{\text{def}}{=} \left\{\mathbf{z} \in \mathbb{R}^n \mid \mathbf{Az} \leq \mathbf{b}\right\}$ is a convex polytope denoting the linear constraints for the gradient $\frac{\partial}{\partial \mathbf{x}}\mathcal{N}(\mathbf{x};\widehat{\boldsymbol{\theta}})$.

4.1 Conditional variable gradient of DNNs

Consider a DNN $\mathcal{N}$ with variable parameters $\widehat{\boldsymbol{\theta}}$ and input $\mathbf{x}$. The *exact* variable gradient $\frac{\partial}{\partial \mathbf{x}}\mathcal{N}(\mathbf{x};\widehat{\boldsymbol{\theta}})$ of $\mathcal{N}$ is *highly non-linear*, involving quadratic terms from the multiplication between weights and disjunctions from the ReLU activation. Our **key insight** is to relax the non-linear $\frac{\partial}{\partial \mathbf{x}}\mathcal{N}(\mathbf{x};\widehat{\boldsymbol{\theta}})$ to a *linear conditional variable gradient* $\frac{\partial}{\partial \mathbf{x}}\widetilde{\mathcal{N}}(\mathbf{x};\widehat{\boldsymbol{\theta}})$ that is sound under a *linear activation condition* $\widetilde{\boldsymbol{\varphi}}$.

Definition 4.2. The conditional variable gradient $\frac{\partial}{\partial \mathbf{x}}\widetilde{\mathcal{N}}(\mathbf{x};\widehat{\boldsymbol{\theta}})$ of a DNN $\mathcal{N}$ in terms of $\widehat{\boldsymbol{\theta}}$ with respect to the input $\mathbf{x}$, under a *poly-size linear* activation condition $\widetilde{\boldsymbol{\varphi}}$, is a *poly-size linear expression* that equals the exact variable gradient $\frac{\partial}{\partial \mathbf{x}}\mathcal{N}(\mathbf{x};\widehat{\boldsymbol{\theta}})$ if the activation condition $\widetilde{\boldsymbol{\varphi}}$ is satisfied.

In other words, for any assignment θ to the variable parameters $\widehat{\boldsymbol{\theta}}$ that satisfies the activation condition $\widetilde{\boldsymbol{\varphi}}$, $\frac{\partial}{\partial \mathbf{x}}\widetilde{\mathcal{N}}(\mathbf{x};\theta) = \frac{\partial}{\partial \mathbf{x}}\mathcal{N}(\mathbf{x};\theta)$. The size of the condition $\widetilde{\boldsymbol{\varphi}}$ and the expression $\frac{\partial}{\partial \mathbf{x}}\widetilde{\mathcal{N}}(\mathbf{x};\widehat{\boldsymbol{\theta}})$ is polynomial in the size of the *edited layers* of the DNN, i.e., the number of edited layers, parameters, and the input and output dimensions of each edited layer. In practice, ProGrad allows editing only the last few layers of a DNN. We defer the construction of such conditional variable gradient $\frac{\partial}{\partial \mathbf{x}}\widetilde{\mathcal{N}}(\mathbf{x};\widehat{\boldsymbol{\theta}})$ of $\mathcal{N}$ to Section 4.3, and first show how it can be used to solve the provable gradient editing problem.

4.2 Provable gradient editing via conditional variable gradient of DNNs

The following theorem shows how the conditional variable gradient of DNNs can be used to solve the provable gradient editing problem.

Theorem 4.3. *Given a provable gradient editing problem (Definition 4.1) for DNN $\mathcal{N}$ and parameters θ with input $\mathbf{x}$ and a gradient constraint $\mathrm{Q} \stackrel{\text{def}}{=} \left\{\mathbf{z} \in \mathbb{R}^n \mid \mathbf{Az} \leq \mathbf{b}\right\}$. Let $\frac{\partial}{\partial \mathbf{x}}\widetilde{\mathcal{N}}(\mathbf{x};\widehat{\boldsymbol{\theta}})$ be the conditional variable gradient of the DNN $\mathcal{N}$ with respect to the input $\mathbf{x}$, under the activation condition $\widetilde{\boldsymbol{\varphi}}$. The following linear program can be solved in polynomial time in the size of the edited layers of the DNN $\mathcal{N}$, and its solution is a solution to the provable gradient editing problem.*

$$\min \|\widehat{\boldsymbol{\theta}} - \theta\| \quad \textit{s.t.} \quad \widetilde{\boldsymbol{\varphi}} \wedge \mathbf{A}\,\frac{\partial}{\partial \mathbf{x}}\widetilde{\mathcal{N}}(\mathbf{x};\widehat{\boldsymbol{\theta}}) \leq \mathbf{b} \tag{15}$$

4.3 Conditional variable gradient of fully-connected ReLU DNN

This section presents how ProGrad focuses on the backward pass and constructs the conditional variable gradient of a fully-connected ReLU DNN $\mathcal{N}$ with respect to the input $\mathbf{x}$ and its parameters $\widehat{\boldsymbol{\theta}}$.

Definition 4.4. Consider an L-layer feed-forward ReLU DNN $\mathcal{N}$ (Definition 3.1) with parameters $\widehat{\boldsymbol{\theta}} \stackrel{\text{def}}{=} \{\widehat{\mathbf{W}}^{(0)}, \mathbf{W}^{(1)}, \ldots, \mathbf{W}^{(L-1)}, \widehat{\mathbf{b}}^{(0)}, \widehat{\mathbf{b}}^{(1)}, \ldots, \widehat{\mathbf{b}}^{(L-1)}\}$, where the first-layer weight $\widehat{\mathbf{W}}^{(0)}$ and all-layer biases $\widehat{\mathbf{b}}^{(\ell)}$ are variables. For an input $\mathbf{x} \in \mathbb{R}^n$, the conditional variable gradient $\frac{\partial}{\partial \mathbf{x}} \widetilde{\mathcal{N}}(\mathbf{x}; \widehat{\boldsymbol{\theta}})$ of a DNN $\mathcal{N}$ in terms of $\widehat{\boldsymbol{\theta}}$ with respect to the input $\mathbf{x}$ is defined by the chain rule as

$$\frac{\partial}{\partial \mathbf{x}} \widetilde{\mathcal{N}}(\mathbf{x}; \widehat{\boldsymbol{\theta}}) \stackrel{\text{def}}{=} \prod_{\ell=L-1}^{0} \frac{\partial \widetilde{\mathbf{x}}^{(\ell+1)}}{\partial \widetilde{\mathbf{z}}^{(\ell)}} \frac{\partial \widetilde{\mathbf{z}}^{(\ell)}}{\partial \widetilde{\mathbf{x}}^{(\ell)}} \tag{16}$$

with the activation condition $\widetilde{\boldsymbol{\varphi}} \stackrel{\text{def}}{=} \bigwedge_\ell \widetilde{\boldsymbol{\varphi}}^{(\ell)}$ defined from each layer ℓ.

For the conditional ReLU activation $\widetilde{\mathbf{x}}^{(\ell+1)} \stackrel{\text{def}}{=} \operatorname{diag}\big(\mathbf{1}_{\mathbf{z}^{(\ell)}>0}\big)\, \widetilde{\mathbf{z}}^{(\ell)}$ of non-last layer $\ell < L-1$, the conditional variable gradient $\frac{\partial \widetilde{\mathbf{x}}^{(\ell+1)}}{\partial \widetilde{\mathbf{z}}^{(\ell)}}$ of the ℓ-th layer post-activation output $\widetilde{\mathbf{x}}^{(\ell+1)}$ with respect to the pre-activation output $\widetilde{\mathbf{z}}^{(\ell)}$ is

$$\frac{\partial \widetilde{\mathbf{x}}^{(\ell+1)}}{\partial \widetilde{\mathbf{z}}^{(\ell)}} \stackrel{\text{def}}{=} \operatorname{diag}\big(\mathbf{1}_{\mathbf{z}^{(\ell)}>0}\big) \tag{17}$$

with the ℓ-th layer activation condition $\widetilde{\boldsymbol{\varphi}}^{(\ell)}$

$$\widetilde{\boldsymbol{\varphi}}^{(\ell)} \stackrel{\text{def}}{=} \operatorname{diag}\big(\mathbf{1}_{\mathbf{z}^{(\ell)}>0}\big)\, \widetilde{\mathbf{z}}^{(\ell)} > 0 \;\wedge\; \operatorname{diag}\big(\mathbf{1}_{\mathbf{z}^{(\ell)}\leq 0}\big)\, \widetilde{\mathbf{z}}^{(\ell)} \leq 0 \tag{18}$$

The activation condition $\widetilde{\boldsymbol{\varphi}}^{(\ell)}$ is constraining the ReLU activation pattern of the pre-activation conditional variable output $\widetilde{\mathbf{z}}^{(\ell)}$ to be the same as a constant pre-activation output $\mathbf{z}^{(\ell)}$, so that if $\widetilde{\boldsymbol{\varphi}}^{(\ell)}$ is satisfied, $\operatorname{diag}\big(\mathbf{1}_{\widetilde{\mathbf{z}}^{(\ell)}>0}\big) = \operatorname{diag}\big(\mathbf{1}_{\mathbf{z}^{(\ell)}>0}\big)$. The choice of the constant $\mathbf{z}^{(\ell)}$ is arbitrary, the default choice is the pre-activation output of the original DNN $\mathcal{N}$. For the last layer $\ell = L-1$ without ReLU activation, $\frac{\partial \widetilde{\mathbf{x}}^{(L)}}{\partial \widetilde{\mathbf{z}}^{(L-1)}} \stackrel{\text{def}}{=} \operatorname{diag}(\mathbf{1})$ is the identity matrix.

As seen in Equations 18 and 10, `ProGrad` and `APRNN` share the same idea of using activation condition to constrain each input in the edit set to lie on a specific linear piece of the edited DNN. Note that the linear piece is not fixed but variable, because it is expressed as a closed-form linear expression in terms of the editable DNN parameters.

For the conditional affine transformation $\widetilde{\mathbf{z}}^{(\ell)} \stackrel{\text{def}}{=} \mathbf{W}^{(\ell)} \widetilde{\mathbf{x}}^{(\ell)} + \widehat{\mathbf{b}}^{(\ell)}$ of the non-first layer $\ell > 0$ with constant weight $\mathbf{W}^{(\ell)}$ and conditional variable input $\widetilde{\mathbf{x}}^{(\ell)}$, the conditional variable gradient $\frac{\partial \widetilde{\mathbf{z}}^{(\ell)}}{\partial \widetilde{\mathbf{x}}^{(\ell)}}$ of the pre-activation output $\widetilde{\mathbf{z}}^{(\ell)}$ with respect to the layer input $\widetilde{\mathbf{x}}^{(\ell)}$ is

$$\frac{\partial \widetilde{\mathbf{z}}^{(\ell)}}{\partial \widetilde{\mathbf{x}}^{(\ell)}} \stackrel{\text{def}}{=} \mathbf{W}^{(\ell)} \tag{19}$$

For the conditional affine transformation $\widetilde{\mathbf{z}}^{(0)} \stackrel{\text{def}}{=} \widehat{\mathbf{W}}^{(0)} \mathbf{x}^{(0)} + \widehat{\mathbf{b}}^{(0)}$ of the first layer $\ell = 0$ with variable weight $\widehat{\mathbf{W}}^{(0)}$ and constant input $\mathbf{x}^{(0)}$, the conditional variable gradient $\frac{\partial \widetilde{\mathbf{z}}^{(0)}}{\partial \mathbf{x}^{(0)}}$ of the pre-activation output $\widetilde{\mathbf{z}}^{(0)}$ with respect to the layer input $\mathbf{x}^{(0)}$ is

$$\frac{\partial \widetilde{\mathbf{z}}^{(0)}}{\partial \mathbf{x}^{(0)}} \stackrel{\text{def}}{=} \widehat{\mathbf{W}}^{(0)} \tag{20}$$ ■

Theorem 4.5. *The conditional variable gradient constructed in Definition 4.4 is valid and satisfies the conditions stated in Definition 4.2.*

5 Experimental Evaluation

We have implemented `ProGrad` in `PyTorch` [26] and use `Gurobi` [10] as the LP solver. All experiments were run on a machine with dual Intel Xeon Platinum 8362 Processors, 32-Core 2.8GHz with 1.5 TB of memory, SSD, and NVIDIA H100 GPU with 80 GB of GPU memory running Ubuntu 22.04. Additional details can be found in Appendix C.

Table 1: **Enforcing hard Grad-CAM constraints on ResNet DNNs.** Comparison of the (pixel-level) Grad-CAM constraint satisfaction rate (Constr. Sat.), minimum cosine similarity (Min. Cos.) and minimum intersection over union (Min. IoU) between the expected and edited Grad-CAMs, and the top-1 accuracy (Acc.) of the edited DNNs on the ILSVRC 2012 IMAGENET validation set.

(a) Enforcing hard Grad-CAM constraints on `ResNet152`

Method	100 images with ε=1e-2				1,000 images with ε=5e-2			
	Constr. Sat.	Min. Cos.	Min. IoU	Acc.	Constr. Sat.	Min. Cos.	Min. IoU	Acc.
Original	7.06%	34.66%	4.35%	78.31%	30.85%	24.51%	0.00%	78.31%
`EWA`	6.61%	73.53%	20.00%	77.27%	30.47%	26.31%	0.00%	73.32%
`SWA`	6.27%	49.95%	71.43%	77.79%	34.34%	65.56%	4.35%	76.55%
`SSWA`	6.37%	56.54%	71.43%	77.79%	34.32%	65.36%	4.35%	76.56%
`ProGrad`	**100.00%**	**99.97%**	**84.62%**	**78.08%**	**100.00%**	**99.44%**	**50.00%**	**77.43%**

(b) Enforcing hard Grad-CAM constraints on `ResNet50`

Method	100 images with ε=1e-2				1,000 images with ε=5e-2			
	Constr. Sat.	Min. Cos.	Min. IoU	Acc.	Constr. Sat.	Min. Cos.	Min. IoU	Acc.
Original	7.35%	24.15%	0.00%	76.13%	31.21%	-68.81%	0.00%	76.13%
`EWA`	5.39%	65.44%	0.00%	73.42%	23.84%	37.95%	0.00%	65.86%
`SWA`	6.80%	83.25%	50.00%	75.24%	31.64%	-1.84%	4.35%	73.67%
`SSWA`	6.71%	83.01%	41.18%	75.25%	31.67%	-1.00%	4.35%	73.72%
`ProGrad`	**100.00%**	**99.96%**	**84.62%**	**75.38%**	**100.00%**	**99.31%**	**50.00%**	**74.33%**

5.1 Enforcing hard Grad-CAM constraints on ResNet DNNs for IMAGENET

In this experiment, we edit `ResNet152` and `ResNet50` DNNs from `torchvision` [18] so that the Grad-CAM attributions [29] for a set of images are ε-close to their expected attributions. We compare `ProGrad` to the state-of-the-art Grad-CAM fine-tuning methods `EWA`, `SWA` and `SSWA` [25].

Edit set. The edit set consists of *misclassified* images that have *deviated* Grad-CAM attributions for their expected class. These images are from the IMAGENET-C dataset [11] that are corrupted with Gaussian noise, whose original uncorrupted version is correctly classified. For each misclassified image in the edit set, we take the Grad-CAM attribution of the corresponding original correctly-classified image as the expected Grad-CAM. For each DNN, we construct two such edit sets: **(i)** 100 images from the first 50 classes with ε=1e-2; **(ii)** 1,000 images from the first 200 classes with ε=5e-2.

Grad-CAM constraints. For each image $\mathbf{x}$ in the edit sets and its expected Grad-CAM $\mathbf{L}$ for the expected class, let $\mathbf{L}'$ denote the Grad-CAM on the edited DNN $\mathcal{N}'$; let $\mathbf{L}_\mathrm{n}$ and $\mathbf{L}'_\mathrm{n}$ be the min-max-normalized expected and edited Grad-CAMs; we use $\|\mathbf{L}_\mathrm{n} - \mathbf{L}'_\mathrm{n}\|_\infty \leq \varepsilon$ as the constraint.

Results. As shown in Table 1, `ProGrad` *is the only method able to enforce the hard Grad-CAM constraints, achieving a 100% constraint satisfaction rate (Constr. Sat.), and it achieves the best accuracy (Acc.).* `ProGrad` also achieves the best similarity between the edited and expected Grad-CAM as measured by cosine similarity (Min. Cos.) and intersection over union (Min. IoU). In the `ResNet152` experiment, `ProGrad` took 22 minutes for 100 images and 3 hours 51 minutes for 1,000 images; in the `ResNet50` experiment, `ProGrad` took 25 minutes for 100 images and 3 hours 37 minutes for 1,000 images. For each baseline, we performed a grid search over hyperparameters with a time limit of 12 hours, and report the best results with the highest constraint satisfaction rate (Constr. Sat.). Although the baselines (`EWA`, `SWA` and `SSWA`) can improve these similarity metrics, they are unable to enforce the hard constraints.

Table 2: **Enforcing hard Integrated Gradients (IG) constraints on Llama 3 and Qwen 3 LLMs.** Comparison of the (token-level) IG constraint satisfaction rate (Constr. Sat.), cosine similarity (Cos.) and intersection over union (IoU) between the expected and edited IG, and the accuracy (Acc.) of the edited LLMs on the SST-2 validation set.

(a) Enforce hard IG constraints on `Llama-3.2-1b-Instruct`

Method	100 samples with ε=5e-2				200 samples with ε=1e-1			
	Constr. Sat.	Cos.	IoU	Acc.	Constr. Sat.	Cos.	IoU	Acc.
Original	25.45%	71.26%	32.34%	60.55%	38.15%	70.07%	30.83%	60.55%
`SFT`	52.67%	51.29%	32.17%	49.08%	67.07%	87.20%	37.14%	50.92%
`DPO`	51.70%	64.20%	23.73%	50.92%	65.88%	66.15%	35.81%	50.92%
`ProGrad`	**100.00%**	**99.69%**	**84.10%**	**60.67%**	**100.00%**	**98.82%**	**71.50%**	**59.63%**

(b) Enforce hard IG constraints on `Qwen3-1.7B`

Method	100 samples with ε=5e-2				200 samples with ε=1e-1			
	Constr. Sat.	Cos.	IoU	Acc.	Constr. Sat.	Cos.	IoU	Acc.
Original	26.55%	64.70%	26.27%	85.89%	35.73%	64.81%	26.77%	85.89%
`SFT`	29.51%	63.09%	25.37%	**89.22%**	37.75%	58.64%	29.52%	51.03%
`DPO`	32.18%	56.52%	30.18%	50.92%	37.72%	60.12%	31.18%	50.92%
`ProGrad`	**100.00%**	**99.75%**	**84.59%**	88.19%	**100.00%**	**99.05%**	**74.00%**	**88.07%**

5.2 Enforcing hard Integrated Gradients constraints on LLMs for SST-2

In this experiment, we edit `Llama-3.2-1b-Instruct` [9] and `Qwen3-1.7B` [48] LLMs so that the Integrated Gradients (IG) attributions [39] for a set of sentences are ε-close to their expected IG attributions as determined by the corresponding teacher LLMs `Llama-3.1-8b-Instruct` and `Qwen3-8B`. We compare `ProGrad` against supervised fine-tuning (`SFT`) and direct preference optimization (`DPO`) [27] as baselines.

Edit set. The edit set consists of *misclassified* sentences that have *deviated* IG attributions for the expected answer token. These sentences are *misclassified* samples from the Stanford Sentiment Treebank 2 (SST-2)[36] training set, but are correctly classified by the corresponding teacher LLM. For each sentence in the edit set, we take the corresponding IG from the teacher LLM as the expected IG attribution. For each LLM, we construct two such edit sets: **(i)** the first 100 misclassified sentences from SST-2 with ε=5e-2; **(ii)** the first 200 misclassified sentences from SST-2 with ε=1e-1.

IG constraints. For each sentence $\mathbf{x}$ in the edit set and the corresponding expected IG $\mathbf{L}$ for the expected answer token, let $\mathbf{L}'$ denote the IG on the edited LLM $\mathcal{N}'$; let $\mathbf{L}_\mathtt{n}$ and $\mathbf{L}'_\mathtt{n}$ denote the min-max-normalized expected and edited IGs. We use $\|\mathbf{L}_\mathtt{n} - \mathbf{L}'_\mathtt{n}\|_\infty \leq \varepsilon$ as the IG constraint.

Results. As shown in Table 2, `ProGrad` *is the only method able to enforce the hard IG constraints and achieves the best accuracy in three out of four experiments.* `ProGrad` also achieves the best similarity between the edited and expected IG as measured by cosine similarity (Cos.) and intersection over union (IoU). In the `Llama-3.2-1b-Instruct` experiment, `ProGrad` took 9 minutes for 100 samples, and 10 minutes for 200 samples. In the `Qwen3-1.7B` experiment, `ProGrad` took 28 minutes for 100 samples, and 32 minutes for 200 samples. For each baseline, we performed a grid search over hyperparameters with a time limit of 4 hours, and report the best results with the highest constraint satisfaction rate (Constr. Sat.). The baselines (`SFT` and `DPO`) failed to enforce the hard constraints, barely improved the similarity metrics, and in most cases decreased the accuracy of the edited LLM on the SST-2 validation set.

Table 3: **Enforce hard gradient constraints on a function approximation DNN.** Comparison of the gradient constraints violation rate (Grad. Violation), as well as the MSE error on the gradient (Grad. Error) and output (Output Error). The GD baseline is regularization-based training on both the DNN output and gradient. The GD + ProGrad method applies ProGrad after the regularization-based training to enforce the hard gradient constraints and minimize the gradient and output errors.

Method	Grad. Violation	Grad. Error	Output Error
GD	4.30%	4.25e-03	1.74e-03
GD + ProGrad	**0.00%**	**1.09e-04**	**1.50e-05**

5.3 Enforce gradient constraints in training a DNN to approximate a target function

In this experiment, we aim to enforce hard gradient constraints on a DNN that approximates a target function. The gradient constraints are derived from the gradient of the target function. This experiment acts as a proxy for safety constraints in control systems and physical invariants in scientific applications. We compare ProGrad against regularized-based training method, which incorporates the DNN gradients in the loss function.

Setup. We use a 4-layer fully-connected DNN $\mathcal{N} : \mathbb{R} \to \mathbb{R}$ with 100 hidden neurons per layer and Tanh activation function. Over the domain $[0, 3\pi]$, the DNN $\mathcal{N}$ is trained to approximate the following function $f : \mathbb{R} \to \mathbb{R}$

$$f(x) = -x\cos(x) + \sin(x) \tag{21}$$

and the gradient $\frac{d\mathcal{N}}{dx}$ of DNN $\mathcal{N}$ approximates the gradient $\frac{df}{dx} : \mathbb{R} \to \mathbb{R}$ of the target function f:

$$\frac{d}{dx} f(x) = x\sin(x) \tag{22}$$

Gradient constraints. The target gradient function $\frac{d}{dx} f(x) = x\sin(x)$ is bounded by the upper bound function $g_u(x) = x$ and the lower bound function $g_l(x) = -x$, and we aim to enforce this hard constraint on the DNN gradient $\frac{d}{dx}\mathcal{N}$ for all training samples $\forall x \in \mathcal{D}$, $-x \le \frac{d}{dx}\mathcal{N}(x) \le x$.

Results. Table 3 presents the results of the experiment. Although regularization-based training on both the DNN output and gradient can achieve good (low) errors, the trained DNN is not free of the gradient constraints violations. Applying ProGrad after the regularization-based training to enforce the hard gradient constraints as well as minimize the gradient and output errors can achieve 0% gradient constraints violation and further improve (decrease) the gradient and output errors. ProGrad took 10 seconds to edit this DNN.

6 Conclusion

We have presented ProGrad, the first efficient approach for provable gradient editing of DNNs that runs in polynomial time in the size of the edited layers. We presented a novel method for constructing conditional variable gradient of DNNs, enabling ProGrad to use an LP solver to find an edit. To demonstrate the effectiveness of ProGrad, we evaluated ProGrad in enforcing hard Grad-CAM constraints on ResNet DNNs for IMAGENET, enforcing hard Integrated Gradients constraints on Llama 3 and Qwen 3 LLMs, and enforcing hard gradient constraints in training a function-approximation DNN. The results highlight the unique capability of ProGrad in enforcing hard constraints on DNN gradients.

Societal Impacts The use of hard constraints to edit DNNs enables learning with fewer data points, and guaranteeing the safety of DNNs. ProGrad can be used to make DNNs safer, trustworthy, and interpretable. However, because ProGrad is a general technique for editing the gradients of DNNs, it could also be misused, for instance, to tamper with the interpretation of a DNN.

Limitations ProGrad is currently limited to enforcing gradient constraints for a (finite) set of input points. Future work could extend ProGrad to enforce gradient constraints for input polytopes by adapting ideas from APRNN [41] and PREPARED [40].

Acknowledgments and Disclosure of Funding

We would like to thank the anonymous reviewers for their feedback and suggestions, which have greatly improved the quality of the paper. This work is supported in part by NSF grant CCF-2048123 and DOE Award DE-SC0022285.

References

[1] Mislav Balunovic and Martin Vechev. Adversarial training and provable defenses: Bridging the gap. In *International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=SJxSDxrKDr`.

[2] Gregory Bonaert, Dimitar I. Dimitrov, Maximilian Baader, and Martin T. Vechev. Fast and precise certification of transformers. In Stephen N. Freund and Eran Yahav, editors, *PLDI '21: 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation, Virtual Event, Canada, June 20-25, 2021*, pages 466–481. ACM, 2021. doi: 10.1145/3453483.3454056. URL `https://doi.org/10.1145/3453483.3454056`.

[3] Salvatore Cuomo, Vincenzo Schiano Di Cola, Fabio Giampaolo, Gianluigi Rozza, Maziar Raissi, and Francesco Piccialli. Scientific machine learning through physics–informed neural networks: Where we are and what's next. *Journal of Scientific Computing*, 92(3):88, 2022.

[4] Claudio Ferrari, Mark Niklas Müller, Nikola Jovanović, and Martin Vechev. Complete verification via multi-neuron relaxation guided branch-and-bound. In *International Conference on Learning Representations*, 2022.

[5] Marc Fischer, Mislav Balunovic, Dana Drachsler-Cohen, Timon Gehr, Ce Zhang, and Martin T. Vechev. DL2: training and querying neural networks with logic. In *Proceedings of the 36th International Conference on Machine Learning (ICML)*, volume 97 of *Proceedings of Machine Learning Research*. PMLR, 2019.

[6] Ben Goldberger, Guy Katz, Yossi Adi, and Joseph Keshet. Minimal modifications of deep neural networks using verification. In Elvira Albert and Laura Kovács, editors, *LPAR 2020: 23rd International Conference on Logic for Programming, Artificial Intelligence and Reasoning, Alicante, Spain, May 22-27, 2020*, volume 73 of *EPiC Series in Computing*, pages 260–278. EasyChair, 2020. doi: 10.29007/699q. URL `https://doi.org/10.29007/699q`.

[7] 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, 2019.

[8] Kshitij Goyal, Sebastijan Dumancic, and Hendrik Blockeel. Deepsade: Learning neural networks that guarantee domain constraint satisfaction. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38 (11):12199–12207, Mar. 2024. doi: 10.1609/aaai.v38i11.29109. URL `https://ojs.aaai.org/index.php/AAAI/article/view/29109`.

[9] Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, Arun Rao, Aston Zhang, Aurelien Rodriguez, Austen Gregerson, Ava Spataru, Baptiste Roziere, Bethany Biron, Binh Tang, Bobbie Chern, Charlotte Caucheteux, Chaya Nayak, Chloe Bi, Chris Marra, Chris McConnell, Christian Keller, Christophe Touret, Chunyang Wu, Corinne Wong, Cristian Canton Ferrer, Cyrus Nikolaidis, Damien Allonsius, Daniel Song, Danielle Pintz, Danny Livshits, Danny Wyatt, David Esiobu, Dhruv Choudhary, Dhruv Mahajan, Diego Garcia-Olano, Diego Perino, Dieuwke Hupkes, Egor Lakomkin, Ehab AlBadawy, Elina Lobanova, Emily Dinan, Eric Michael Smith, Filip Radenovic, Francisco Guzmán, Frank Zhang, Gabriel Synnaeve, Gabrielle Lee, Georgia Lewis Anderson, Govind Thattai, Graeme Nail, Gregoire Mialon, Guan Pang, Guillem Cucurell, Hailey Nguyen, Hannah Korevaar, Hu Xu, Hugo Touvron, Iliyan Zarov, Imanol Arrieta Ibarra, Isabel Kloumann, Ishan Misra, Ivan Evtimov, Jack Zhang, Jade Copet, Jaewon Lee, Jan Geffert, Jana Vranes, Jason Park, Jay Mahadeokar, Jeet Shah, Jelmer van der Linde, Jennifer Billock, Jenny Hong, Jenya Lee, Jeremy Fu, Jianfeng Chi, Jianyu Huang, Jiawen Liu, Jie Wang, Jiecao Yu, Joanna Bitton, Joe Spisak, Jongsoo Park, Joseph Rocca, Joshua Johnstun, Joshua Saxe, Junteng Jia, Kalyan Vasuden Alwala, Karthik Prasad, Kartikeya Upasani, Kate Plawiak, Ke Li, Kenneth Heafield, Kevin Stone, Khalid El-Arini, Krithika Iyer, Kshitiz Malik, Kuenley Chiu, Kunal Bhalla, Kushal Lakhotia, Lauren Rantala-Yeary, Laurens van der Maaten, Lawrence Chen, Liang Tan, Liz Jenkins, Louis Martin, Lovish Madaan, Lubo Malo, Lukas Blecher, Lukas Landzaat, Luke de Oliveira, Madeline Muzzi, Mahesh Pasupuleti, Mannat Singh, Manohar Paluri, Marcin Kardas, Maria Tsimpoukelli, Mathew Oldham, Mathieu Rita, Maya Pavlova, Melanie Kambadur, Mike Lewis, Min Si, Mitesh Kumar

Singh, Mona Hassan, Naman Goyal, Narjes Torabi, Nikolay Bashlykov, Nikolay Bogoychev, Niladri Chatterji, Ning Zhang, Olivier Duchenne, Onur Çelebi, Patrick Alrassy, Pengchuan Zhang, Pengwei Li, Petar Vasic, Peter Weng, Prajjwal Bhargava, Pratik Dubal, Praveen Krishnan, Punit Singh Koura, Puxin Xu, Qing He, Qingxiao Dong, Ragavan Srinivasan, Raj Ganapathy, Ramon Calderer, Ricardo Silveira Cabral, Robert Stojnic, Roberta Raileanu, Rohan Maheswari, Rohit Girdhar, Rohit Patel, Romain Sauvestre, Ronnie Polidoro, Roshan Sumbaly, Ross Taylor, Ruan Silva, Rui Hou, Rui Wang, Saghar Hosseini, Sahana Chennabasappa, Sanjay Singh, Sean Bell, Seohyun Sonia Kim, Sergey Edunov, Shaoliang Nie, Sharan Narang, Sharath Raparthy, Sheng Shen, Shengye Wan, Shruti Bhosale, Shun Zhang, Simon Vandenhende, Soumya Batra, Spencer Whitman, Sten Sootla, Stephane Collot, Suchin Gururangan, Sydney Borodinsky, Tamar Herman, Tara Fowler, Tarek Sheasha, Thomas Georgiou, Thomas Scialom, Tobias Speckbacher, Todor Mihaylov, Tong Xiao, Ujjwal Karn, Vedanuj Goswami, Vibhor Gupta, Vignesh Ramanathan, Viktor Kerkez, Vincent Gonguet, Virginie Do, Vish Vogeti, Vítor Albiero, Vladan Petrovic, Weiwei Chu, Wenhan Xiong, Wenyin Fu, Whitney Meers, Xavier Martinet, Xiaodong Wang, Xiaofang Wang, Xiaoqing Ellen Tan, Xide Xia, Xinfeng Xie, Xuchao Jia, Xuewei Wang, Yaelle Goldschlag, Yashesh Gaur, Yasmine Babaei, Yi Wen, Yiwen Song, Yuchen Zhang, Yue Li, Yuning Mao, Zacharie Delpierre Coudert, Zheng Yan, Zhengxing Chen, Zoe Papakipos, Aaditya Singh, Aayushi Srivastava, Abha Jain, Adam Kelsey, Adam Shajnfeld, Adithya Gangidi, Adolfo Victoria, Ahuva Goldstand, Ajay Menon, Ajay Sharma, Alex Boesenberg, Alexei Baevski, Allie Feinstein, Amanda Kallet, Amit Sangani, Amos Teo, Anam Yunus, Andrei Lupu, Andres Alvarado, Andrew Caples, Andrew Gu, Andrew Ho, Andrew Poulton, Andrew Ryan, Ankit Ramchandani, Annie Dong, Annie Franco, Anuj Goyal, Aparajita Saraf, Arkabandhu Chowdhury, Ashley Gabriel, Ashwin Bharambe, Assaf Eisenman, Azadeh Yazdan, Beau James, Ben Maurer, Benjamin Leonhardi, Bernie Huang, Beth Loyd, Beto De Paola, Bhargavi Paranjape, Bing Liu, Bo Wu, Boyu Ni, Braden Hancock, Bram Wasti, Brandon Spence, Brani Stojkovic, Brian Gamido, Britt Montalvo, Carl Parker, Carly Burton, Catalina Mejia, Ce Liu, Changhan Wang, Changkyu Kim, Chao Zhou, Chester Hu, Ching-Hsiang Chu, Chris Cai, Chris Tindal, Christoph Feichtenhofer, Cynthia Gao, Damon Civin, Dana Beaty, Daniel Kreymer, Daniel Li, David Adkins, David Xu, Davide Testuggine, Delia David, Devi Parikh, Diana Liskovich, Didem Foss, Dingkang Wang, Duc Le, Dustin Holland, Edward Dowling, Eissa Jamil, Elaine Montgomery, Eleonora Presani, Emily Hahn, Emily Wood, Eric-Tuan Le, Erik Brinkman, Esteban Arcaute, Evan Dunbar, Evan Smothers, Fei Sun, Felix Kreuk, Feng Tian, Filippos Kokkinos, Firat Ozgenel, Francesco Caggioni, Frank Kanayet, Frank Seide, Gabriela Medina Florez, Gabriella Schwarz, Gada Badeer, Georgia Swee, Gil Halpern, Grant Herman, Grigory Sizov, Guangyi, Zhang, Guna Lakshminarayanan, Hakan Inan, Hamid Shojanazeri, Han Zou, Hannah Wang, Hanwen Zha, Haroun Habeeb, Harrison Rudolph, Helen Suk, Henry Aspegren, Hunter Goldman, Hongyuan Zhan, Ibrahim Damlaj, Igor Molybog, Igor Tufanov, Ilias Leontiadis, Irina-Elena Veliche, Itai Gat, Jake Weissman, James Geboski, James Kohli, Janice Lam, Japhet Asher, Jean-Baptiste Gaya, Jeff Marcus, Jeff Tang, Jennifer Chan, Jenny Zhen, Jeremy Reizenstein, Jeremy Teboul, Jessica Zhong, Jian Jin, Jingyi Yang, Joe Cummings, Jon Carvill, Jon Shepard, Jonathan McPhie, Jonathan Torres, Josh Ginsburg, Junjie Wang, Kai Wu, Kam Hou U, Karan Saxena, Kartikay Khandelwal, Katayoun Zand, Kathy Matosich, Kaushik Veeraraghavan, Kelly Michelena, Keqian Li, Kiran Jagadeesh, Kun Huang, Kunal Chawla, Kyle Huang, Lailin Chen, Lakshya Garg, Lavender A, Leandro Silva, Lee Bell, Lei Zhang, Liangpeng Guo, Licheng Yu, Liron Moshkovich, Luca Wehrstedt, Madian Khabsa, Manav Avalani, Manish Bhatt, Martynas Mankus, Matan Hasson, Matthew Lennie, Matthias Reso, Maxim Groshev, Maxim Naumov, Maya Lathi, Meghan Keneally, Miao Liu, Michael L. Seltzer, Michal Valko, Michelle Restrepo, Mihir Patel, Mik Vyatskov, Mikayel Samvelyan, Mike Clark, Mike Macey, Mike Wang, Miquel Jubert Hermoso, Mo Metanat, Mohammad Rastegari, Munish Bansal, Nandhini Santhanam, Natascha Parks, Natasha White, Navyata Bawa, Nayan Singhal, Nick Egebo, Nicolas Usunier, Nikhil Mehta, Nikolay Pavlovich Laptev, Ning Dong, Norman Cheng, Oleg Chernoguz, Olivia Hart, Omkar Salpekar, Ozlem Kalinli, Parkin Kent, Parth Parekh, Paul Saab, Pavan Balaji, Pedro Rittner, Philip Bontrager, Pierre Roux, Piotr Dollar, Polina Zvyagina, Prashant Ratanchandani, Pritish Yuvraj, Qian Liang, Rachad Alao, Rachel Rodriguez, Rafi Ayub, Raghotham Murthy, Raghu Nayani, Rahul Mitra, Rangaprabhu Parthasarathy, Raymond Li, Rebekkah Hogan, Robin Battey, Rocky Wang, Russ Howes, Ruty Rinott, Sachin Mehta, Sachin Siby, Sai Jayesh Bondu, Samyak Datta, Sara Chugh, Sara Hunt, Sargun Dhillon, Sasha Sidorov, Satadru Pan, Saurabh Mahajan, Saurabh Verma, Seiji Yamamoto, Sharadh Ramaswamy, Shaun Lindsay, Shaun Lindsay, Sheng Feng, Shenghao Lin, Shengxin Cindy Zha, Shishir Patil, Shiva Shankar, Shuqiang Zhang, Shuqiang Zhang, Sinong Wang, Sneha Agarwal, Soji Sajuyigbe, Soumith Chintala, Stephanie Max, Stephen Chen, Steve Kehoe, Steve Satterfield, Sudarshan Govindaprasad, Sumit Gupta, Summer Deng, Sungmin Cho, Sunny Virk, Suraj Subramanian, Sy Choudhury, Sydney Goldman, Tal Remez, Tamar Glaser, Tamara Best, Thilo Koehler, Thomas Robinson, Tianhe Li, Tianjun Zhang, Tim Matthews, Timothy Chou, Tzook Shaked, Varun Vontimitta, Victoria Ajayi, Victoria Montanez, Vijai Mohan, Vinay Satish Kumar, Vishal Mangla, Vlad Ionescu, Vlad Poenaru, Vlad Tiberiu Mihailescu, Vladimir Ivanov, Wei Li, Wenchen Wang, Wenwen Jiang, Wes Bouaziz, Will Constable, Xiaocheng Tang, Xiaojian Wu, Xiaolan Wang, Xilun Wu, Xinbo Gao, Yaniv Kleinman, Yanjun Chen, Ye Hu, Ye Jia, Ye Qi, Yenda Li, Yilin Zhang, Ying Zhang, Yossi Adi, Youngjin Nam, Yu, Wang, Yu Zhao, Yuchen Hao, Yundi Qian, Yunlu Li, Yuzi He, Zach Rait, Zachary DeVito, Zef Rosnbrick, Zhaoduo Wen, Zhenyu Yang, Zhiwei Zhao, and Zhiyu Ma. The llama 3 herd of models, 2024. URL https://arxiv.org/abs/2407.21783.

[10] Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2022. URL https://www.gurobi.com.

[11] Dan Hendrycks and Thomas G. Dietterich. Benchmarking neural network robustness to common corruptions and perturbations. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL https://openreview.net/forum?id=HJz6tiCqYm.

[12] Nick Hoernle, Rafael Michael Karampatsis, Vaishak Belle, and Kobi Gal. Multiplexnet: Towards fully satisfied logical constraints in neural networks. *Proceedings of the AAAI Conference on Artificial Intelligence*, 36(5):5700–5709, Jun. 2022. doi: 10.1609/aaai.v36i5.20512. URL https://ojs.aaai.org/index.php/AAAI/article/view/20512.

[13] Hanjiang Hu, Yujie Yang, Tianhao Wei, and Changliu Liu. Verification of neural control barrier functions with symbolic derivative bounds propagation. In Pulkit Agrawal, Oliver Kroemer, and Wolfram Burgard, editors, *Proceedings of The 8th Conference on Robot Learning*, volume 270 of *Proceedings of Machine Learning Research*, pages 1797–1814. PMLR, 06–09 Nov 2025. URL https://proceedings.mlr.press/v270/hu25a.html.

[14] Zhiting Hu, Xuezhe Ma, Zhengzhong Liu, Eduard Hovy, and Eric Xing. Harnessing deep neural networks with logic rules. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2410–2420, Berlin, Germany, August 2016. Association for Computational Linguistics. doi: 10.18653/v1/P16-1228. URL https://aclanthology.org/P16-1228.

[15] Jacob Laurel, Rem Yang, Shubham Ugare, Robert Nagel, Gagandeep Singh, and Sasa Misailovic. A general construction for abstract interpretation of higher-order automatic differentiation. *Proceedings of the ACM on Programming Languages*, 6(OOPSLA2):1007–1035, 2022.

[16] Jacob Laurel, Siyuan Brant Qian, Gagandeep Singh, and Sasa Misailovic. Synthesizing precise static analyzers for automatic differentiation. *Proceedings of the ACM on Programming Languages*, 7(OOPSLA2):1964–1992, 2023.

[17] Zenan Li, Zehua Liu, Yuan Yao, Jingwei Xu, Taolue Chen, Xiaoxing Ma, and Jian L\"{u}. Learning with logical constraints but without shortcut satisfaction. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=M2unceRvqhh.

[18] TorchVision maintainers and contributors. Torchvision: Pytorch's computer vision library. https://github.com/pytorch/vision, 2016.

[19] Dmitry Malioutov and Kuldeep S Meel. Mlic: A maxsat-based framework for learning interpretable classification rules. In *International Conference on Principles and Practice of Constraint Programming*, pages 312–327. Springer, 2018.

[20] Yuhao Mao, Mark Niklas Mueller, Marc Fischer, and Martin Vechev. Connecting certified and adversarial training. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=T2lM4ohRwb.

[21] Matthew Mirman, Timon Gehr, and Martin T. Vechev. Differentiable abstract interpretation for provably robust neural networks. In Jennifer G. Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 3575–3583. PMLR, 2018. URL http://proceedings.mlr.press/v80/mirman18b.html.

[22] Mark Niklas Mueller, Franziska Eckert, Marc Fischer, and Martin Vechev. Certified training: Small boxes are all you need. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=7oFuxtJtUMH.

[23] Yatin Nandwani, Abhishek Pathak, Mausam, and Parag Singla. A primal dual formulation for deep learning with constraints. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper_files/paper/2019/file/cf708fc1decf0337aded484f8f4519ae-Paper.pdf.

[24] Alessandro De Palma, Rudy Bunel, Krishnamurthy Dvijotham, M. Pawan Kumar, and Robert Stanforth. Ibp regularization for verified adversarial robustness via branch-and-bound, 2023.

[25] Geondo Park, June Yong Yang, Sung Ju Hwang, and Eunho Yang. Attribution preservation in network compression for reliable network interpretation. *Advances in Neural Information Processing Systems*, 33:5093–5104, 2020.

[26] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Z. Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. *CoRR*, abs/1912.01703, 2019. URL `http://arxiv.org/abs/1912.01703`.

[27] Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. *Advances in neural information processing systems*, 36:53728–53741, 2023.

[28] Maziar Raissi, Paris Perdikaris, and George E Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational physics*, 378:686–707, 2019.

[29] Ramprasaath R Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-CAM: visual explanations from deep networks via gradient-based localization. *International journal of computer vision*, 128:336–359, 2020.

[30] Ramprasaath Ramasamy Selvaraju, Stefan Lee, Yilin Shen, Hongxia Jin, Shalini Ghosh, Larry P. Heck, Dhruv Batra, and Devi Parikh. Taking a HINT: leveraging explanations to make vision and language models more grounded. In *ICCV*, pages 2591–2600. IEEE, 2019.

[31] Zhouxing Shi, Yihan Wang, Huan Zhang, Jinfeng Yi, and Cho-Jui Hsieh. Fast certified robust training with short warmup. In *ICML 2021 Workshop on Adversarial Machine Learning*, 2021. URL `https://openreview.net/forum?id=_jUobmvki51`.

[32] Zhouxing Shi, Yihan Wang, Huan Zhang, J Zico Kolter, and Cho-Jui Hsieh. Efficiently computing local lipschitz constants of neural networks via bound propagation. *Advances in Neural Information Processing Systems*, 35:2350–2364, 2022.

[33] Zhouxing Shi, Qirui Jin, J Zico Kolter, Suman Jana, Cho-Jui Hsieh, and Huan Zhang. Formal verification for neural networks with general nonlinearities via branch-and-bound. 2023.

[34] Gagandeep Singh, Timon Gehr, Matthew Mirman, Markus Püschel, and Martin T. Vechev. Fast and effective robustness certification. In Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 10825–10836, 2018. URL `https://proceedings.neurips.cc/paper/2018/hash/f2f446980d8e971ef3da97af089481c3-Abstract.html`.

[35] Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin T. Vechev. An abstract domain for certifying neural networks. *Proc. ACM Program. Lang.*, 3(POPL):41:1–41:30, 2019. doi: 10.1145/3290354. URL `https://doi.org/10.1145/3290354`.

[36] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Y. Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing, EMNLP 2013, 18-21 October 2013, Grand Hyatt Seattle, Seattle, Washington, USA, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 1631–1642. ACL, 2013. URL `https://aclanthology.org/D13-1170/`.

[37] Matthew Sotoudeh and Aditya Thakur. Prdnn. `https://github.com/95616ARG/PRDNN`, 2021.

[38] Kaustubh Sridhar, Souradeep Dutta, James Weimer, and Insup Lee. Guaranteed conformance of neurosymbolic models to natural constraints. In *Learning for Dynamics and Control Conference*, pages 76–89. PMLR, 2023.

[39] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic attribution for deep networks. In *International conference on machine learning*, pages 3319–3328. PMLR, 2017.

[40] Zhe Tao and Aditya Thakur. Provable editing of deep neural networks using parametric linear relaxation. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL `https://openreview.net/forum?id=IGhpUd496D`.

[41] Zhe Tao, Stephanie Nawas, Jacqueline Mitchell, and Aditya V. Thakur. Architecture-preserving provable repair of deep neural networks. *Proc. ACM Program. Lang.*, 7(PLDI), jun 2023. doi: 10.1145/3591238. URL `https://doi.org/10.1145/3591238`.

[42] Leandro von Werra, Younes Belkada, Lewis Tunstall, Edward Beeching, Tristan Thrush, Nathan Lambert, Shengyi Huang, Kashif Rasul, and Quentin Gallouédec. TRL: Transformer Reinforcement Learning, 2025. URL `https://github.com/huggingface/trl`.

[43] Shiqi Wang, Huan Zhang, Kaidi Xu, Xue Lin, Suman Jana, Cho-Jui Hsieh, and J. Zico Kolter. Beta-crown: Efficient bound propagation with per-neuron split constraints for neural network robustness verification. In Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan, editors, *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 29909–29921, 2021. URL `https://proceedings.neurips.cc/paper/2021/hash/fac7fead96dafceaf80c1daffeae82a4-Abstract.html`.

[44] Tianhao Wei, Hanjiang Hu, Luca Marzari, Kai S Yun, Peizhi Niu, Xusheng Luo, and Changliu Liu. Modelverification. jl: a comprehensive toolbox for formally verifying deep neural networks. In *International Conference on Computer Aided Verification*, pages 395–408. Springer, 2025.

[45] Jingyi Xu, Zilu Zhang, Tal Friedman, Yitao Liang, and Guy Van den Broeck. A semantic loss function for deep learning with symbolic knowledge. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 5502–5511. PMLR, 10–15 Jul 2018. URL `https://proceedings.mlr.press/v80/xu18h.html`.

[46] Kaidi Xu, Huan Zhang, Shiqi Wang, Yihan Wang, Suman Jana, Xue Lin, and Cho-Jui Hsieh. Fast and complete: Enabling complete neural network verification with rapid and massively parallel incomplete verifiers. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. URL `https://openreview.net/forum?id=nVZtXBI6LNn`.

[47] Ziwei Xu, Yogesh Rawat, Yongkang Wong, Mohan S Kankanhalli, and Mubarak Shah. Don't pour cereal into coffee: Differentiable temporal logic for temporal action segmentation. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 14890–14903. Curran Associates, Inc., 2022. URL `https://proceedings.neurips.cc/paper_files/paper/2022/file/5f96a21345c138da929e99871fda138e-Paper-Conference.pdf`.

[48] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

[49] Huan Zhang, Tsui-Wei Weng, Pin-Yu Chen, Cho-Jui Hsieh, and Luca Daniel. Efficient neural network robustness certification with general activation functions. In Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 4944–4953, 2018. URL `https://proceedings.neurips.cc/paper/2018/hash/d04863f100d59b3eb688a11f95b0ae60-Abstract.html`.

[50] 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.

A Proofs

Theorem 4.3. *Given a provable gradient editing problem (Definition 4.1) for DNN $\mathcal{N}$ and parameters θ with input $\mathbf{x}$ and a gradient constraint $\mathrm{Q} \stackrel{\text{def}}{=} \{\mathbf{z} \in \mathbb{R}^n \mid \mathbf{A}\mathbf{z} \leq \mathbf{b}\}$. Let $\frac{\partial}{\partial \mathbf{x}}\widetilde{\mathcal{N}}(\mathbf{x}; \widehat{\boldsymbol{\theta}})$ be the conditional variable gradient of the DNN $\mathcal{N}$ with respect to the input $\mathbf{x}$, under the activation condition $\widetilde{\varphi}$. The following linear program can be solved in polynomial time in the size of the edited layers of the DNN $\mathcal{N}$, and its solution is a solution to the provable gradient editing problem.*

$$\min \|\widehat{\boldsymbol{\theta}} - \theta\| \quad \textit{s.t.} \quad \widetilde{\varphi} \ \wedge \ \mathbf{A}\,\frac{\partial}{\partial \mathbf{x}}\widetilde{\mathcal{N}}(\mathbf{x}; \widehat{\boldsymbol{\theta}}) \leq \mathbf{b} \tag{15}$$

Proof. We first show that **Equation 15 is a linear program that can be solved in polynomial time in the size of the DNN**. By the definition of conditional variable gradient (Definition 4.2), **(i)** $\frac{\partial}{\partial \mathbf{x}}\widetilde{\mathcal{N}}(\mathbf{x}; \widehat{\boldsymbol{\theta}})$ is a linear expression and $\widetilde{\varphi}$ is a linear formula, hence Equation 15 is a linear program; **(ii)** the sizes of both $\frac{\partial}{\partial \mathbf{x}}\widetilde{\mathcal{N}}(\mathbf{x}; \widehat{\boldsymbol{\theta}})$ and $\widetilde{\varphi}$ are polynomial in the size of the DNN, i.e., the number of parameters, layers, and the input and output dimensions of each layer, hence Equation 15 can be solved in polynomial time in the size of the DNN.

We then show that **any solution to Equation 15 is a solution to the provable gradient editing problem (Equation 14 in Definition 4.1)**. By the definition of conditional variable gradient (Definition 4.2), for any solution θ' to the variable parameters $\widehat{\boldsymbol{\theta}}$ that satisfies the activation condition $\widetilde{\varphi}$, $\frac{\partial}{\partial \mathbf{x}}\widetilde{\mathcal{N}}(\mathbf{x}; \theta') = \frac{\partial}{\partial \mathbf{x}}\mathcal{N}(\mathbf{x}; \theta')$, hence $\mathbf{A}\frac{\partial}{\partial \mathbf{x}}\mathcal{N}(\mathbf{x}; \theta') \leq \mathbf{b}$ holds and Equation 14 is satisfied. ☐

Theorem 4.5. *The conditional variable gradient constructed in Definition 4.4 is valid and satisfies the conditions stated in Definition 4.2.*

Proof. We will recall Definition 4.4 and prove this theorem step by step. Consider an L-layer feed-forward ReLU deep neural network (DNN) $\mathcal{N}$ with parameters $\widehat{\boldsymbol{\theta}} \stackrel{\text{def}}{=} \{\widehat{\mathbf{W}}^{(0)}, \mathbf{W}^{(1)}, \ldots, \mathbf{W}^{(L-1)}, \widehat{\mathbf{b}}^{(0)}, \widehat{\mathbf{b}}^{(1)}, \ldots, \widehat{\mathbf{b}}^{(L-1)}\}$, where the first-layer weight $\widehat{\mathbf{W}}^{(0)}$ and all-layer biases $\widehat{\mathbf{b}}^{(\ell)}$ are variables. For an input $\mathbf{x} \in \mathbb{R}^n$, the conditional variable gradient $\frac{\partial}{\partial \mathbf{x}}\widetilde{\mathcal{N}}(\mathbf{x}; \widehat{\boldsymbol{\theta}})$ of a DNN $\mathcal{N}$ in terms of $\widehat{\boldsymbol{\theta}}$ with respect to the input $\mathbf{x}$ is defined by the chain rule as

$$\frac{\partial}{\partial \mathbf{x}}\widetilde{\mathcal{N}}(\mathbf{x}; \widehat{\boldsymbol{\theta}}) \stackrel{\text{def}}{=} \prod_{\ell=L-1}^{0} \frac{\partial \widetilde{\mathbf{x}}^{(\ell+1)}}{\partial \widetilde{\mathbf{z}}^{(\ell)}} \frac{\partial \widetilde{\mathbf{z}}^{(\ell)}}{\partial \widetilde{\mathbf{x}}^{(\ell)}} \tag{23}$$

with the activation condition $\widetilde{\varphi} \stackrel{\text{def}}{=} \bigwedge_\ell \widetilde{\varphi}^{(\ell)}$ defined from each layer ℓ.

We will show below that **(i) except for $\frac{\partial \widetilde{\mathbf{z}}^{(0)}}{\partial \mathbf{x}^{(0)}}$ for the first layer $\ell = 0$ is a linear expression, all other $\frac{\partial \widetilde{\mathbf{x}}^{(\ell+1)}}{\partial \widetilde{\mathbf{z}}^{(\ell)}}$ and $\frac{\partial \widetilde{\mathbf{z}}^{(\ell)}}{\partial \widetilde{\mathbf{x}}^{(\ell)}}$ are *constant* matrices (ii) $\widetilde{\varphi}^{(\ell)}$ for each layer ℓ are linear formulas, $\frac{\partial \widetilde{\mathbf{z}}^{(0)}}{\partial \mathbf{x}^{(0)}}$ is a linear expression, and their sizes are polynomial in the number of input and output neurons as well as parameters of the layer ℓ; (iii) for each layer ℓ, $\frac{\partial \widetilde{\mathbf{z}}^{(\ell)}}{\partial \widetilde{\mathbf{x}}^{(\ell)}} = \frac{\partial \widetilde{\mathbf{z}}^{(\ell)}}{\partial \widetilde{\mathbf{x}}^{(\ell)}}$, and $\widetilde{\varphi}^{(\ell)}$ implies $\frac{\partial \widetilde{\mathbf{x}}^{(\ell+1)}}{\partial \widetilde{\mathbf{z}}^{(\ell)}} = \frac{\partial \widetilde{\mathbf{x}}^{(\ell+1)}}{\partial \widetilde{\mathbf{z}}^{(\ell)}}$.** Consequently, we can conclude that **(i)** $\frac{\partial}{\partial \mathbf{x}}\widetilde{\mathcal{N}}(\mathbf{x}; \widehat{\boldsymbol{\theta}})$ is a linear expression, because it is a product of one linear expression ($\frac{\partial \widetilde{\mathbf{z}}^{(0)}}{\partial \mathbf{x}^{(0)}}$) and many constant matrices (all other $\frac{\partial \widetilde{\mathbf{x}}^{(\ell+1)}}{\partial \widetilde{\mathbf{z}}^{(\ell)}}$ and $\frac{\partial \widetilde{\mathbf{z}}^{(\ell)}}{\partial \widetilde{\mathbf{x}}^{(\ell)}}$); **(ii)** $\widetilde{\varphi} \stackrel{\text{def}}{=} \bigwedge_\ell \widetilde{\varphi}^{(\ell)}$ is a linear formula, the sizes of $\frac{\partial}{\partial \mathbf{x}}\widetilde{\mathcal{N}}(\mathbf{x}; \widehat{\boldsymbol{\theta}})$ and $\widetilde{\varphi}$ are polynomial in the size of the DNN $\mathcal{N}$, i.e., the number of parameters, layers, and the input and output dimensions of each layer; **(iii)** by induction, $\widetilde{\varphi}$ implies $\frac{\partial}{\partial \mathbf{x}}\widetilde{\mathcal{N}}(\mathbf{x}; \widehat{\boldsymbol{\theta}}) = \frac{\partial}{\partial \mathbf{x}}\mathcal{N}(\mathbf{x}; \widehat{\boldsymbol{\theta}})$. Hence, the conditional variable gradient constructed in Definition 4.4 is valid and satisfies the conditions stated in Definition 4.2.

We first prove those conditions for the affine transformation. For the conditional affine transformation $\widetilde{\mathbf{z}}^{(0)} \stackrel{\text{def}}{=} \widehat{\mathbf{W}}^{(0)}\mathbf{x}^{(0)} + \widehat{\mathbf{b}}^{(0)}$ of the first layer $\ell = 0$ with variable weight $\widehat{\mathbf{W}}^{(0)}$ and constant input $\mathbf{x}^{(0)}$, the conditional variable gradient $\frac{\partial \widetilde{\mathbf{z}}^{(0)}}{\partial \mathbf{x}^{(0)}}$ of the pre-activation output $\widetilde{\mathbf{z}}^{(0)}$ with respect to the layer input $\mathbf{x}^{(0)}$ is

$$\frac{\partial \widetilde{\mathbf{z}}^{(0)}}{\partial \mathbf{x}^{(0)}} \stackrel{\text{def}}{=} \widehat{\mathbf{W}}^{(0)} \tag{24}$$

which **(i) is a matrix of variable weights whose size is the same as $\mathbf{W}^{(\ell)}$, hence linear; (ii) unconditionally equals to the exact gradient $\frac{\partial \widetilde{\mathbf{z}}^{(0)}}{\partial \mathbf{x}^{(0)}} \overset{\text{def}}{=} \widehat{\mathbf{W}}^{(0)}$.**

For the conditional affine transformation $\widetilde{\mathbf{z}}^{(\ell)} \overset{\text{def}}{=} \mathbf{W}^{(\ell)} \widetilde{\mathbf{x}}^{(\ell)} + \widehat{\mathbf{b}}^{(\ell)}$ of the non-first-layer $\ell > 0$ with constant weight $\mathbf{W}^{(\ell)}$ and conditional variable input $\widetilde{\mathbf{x}}^{(\ell)}$, the conditional variable gradient $\frac{\partial \widetilde{\mathbf{z}}^{(\ell)}}{\partial \widetilde{\mathbf{x}}^{(\ell)}}$ of the pre-activation output $\widetilde{\mathbf{z}}^{(\ell)}$ with respect to the layer input $\widetilde{\mathbf{x}}^{(\ell)}$ is

$$\frac{\partial \widetilde{\mathbf{z}}^{(\ell)}}{\partial \widetilde{\mathbf{x}}^{(\ell)}} \overset{\text{def}}{=} \mathbf{W}^{(\ell)} \tag{25}$$

which **(i) is a constant matrix, hence linear; (ii) unconditionally equals to the exact gradient $\frac{\partial \widetilde{\mathbf{z}}^{(\ell)}}{\partial \widetilde{\mathbf{x}}^{(\ell)}} \overset{\text{def}}{=} \mathbf{W}^{(\ell)}$.**

For the conditional ReLU activation $\widetilde{\mathbf{x}}^{(\ell+1)} \overset{\text{def}}{=} \operatorname{diag}\big(\mathbf{1}_{\mathbf{z}^{(\ell)}>0}\big)\, \widetilde{\mathbf{z}}^{(\ell)}$ of non-last layer $\ell < L-1$, let p be the number of output neurons of the layer ℓ. **We first show that $\frac{\partial \widetilde{\mathbf{x}}^{(\ell+1)}}{\partial \widetilde{\mathbf{z}}^{(\ell)}}$ is a constant matrix, and $\widetilde{\varphi}^{(\ell)}$ is a linear formula whose size is polynomial in the number of output neurons p of the layer ℓ.** Recall that the conditional variable gradient $\frac{\partial \widetilde{\mathbf{x}}^{(\ell+1)}}{\partial \widetilde{\mathbf{z}}^{(\ell)}}$ of the ℓ-th layer post-activation output $\widetilde{\mathbf{x}}^{(\ell+1)}$ with respect to the pre-activation output $\widetilde{\mathbf{z}}^{(\ell)}$ is

$$\frac{\partial \widetilde{\mathbf{x}}^{(\ell+1)}}{\partial \widetilde{\mathbf{z}}^{(\ell)}} \overset{\text{def}}{=} \operatorname{diag}\big(\mathbf{1}_{\mathbf{z}^{(\ell)}>0}\big) \tag{26}$$

with the ℓ-th layer activation condition $\widetilde{\varphi}^{(\ell)}$

$$\widetilde{\varphi}^{(\ell)} \overset{\text{def}}{=} \operatorname{diag}\big(\mathbf{1}_{\mathbf{z}^{(\ell)}>0}\big)\, \widetilde{\mathbf{z}}^{(\ell)} > 0 \;\wedge\; \operatorname{diag}\big(\mathbf{1}_{\mathbf{z}^{(\ell)}\leq 0}\big)\, \widetilde{\mathbf{z}}^{(\ell)} \leq 0 \tag{27}$$

$\frac{\partial \widetilde{\mathbf{x}}^{(\ell+1)}}{\partial \widetilde{\mathbf{z}}^{(\ell)}} \in \mathbb{R}^{p\times p}$ is a constant matrix of size $p \times p$, and $\widetilde{\varphi}^{(\ell)}$ is a conjunction of p linear inequalities, hence $\frac{\partial \widetilde{\mathbf{x}}^{(\ell+1)}}{\partial \widetilde{\mathbf{z}}^{(\ell)}}$ is a constant matrix, $\widetilde{\varphi}^{(\ell)}$ is a linear formula whose size is polynomial in the number of output neurons p of the layer ℓ.

We then show that $\widetilde{\varphi}^{(\ell)}$ implies $\frac{\partial \widetilde{\mathbf{x}}^{(\ell+1)}}{\partial \widetilde{\mathbf{z}}^{(\ell)}} = \frac{\partial \widehat{\mathbf{x}}^{(\ell+1)}}{\partial \widetilde{\mathbf{z}}^{(\ell)}}$ where $\widehat{\mathbf{x}}^{(\ell+1)} \overset{\text{def}}{=} \mathtt{ReLU}(\widetilde{\mathbf{z}}^{(\ell)})$. The activation condition $\widetilde{\varphi}^{(\ell)}$ constrains the ReLU activation pattern of the pre-activation conditional variable output $\widetilde{\mathbf{z}}^{(\ell)}$ to be the same as a constant pre-activation output $\mathbf{z}^{(\ell)}$. Hence, if $\widetilde{\varphi}^{(\ell)}$ is satisfied, then **(i)** for any $\mathbf{z}_i^{(\ell)} > 0$, we have the conditional variable $\widetilde{\mathbf{z}}_i^{(\ell)} > 0$, hence $\widehat{\mathbf{x}}_i^{(\ell+1)} = \widetilde{\mathbf{z}}_i^{(\ell)}$ and $\mathbf{1}_{\widetilde{\mathbf{z}}_i>0} = 1$; **(ii)** for any $\mathbf{z}_i^{(\ell)} \leq 0$, we have the conditional variable $\widetilde{\mathbf{z}}_i^{(\ell)} \leq 0$, hence $\widehat{\mathbf{x}}_i^{(\ell+1)} = 0$ and $\mathbf{1}_{\widetilde{\mathbf{z}}_i>0} = 0$. Therefore, we proved that $\widetilde{\varphi}^{(\ell)}$ implies $\operatorname{diag}\big(\mathbf{1}_{\mathbf{z}^{(\ell)}>0}\big) = \operatorname{diag}\big(\mathbf{1}_{\widetilde{\mathbf{z}}^{(\ell)}>0}\big)$, which is our goal $\frac{\partial \widetilde{\mathbf{x}}^{(\ell+1)}}{\partial \widetilde{\mathbf{z}}^{(\ell)}} = \frac{\partial \widehat{\mathbf{x}}^{(\ell+1)}}{\partial \widetilde{\mathbf{z}}^{(\ell)}}$.

For the last layer $\ell = L-1$ without ReLU activation, $\frac{\partial \widetilde{\mathbf{x}}^{(L)}}{\partial \widetilde{\mathbf{z}}^{(L-1)}} = \frac{\partial \widehat{\mathbf{x}}^{(L)}}{\partial \widetilde{\mathbf{z}}^{(L-1)}} = \operatorname{diag}(\mathbf{1})$ is an identity matrix and $\widetilde{\varphi}^{(L-1)} = \top$ has a constant size. Hence, **for the last layer $\ell = L-1$, $\frac{\partial \widetilde{\mathbf{x}}^{(L)}}{\partial \widetilde{\mathbf{z}}^{(L-1)}}$ is a constant matrix, $\widetilde{\varphi}^{(L-1)}$ is a linear formula of constant size, and $\widetilde{\varphi}^{(L-1)}$ trivially implies $\frac{\partial \widetilde{\mathbf{x}}^{(L)}}{\partial \widetilde{\mathbf{z}}^{(L-1)}} = \frac{\partial \widehat{\mathbf{x}}^{(L)}}{\partial \widetilde{\mathbf{z}}^{(L-1)}}$.** $\square$

B General provable gradient editing of DNNs

In this section, we present the extension of our approach presented in Section 4 to solve the general provable gradient editing, which handles **(i) efficient editing**: our approach allows editing only the last few layers of the DNN while keeping the rest of the DNN unchanged, which enables efficient editing of large DNNs; **(ii) general DNN architectures**: the editing is not restricted to multi-layer perceptron (MLP) fully-connected ReLU DNNs, but applies to DNNs of any architecture with an MLP ReLU DNN as the last few layers; **(iii) set of inputs**: our approach can edit the DNN for a set of inputs at the same time, guaranteeing constraint satisfaction for all of them; **(iv) general linear constraints**: our approach allows any linear constraints over the set of inputs, as well as their corresponding DNN outputs and gradients, not restricted to constraints that only talk about individual inputs. *The effectiveness and efficiency of our approach for solving the general provable gradient editing problem is demonstrated in our experiments (Section 5).*

Definition B.1. Given a DNN $\mathcal{N} \stackrel{\text{def}}{=} \mathcal{N}^{(\text{up})} \circ \mathcal{N}^{(\text{down})}$, which is a composition of a *general* DNN $\mathcal{N}^{(\text{up})}\colon \mathbb{R}^p \to \mathbb{R}^n$ of *any differentiable architecture*, and an L-layer fully-connected ReLU DNN $\mathcal{N}^{(\text{down})}\colon \mathbb{R}^n \to \mathbb{R}^m$ with parameters $\theta \stackrel{\text{def}}{=} \{\mathbf{W}^{(0)}, \mathbf{W}^{(1)}, \dots, \mathbf{W}^{(L-1)}, \mathbf{b}^{(0)}, \mathbf{b}^{(1)}, \dots, \mathbf{b}^{(L-1)}\}$ as defined in Definition 3.1. Let $\widehat{\boldsymbol{\theta}} \stackrel{\text{def}}{=} \{\widehat{\mathbf{W}}^{(0)}, \mathbf{W}^{(1)}, \dots, \mathbf{W}^{(L-1)}, \widehat{\mathbf{b}}^{(0)}, \widehat{\mathbf{b}}^{(1)}, \dots, \widehat{\mathbf{b}}^{(L-1)}\}$ be the new parameters of $\mathcal{N}^{(\text{down})}$. Given a set $\mathcal{S} \subseteq \{\mathbf{x} \mid \mathbf{x} \in \mathbb{R}^n\} \times \{\mathrm{Q} \mid \mathrm{Q} \subseteq \mathbb{R}^{m\times n}\}$ of pairs $(\mathbf{x}, \mathrm{Q})$, where $\mathbf{x} \in \mathbb{R}^n$ is a DNN input, and $\mathrm{Q} \stackrel{\text{def}}{=} \{\mathbf{J} \in \mathbb{R}^{m\times n} \mid \mathbf{A}\,\mathrm{vec}(\mathbf{J}) \le \mathbf{b}\}$ is the corresponding linear constraints on the Jacobian $\mathbf{J} \stackrel{\text{def}}{=} \frac{\partial}{\partial \mathbf{x}}\mathcal{N}(\mathbf{x}; \widehat{\boldsymbol{\theta}}) \in \mathbb{R}^{m\times n}$ of the DNN $\mathcal{N}$ with respect to the input $\mathbf{x} \in \mathbb{R}^n$, we use $\vec{\mathbf{J}}$ to denote the Jacobian matrix $\mathbf{J}$ flattened as a vector. The **provable gradient editing problem** is to find new parameters $\widehat{\boldsymbol{\theta}}$ that

$$\min \|\widehat{\boldsymbol{\theta}} - \theta\| \quad \text{s.t.} \quad \bigwedge_{(\mathbf{x},\mathrm{Q})\in\mathcal{S}} \frac{\partial}{\partial \mathbf{x}}\mathcal{N}(\mathbf{x}; \widehat{\boldsymbol{\theta}}) \in \mathrm{Q} \tag{28}$$

The problem above can be solved using conditional variable gradient of DNNs defined in Definition 4.4:

$$\min \|\widehat{\boldsymbol{\theta}} - \theta\| \quad \text{s.t.} \quad \bigwedge_{(\mathbf{x},\mathrm{Q})\in\mathcal{S}} \widetilde{\varphi}_{\mathbf{z}^{(\text{up})}} \wedge \mathbf{A}\frac{\partial \mathbf{z}^{(\text{up})}}{\partial \mathbf{x}} \frac{\partial}{\partial \mathbf{z}^{(\text{up})}} \widetilde{\mathcal{N}}^{(\text{down})}(\mathbf{z}^{(\text{up})}; \widehat{\boldsymbol{\theta}}) \le \mathbf{b} \tag{29}$$

where $\mathbf{z}^{(\text{up})} \stackrel{\text{def}}{=} \mathcal{N}^{(\text{up})}(\mathbf{x})$ denotes the input to $\mathcal{N}^{(\text{down})}$, $\widetilde{\varphi}_{\mathbf{z}^{(\text{up})}}$ denotes the activation condition constraint for $(\mathbf{x}, \mathrm{Q}) \in \mathcal{S}$, $\frac{\partial \mathbf{z}^{(\text{up})}}{\partial \mathbf{x}}$ denotes the Jacobian matrix of $\mathbf{z}^{(\text{up})}$ with respect to $\mathbf{x}$. ■

B.1 Analysis of the LP Size

First, we would like to emphasize that the size of the formulated linear program (LP) *does not* necessarily depend on the size (depth and width) of the entire DNN; viz., the size of the LP does not necessarily grow with the size of the DNN. Because `ProGrad` *does not require editing from the first layer; in practice,* `ProGrad` *allows editing only the last few layers ($\mathcal{N}^{(\text{down})}$) of the DNN, while keeping the rest of the DNN ($\mathcal{N}^{(\text{up})}$) unchanged, enabling efficient editing of large DNNs.*

In this section, we analyze the size of the LP for formulating the conditional variable gradient for one input in terms of the depth and width of the *non-frozen editing-layers* $\mathcal{N}^{(\text{down})}$, which are the the last L layers of an arbitrarily large DNN. For ease of presentation, we assume $\mathcal{N}$ is a feed-forward ReLU DNN as defined in Definition 3.1. Let n_ℓ and $n_{\ell+1}$ for $0 \le \ell < L$ be the input and output widths of the ℓ-th layer, viz., $\mathbf{x}^{(\ell)} \in \mathbb{R}^{n_\ell}$, $\mathbf{z}^{(\ell)} \in \mathbb{R}^{n_{\ell+1}}$, $\mathbf{W}^{(\ell)} \in \mathbb{R}^{n_{\ell+1}\times n_\ell}$ and $\mathbf{b}^{(\ell)} \in \mathbb{R}^{n_{\ell+1}}$. We will show that the size of the formulated LP for each edited layer is polynomial in the input and output widths of that layer, hence the LP size for encoding the conditional variable gradient of a large DNN $\mathcal{N}$ for one input is polynomial only in the size of the last L edited layers $\mathcal{N}^{(\text{down})}$.

LP size for encoding the pre-activation conditional variable output. For the first layer $\ell = 0$, encoding the pre-activation conditional variable output $\widetilde{\mathbf{z}}^{(0)}$ (Equation 7)

$$\widetilde{\mathbf{z}}^{(0)} \stackrel{\text{def}}{=} \widehat{\mathbf{W}}^{(0)}\mathbf{x}^{(0)} + \widehat{\mathbf{b}}^{(0)} \tag{7}$$

uses $\left|\widetilde{\mathbf{z}}^{(0)}\right| = n_1$ constraints, $\left|\widetilde{\mathbf{z}}^{(0)}\right| + \left|\widehat{\mathbf{W}}^{(0)}\right| + \left|\widehat{\mathbf{b}}^{(0)}\right| = n_1 + n_0 n_1 + n_1 = n_0 n_1 + 2n_1$ variables and less than $\left|\widetilde{\mathbf{z}}^{(0)}\right| \times (1 + \left|\widehat{\mathbf{W}}^{(0)}\right| + 1) = n_0 n_1^2 + 2n_1$ non-zero entries in the formulated LP.

For other layers $0 < \ell < L$, encoding the pre-activation conditional variable output $\widetilde{\mathbf{z}}^{(\ell)}$ (Equation 8)

$$\widetilde{\mathbf{z}}^{(\ell)} \overset{\text{def}}{=} \mathbf{W}^{(\ell)} \widetilde{\mathbf{x}}^{(\ell)} + \widehat{\mathbf{b}}^{(\ell)} \tag{8}$$

uses $\left|\widetilde{\mathbf{z}}^{(\ell)}\right| = n_{\ell+1}$ constraints, $\left|\widetilde{\mathbf{z}}^{(\ell)}\right| + \left|\widetilde{\mathbf{x}}^{(\ell)}\right| + \left|\widehat{\mathbf{b}}^{(\ell)}\right| = n_{\ell+1} + n_\ell + n_{\ell+1} = n_\ell + 2n_{\ell+1}$ variables and less than $\left|\widetilde{\mathbf{z}}^{(\ell)}\right| \times (1 + \left|\widetilde{\mathbf{x}}^{(\ell)}\right| + 1) = n_\ell n_{\ell+1} + 2n_{\ell+1}$ non-zero entries in the formulated LP.

LP size for encoding the post-activation conditional layer output. For any layer $0 \leq \ell < L$, encoding the ℓ-th layer post-activation conditional variable layer output $\widetilde{\mathbf{x}}^{(\ell+1)}$ (Equation 9)

$$\widetilde{\mathbf{x}}^{(\ell+1)} \overset{\text{def}}{=} \operatorname{diag}\left(\mathbf{1}_{\mathbf{z}^{(\ell)}>0}\right) \widetilde{\mathbf{z}}^{(\ell)} \tag{9}$$

uses $\left|\widetilde{\mathbf{x}}^{(\ell+1)}\right| = n_{\ell+1}$ constraints, $\left|\widetilde{\mathbf{x}}^{(\ell+1)}\right| + \left|\widetilde{\mathbf{z}}^{(\ell)}\right| = n_{\ell+1} + n_{\ell+1} = 2n_{\ell+1}$ variables and less than $2 \times \left|\widetilde{\mathbf{x}}^{(\ell+1)}\right| = 2n_{\ell+1}$ non-zero entries in the formulated LP.

LP size for encoding the activation condition. For any non-last layer $0 \leq \ell < L-1$, encoding the ℓ-th layer activation condition $\widetilde{\varphi}^{(\ell)}$ (Equation 10)

$$\widetilde{\varphi}^{(\ell)} \overset{\text{def}}{=} \operatorname{diag}\left(\mathbf{1}_{\mathbf{z}^{(\ell)}>0}\right) \widetilde{\mathbf{z}}^{(\ell)} > 0 \ \wedge \ \operatorname{diag}\left(\mathbf{1}_{\mathbf{z}^{(\ell)}\leq 0}\right) \widetilde{\mathbf{z}}^{(\ell)} \leq 0 \tag{10}$$

uses $2 \times \left|\widetilde{\mathbf{z}}^{(\ell)}\right| = 2n_{\ell+1}$ constraints, $\left|\widetilde{\mathbf{z}}^{(\ell)}\right| = n_{\ell+1}$ variables and less than $2 \times \left|\widetilde{\mathbf{z}}^{(\ell)}\right| = 2n_{\ell+1}$ non-zero entries in the formulated LP.

For the last layer $\ell = L-1$, there is no ReLU activation, hence no activation condition to encode.

LP size for encoding the conditional variable gradient. Encoding the conditional variable gradient for the first editing layer $\frac{\partial \widetilde{\mathbf{z}}^{(0)}}{\partial \mathbf{x}^{(0)}}$ (Equation 20)

$$\frac{\partial \widetilde{\mathbf{z}}^{(0)}}{\partial \mathbf{x}^{(0)}} \overset{\text{def}}{=} \widehat{\mathbf{W}}^{(0)} \tag{20}$$

uses $\left|\frac{\partial \widetilde{\mathbf{z}}^{(0)}}{\partial \mathbf{x}^{(0)}}\right| = n_1 n_0$ constraints, $\left|\frac{\partial \widetilde{\mathbf{z}}^{(0)}}{\partial \mathbf{x}^{(0)}}\right| + \left|\widehat{\mathbf{W}}^{(0)}\right| = n_0 n_1 + n_0 n_1$ variables, and less than $2 \times (\left|\frac{\partial \widetilde{\mathbf{z}}^{(0)}}{\partial \mathbf{x}^{(0)}}\right| + \left|\widehat{\mathbf{W}}^{(0)}\right|) = 2n_0 + 2n_1$ non-zero entries in the formulated LP.

Because the gradient for $\mathcal{N}^{(\text{down})(1:L)}$ is constant, encoding the variable gradient for $\frac{\partial}{\partial \mathbf{x}} \widetilde{\mathcal{N}}^{(\text{down})}(\mathbf{x}; \widehat{\boldsymbol{\theta}})$ uses $n_0 n_L$ constraints, $n_0 n_L + n_1 n_L$ variables, and less than $n_0 n_1 \times n_0 n_L + n_0 n_L$ non-zero entries in the formulated LP.

Because the gradient for the frozen layers $\mathcal{N}^{(\text{up})}$ is constant, let n denote the input width to $\mathcal{N}$, encoding the variable gradient for $\mathcal{N}$ uses $n n_L$ constraints, $n n_L + n_0 n_L$ variables, and less than $n n_0 \times n n_L + n n_L$ non-zero entries in the formulated LP.

Therefore, by induction, we have that the size of the formulated LP for encoding the conditional variable gradient for one input is polynomial in the input and output widths of each edited layer, hence polynomial in the size of the last L edited layers $\mathcal{N}^{(\text{down})}$ instead of the entire DNN $\mathcal{N}$. When editing the DNN gradients for a set of inputs, the size of the formulated LP grows linearly with the number of inputs in the set.

C Evaluation Details

C.1 Enforcing hard Grad-CAM constraints on ResNet DNNs for IMAGENET.

In this section we present the experiment details for Section 5.1.

Setup details. For the two edit sets, (**i**) the 100 images with ε=1e-2 edit set consists of the first 100 images from the first 50 classes of the IMAGENET-C dataset with Gaussian noise and severity 1, for which the original uncorrupted images are correctly classified, while the corrupted images are top-1 misclassified but top-5 correctly classified; (**ii**) the 1,000 images with ε=5e-2 edit set consists of the first 1,000 images from the first 200 classes of the IMAGENET-C dataset with Gaussian noise and severity 1, for which the original uncorrupted images are correctly classified, while the corrupted images are top-1 misclassified but top-5 correctly classified. Both cosine similarity and IoU are computed on the min-max-normalized Grad-CAM attributions, where IoU uses a 75% percentile threshold.

The `ResNet152` DNN is from `torchvision` with the pre-trained weights `ResNet152_Weights.IMAGENET1K_V1`: `https://docs.pytorch.org/vision/main/models/generated/torchvision.models.resnet152.html#torchvision.models.ResNet152_Weights`. The `ResNet50` DNN is from `torchvision` with the pre-trained weights `ResNet50_Weights.IMAGENET1K_V1`: `https://docs.pytorch.org/vision/main/models/generated/torchvision.models.resnet50.html#torchvision.models.ResNet50_Weights`.

For each gradient-descent-based baseline, we perform a grid search over the learning rate, batch size, number of epochs, and transfer weight hyperparameters, with a time limit of 12 hours, and report the best results with the best constraint satisfaction rate.

Setup for `EWA` baseline. The hyperparameters are taken from the original `EWA` experiments. We use the SGD optimizer with learning rate $1e-3$, momentum 0.9 and weight decay $1e-4$. We fine-tune the last layer of `ResNet152` for 300 epochs with a batch size of 100. For the 100 images with $\varepsilon = 1e-2$ experiment, we use the transfer weight $1e4$; for the 1000 images with $\varepsilon = 5e-2$ experiment, we use the transfer weight $1e3$.

Setup for `SWA` baseline. The hyperparameters are taken from the original `SWA` experiments. We use the SGD optimizer with learning rate $1e-3$, momentum 0.9 and weight decay $1e-4$. We fine-tune the last layer of `ResNet152` for 300 epochs with a batch size of 100. For the 100 images with $\varepsilon = 1e-2$ experiment, we use the transfer weight $1e4$; for the 1000 images with $\varepsilon = 5e-2$ experiment, we use the transfer weight $1e3$.

Setup for `SSWA` baseline. The hyperparameters are taken from the original `SSWA` experiments. We use the SGD optimizer with learning rate $1e-3$, momentum 0.9 and weight decay $1e-4$. We use the `SSWA` erase rate 0.3, and fine-tune the last layer of `ResNet152` for 300 epochs with a batch size of 100. For the 100 images with $\varepsilon = 1e-2$ experiment, we use the transfer weight $1e4$; for the 1000 images with $\varepsilon = 5e-2$ experiment, we use the transfer weight $1e3$.

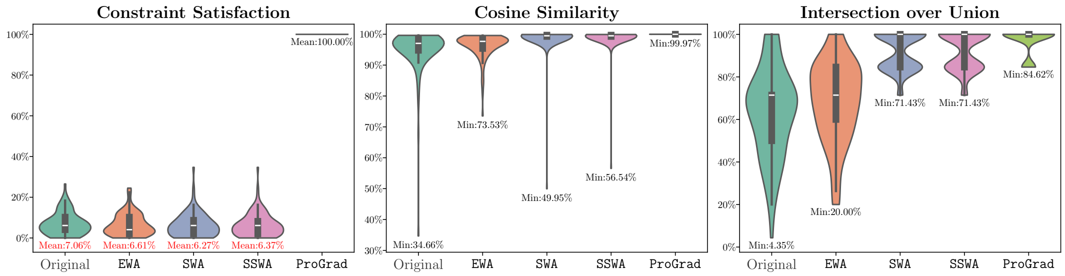
Setup for `ProGrad`. We edit the last layer of both ResNet DNNs, with the changes to the parameters bounded by $[-3, 3]$.

Additional statistics for Grad-CAM similarity metrics. Figure 4 shows the statistics of various Grad-CAM similarity metrics, viz., the constraint satisfaction rate (Constr. Sat.), cosine similarity (Cos.) and intersection over union (IoU) between the expected and edited Grad-CAM attributions for the `ResNet152` experiments.

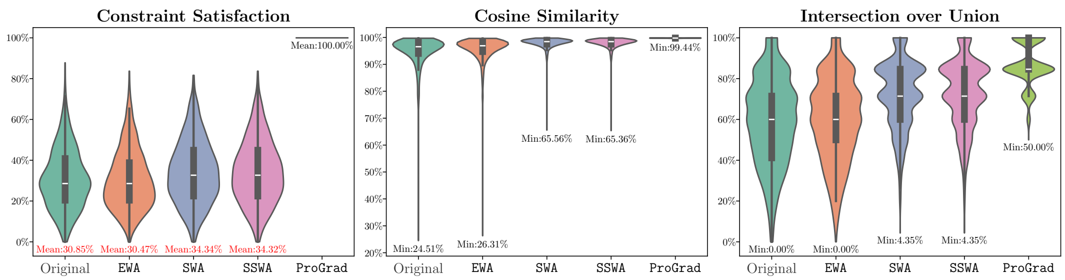
Additional generalization results. In Table 4 we present additional generalization results for the `ResNet152` experiment on enforcing hard Grad-CAM constraints with 1000 samples in Section 5.1. As evidenced by the results, `ProGrad` improves those metrics comparing to the original model, and achieves the best generalization comparing to the baselines.

C.2 Enforce hard Integrated Gradients constraint for SST-2 Llama LLM.

In this section we present the experiment details for Section 5.2.



(a) Statistics of the 100 images with ε=1e-2 experiment on `ResNet152`.



(b) Statistics of the 1,000 images with ε=5e-2 experiment on `ResNet152`.

Figure 4: Additional statistics for Grad-CAM similarity metrics of Table 1 in the main paper (Section 5.1), viz., the (pixel-level) Grad-CAM constraint satisfaction rate (Constr. Sat.), cosine similarity (Cos.) and intersection over union (IoU) between the expected and edited Grad-CAMs.

Table 4: Additional generalization results for the `ResNet152` experiment on enforcing hard Grad-CAM constraints with 1000 samples in Section 5.1. For a generalization set consisting of 4,000 images that's similar to the 1,000-image edit set but with different corruption levels, we present the constraint satisfaction rate (Constr. Sat.) with ε=1e-1, cosine similarity (Cos.) and intersection over union (IoU) on the generalization set, as well as the top-1 accuracy (Acc.) of the edited DNN on the ILSVRC 2012 ImageNet validation set. As evidenced by the results, `ProGrad` improves those metrics comparing to the original model, and achieves the best generalization comparing to the baselines.

Method	Constr. Sat.	Min. Cos.	Min. IoU	Acc.
Original	32.98%	77.53%	39.42%	78.31%
`EWA`	35.71%	82.41%	39.89%	73.32%
`SWA`	36.15%	84.74%	45.28%	76.55%
`SSWA`	36.16%	84.74%	45.26%	76.56%
`ProGrad`	**42.35%**	**85.75%**	**47.43%**	**77.43%**

Setup details. For both constant and conditional variable integrated gradients, we use Gauss-Legendre quadrature to approximate the integral with 25 steps in `bfloat16` precision. Following the common practice, we use the zero point in the token space as the baseline, i.e., a vector of token ID 0. We apply hard constraints and evaluate IG similarity metrics on the IG of the actual sentences from the SST-2 dataset, ignoring the template and special tokens. Both cosine similarity and IoU are computed on the min-max-normalized IG attributions, where IoU uses a 50% percentile threshold. The `Llama-3.2-1b-Instruct` LLM for editing is from `https://huggingface.co/meta-llama/Llama-3.2-1B-Instruct`; The teacher `Llama-3.1-8b-Instruct` LLM for computing the expected IG is from `https://huggingface.co/meta-llama/Llama-3.1-8B-Instruct`. The `Qwen3-1.7B` LLM for editing is from `https://huggingface.co/Qwen/Qwen3-1.7B`; The teacher `Qwen3-8B` LLM for computing the expected IG is from `https://huggingface.co/Qwen/Qwen3-8B`.

For each gradient-descent-based baseline, we perform a grid search over the learning rate, batch size, number of epochs, and trainable parameters, with a time limit of 4 hours, and report the best results with the best constraint satisfaction rate.

Setup for `SFT` baselines. We use `trl` [42] with a learning rate from {1e-3, 1e-4}, epochs from $\{1, 2, 3, 4, 5\}$, `bfloat16` precision, and trainable parameters from the language modelling head (`lm_head`) only or all layers. In Table 2 we report the best results among all hyperparameter combinations with the highest constraint satisfaction rate. For the Llama 3 experiment with 100 samples, the best result uses a learning rate of 1e-3, fine-tunes all layers for 1 epoch; for the Llama 3 experiment with 200 samples, the best result uses a learning rate of 1e-3, fine-tunes all layers for 3 epochs; for the Qwen 3 experiment with 100 samples, the best result uses a learning rate of 1e-4, fine-tunes all layers for 5 epochs; for the Qwen 3 experiment with 200 samples, the best result uses a learning rate of 1e-3, fine-tunes all layers for 3 epochs.

Setup for `DPO` baselines. We use `trl` [42] with a learning rate from {1e-3, 1e-4}, epochs from $\{1, 2, 3, 4, 5\}$, `bfloat16` precision, and trainable parameters from the language modelling head (`lm_head`) only or all layers. In Table 2 we report the best results among all hyperparameter combinations with the highest constraint satisfaction rate. For the Llama 3 experiment with 100 samples, the best result uses a learning rate of 1e-3, fine-tunes all layers for 1 epoch; for the Llama 3 experiment with 200 samples, the best result uses a learning rate of 1e-3, fine-tunes all layers for 1 epoch; for the Qwen 3 experiment with 100 samples, the best result uses a learning rate of 1e-3, fine-tunes `lm_head` for 1 epoch; for the Qwen 3 experiment with 200 samples, the best result uses a learning rate of 1e-3, fine-tunes `lm_head` for 1 epoch.

Setup for `ProGrad`. For both LLMs, we edit the language modelling head (`lm_head`) with the changes to the parameters bounded by $[-10, 10]$.

C.3 Enforce gradient constraints in training a DNN to approximate a target function

Setup. We use a 4-layer fully-connected DNN $\mathcal{N} : \mathbb{R} \to \mathbb{R}$ with 100 hidden neurons per layer and Tanh activation function. Over the domain $[0, 3\pi]$, we uniformly sample 2,048 points as the dataset $\mathcal{D}$. The DNN $\mathcal{N}$ is trained to approximate the following function $f : \mathbb{R} \to \mathbb{R}$

$$f(x) = -x\cos(x) + \sin(x) \tag{30}$$

and the gradient $\frac{d\mathcal{N}}{dx}$ of DNN $\mathcal{N}$ approximates the gradient $\frac{df}{dx} : \mathbb{R} \to \mathbb{R}$ of the target function f:

$$\frac{d}{dx}f(x) = x\sin(x) \tag{31}$$

Gradient constraint. The target gradient function $\frac{d}{dx}f(x) = x\sin(x)$ is bounded by the upper bound function $g_u(x) = x$ and the lower bound function $g_l(x) = -x$, and we aim to enforce this hard constraint on the DNN gradient $\frac{d}{dx}\mathcal{N}$: for all training samples $\forall x \in \mathcal{D}$, $-x \leq \frac{d}{dx}\mathcal{N}(x) \leq x$.

Setup for `GD` baseline. We use the Adam optimizer with learning rate 0.01, and exponential learning rate decay with $\gamma = 0.997$. We train the DNN for 300 epochs with a batch size of 64, using the following loss function to learn both the DNN output and the gradient:

$$\mathcal{L}(x) = \mathtt{MSE}\big(\mathcal{N}(x), f(x)\big) + \mathtt{MSE}\big(\frac{d}{dx}\mathcal{N}(x), \frac{d}{dx}f(x)\big) \tag{32}$$

Setup for `ProGrad`. We edit the last layer of the `GD` trained DNN, with the following hard constraints

$$\forall \mathbf{x} \in \mathcal{D}.\ -\mathbf{x} \leq \frac{d}{d\mathbf{x}}\mathcal{N}(\mathbf{x}; \widehat{\boldsymbol{\theta}}) \leq \mathbf{x}$$

while minimizing the difference between the DNN outputs and gradients with the reference outputs and gradients.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The theoretical claims in the abstract and introduction are presented in Section 4 and proved in the appendix. The empirical claims in the abstract and introduction are justified in Section 5.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: The limitations are discussed in Section 6.

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [Yes]

Justification: The full set of assumptions and informal proof sketch are provided in Section 4 complemented by formal proofs in the appendix.

Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: Section 4 complemented by the appendix fully disclose all information needed to reproduce the technique, and Section 5 complemented by the appendix fully disclose all information needed to reproduce the experimental results.

Guidelines:

- The answer NA means that the paper does not include experiments.
- If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [No]

   Justification: The experiments Section 5.1–5.2 use open-access benchmarks. For the implementation of our tool, we could not make the code open access due to ongoing IP restrictions.

   Guidelines:

   - The answer NA means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: The experimental settings are presented in Section 5 complemented by further details in the appendix.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [No]

   Justification: Because the proposed algorithm is deterministic, we instead evaluate our approach on a variety of benchmarks and setups to demonstrate its efficacy and efficiency.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.

- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: The evaluation platform and runtime are provided in Section 5 and the appendix.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics.

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: This is discussed in Sections 1 and 6.

    Guidelines:

    - The answer NA means that there is no societal impact of the work performed.

- If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

    Justification: This paper proposes an efficient and provable gradient editing approach of DNNs, which does not involve releasing of data and models that have a high risk for misuse.

    Guidelines:

    - The answer NA means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: Section 5 and the appendix cites the original papers that produced the baselines, tools, models and benchmarks used in the paper.

    Guidelines:

    - The answer NA means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.

- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [NA]

    Justification: We do not introduce new assets in the paper.

    Guidelines:

    - The answer NA means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [NA]

    Justification: This paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [NA]

    Justification: This paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.

- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does not impact the core methodology, scientific rigorousness, or originality of the research, declaration is not required.

    Answer: [NA]

    Justification: The core method development in this research does not involve LLMs as any important, original, or non-standard components.

    Guidelines:

    - The answer NA means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy (`https://neurips.cc/Conferences/2025/LLM`) for what should or should not be described.